# PATCH-LEVEL TRAINING FOR LARGE LANGUAGE MODELS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The prohibitive training costs of Large Language Models (LLMs) have emerged as a significant bottleneck in the development of next-generation LLMs. In this paper, we show that it is possible to significantly reduce the training costs of LLMs without sacrificing their performance. Specifically, we introduce patch-level training for LLMs, in which multiple tokens are aggregated into a unit of higher information density, referred to as a 'patch', to serve as the fundamental text unit for training LLMs. During patch-level training, we feed the language model shorter sequences of patches and train it to predict the next patch, thereby processing the majority of the training data at a significantly reduced cost. Following this, the model continues token-level training on the remaining training data to align with the inference mode. Experiments on a diverse range of models (370M-2.7B parameters) demonstrate that patch-level training can reduce the overall training costs to $0.5\times$, without compromising the model performance compared to token-level training.

## 1 INTRODUCTION

Large Language Models (LLMs, Achiam et al., 2023; Touvron et al., 2023a;b; Team et al., 2023; Bai et al., 2023) have achieved remarkable progress in language understanding and generation, which are primarily attributed to their unprecedented model capacity and the corresponding growth in the volume of training data they require (Kaplan et al., 2020; Hoffmann et al., 2022). However, this scaling up comes with a substantial rise in computational costs, making the training efficiency of LLMs a critical concern. Despite the ongoing efforts on efficient LLMs (Wan et al., 2023), it remains a formidable challenge to reduce training costs without compromising the model performance.

Specifically, the amount of compute (FLOPs) required for training LLMs is approximately proportional to both the number of model parameters $N$ and the number of text units (i.e., tokens) $D$ in the training data. This relationship can be expressed as:

$$C \approx 6ND. \tag{1}$$

Therefore, strategies for reducing training costs can target either the reduction in the number of model parameters $N$ or the number of text units $D$. One prominent approach to reduce the parameter size $N$ is called model growth (Gong et al., 2019; Yang et al., 2020; Chen et al., 2022). Rather than equipping the model with full parameters from the beginning, it advocates for a progressive expansion of the model's parameter size throughout the training phase, thereby reducing the average parameter size during training. Nonetheless, a model's performance hinges on an adequate number of parameters to store extensive knowledge and develop intricate reasoning capabilities. The inadequacy of parameters inherently limits the scope of knowledge and capabilities a model can develop, rendering it challenging to match the performance of training with the full parameter set.

The second pathway to lowering training costs is reducing the number of text units $D$ within the training data. This direction remains largely unexplored but intuitively holds more promise, as the knowledge embedded within training data is sparsely distributed across numerous tokens, with each token encapsulating a minimal amount of information. This sparse distribution of information results in a scenario where, despite the substantial computational costs incurred during each learning step, only a small fraction of model parameters that are relevant to the current token are effectively updated. By increasing the amount of information the model processes at each learning step—that is, by

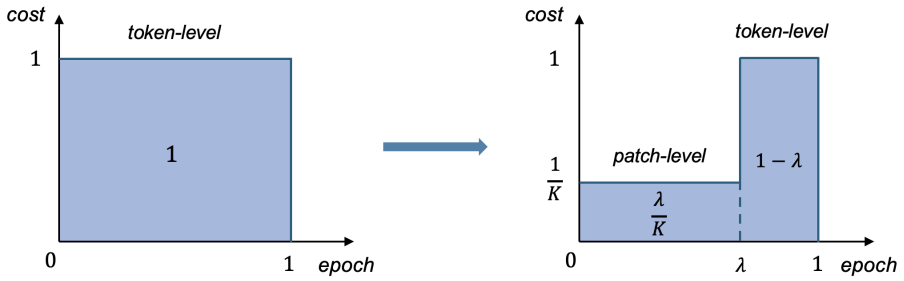


Figure 1: Visualization of overall training costs with patch compression for a fraction $\lambda$ of training data and patch size $K$.

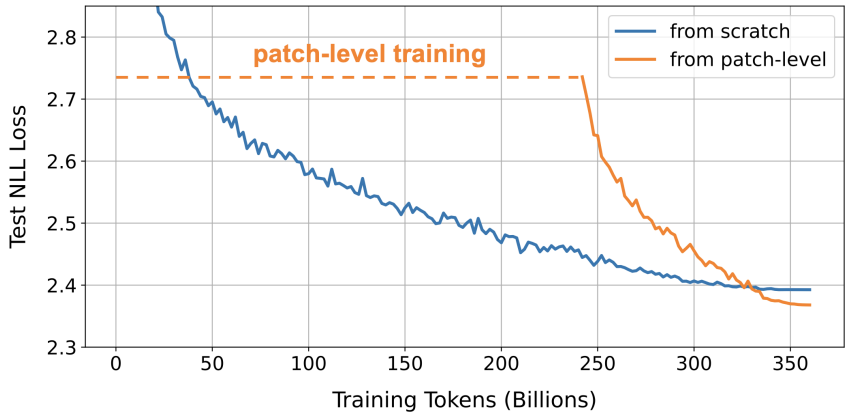


Figure 2: Negative log-likelihood (NLL) loss on test set w.r.t the number of processed tokens during the training of 370M-parameter Transformers.

augmenting the information density of text units and thus reducing the number of text units $D$—we could potentially boost training efficiency significantly without sacrificing model performance.

Building on these insights, this paper introduces patch-level training for large language models, in which multiple tokens are aggregated into a unit of higher information density, referred to as a 'patch', to serve as the fundamental text unit for training LLMs. Specifically, we divide the training process into two stages: patch-level training and token-level training. During patch-level training, we feed the language model shorter sequences of patches and train it to predict the next patch, thereby processing the majority of the training data at a significantly reduced cost. The resulting parameters are used to initialize the token-level model, which then continues training on the remaining data to adapt the knowledge gained during patch-level training to the token-level.

Figure 1 illustrates the efficiency advantage of patch-level training, where the area of the shape represents the overall training costs. With a patch size of $K$, the amount of compute required for patch-level training is $1/K$ of that required for token-level training. When a fraction $\lambda$ of the training data is compressed into patches, the overall training costs are reduced to $\lambda/K + 1 - \lambda$ times the original costs. For instance, to halve the training costs, one could set the patch size $K = 4$ and conduct patch-level training on $\lambda = 2/3$ of the training data.

Employing the above settings ($K = 4, \lambda = 2/3$), we train a series of LLMs of varying sizes (370M-2.7B parameters) on the Pile dataset (Gao et al., 2020). Figure 2 illustrates the trend of NLL loss against the number of training tokens for the 370M model. After initialization with patch-level training, the model experiences a rapid decrease in loss as it continues token-level training on the remaining data. Remarkably, it achieves an even lower loss in comparison with training from scratch, while reducing training costs by half. By further adjusting the hyperparameter settings, even higher acceleration rates can be achieved, with only a slight sacrifice in model performance.

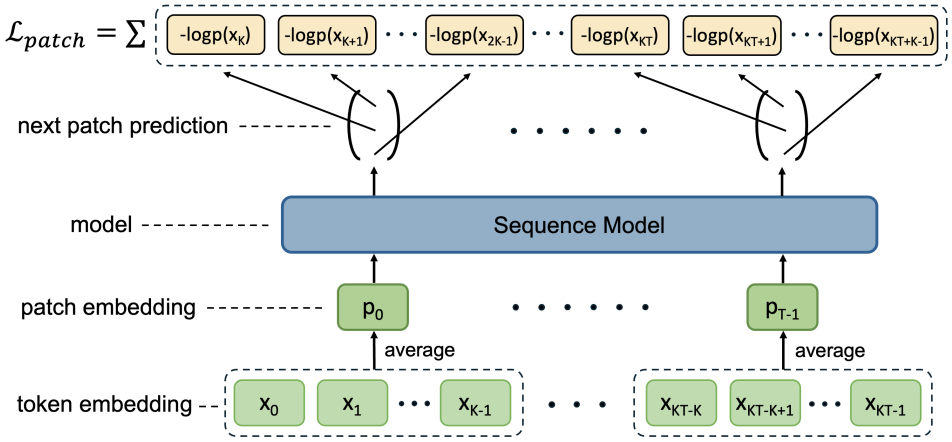


Figure 3: Overview of patch-level training. Every consecutive $K$ token embeddings are averaged to form the patch embedding. The sequence model is fed the patch sequence and trained to predict the next patch. The cross-entropy loss is computed based on each patch prediction vector and all the subsequent $K$ tokens in its next patch.

## 2 Patch-Level Training

In this section, we outline the patch-level training approach for large language models, as illustrated in Figure 3. Initially, the token sequence is transformed into a patch sequence by compressing every $K$ consecutive tokens into a single patch. This patch sequence is then fed into the sequence model, and the model is trained to predict all tokens in the next patch. The knowledge acquired during patch-level training is subsequently transferred to the token-level model. Specifically, we use the parameters obtained from the patch-level model to initialize the token-level model, and then proceed with token-level training on the remaining data.

While formulating the patch-level model structure, our goal is to minimize the discrepancy between patch-level and token-level models, thereby ensuring that the knowledge gained during patch-level training can be smoothly transferred to the token-level model. In practice, we observed that it is crucial for at least one end (input or output) of the patch-level model to remain consistent with the token-level model, which acts as an anchor to align their representations. The detailed ablation is presented in section 3.7.

Given the context length $T$ for token-level training, we set the context length for patch-level training as $KT$, which is then compressed to a patch sequence of length $T$ to maintain consistency with the subsequent token-level training. To avoid introducing unnecessary parameters during token-to-patch compression, we represent the patch embedding as the average of its associated token embeddings. Let $p_i$ be the $i$-th patch, $x_{iK+k}$ be the $k$-th token in the $i$-th patch, and $E$ be the embedding function. The patch embedding is:

$$E(p_i) = \frac{1}{K} \sum_{k=0}^{K-1} E(x_{iK+k}). \tag{2}$$

The patch-level model is trained through next patch prediction, i.e., predicting the $K$ tokens in the next patch. The simultaneous prediction of multiple tokens has been explored in speculative decoding, which typically employs multiple output heads and each head is responsible for predicting a distinct token (Cai et al., 2024; Lin et al., 2024). However, this approach would also entail additional parameters that may be unfavorable for the subsequent knowledge transfer. Instead, we maintain a single output head and make its prediction cover all tokens in the next patch. Specifically, we calculate the cross-entropy loss for all the subsequent $K$ tokens based on the same patch prediction $P(\cdot|p_{<i})$, resulting in the following loss function:

$$\mathcal{L}_{patch} = -\sum_{i=1}^{T} \sum_{k=0}^{K-1} \log P(x_{iK+k}|p_{<i}). \tag{3}$$

Since the model finally works at the token-level, it is essential to reserve some training data to adapt the patch-level model to token-level. Specifically, we conduct patch-level training on a fraction $\lambda$ of the training data, and then use the resulting parameters to initialize the token-level model. Following this, the token-level model continues training on the remaining data to adapt the knowledge gained during patch-level training to the token-level. As illustrated in Figure 1, the overall training costs are reduced to $\lambda/K + 1 - \lambda$ times the original costs of token-level training. When the amount of training data is limited, this approach can also be utilized for efficient multi-epoch training. For example, given a budget of $N$ epochs, we can conduct patch-level training on the first $N\lambda$ epochs, and then switch to token-level training for $N(1-\lambda)$ epochs.

# 3 EXPERIMENTS

## 3.1 SETUP

**Datasets.** We evaluate our approach on standard language modeling tasks, using the Pile dataset (Gao et al., 2020) containing 360B tokens for training [1]. We assess the performance of LLMs from multiple aspects, including their perplexity, zero-shot accuracy, and instruction-following ability. Perplexity is calculated on the WikiText-103 test set (Merity et al., 2017). We evaluate the zero-shot capabilities of language models on 6 NLP benchmarks, including MMLU (Hendrycks et al., 2021), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2020), ARC-E, and ARC-C (Clark et al., 2018) [2]. For the pre-trained LLMs, we conduct instruction fine-tuning using the Alpaca dataset by GPT4 (Taori et al., 2023), and then evaluate their instruction-following abilities on MT-Bench (Zheng et al., 2024).

**Models.** We use the Transformer backbone (Vaswani et al., 2017) and adopt most of the architecture designs from LLaMA (Touvron et al., 2023a). We apply pre-normalization using RMSNorm (Zhang & Sennrich, 2019), use the SwiGLU activation function (Shazeer, 2020), and rotary positional embeddings (Su et al., 2021). We also apply FlashAttention-2 (Dao, 2024) to accelerate attention computation. We scale the model demension and obtain 4 different sizes of Transformers: Transformer-370M (hidden_size=1024, intermediate_size=2752, hidden_layers=24, attention_heads=16), Transformer-780M (hidden_size=1536, intermediate_size=4128, hidden_layers=24, attention_heads=16), Transformer-1.3B (hidden_size=2048, intermediate_size=5504, hidden_layers=24, attention_heads=16), Transformer-2.7B (hidden_size=2560, intermediate_size=6880, hidden_layers=32, attention_heads=32).

**Implementation Details.** Unless otherwise specified, the patch size $K$ is 4. The context length for token-level training $2048$. For patch-level training, the context length is the patch size $K * 2048$. The global batch size is $2M$ tokens, and the total number of training steps is $N = 180000$. For patch-level training, the number of training steps is $N\lambda$, and then the model proceeds with token-level training for $N(1-\lambda)$ steps. After patch-level training, only the obtained model parameters are used for initialization, and all other states like the optimizer and learning rate scheduler are reset. We use the tokenizer of LLaMA2, whose vocabulary size is $32000$. Our models are optimized by the AdamW optimizer (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 1e-8$. The learning rate is $3e-4$ and the cosine learning rate schedule is applied with warmup of $2000$ steps. We use a weight decay of $0.1$ and gradient clipping of $1.0$, and no dropout is applied during training.

## 3.2 MAIN RESULTS

We train a series of LLMs of varying sizes (370M-2.7B parameters) on the Pile dataset. We employ patch-level training with the settings of $K = 4, \lambda = 2/3$, which theoretically reduces the training costs to $0.5\times$. Please refer to Appendix B for the actual speed measurement. For the Transformer-370M, we also explore other choices of $\lambda$ to evaluate its impact. Table 1 presents the performance comparison of the resulting models. Remarkably, our approach consumes only half of the compute

[1] Previous works generally refer to the Pile dataset as having 300B tokens, but our actual measurement is 360B. The discrepancy is likely due to differences in tokenizers; we use the LLaMA2 tokenizer, which has a relatively small vocabulary, possibly resulting in more tokens. The perplexity scores are also incomparable with models using other tokenizers.

[2] https://github.com/EleutherAI/lm-evaluation-harness

Table 1: Performance comparison of Transformers trained on the Pile dataset. $\lambda$ denotes the proportion of training data used for patch-level training, with the patch size $K$ fixed at 4. 'PPL' represents the perplexity score on the WikiText-103 test set. For zero-shot evaluations, we report the normalized accuracy across 6 NLP benchmarks. 'Average' means the average zero-shot accuracy.

| **Model Type** | **Cost** | **PPL** | **MMLU** | **HellaSwag** | **PIQA** | **WinoG** | **ARC-E** | **ARC-C** | **Average** |
|---|---|---|---|---|---|---|---|---|---|
| Transformer-370M | 1.0× | 10.9 | 22.9 | 40.8 | 67.5 | 53.1 | 44.3 | 24.7 | 42.2 |
| + Patch ($\lambda = 1/2$) | 0.625× | 10.6 | 23.5 | 42.0 | 67.9 | 52.1 | 46.1 | 25.6 | 42.9 |
| + Patch ($\lambda = 2/3$) | 0.5× | 10.7 | 23.7 | 41.1 | 68.0 | 51.9 | 46.0 | 24.2 | 42.5 |
| + Patch ($\lambda = 4/5$) | 0.4× | 11.0 | 23.3 | 40.5 | 67.5 | 51.7 | 44.9 | 24.5 | 42.1 |
| Transformer-780M | 1.0× | 9.2 | 24.4 | 48.5 | 69.0 | 55.4 | 49.0 | 26.7 | 45.5 |
| + Patch ($\lambda = 2/3$) | 0.5× | 9.1 | 24.1 | 49.1 | 70.6 | 54.8 | 51.3 | 28.2 | 46.3 |
| Transformer-1.3B | 1.0× | 8.2 | 23.9 | 54.5 | 71.2 | 57.3 | 55.1 | 28.9 | 48.5 |
| + Patch ($\lambda = 2/3$) | 0.5× | 8.2 | 24.3 | 54.1 | 71.6 | 57.8 | 55.6 | 30.4 | 49.0 |
| Transformer-2.7B | 1.0× | 7.1 | 25.3 | 62.2 | 74.3 | 61.5 | 61.2 | 34.3 | 53.1 |
| + Patch ($\lambda = 2/3$) | 0.5× | 7.2 | 25.4 | 61.9 | 74.9 | 62.4 | 61.9 | 34.6 | 53.5 |

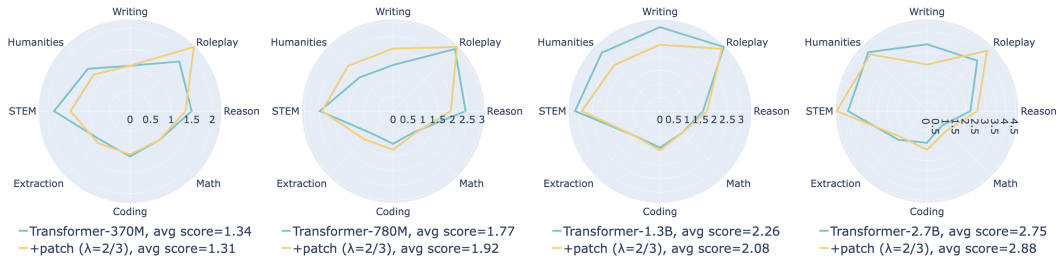


Figure 4: Instruction-following abilities evaluated on MT-bench, a multi-turn question set.

and incurs almost no performance loss. It matches the baseline model in terms of perplexity and even demonstrates a consistent gain in zero-shot evaluations, raising the average accuracy by approximately 0.5%. The model performance is also influenced by the choice of $\lambda$. Within the range of values we set, a smaller $\lambda$ leads to better model performance but also entails larger training costs. A more detailed study on the hyperparameter $\lambda$ will be presented in Section 3.6.

We further conduct instruction fine-tuning using the Alpaca dataset by GPT4 to examine the impact of patch-level training on the model's instruction-following ability. We evaluate our models using MT-Bench, a multi-turn question set, and present the experimental results in Figure 4. As can be seen, our approach maintains a similar instruction-following ability to the original models, with some experiencing a score decrease (Transformer-370M, Transformer-1.3B) and others showing an improvement (Transformer-780M, Transformer-2.7B), which can be viewed as regular variations.

Our primary motivation for patch-level training is to enhance the model's knowledge acquisition efficiency. Interestingly, experimental results show that this approach can sometimes lead to performance improvements, which is beyond our initial expectation. We initially thought that the extended context length in patch-level training contributes to the improvements. However, when we decreased the context length during patch-level training from $KT = 8192$ to $T = 2048$ for Transformer-370M ($\lambda = 1/2$), the model performance only experiences a slight decline (PPL $\uparrow$ 0.06, zero-shot accuracy $\downarrow$ 0.2), yet still surpasses the baseline, implying that context length is not the primary factor. We hypothesize that two other factors might be responsible for these improvements: firstly, the patch-level initialization could potentially serve as a form of regularization; secondly, by compressing consecutive tokens into patches, the model might more effectively recognize and capture long-range dependencies due to the reduced token distance.

Table 2: Performance comparison of Transformers trained on 60B tokens for 6 epochs.

| Model Type | Cost | PPL | MMLU | HellaSwag | PIQA | WinoG | ARC-E | ARC-C | Average |
|---|---|---|---|---|---|---|---|---|---|
| Transformer-370M | 1.0× | 11.0 | 23.6 | 40.8 | 66.5 | 50.8 | 44.8 | 25.2 | 42.0 |
| + Patch ($\lambda = 1/2$) | 0.625× | 10.4 | 23.9 | 43.3 | 67.5 | 55.6 | 44.4 | 26.1 | 43.5 |
| + Patch ($\lambda = 2/3$) | 0.5× | 10.5 | 24.7 | 42.4 | 67.9 | 51.9 | 45.3 | 24.7 | 42.8 |
| + Patch ($\lambda = 4/5$) | 0.4× | 10.7 | 23.0 | 41.5 | 67.0 | 52.0 | 45.1 | 25.4 | 42.3 |

### 3.3 Multi-Epoch Training

Given that patch-level training consumes training data more rapidly, it is more data-hungry compared to token-level training. Consequently, it is essential to consider scenarios where training data is relatively limited and assess the performance of patch-level training when training data is reused for multi-epoch training (Muennighoff et al., 2023). We randomly extract a subset of 60B tokens from the Pile dataset and increase the number of training epochs to $N = 6$. In this way, the model is first trained on patch-level for $N\lambda$ epochs, followed by $N(1 - \lambda)$ epochs of token-level training.

The results presented in Table 2 demonstrate that patch-level training maintains its superiority in terms of training efficiency and performance on multi-epoch training. Intriguingly, when both consuming 360B tokens, patch-level training for multiple epochs even outperforms its single-epoch variant in Table 1. This unexpected advantage may stem from the synergistic effect of integrating patch-level and token-level training on the same data, which likely enhances model regularization. It also suggests that our approach can be data-efficient by initializing the model with patch-level training for one or multiple epochs, offering a promising direction for boosting model performance.

### 3.4 Scaling Properties

In the above, we have validated the effectiveness of patch-level training across several model sizes (370M-2.7B), using a training set of 360B tokens. However, state-of-the-art LLMs are generally trained on model sizes and datasets that are at least an order of magnitude larger than our settings. Therefore, it is crucial to know the scaling properties of patch-level training, i.e., how it performs when applied to larger training datasets and models.

In Table 1, we notice a trend related to the model size: the performance advantage of patch-level training appears to decrease as the model size increases. Table 3 describes this trend more precisely, indicating that the model with patch-level training experiences smaller performance gains from the increase in model size. On the other hand, Table 4 presents the changes of cross-entropy loss when maintaining a constant model size and varying the size of the training data. As the data size increases, the performance of patch-level training improves at a faster rate compared to the baseline model.

Table 3: Test losses when scaling the model size from 370M to 2.7B and training on the Pile dataset (360B tokens). '↓' indicates the loss reduction compared to the previous model size.

| Model Size | 370M | 780M | 1.3B | 2.7B |
|---|---|---|---|---|
| Transformer | 2.392 | 2.217 (↓0.175) | 2.102 (↓0.115) | 1.961 (↓0.141) |
| + Patch ($\lambda = 2/3$) | 2.368 | 2.208 (↓0.160) | 2.108 (↓0.100) | 1.980 (↓0.128) |

Table 4: Test losses of Transformer-370M when scaling the size of training data from 45B to 360B. '↓' indicates the loss reduction compared to the previous data size. The batch size is adjusted to maintain a consistent number of training steps.

| Data Size | 45B | 90B | 180B | 360B |
|---|---|---|---|---|
| Transformer | 2.526 | 2.460 (↓0.066) | 2.423 (↓0.037) | 2.392 (↓0.031) |
| + Patch ($\lambda = 2/3$) | 2.553 | 2.468 (↓0.085) | 2.413 (↓0.055) | 2.368 (↓0.045) |

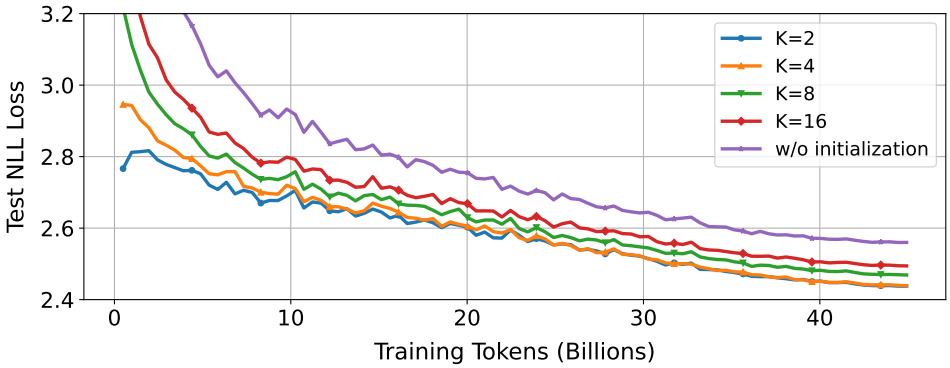


Figure 5: Test losses of Transformer-370M w.r.t the number of processed tokens. Models are initialized by patch-level training with patch size $K$.

This phenomenon can be explained from the perspective of knowledge transfer. As the data size increases, more training data is employed to adjust the model from patch-level to token-level, facilitating a smoother knowledge transfer process. However, an increase in model size implies a greater number of model parameters to be transferred to the token-level, which raises the level of transfer difficulty and necessitates more training data. Based on this explanation, patch-level training is better suited for scenarios with abundant training data.

Note that the above conclusions are drawn under the settings of $K = 4, \lambda = 2/3$, which may vary with changes in the patch size $K$ and the patch-level data fraction $\lambda$. At present, we have not identified a general scaling law for patch-level training that incorporates $K$ and $\lambda$. Instead, we have made some observations regarding their effects on model performance, which will be discussed in the following.

### 3.5 Effect of Patch Size $K$

We investigate the effect of patch size under the settings of 90B training tokens, 370M model parameters, a batch size of 512K, and $\lambda = 1/2$. The results are shown in Figure 5. Across different patch sizes, the loss curves for patch sizes $K = 2$ and $K = 4$ are nearly indistinguishable, while further increasing the patch size to 8 or 16 results in a certain performance decline. Despite this, these models still exhibit significant performance improvements when compared to the model trained from scratch, which does not benefit from the initialization of patch-level training.

Overall, the patch size of $K = 4$ strikes a favorable trade-off between training efficiency and performance. Considering that larger patch sizes can process more data at the same cost, we also experiment with patch-level training using $K = 8$ on 90B tokens, which costs similar compute as $K = 4$ on 45B tokens. Following this, both models proceed with token-level training on 45B tokens, and coincidentally, their loss curves are nearly identical. In this context, the advantage of $K = 4$ lies in its data efficiency, as it achieves similar performance while consuming less data.

### 3.6 Effect of $\lambda$

The hyperparameter $\lambda$ allocates the ratio of training data between patch-level and token-level training. A larger $\lambda$ results in more tokens being compressed into patches, leading to a higher acceleration rate, but it may also leave insufficient data to adjust the model to the token-level. In this section, we investigate the effect of $\lambda$ under the settings of 370M model parameters, a batch size of 512K, and a patch size of $K = 4$. We consider two scenarios:

1. Unlimited computational budget: We assess the impact of varying $\lambda$ while keeping the data size constant (90B tokens). The results are shown in Figure 6.
2. Unlimited training data: We identify the optimal $\lambda$ under a fixed amount of computational budget (tokens + patches = 56.25B). For example, when $\lambda = 1/2$, the size of training data should be 90B tokens, with 45B tokens being compressed into 11.25B patches. The results are shown in Figure 7.

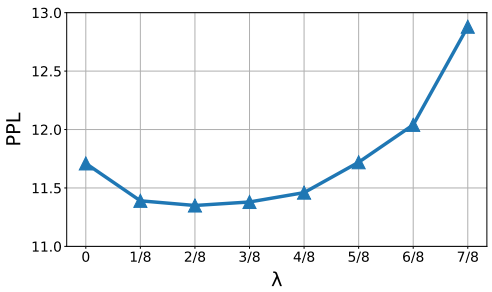


Figure 6: Effect of varying $\lambda$ while keeping the data size constant.

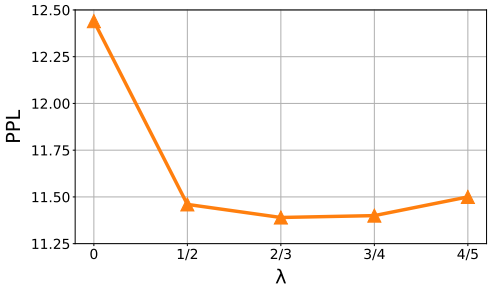


Figure 7: Effect of varying $\lambda$ while keeping the computational cost constant.

Figure 6 shows that the model performance initially rises and later falls as $\lambda$ increases, with a turning point near $\lambda = 1/4$. The performance improvements when $\lambda < 1/4$ can be attributed to the inherent benefits of patch-level training, as analyzed in Section 3.2. When $\lambda$ exceeds $3/4$, further increasing $\lambda$ leaves insufficient data to adjust the model to the token-level, leading to a rapid decline in performance. Figure 7, on the other hand, shows that when computational budget are limited, the optimal value for $\lambda$ is around $2/3$. Note that these conclusions are specific to the current settings and should be used as a reference only. The optimal $\lambda$ may vary depending on factors such as data size and patch size. To determine the optimal value of $\lambda$ in any scenario, it is essential to establish the scaling law for patch-level training.

### 3.7 Effect of Architecture

In the current setup, the architecture of the patch-level model is identical to that of the token-level model, facilitating smoother model transfer but also leading to some concerns. For instance, the token-to-patch transformation and the prediction of the next patch do not take into account the order of tokens within a patch. One may conjecture that patch-level models might benefit from adopting architectures that are better suited for representing and predicting patches, where the additional parameters introduced in such architectures could simply be discarded in the next stage.

We evaluate this strategy under the settings of 90B training tokens, 370M model parameters, a batch size of 512K, and $\lambda = 1/2$. Specifically, we incorporate a linear projection layer at both the input and output sides of the model. On the input side, the conversion of token embeddings into patch embeddings is facilitated through a linear projection $w_{in} \in \mathbb{R}^{Kd \times d}$. On the output side, a linear projection $w_{out} \in \mathbb{R}^{d \times Kd}$ is employed to transform patch-level representations back into token-level representations, followed by $K$ softmax layers to obtain the probability distribution of each token. The effects of these two modules are detailed in Table 5.

Table 5: Impact of architecture modifications in patch-level models. '+InputProj' and '+OutputProj' denote the incorporation of linear projections at model's input and output, respectively. 'Patch PPL' and 'Token PPL' are perplexities of the patch-level and token-level model, respectively.

| **Model** | **Transformer** | **+InputProj** | **+OutputProj** | **+Both** |
|---|---|---|---|---|
| Patch PPL | 159.17 | 146.93 | 99.48 | 86.50 |
| Token PPL | 11.46 | 11.63 | 11.50 | 12.33 |

Overall, while these modifications are effective in reducing the patch-level loss, they do not translate into benefits for the subsequent token-level training. Particularly, when linear projections are applied at both the model input and output, the performance of the subsequent token-level model significantly declines. It suggests that it is crucial for at least one end (input or output) of the patch-level model to remain consistent with the token-level model, which acts as an anchor to align their representations. It also shows that there is no direct correlation between patch-level loss and the final performance of the token-level model, so a large loss during patch-level training does not imply ineffective learning. Therefore, we opt to preserve the original Transformer architecture for patch-level training.

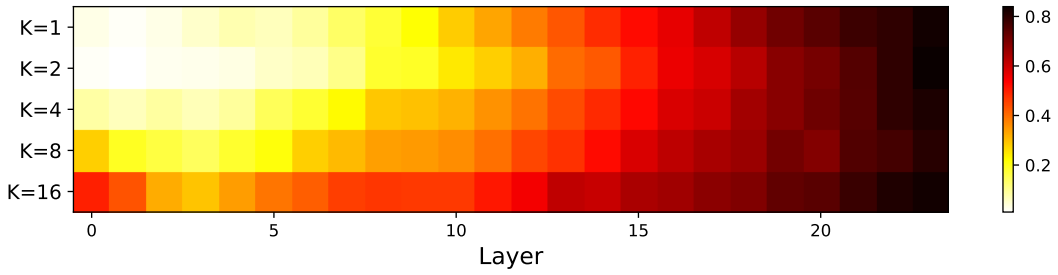


Figure 8: Percentage of activated neurons for models of different patch sizes. Output neurons of each model layer (FFN output) with an absolute value greater than 0.5 are classified as activated.

### 3.8 Neuron Activation

In this section, we quantitatively explain why patch-level training leads to better learning efficiency from the perspective of neuron activation. The training of LLMs is essentially a process of embedding knowledge from the training set into the model's parameters. During this process, the model employs all of its parameters to encode every token and updates the relevant parameters based on the gradient feedback. We argue that this is an inefficient process for large models, as the knowledge encapsulated in each token is only associated with a small subset of model parameters, resulting in a limited number of effectively activated and updated parameters.

We substantiate this by measuring the percentage of activated neurons for models of different patch sizes, as depicted in Figure 8. In the token-level model ($K=1$), only a small proportion of neurons are activated, suggesting that the model has enough capacity to handle text units with higher information density, thus indicating significant room for improvement in training efficiency. By grouping multiple tokens into a patch, the information density processed at each step is increased, which is manifested as increased neuron activation rates. From this observation, it becomes evident that patch-level training makes more comprehensive utilization of the model's capabilities, therefore demonstrating higher training efficiency.

## 4 Related Work

**Model Growth.** Our approach draws inspiration from transfer learning, reducing training costs by transferring knowledge acquired at a lower training cost (patch-level) to a model with a higher training cost (token-level). A similar strategy has been employed in studies of model growth, which train large models at a relatively lower cost by progressively increasing the model size during training. For example, Gong et al. (2019); Yang et al. (2020) improve the training efficiency by transferring knowledge from a shallow model to a deep model, where model layers are progressively stacked during training. Gu et al. (2021) further proposes progressive compound growth, where the model grows at multiple dimensions during training, including the context length, model width, and the number of layers. Subsequent studies primarily focus on the initialization problem during the model growth process, i.e., how to expand the small model into a large one. Chen et al. (2022); Yao et al. (2024) aim to achieve function-preserving growth (Chen et al., 2015) that the post-growth model have the same function as the pre-growth model, which intuitively ensures smooth knowledge transfer. Wang et al. (2023); Pan et al. (2023) introduce learnable linear operators that linearly map the parameters of the small model to initialize the large model. Compared to model growth, patch-level training is more flexible and generalizable as it does not necessitate specialized model architectures or carefully crafted model mapping strategies. More importantly, patch-level training merely alters the information density of text units while preserving the full set of model parameters. Therefore, it enables the model to develop its complete capabilities without compromising its performance.

**Multi-Token Prediction.** Our approach improves training efficiency by concurrently predicting all tokens in the next patch. Similar attempts of multi-token prediction have been made in the past to improve the inference efficiency, including non-autoregressive generation (Gu et al., 2018) and speculative decoding (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023). Non-autoregressive

generation reduces the number of decoding iterations by generating all tokens at once, involving techniques such as knowledge distillation (Kim & Rush, 2016; Shao et al., 2022), training objectives (Shao et al., 2021; Ghazvininejad et al., 2020; Du et al., 2021; Shao & Feng, 2022; Ma et al., 2023), latent modeling (Kaiser et al., 2018; Ma et al., 2019; Shu et al., 2020), iterative decoding (Lee et al., 2018; Ghazvininejad et al., 2019; Gu et al., 2019), and expressive model architectures (Libovický & Helcl, 2018; Huang et al., 2022; Gui et al., 2023). Despite the reduction in decoding iterations, the computational demands (FLOPs) do not decrease and may even rise for certain structures, leading to increased training costs. Speculative decoding is a novel decoding paradigm for accelerating LLM inference. During each step of decoding, it drafts several future tokens efficiently and then verifies them in parallel. The model for generating draft tokens can either be a relatively small model (Leviathan et al., 2023; Chen et al., 2023) or a non-autoregressive model that generates multiple tokens in parallel (Cai et al., 2024; Lin et al., 2024; Fu et al., 2024; Li et al., 2024). Recently, Gloeckle et al. pointed out that besides accelerating the inference, multi-token prediction also serves as an auxiliary training task to enhance the training signal. The key difference between our approach and these works lies in our utilization of multi-token prediction for reducing sequence length during training, with the goal of improving training efficiency. Regarding the model structure, we avoid introducing extra parameters and employ a single head for multi-token prediction.

**Patch-Level Models.** The concept of handling input data at the patch-level has emerged as a pivotal strategy for enhancing computational efficiency and capturing local features. Convolutional Neural Networks (CNNs, Lecun et al., 1998) are perhaps the earliest attempt of patch-level processing, utilizing kernel filters to extract local features from images. More recently, Vision Transformers (Dosovitskiy et al., 2021) have revolutionized image processing by employing CNNs to encode an image into non-overlapping image patches, thereby enabling Transformers to efficiently capture both local and global features. Similarly, speech models also rely on CNNs to compress high-frequency waveforms into hidden state representations (Baevski et al., 2020; Hsu et al., 2021), which can be interpreted as speech patches. Dosovitskiy et al. (2021) suggested that the Vision Transformer should be fine-tuned at higher resolution, a strategy similar to ours, which improves data precision during the later phases of training. In subsequent studies, the impact of patch sizes has also garnered attention. Beyer et al. (2023) proposed to randomize the patch size at training time, which creates models capable of handling diverse patch sizes and adapting to different compute budgets. Further exploration by Anagnostidis et al. (2024) revealed that training with larger patch sizes is generally more computationally efficient. This insight led to a training strategy that progressively decreases the patch size, enabling more efficient training of vision Transformers. For textual data, characters, the basic building blocks of text, can be downsampled into more compact representations (Clark et al., 2022) or merged into tokens (Sennrich et al., 2016; Kudo & Richardson, 2018). Recently, there have been attempts to further compress tokens into patches, with the model directly processing the patch sequence (Nawrot et al., 2022; Mujika, 2023; Yu et al., 2024; Ho et al., 2024). However, it remains necessary to upsample the patch representation and input it into a token-level autoregressive model for the likelihood inference.

## 5 Conclusion

This paper introduces patch-level training, an efficient training approach for large language models, in which multiple tokens are aggregated into a unit of higher information density, referred to as a 'patch', to serve as the fundamental text unit for training LLMs. During patch-level training, the model reads training data in patches and learns to predict the next patch. Following this, a small amount of training data is utilized to adjust the model to the token-level. Experimental results show that this approach can cut LLM training costs by 50% while maintaining comparable performance.

Yet, our exploration of patch-level training is still in its infancy, and advancements in the following directions could further enhance this methodology: assessing the scalability of patch-level training by evaluating its performance on larger models and datasets; establishing an empirical scaling law for patch-level training, ideally incorporating both $K$ and $\lambda$; developing advanced training techniques to accommodate larger $K$ and $\lambda$, thereby pushing acceleration rates to a higher level; further investigating the potential of patch-level training in multi-epoch training; exploring the applicability of patch-level training to other data modalities, such as images, speech, and video.

## REFERENCES


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Sotiris Anagnostidis, Gregor Bachmann, Imanol Schlag, and Thomas Hofmann. Navigating scaling laws: Compute optimality in adaptive model training. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1511–1530. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/anagnostidis24a.html.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Lucas Beyer, Pavel Izmailov, Alexander Kolesnikov, Mathilde Caron, Simon Kornblith, Xiaohua Zhai, Matthias Minderer, Michael Tschannen, Ibrahim Alabdulmohsin, and Filip Pavetic. Flexivit: One model for all patch sizes. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14496–14506, 2023. doi: 10.1109/CVPR52729.2023.01393.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr. 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

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

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. bert2BERT: Towards reusable pretrained language models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2134–2148, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.151. URL https://aclanthology.org/2022.acl-long.151.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022. doi: 10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Cunxiao Du, Zhaopeng Tu, and Jing Jiang. Order-agnostic cross entropy for non-autoregressive machine translation. In *International conference on machine learning*, pp. 2849–2859. PMLR, 2021.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=eDjvSFOkXw.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 6112–6121, 2019. URL https://www.aclweb.org/anthology/D19-1633.

Marjan Ghazvininejad, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. Aligned cross entropy for non-autoregressive machine translation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3515–3523. PMLR, 2020. URL http://proceedings.mlr.press/v119/ghazvininejad20a.html.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Forty-first International Conference on Machine Learning*.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of BERT by progressively stacking. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2337–2346. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/gong19a.html.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1l8BtlCb.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 11179–11189, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/675f9820626f5bc0afb47b57890b466e-Abstract.html.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. On the transformer growth for progressive BERT training. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5174–5180, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.406. URL https://aclanthology.org/2021.naacl-main.406.

Shangtong Gui, Chenze Shao, Zhengrui Ma, xishan zhang, Yunji Chen, and Yang Feng. Non-autoregressive machine translation with probabilistic context-free grammar. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 5598–5615. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/11c7f1dd168439884b6dfb43a7891432-Paper-Conference.pdf.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Namgyu Ho, Sangmin Bae, Taehyeon Kim, Hyunjik Jo, Yireun Kim, Tal Schuster, Adam Fisch, James Thorne, and Se-Young Yun. Block transformer: Global-to-local language modeling for fast inference. *arXiv preprint arXiv:2406.02657*, 2024.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021.

Fei Huang, Hao Zhou, Yang Liu, Hang Li, and Minlie Huang. Directed acyclic transformer for non-autoregressive machine translation. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9410–9428. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/huang22m.html.

Lukasz Kaiser, Samy Bengio, Aurko Roy, Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2390–2399, StockholmsmÃ€ssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/kaiser18a.html.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1139. URL https://www.aclweb.org/anthology/D16-1139.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Eduardo Blanco and Wei Lu (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1182, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1149. URL https://www.aclweb.org/anthology/D18-1149.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Zeping Li, Xinlong Yang, Ziheng Gao, Ji Liu, Zhuang Liu, Dong Li, Jinzhang Peng, Lu Tian, and Emad Barsoum. Amphista: Accelerate llm inference with bi-directional multiple drafting heads in a non-autoregressive style. *arXiv preprint arXiv:2406.13170*, 2024.

Jindřich Libovický and Jindřich Helcl. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3016–3021, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1336. URL https://www.aclweb.org/anthology/D18-1336.

Feng Lin, Hanling Yi, Hongbin Li, Yifan Yang, Xiaotian Yu, Guangming Lu, and Rong Xiao. Bita: Bi-directional tuning for lossless acceleration in large language models. *arXiv preprint arXiv:2401.12522*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4282–4292, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1437. URL https://www.aclweb.org/anthology/D19-1437.

Zhengrui Ma, Chenze Shao, Shangtong Gui, Min Zhang, and Yang Feng. Fuzzy alignments in directed acyclic graph for non-autoregressive machine translation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=LSz-gQyd0zE.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Niklas Muennighoff, Alexander M Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=j5BuTrEj35.

Asier Mujika. Hierarchical attention encoder decoder. *arXiv preprint arXiv:2306.01070*, 2023.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 1559–1571, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.117. URL https://aclanthology.org/2022.findings-naacl.117.

Yu Pan, Ye Yuan, Yichun Yin, Zenglin Xu, Lifeng Shang, Xin Jiang, and Qun Liu. Reusing pretrained models by multi-linear operators for efficient training. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=RgNXKIrWyU.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, Apr. 2020. doi: 10.1609/aaai.v34i05.6399. URL https://ojs.aaai.org/index.php/AAAI/article/view/6399.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Chenze Shao and Yang Feng. Non-monotonic latent alignments for ctc-based non-autoregressive machine translation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 8159–8173. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/35f805e65c77652efa731edc10c8e3a6-Paper-Conference.pdf.

Chenze Shao, Yang Feng, Jinchao Zhang, Fandong Meng, and Jie Zhou. Sequence-Level Training for Non-Autoregressive Neural Machine Translation. *Computational Linguistics*, 47(4):891–925, 12 2021. ISSN 0891-2017. doi: 10.1162/coli_a_00421. URL https://doi.org/10.1162/coli_a_00421.

Chenze Shao, Xuanfu Wu, and Yang Feng. One reference is not enough: Diverse distillation with reference selection for non-autoregressive translation. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3779–3791, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.277. URL https://aclanthology.org/2022.naacl-main.277.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Raphael Shu, Jason Lee, Hideki Nakayama, and Kyunghyun Cho. Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8846–8853. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6413.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 1, 2023.

Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. Learning to grow pretrained models for efficient transformer training. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=cDYRS5iZ16f.

Cheng Yang, Shengnan Wang, Chao Yang, Yuechuan Li, Ru He, and Jingqiao Zhang. Progressively stacking 2.0: A multi-stage layerwise training method for bert training speedup. *arXiv preprint arXiv:2011.13635*, 2020.

Yiqun Yao, Zheng Zhang, Jing Li, and Yequan Wang. Masked structural growth for 2x faster language model pre-training. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=rL7xsg1aRn.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. Megabyte: Predicting million-byte sequences with multiscale transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

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

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1Ddp1-Rb.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

## A PSEUDOCODE

```
# Model input
num_patches = seq_length // self.patch_size
inputs_embeds = inputs_embeds.view(batch_size, num_patches, self.
    patch_size, -1).mean(2)
position_ids = position_ids[:, :num_patches]

...

# Model output
shift_logits = logits[..., :-1, :].reshape(-1, self.config.vocab_size)
shift_labels = labels[..., self.patch_size:].reshape(-1, self.patch_size)
loss = 0
log_probs = F.log_softmax(shift_logits, dim=1)
for i in range(self.patch_size):
    loss = loss + F.nll_loss(log_probs, shift_labels[:, i])
loss = loss / self.patch_size
```

## B SPEED MEASUREMENT

The practical speed-up achieved through patch-level training falls short of the theoretical $K\times$ improvement, primarily due to overheads of data loading and gradient synchronization. Table 6 gives the actual running speed of patch-level training in comparison with the token-level baseline setting (patch_size=1, block_size=2048, per_device_train_batch_size=4, accumulation_steps=8), measured on 8 NVIDIA A100 GPUs. Increasing the patch size to 4 and the block length to 8192 results in a speed-up of approximately 3.8×, albeit with some efficiency loss due to data loading. In practice, the gradient accumulation steps should be reduced to 2 to keep the total batch size constant, which brings the speed-up down to around 3.5× due to more frequent gradient synchronizations. Consequently, the actual runtime is approximately $\frac{2}{3} \times \frac{1}{3.5} + \frac{1}{3} \approx 0.523\times$, which is slightly larger than the theoretical cost of $0.5\times$.

Table 6: Speed comparison between token-level and patch-level training.

| Settings | Tokens / sec | Speed-up |
|---|---|---|
| Baseline | 247.1K | 1.0× |
| patch_size=4, block_size=8192 | 933.5K | 3.78× |
| patch_size=4, block_size=8192, accumulation_steps=2 | 864.2K | 3.50× |

## C MIXUP TRAINING

The core idea of this work is to aggregate multiple tokens into a unit of higher information density, which we refer to as a 'patch'. Our current strategy compresses every consecutive $K$ tokens into a single patch. Here we explore an alternative approach inspired by mixup data augmentation (Zhang et al., 2018). This method involves randomly selecting $K$ training samples and mixing their tokens position-wise to form a patch sequence. Similarly, we train the model with the next patch prediction objective.

We conduct experiments under the settings of 90B training tokens, 370M model parameters, a batch size of 512K, and $\lambda = 1/2$. The results are shown in Figure 9. Mixup training offers some improvements over the model trained from scratch; however, its performance significantly lags behind that of patch-level training. The primary difference between these methods lies in the composition of the patches, suggesting that it is more beneficial to form patches from consecutive tokens. We hypothesize that by compressing consecutive tokens into patches, the model may more effectively recognize and capture long-range dependencies, thanks to the reduced distance between tokens.

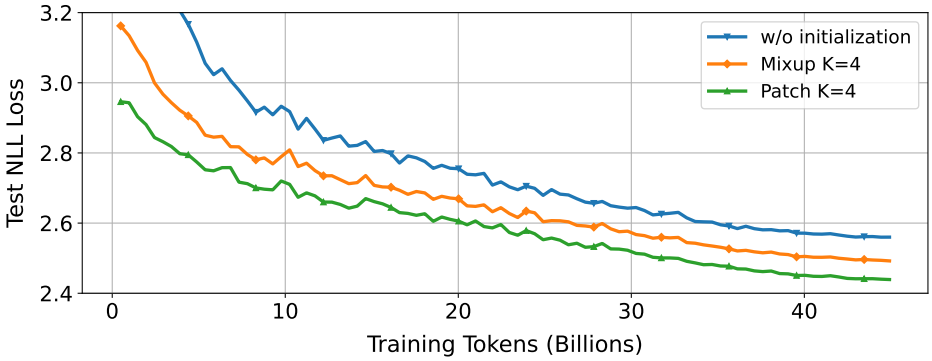


Figure 9: Test losses of Transformer-370M w.r.t the number of processed tokens. Models are initialized by mixup training or patch-level training with patch size $K$.

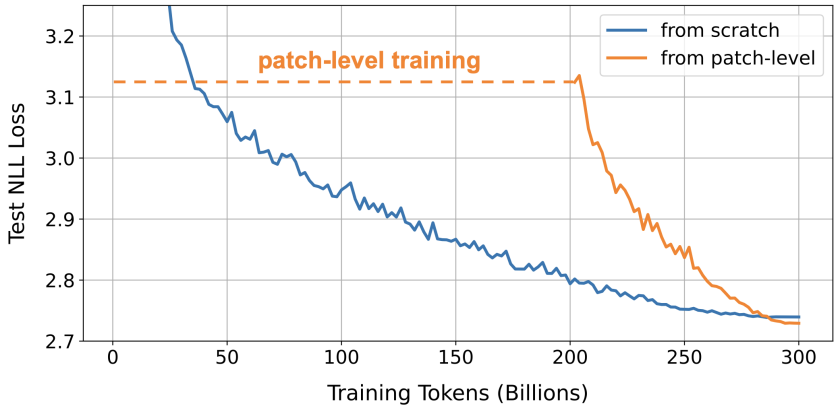


Figure 10: Negative log-likelihood (NLL) loss on test set w.r.t the number of processed tokens during the training of 370M-parameter Transformers. The LLaMA3 tokenizer is used.

## D EFFECT OF TOKENIZER

Our approach enhances training efficiency by increasing the information density of text units and reducing their quantity. Another dimension that works similarly is the choice of tokenizer, as a tokenizer with a larger vocabulary typically represents the same data with fewer tokens. This raises an interesting question about how the efficacy of patch-level training is affected by the tokenizer. The LLaMA2 tokenizer, used in our initial experiments, has a vocabulary size of 32,000. Here, we extend our study to include the LLaMA3 tokenizer (Dubey et al., 2024), whose vocabulary size is 128,000, to observe whether our approach remains effective with tokenizers that have a higher compression rate.

We continue to use the Pile dataset for the training, which comprises approximately 300B tokens when encoded with the LLaMA3 tokenizer, corresponding to about 85% of the token count compared to the LLaMA2 tokenizer. Figure 10 depicts the loss curves for the models trained using patch-level training ($K = 4, \lambda = 2/3$) and from scratch at the token-level. Table 7 presents a performance comparison between the two approaches. Remarkably, patch-level training remains its effectiveness, achieving slightly better performance than token-level training while halving the training cost.

The choice of patch size, we hypothesize, is also swayed by the tokenizer's compression rate. When the token representations are less compact, a larger patch size might be employed to increase information density, and vice versa. We validate this hypothesis by comparing the effects of patch sizes $K = 2$ and $K = 4$ under the settings of 75B training tokens, 370M model parameters, a batch size of 512K, $\lambda = 1/2$, and using the LLaMA3 tokenizer. The results are illustrated in Figure 11.

Table 7: Performance comparison of Transformers trained on the Pile dataset. The LLaMA3 tokenizer is used.

| Model Type | Cost | PPL | MMLU | HellaSwag | PIQA | WinoG | ARC-E | ARC-C | Average |
|---|---|---|---|---|---|---|---|---|---|
| Transformer-370M | 1.0× | 15.48 | 22.9 | 41.2 | 67.3 | 54.1 | 45.6 | 25.8 | 42.8 |
| + Patch ($\lambda = 2/3$) | 0.5× | 15.32 | 23.1 | 41.1 | 68.1 | 53.2 | 45.9 | 26.2 | 42.9 |

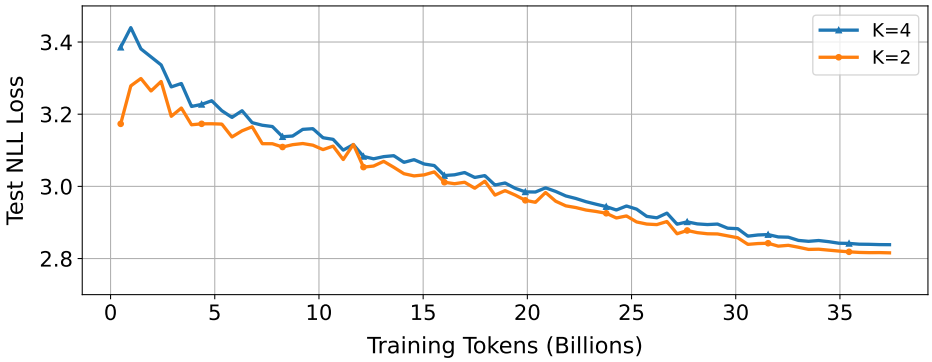


Figure 11: Test losses of Transformer-370M w.r.t the number of processed tokens. Models are initialized by patch-level training with patch size $K$. The LLaMA3 tokenizer is used.

It is observed that there is a slight performance gap between patch sizes $K = 2$ and $K = 4$ at this setting, whereas, in comparison, the loss curves for these patch sizes using the LLaMA2 tokenizer are almost indistinguishable in Figure 5. This confirms our hypothesis that the suitable patch size varies with the tokenizer. Nevertheless, we posit that a patch size of $K = 4$ is still more advantageous under this setting, offering significant efficiency improvements at the cost of a minor performance trade-off.