Efficient Transformer Training: Reducing Padding Overhead with Length Truncation and Bucketing

Jiyeon Lee
AIFFEL Research
Republic of Korea
jylee4747@gmail.com

Abstract

Training Transformer-based language models often suffers from significant computational inefficiency due to excessive padding caused by variable-length input sequences. In this work, we investigate two simple yet effective strategies for mitigating this inefficiency: sequence length truncation based on a fixed maximum length (*max_len*) and length-based bucketing. Using a KLUE BERT classifier on the Korean NSMC sentiment dataset, we systematically compare four configurations that combine or isolate length truncation and bucketing. Our experiments show that bucketing alone reduces training time by more than 40% without degrading classification performance, while coverage-aware *max_len* truncation yields an additional average speedup of 5.8% with negligible impact on accuracy and loss. Furthermore, the runtime reduction due to truncation is amplified by a factor of approximately 8.6 when combined with bucketing, which we explain through the quadratic computational complexity of self-attention. Overall, our results demonstrate that combining sequence length truncation with bucketing provides a practical, model-agnostic, and easy-to-implement strategy for optimizing Transformer training efficiency in resource-constrained environments.

1 Introduction

Transformer-based models have become the dominant architecture in natural language processing due to their strong modeling capacity and scalability. However, this performance comes at the cost of high computational complexity, making training efficiency a critical concern in resource-constrained environments such as Google Colab.

A common approach for handling variable-length text inputs is to pad all sequences to a fixed maximum length. While convenient for batching, this strategy introduces substantial inefficiency: short sequences receive large amounts of padding, causing the model to waste computation on tokens that carry no semantic information.

This issue is particularly pronounced in real-world datasets, which rarely exhibit uniform length distributions. Instead, most datasets consist primarily of short sentences accompanied by a small number of much longer examples. Under random batching, these long sequences often determine the effective padding length of entire mini-batches, resulting in significant computational waste.

Motivated by this observation, we investigate length-based bucketing as a practical strategy for improving training efficiency. By grouping sequences of similar lengths into the same mini-batch, bucketing directly mitigates padding inefficiency caused by length imbalance. We further examine how bucketing interacts with sequence length truncation based on a fixed *max_len*, which constrains the extreme tail of the length distribution.

Through controlled experiments on a Korean sentiment classification task, we show that bucketing serves as the primary mechanism for exploiting length imbalance in practice, and that sequence

length truncation becomes significantly more effective when combined with bucketing. Together, these techniques yield substantial reductions in training time without degrading model performance.

Contributions The main contributions of this work are as follows:

- We analyze padding-induced inefficiency in Transformer training from the joint perspective of model characteristics and empirical data length distributions.
- We systematically evaluate the effects of length-based bucketing and sequence length truncation, both individually and in combination.
- We show that bucketing alone yields large training-time reductions, while truncation provides additional efficiency gains primarily when paired with bucketing.
- We provide an intuitive, data-driven explanation of why bucketing amplifies the effectiveness of truncation under skewed length distributions.

2 Background and Dataset

This section establishes the theoretical and empirical background that motivates the proposed length-aware training strategies. We first explain why Transformer-based models are inherently sensitive to sequence length, then discuss how real-world datasets exhibit strong length imbalance. Finally, we position our approach relative to prior work that addresses efficiency by modifying model architectures.

2.1 Length Sensitivity of Self-Attention

Transformer models rely on self-attention mechanisms that explicitly model pairwise interactions between all tokens in an input sequence [3]. A fundamental property of self-attention is that its computational and memory complexity scale quadratically with respect to the input sequence length L.

For a dataset consisting of N sequences with lengths $\{L_i\}_{i=1}^{N}$, the total computational cost of self-attention can be expressed as

$$\sum_{i=1}^{N} O(L_i^2). \tag{1}$$

This formulation highlights that longer sequences contribute disproportionately to overall computational cost.

In practical mini-batch training, sequences of varying lengths must be padded to a common length. Importantly, padding tokens are processed identically to real tokens inside the self-attention layers and therefore incur the same computational cost, despite carrying no semantic information [1]. As a result, even a small number of long sequences can dominate the computation time of an entire batch.

This quadratic length sensitivity has been widely recognized as a key efficiency bottleneck in Transformer-based architectures [2], motivating a broad line of research on efficient sequence modeling.

2.2 Architecture-Based Approaches to Efficient Attention

Several studies have attempted to reduce the computational burden of self-attention by modifying the model architecture itself. One representative line of work replaces attention mechanisms with convolution-based alternatives. For example, Wu et al. [4] propose lightweight and dynamic convolutions that achieve linear-time complexity with respect to sequence length while maintaining competitive performance.

While such approaches improve asymptotic efficiency, they require architectural changes and deviate from standard Transformer designs. As a result, they are not always directly compatible with pretrained Transformer models or existing training pipelines.

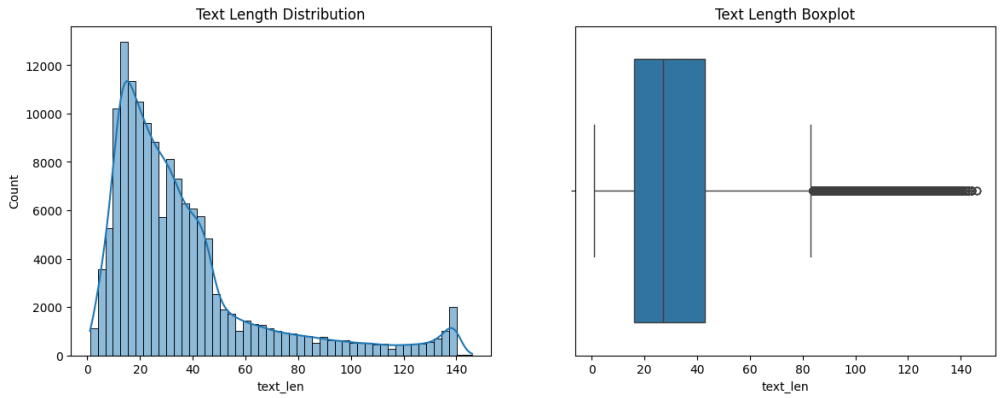

Figure 1: Length distribution of the NSMC dataset before preprocessing. Most samples are short, while a small number of long reviews form a heavy right tail and appear as outliers. This imbalance implies that padding cost during batch-based Transformer training can be dominated by a few long sequences.

In contrast, the focus of this work is not to alter the model architecture, but to improve training efficiency through data preprocessing and batching strategies that can be applied to off-the-shelf Transformer models without modification.

2.3 Length Imbalance in Real-World Text Data

Real-world natural language datasets rarely consist of sequences with uniform lengths. Instead, they typically exhibit a highly skewed distribution in which most samples are relatively short, while a small fraction of samples are significantly longer and form a long right tail.

This characteristic is clearly observed in the NSMC (Naver Sentiment Movie Corpus) dataset used in this study. Both character-level and token-level length distributions reveal that the majority of movie reviews fall within short length ranges, whereas a small number of long reviews substantially extend the upper bound of the distribution. Boxplot analysis further confirms the presence of numerous outliers.

Under naive random batching, such length imbalance causes the padding length of a mini-batch to be determined by a small number of long sequences, resulting in substantial computational waste for the majority of shorter samples.

2.4 Dataset Description

We use the NSMC dataset, a widely adopted benchmark for Korean sentiment classification. The dataset consists of movie reviews collected from Naver, each annotated with a binary sentiment label indicating positive or negative sentiment. We follow the official train/test split provided by Hugging Face.

The wide range of input lengths in NSMC makes it particularly suitable for studying padding-induced inefficiencies and evaluating length-aware training strategies under realistic data conditions.

2.5 Motivation for Bucketing and Truncation

Given the strong length sensitivity of self-attention and the pronounced length imbalance observed in the dataset, length-based bucketing emerges as a natural and effective solution. By grouping sequences of similar lengths into the same mini-batches, bucketing prevents a small number of long sequences from inflating the padding cost for many short ones, without discarding data or modifying the model architecture.

Sequence length truncation based on a fixed *max_len* provides a complementary mechanism by constraining the extreme tail of the length distribution. As illustrated in Figure 2, truncation reduces the global upper bound of sequence length while preserving the overall distributional structure.

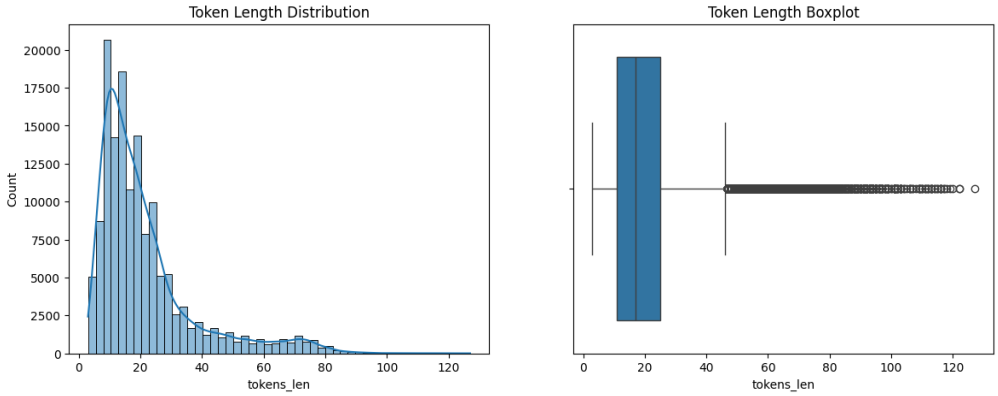

Figure 2: Length distribution of the NSMC dataset after max_len-based truncation. The extreme tail of the distribution is constrained while the overall skewed structure remains, indicating reduced length variance without fundamental distortion of the data distribution.

However, truncation alone does not eliminate padding inefficiency at the mini-batch level. Even after truncation, random batching may still mix sequences of substantially different lengths. This observation suggests that truncation is most effective when combined with bucketing.

From this perspective, truncation reduces the global range of sequence lengths, while bucketing minimizes padding variance within batches. Together, these two strategies act in a complementary manner, directly motivating the experimental design explored in the following section.

3 Experiment

Based on the dataset analysis presented in the previous section, this section describes the experimental setup designed to evaluate the effects of length-based preprocessing and bucketing on training efficiency and model performance. All experiments are conducted under controlled conditions, with only the length handling strategy varied across settings.

3.1 Task and Dataset

We conduct experiments on a binary sentiment classification task using Korean movie reviews. Each input consists of a single review sentence, and the output is a binary label indicating sentiment polarity, where positive reviews are labeled as 1 and negative reviews as 0. The NSMC dataset provided by Hugging Face is used with the official train/test split.

The train and test splits are loaded using the Hugging Face `datasets` library and converted into Pandas DataFrames. Only the review text (`document`) and the sentiment label (`label`) are retained for analysis. A text preprocessing function is applied to normalize raw reviews by removing unnecessary whitespace and normalizing repeated punctuation. The cleaned text is stored in a `clean_text` column and used as input to the tokenizer.

The cleaned text is tokenized using the KLUE BERT tokenizer, and the token length of each sentence is computed. A full training dataset (`train_full_df`) is constructed using all samples. To analyze the effect of sequence length truncation, an additional dataset (`train_len95_df`) is created by retaining only samples with token lengths less than or equal to 95, a threshold selected based on coverage analysis of the token length distribution.

For each configuration, the data is split into training and validation sets at an 8:2 ratio using stratified sampling. Tokenization is performed without padding or truncation, producing only token sequences. The resulting data is converted into Hugging Face `Dataset` objects and kept in Python list format to enable length-based bucketing during training. The test dataset undergoes the same preprocessing and tokenization steps and is likewise stored in Python list format.

3.2 Model and Training Setup

We use `AutoModelForSequenceClassification` with the `klue/bert-base` checkpoint. The model is configured with two output labels corresponding to the binary sentiment classification task.

Dynamic padding is applied at the mini-batch level using `DataCollatorWithPadding`. When combined with length-based bucketing, this approach minimizes padding within each batch and reduces computational overhead.

Training arguments are defined using a helper function. The main hyperparameters are as follows: learning rate 2×10^{-5}, batch size 8, three training epochs, weight decay of 0.01, and evaluation at each epoch. Bucketing is controlled via the `group_by_length` flag. All experiments are logged using Weights & Biases.

Model performance is evaluated using accuracy from the Hugging Face `evaluate` library. Predictions are obtained by applying an argmax operation to the output logits. Each experiment initializes a new model instance to ensure identical starting weights. Training and validation are performed using the Hugging Face `Trainer`. After training, the model is evaluated on the test set, and metrics related to performance and runtime are collected for analysis.

3.3 Length Handling Configurations

We consider four experimental configurations by crossing two factors: application of length truncation (max_len=95) and application of bucketing. All other settings are kept constant.

- **full_plain**: no truncation, no bucketing
- **len95_plain**: truncation only
- **full_bucket**: bucketing only
- **len95_bucket**: truncation and bucketing

For each configuration, we collect test loss, test accuracy, runtime, samples per second, and steps per second. All experiments are conducted on a single NVIDIA A100 GPU. Although the A100 is a high-performance accelerator, we deliberately restricted the batch size to 8 (as defined in Section 3.2) to simulate the memory-constrained training conditions typical of personal computing resources (e.g., Google Colab free tier). Training logs and evaluation metrics are recorded using Weights & Biases for further analysis and visualization.

4 Results

This section reports experimental results comparing classification performance and training efficiency across four configurations: *full_plain*, *len95_plain*, *full_bucket*, and *len95_bucket*. These configurations differ only in whether sequence length truncation (max_len=95) and/or length-based bucketing is applied; all other factors are held constant.

4.1 Classification Performance

Across all configurations, the model achieved consistently high classification performance. Test accuracy exceeded 90% in all cases, and test loss converged below 0.5, indicating stable optimization regardless of the length handling strategy.

Applying sequence length truncation (*len95*) resulted in a very small performance change compared to the full-length setting. On average, accuracy decreased by less than 0.1 percentage points, while loss increased by approximately 0.01. These differences are negligible in practice, suggesting that truncating long sequences based on coverage has minimal impact on semantic information required for sentiment classification.

Similarly, enabling length-based bucketing caused no meaningful degradation in performance. Accuracy differences across bucketing and non-bucketing settings remained within 0.001, and loss values were comparable or slightly lower in some cases, indicating that bucketing does not negatively affect convergence behavior.

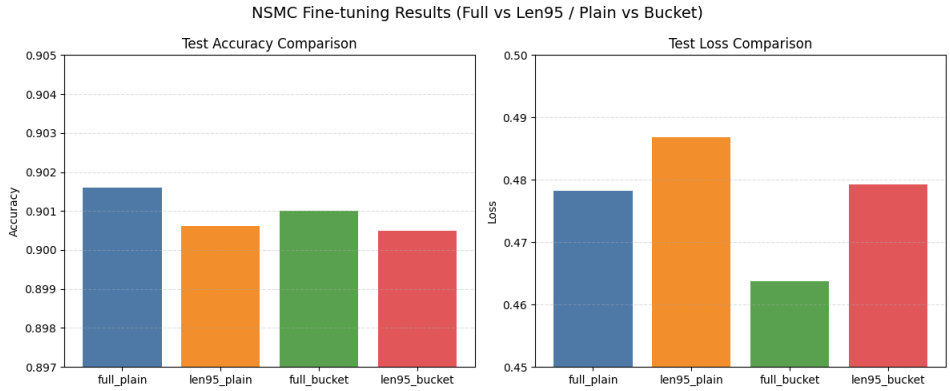

Figure 3: Test accuracy and loss across four experimental settings. All configurations achieve accuracy above 90% and loss below 0.5, demonstrating that neither sequence length truncation nor bucketing significantly degrades classification performance.

Figure 3 summarizes these trends, confirming that performance remains stable across all experimental conditions.

4.2 Training Time Comparison

In contrast to the small differences in classification performance, training time varied substantially depending on how sequence length was handled.

Without bucketing, the training times of the *full_plain* and *len95_plain* configurations were very similar, differing by less than one minute. This indicates that max_len-based truncation alone provides only a limited efficiency benefit when batches still contain sequences with highly diverse lengths.

When bucketing was applied, training time was reduced dramatically. Both bucketing configurations achieved more than a 40% reduction in total training time compared to their non-bucketing counterparts. Among all settings, the *len95_bucket* configuration exhibited the shortest training time.

More importantly, the effect of sequence length truncation was strongly amplified in the presence of bucketing. While truncation alone led to only a minor speedup, combining truncation with bucketing increased the relative runtime reduction by a factor of approximately **8.6**. In practical terms, this corresponds to several additional minutes of training time saved without any measurable loss in classification performance.

These results indicate that bucketing serves as the primary mechanism for eliminating padding-induced inefficiency, while sequence length truncation provides a complementary optimization whose impact becomes substantially larger when both strategies are applied together.

5 Discussion

In this section, we interpret the experimental results by connecting the theoretical properties of self-attention, the empirical length distribution of the dataset, and the observed effects of bucketing and sequence length truncation.

5.1 From Length Sensitivity of Attention to Length Imbalance in Data

As discussed in Section 2, Transformer models based on self-attention are highly sensitive to input sequence length: longer sequences incur disproportionately higher computational cost than shorter ones. In practice, however, natural language datasets rarely consist of uniformly long sentences. Instead, they exhibit highly skewed length distributions in which most examples are short, while a relatively small number of very long examples form a heavy tail.

The NSMC dataset used in this work follows exactly this pattern. Our length analysis in Section 2 showed that the majority of Korean movie reviews are concentrated in a short length range, whereas

a small fraction of outlier sentences substantially extends the upper tail. This implies that, in naive batching, a few long sequences can determine the effective sequence length for an entire mini-batch, forcing the model to spend a large portion of computation on padding tokens that carry no semantic information.

5.2 Why Bucketing is Effective Under Length Imbalance

Given this mismatch between the computational behavior of self-attention and the empirical length distribution of the data, length-based bucketing is a natural remedy. By grouping sequences of similar lengths into the same mini-batch, bucketing prevents a handful of very long sentences from inflating the padding cost for many short ones.

Our experimental results confirm this expectation. When bucketing was applied without any sequence length truncation, total training time decreased by approximately 43% compared to the non-bucketing baseline, while test accuracy and loss remained essentially unchanged. This shows that, under a highly imbalanced length distribution, simply reordering the data into length-homogeneous batches is sufficient to realize large computational gains without compromising model quality.

5.3 Why Bucketing Works Even Better with Truncation

Sequence length truncation using a coverage-aware max_len provides a second, complementary way to address the same underlying issue. Whereas bucketing acts on the batch formation process, truncation modifies the dataset itself by removing only the extreme tail of the length distribution. In our setting, max_len was chosen so that almost all token sequences were preserved, and only a small fraction of excessively long sentences was shortened.

On its own, this preprocessing step led to only modest runtime improvements when bucketing was not enabled, because batches still contained mixtures of short and relatively long sequences. However, when truncation and bucketing were combined, their effects reinforced each other. Truncation reduced the global length range that bucketing had to manage, and bucketing in turn made more consistent use of the shortened sequences by grouping them with similarly short examples.

Empirically, this interaction manifested as a substantial amplification of the runtime reduction. With bucketing enabled, the total training time reduction increased to over 50% when truncation was applied, reaching approximately 52% in our experiments. The additional speedup attributable to truncation was several times larger under bucketing than in the plain setting, corresponding to a relative amplification factor of about 8.6. In other words, bucketing does not merely coexist with truncation; it makes truncation significantly more effective.

5.4 Putting the Pieces Together

Overall, the results follow a coherent narrative:

- **Theory**: self-attention is highly sensitive to sequence length, so excessive padding on long batches is computationally expensive.
- **Data**: the NSMC dataset exhibits a skewed length distribution, where most sentences are short but a minority of long sentences determines batch-wise padding cost.
- **Method 1 (Bucketing)**: grouping sequences by length directly targets this mismatch and yields large training-time reductions (around 43%) with negligible changes in accuracy and loss.
- **Method 2 (Truncation + Bucketing)**: constraining the extreme tail with max_len and then applying bucketing further enhances efficiency, pushing training-time reductions beyond 50% and amplifying the benefit of truncation by an estimated factor of 8.6.

These observations suggest that bucketing should be viewed as the primary mechanism for exploiting length imbalance in real-world datasets, while sequence length truncation acts as a complementary strategy that unlocks its full potential. Together, they provide a simple yet powerful recipe for making Transformer training substantially more efficient under realistic, skewed length distributions.

6 Conclusion

In this study, we systematically analyzed the effects of bucketing and max_len-based sequence length truncation as practical strategies for reducing training time in Transformer-based language models, with a particular focus on mitigating computational waste caused by excessive padding.

Experimental results demonstrate that bucketing is a highly effective technique for improving training efficiency, achieving more than a 40% reduction in training time while maintaining stable classification performance. Across all experimental settings, test accuracy remained above 90% and loss values converged reliably, indicating that bucketing does not degrade model quality despite significantly reducing computational cost.

Sequence length truncation based on max_len provided a moderate but consistent runtime improvement of approximately 5.8% when applied independently. In this work, max_len was set to 95 based on the 99.9% coverage of the token length distribution (with the exact coverage point being 93, adjusted upward for implementation simplicity). Given the negligible changes in accuracy and loss, this preprocessing strategy represents an efficient trade-off between performance preservation and computational reduction. However, its standalone effect remained limited in non-bucketing settings.

The most significant finding of this study is that combining sequence length truncation with bucketing leads to a strong amplification of training speed improvements. When bucketing was enabled, the runtime reduction effect of truncation was amplified by approximately 8.6 times compared to the plain setting, with the *len95_bucket* configuration achieving the shortest training time among all cases. This behavior is consistent with the quadratic computational complexity of self-attention mechanisms, where reducing both the global length upper bound and per-batch padding results in a substantial decrease in overall computation.

Overall, this work empirically shows that applying an appropriate max_len during preprocessing to constrain input length, followed by bucketing to minimize padding at the batch level, constitutes a highly effective strategy for optimizing training speed. The proposed approach offers practical guidance for efficient training of Transformer-based models in environments with limited GPU resources.

Future Work Despite the effectiveness of the proposed approach, several directions remain for future research. First, the evaluation in this study focused primarily on accuracy and loss, which may not fully capture all aspects of model behavior. Incorporating additional metrics such as precision, recall, F1-score, or calibration measures would enable a more comprehensive assessment of performance trade-offs.

Second, this work did not include qualitative analysis of model predictions. Future studies could investigate how length truncation and bucketing influence the types of errors made by the model, particularly for long or context-rich sentences. Finally, extending the analysis to other natural language processing tasks with different length characteristics, such as question answering or natural language inference, would help evaluate the generalizability of the proposed strategy.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.

[2] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 2020.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[4] Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*, 2019.