MultiTok: Variable-Length Tokenization for Efficient LLMs Adapted from LZW Compression

Noel Elias[1], Homa Esfahanizadeh[2], Kaan Kale[3], Sriram Vishwanath[1], and Muriel Médard[4]

[1]University of Texas at Austin, Austin, TX, USA, Emails: {nelias,sriram}@utexas.edu
[2]Nokia Bell Labs, Murray Hill, NJ, USA, Email: homa.esfahanizadeh@nokia-bell-labs.com
[3]Boğaziçi University, Istanbul, Türkiye, Email: huseyin.kale@std.bogazici.edu.tr
[4]Massachusetts Institute of Technology (MIT), Cambridge, USA, Email: medard@mit.edu

Abstract—**Large language models have drastically changed the prospects of AI by introducing technologies for more complex natural language processing. However, current methodologies to train such LLMs require extensive resources including but not limited to large amounts of data, expensive machinery, and lengthy training. To solve this problem, this paper proposes a new tokenization method inspired by universal Lempel-Ziv-Welch data compression that compresses repetitive phrases into multi-word tokens. With MultiTok as a new tokenizing tool, we show that language models are able to be trained notably more efficiently while offering a similar accuracy on more succinct and compressed training data. In fact, our results demonstrate that MultiTok achieves a comparable performance to the BERT standard as a tokenizer while also providing close to 2.5x faster training with more than 30% less training data.**

Index Terms—**compression, efficiency, language models, universal compression, multi-word tokenization.**

I. INTRODUCTION

In recent times, the landscape of AI-based applications has dramatically changed with the introduction of Large language models (LLMs). With the emergence of technologies like ChatGPT, LLaMA, etc., users are able to utilize LLMs for natural language processing tasks including text generation, code generation, summarization, and even complex reasoning [1]. However, to achieve these capabilities, LLMs must utilize massive amounts of diverse training data. In particular, most LLMs need over 1 billion tunable parameters to learn from such large amounts of data. Consequently, it often takes a notably long time to train advanced LLMs, utilizing expensive GPU machinery. In fact, according to [1], the time needed to train language models increases exponentially with the training data size. On the other hand, the training data size for language models has grown 3x per year since 2010 [2]. Thus, these extensive resource requirements and lengthy training time for LLMs call attention to the need for more efficient techniques, motivating training on compressed data, which is our focus.

To mitigate challenges regarding the massive training data, data compression techniques have been deployed to reduce the size of the data and minimize additional overhead. Current techniques for training data compression include methods like PCA [3], t-SNE [4], and mutual information-based methods [5] for feature dimension reduction, among others. Specifically, such techniques work by reducing the resulting token embedding size to lower-dimensional spaces.

Additionally, methods such as transferring knowledge across language models [6] and creating sparse representation for data [7] have been utilized to reduce the training cost. These methods utilize post-processing techniques to reduce model parameters. However, many of these solutions are not dynamic and still require some baseline tokenization or rely on specific linear data distributions. There also exists methods to reduce inference cost via removing unnecessary neural parameters through quantization [8]. Still, these methodologies are not as effective since they are post processing methods, require a large amount of resources, and are computationally costly to run. A promising direction includes using compression algorithms on training data to expedite the training [9]. One of the most popular lossless compression algorithms is the Lempel-Ziv-Welch (LZW) algorithm [10]. The LZW algorithm is a universal compression algorithm that works by compressing repeated patterns of bits within a dataset. As such, it achieves a high compression ratio and performs better with larger amounts of data [11].

In this paper, we introduce `MultiTok`, a new tokenization method inspired by the LZW compression algorithm for the efficient LLM training tasks. We focus on LLMs where the inputted data is first tokenized utilizing an encoder, extracted into embedding vectors representing the semantic and contextual information, and finally fed into a transformer model for text classification. `MultiTok` provides a novel variable-length tokenizer in the sense that each token can represent a variable number of words. The advantages of `MultiTok` are it (i) dynamically compresses the necessary training data by close to 33% (ii) allows the LLM model to be trained close to 2.5x times faster and (iii) maintains performance comparable to the baseline, e.g., Bidirectional Encoder Representations from Transformers (BERT) [12] standard. Our tokenizing schemes can mark the beginning of the use of information-theoretic approaches to provide efficient, secure, and robust LLM systems. The code and data can be accessed at: https://github.com/noelkelias/multitok.

II. PROBLEM SETTING

We denote the set of all samples of a distribution by $\mathcal{X}$. Each sample $x \in \mathcal{X}$ in the training data is described as a sequence of tokens, i.e.,

$$x = \{x_1, \ldots, x_{n(x)}\}, \tag{1}$$

where $n(x)$ is the number of tokens in the sample x. For instance, each x_i can be a single-word token, or alternatively can be obtained from more advanced tokenization schemes in the literature [12]. All tokens are recorded in a dictionary D, where each token is then paired with a unique embedding vector for the LLM tasks.

The focus of our paper is on building an efficient multi-word tokenization scheme T. Our method is based on converting the original tokens of each sample into a smaller number of tokens, by taking a consecutive number of original tokens and grouping them together into a new token,

$$x = \{x_1, \ldots, x_{n(x)}\} \to T(x) = \{y_1, \ldots, y_{m(x)}\}. \quad (2)$$

Here, $m(x) < n(x)$, and the new shorter sequence of tokens is a consecutive disjoint split of tokens in x.

We consider several performance goals: The utility goal is to train a competitive downstream classification with a high generalization accuracy using the proposed tokenization scheme. The utility is often quantified via the generalization accuracy over some validation data. The efficiency goal is two-fold: compression (having fewer tokens per sample) and convergence (reducing the training time).

Definition 1. The ratio of the number of tokens in the dataset, before and after applying T is defined as the compression ratio,

$$r = \frac{\sum_{x \in \mathcal{X}} |T(x)|}{\sum_{x \in \mathcal{X}} |x|} = \frac{\sum_{x \in \mathcal{X}} m(x)}{\sum_{x \in \mathcal{X}} n(x)}.$$

Definition 2. Let l_i be the training loss that the model exhibits on the i-th epoch. The training time is defined as

$$C(\epsilon) = \min\{i | l_{i+1}, \ldots, l_{i+10} < \epsilon\}.$$

Utility and efficiency goals are often competing requirements. In this paper, we design a tokenizer that optimizes for both goals and provides parameters that offer a utility-efficiency trade-off.

III. MULTITOK: A VARIABLE-LENGTH TOKENIZATION

MultiTok works by building a dictionary while encoding the samples into a sequence of tokens, inspired by the LZW compression scheme [10]. We first briefly explain the method: Within a training window of size w, MultiTok looks ahead from the current word, starting from the first word of the first training sample, to identify the largest multi-word token it has previously encountered (already in the dictionary). A new token, comprising this largest known token and the next word, is then added to the dictionary. The process is repeated with the newly added word as the focus. Once all words in a given sample have been processed, the algorithm moves to the next sample, continuing until the entire dataset has been used to build a more comprehensive dictionary. This enables MultiTok to add more common phrases into the dictionary, making it larger, but reducing the average number of tokens needed to represent a sample in the training data.

An overview of the MultiTok tokenization scheme is illustrated as Algorithm 1. The input of the algorithm is the training dataset $\mathcal{X}$, an initial dictionary of all possible single-word tokens D, and the maximum number of words that a MultiTok token can represent as w. In Line 3, we initialize the MultiTok vector $T(x)$ for sample x and the current length of our dictionary as idx. Through Lines 4 and 5, we iterate through each word within x. For each word, we look w-word ahead to form a multi-word phrase of maximum length w, starting from the shortest phrase. When we find the smallest multi-word phrase that is not already in our dictionary $\mathcal{D}$, we will add it to the dictionary (lines 6 and 7). Then, we add the current largest known multi-word token to our MultiTok vector $T(x)$ and move on to encode the next available word (Line 7). If we have reached the end of our sample, we add the index of the remaining phrase to our MultiTok vector in Line 10. Finally, we add the vector of tokens representing the sample x to the set $\mathcal{T}$.

We note that $x[i{:}j]$ is the sub-vector of x with the elements from indices i to j. Further, the operator $+=$ denotes appending an element to the end of a vector. During inference, for tokenization, we start from the first word of a sample, and identify the largest match in the dictionary with the length at most w_{test} (called testing window). We note that w_{test} can be different than w (called training window) in Algorithm 1.

Algorithm 1: MultiTok Tokenization Pseudocode

Input: $\mathcal{X}, D, w$
Output: $\mathcal{T} = \{T(x)\}_{x \in \mathcal{X}}$
1:   $\mathcal{T} = \{\}$
2:   **for** $x \in \mathcal{X}$ **do**
3:     $i = 1, T(x) = [], idx = |\mathcal{D}| + 1,$
4:     **while** $i \leq |x|$ **do**
5:       **for** $j \in [i, \min(i + w, |x|)]$ **do**
6:         **if** $x[i:j] \notin D$ **then**
7:           $D(x[i:j]) = idx, \ idx = idx + 1$
8:           $T(x) += D(x[i:j-1]) \ i := j$
9:           **break**
10:       **if** $j = |x|$ **then** $T(x) += D(x[i:j])$
11:     $\mathcal{T} = \mathcal{T} \cup T(x)$

We also propose a post-processing technique, where we conduct frequency analysis on the MultiTok dictionary. Particularly, multi-word tokens that only appear a few number of times within the encoded training sample are pruned. As a result, during the encoding, those larger tokens are replaced by their original single-word tokens. The motivation for this step is to provide a smaller dictionary of more frequently utilized tokens for a training task rather than including multi-word tokens that might have been rarely used during tokenization.

Finally, we present the end-to-end tokenization pipeline utilizing Algorithm 1 through an example in Fig. 1. We begin by tokenizing the phrase "Alice goes to the Wonderland and Bob goes to the Wonderland Zoo". We first find the largest existing phrase in the inputted dictionary (left) and thus tokenize "Alice" as 2. Then, we add "Alice goes" to the MultiTok dictionary (right) as a new token with index 10. We continue this process until the second mention of the phrase "goes to". This phrase is already in the MultiTok dictionary and is thus encoded as 10 instead of tokens 1 and
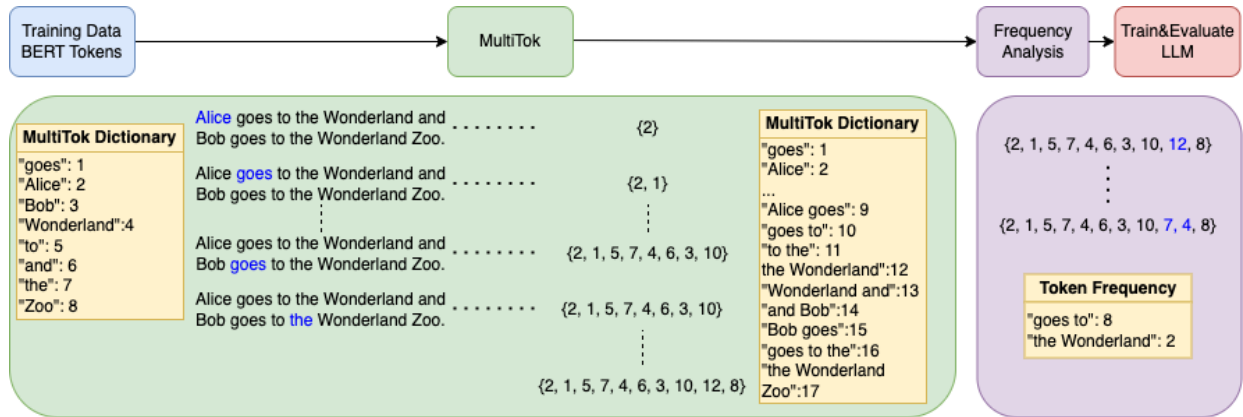
Fig. 1: A toy example showing how `MultiTok` dictionary is constructed and used for tokenization.

5. Now, we add "goes to the" to the `MultiTok` dictionary with index 16. Afterwards, the tokenized samples undergo frequency analysis (purple). Particularly, we prune tokens like token 12 which were only used twice within the entire training data set $\mathcal{X}$. Thus, its instances in the dataset are replaced with the two tokens that it compresses, i.e., 7 and 4. The resulting tokenized sequence is inputted into the LLM for training.

IV. SIMULATIONS RESULTS

In this section, we demonstrate the empirical performance of our tokenizer, `MultiTok`, with respect to the utility and efficiency goals described by Definitions 1 and 2. Further, we show various goals such as utility, accuracy, and training time can compete, and we demonstrate some parameters that can offer a trade-off among them.

A. Experimental Setup

The general pipeline includes tokenizing the training samples with the `MultiTok` and BERT tokenization schemes. These tokens were then assigned corresponding random vector embedding and inputted to train a selected model for the textual binary classification task.

Training. The datasets utilized in this section are the IMDB [13], the sst2 [14], and the AG-News [15] datasets. The first two are collections of movie reviews, each annotated with a binary sentiment. The third is a set of news pieces from different categories. Our experiments were performed on Google Colab instances (Xeon Processors @2.3Ghz with 12GB of RAM). The selected model utilizes a random vector embedding of dimension 100 per token. This is attached to a deep neural network with an LSTM layer and two hidden linear layers with dimensions 100 and 50, respectively. Finally, these nodes are flattened to an output layer with the binary classification output. All models were trained for 30 epochs utilizing batch sizes of 1000, the Adam's optimizer, binary cross-entropy loss, and a learning rate of 0.01. The complete model configuration and hyperparameters can be found in our code repository https://github.com/noelkelias/multitok.

Evaluation. All our results are the averages over 20 trials with the same parameters. Specifically, we record the model's average training loss, training accuracy, and testing accuracy

TABLE I: results on the IMDB dataset with 20+ trials

Test	Comp. (r)	Accuracy	AUC
BERT	1.1	0.739	0.81
No Compression	1	0.732	0.81
MultiTok, 100%, Max-Max	0.57	0.644	0.68
MultiTok, 100%, Max-2	0.57	0.637	0.71
MultiTok, 100%, Max-1	0.57	0.50	0.5
MultiTok, 100%, 2-2	0.63	0.705	0.76
MultiTok, 50%, Max-Max	0.8	0.659	0.71
MultiTok, 50%, Max-1	0.8	0.739	0.77
MultiTok, 50%, 2-2	0.83	0.704	0.75
MultiTok, 50%, 2-1	0.83	0.747	0.81
MultiTok, 25%, 2-1	0.93	0.713	0.80
BERT-MultiTok, 50%, 2-1	0.75	0.727	0.78
BERT-MultiTok, 25%, 2-1	0.91	0.745	0.82
MultiTok, 100%, 2-2, ≥ 2	0.66	0.724	0.79
MultiTok, 100%, Max-Max, ≥ 2	0.65	0.707	0.76

per epoch. In addition, the compression ratio as well as the average area under the curve (AUC) given by the receiver operating characteristic curve (ROC) for each test were recorded and graphed. Our results for `MultiTok` were compared with BERT tokenization [12] and single-word tokenization where each unique word is considered as a token. For all these schemes, we use random vectors for embedding of each token.

Our first round of experiments are focused on studying the performance `MultiTok` algorithm for various sizes of the training window and the testing window. In the second experiment, we apply `MultiTok` encoding on the BERT tokens to have a cascade of two tokenization schemes, where `MultiTok` takes BERT tokens as units of data for compression. Finally, we apply the frequency-based post processing on the `MultiTok` Tokens for further improvements. We note that the title of rows in the table can be read as the type of tokenization, the percentage of training data to which the tokenizer is applied, and the training and testing window sizes.

B. MultiTok Tokenization

Table I shows the results for tokenization schemes with different parameters. According to this table, the best performance is obtained by the presented `MultiTok` tokenization when the size of training and testing windows are 2 and 1,
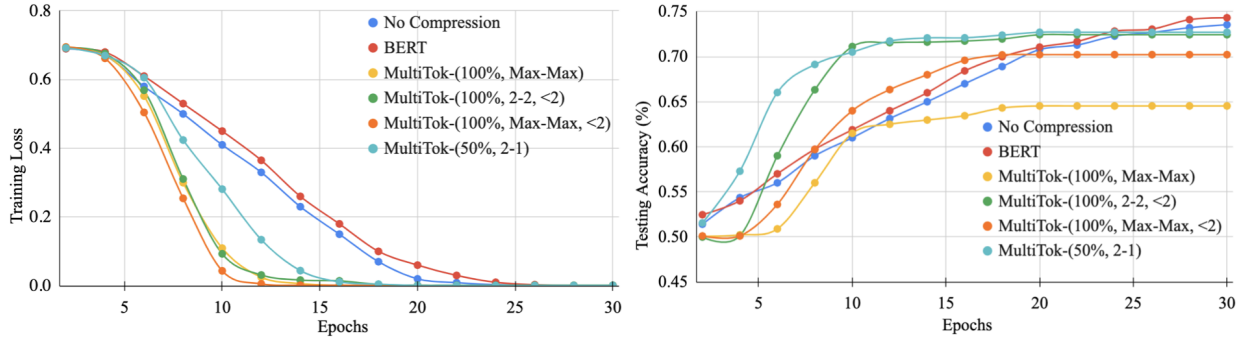
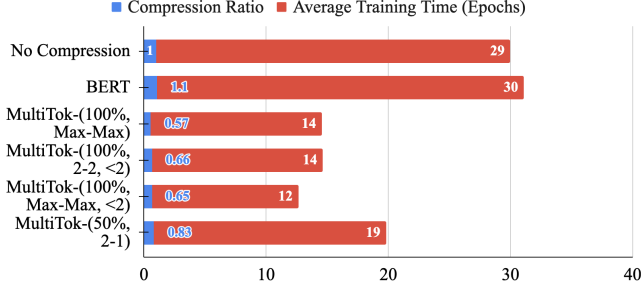Fig. 2: Training (left) & Testing Graphs (right) for the IMDB dataset.

Fig. 3: `MultiTok` demonstrates an average training time $(C(\epsilon)|\epsilon = 0.01)$ that is almost 2-3 times faster than the BERT.

respectively, and the tokenization dictionary is applied to 50% of samples in the training data. We remind that the testing window size 1 means that during the inference from the trained downstream model, we used single-word tokens, and multi-word tokens were only used during training. In this scenario, `MultiTok` tokenization exhibits a slightly better performance compared to the standard BERT's performance (0.747 compared to 0.739) on the IMDB dataset, while compressing the training data by 17%.

As the amount of data that the `MultiTok` tokenization is applied to increases (the size of tokenized training data decreases), the performance of the downstream model decreases. Similarly, as the training window of the `MultiTok` tokenization scheme increases, the predictive performance of the downstream model decreases, as `MultiTok` groups tokens that rarely happen together in the training samples. Thus, there is an inverse relationship between the `MultiTok` tokenization amount and its training window; and the predictive performance of the downstream models. This is a reasonable observation as compression and performance are often competing goals.

C. MultiTok Tokenization on BERT Tokens

In this experiment, we used `MultiTok` tokenization to further compress the repetitive sequences within BERT tokens, and consequently, reduce the training data size while achieving similar performance to that of normal uncompressed BERT tokens. We observed that we could compressed the training data that are tokenized via BERT by 9% using `MultiTok`, while preserving the accuracy of the downstream model. However, when we further reduced training data size, a higher

compression is obtained at the cost of slight degradation in the performance. This suggests a notion of using a 'golden ratio' of `MultiTok` compression to maximize efficiency and performance. This 'golden ratio' may vary based on the repetition and succinctness found within different datasets.

D. MultiTok Tokenization with Post-Processing

Our results in this experiment indicates that the post processing of `MultiTok` Tokenization with frequency analysis component dramatically increases the performance of the resulting tokens. We remind that without this post processing, there is a significant decrease in the model's performance when the full capacity of `MultiTok` is utilized (when `MultiTok` is applied to the entire dataset with maximum window sizes), see Table I. However, by pruning less-utilized tokens, we were able to circumvent any impact to accuracy, as shown by the last set of experimental results. Specifically for experiment "`MultiTok`, 100%, 2-2, ≥ 2" (`MultiTok` is applied to the entire dataset with a training window size of 2), we see that by simply removing all multi-word tokens that only appear once in the training data, we get an accuracy of around 72%. This is comparable to the BERT standard and reduces the training data size by 34% .

Next, we provide an empirical training convergence analysis, which shows that `MultiTok` not only performs as good as the BERT standard in terms of the accuracy and AUC, but also results in a faster training process for the downstream models. According to Fig. 2, using `MultiTok` tokens results in a comparable accuracy almost $2.5x$ faster than when using BERT tokens. Specifically, at epoch 12 in Fig. 2, the loss of the `MultiTok` experiments have almost achieved their stable minimum value, being at least 0.3 lower than the training loss for the baselines at the same epoch. In addition, as shown in Fig. 3, the average training time (as defined in Definition 2) for `MultiTok` is around 12-13 epochs, which is significantly lower than the other counterparts. Significantly, `MultiTok` has the lowest compression ratio out of the selected experiments, where the `MultiTok` tokens not only compressed the inputted training data by a factor of around 3, but also helped the model learn the training data almost $2.5x$ faster. This confirms that with `MultiTok` tokenization, the model is able to learn the training data faster and more efficiently as opposed to the baseline tokenization schemes.

REFERENCES

[1] Z. Wan, X. Wang, C. Liu, S. Alam, Y. Zheng, Z. Qu, S. Yan, Y. Zhu, Q. Zhang, M. Chowdhury *et al.*, "Efficient large language models: A survey," *arXiv preprint arXiv:2312.03863*, 2023.

[2] Epoch AI, "Key trends and figures in machine learning," 2023, accessed: 2024-07-19. [Online]. Available: https://epochai.org/trends

[3] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303–342, 1993.

[4] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[5] K. Kale, H. Esfahanizadeh, N. Elias, O. Baser, M. Médard, and S. Vishwanath, "TexShape: Information theoretic sentence embedding for language models," in *2024 IEEE International Symposium on Information Theory (ISIT)*, 2024, pp. 2038–2043.

[6] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" *arXiv preprint arXiv:1909.01066*, 2019.

[7] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine learning*, vol. 42, pp. 143–175, 2001.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] R. R. Gajjala, S. Banchhor, A. M. Abdelmoniem, A. Dutta, M. Canini, and P. Kalnis, "Huffman coding based encoding techniques for fast distributed deep learning," in *Proceedings of the 1st Workshop on Distributed Machine Learning*, 2020, pp. 21–27.

[10] Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[11] A. Gupta, A. Bansal, and V. Khanduja, "Modern lossless compression techniques: Review, comparison and analysis," in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, 2017, pp. 1–8.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[13] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[14] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://www.aclweb.org/anthology/D13-1170

[15] X. Zhang, J. J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *NIPS*, 2015.