Combined Tokenization Processing

Yaroslava Bryukhanova, Nazgul Salikhova,
Mikhail Romanov
Innopolis University

***Abstract*—Tokenization is an influential part of the Transformer model. Breaking down the text into the units to analyze, it significantly affects the quality of the text processing. There exist multiple algorithms producing different tokenizations. All the tokenization algorithms have their advantages and disadvantages, so combining various tokenization algorithms in a single model may compensate for the disadvantages and happen to be beneficial for the quality of the text processing. In this paper, we explore an experimental technique that combines three different tokenizers to investigate its potential impact on model performance.**

Introduction

Tokenization is a fundamental step in natural language processing (NLP), directly influencing the efficiency and performance of downstream models. Subword tokenization methods such as Byte-Pair Encoding (BPE) [7], Unigram [5], and WordPiece [6] have become standard choices in modern NLP architectures, particularly for transformer-based models. Each method has distinct advantages: BPE efficiently handles rare words through subword segmentation, Unigram provides a probabilistic approach that optimizes vocabulary selection, and WordPiece balances efficiency and expressiveness in handling word structures.

However, selecting a single tokenization strategy often leads to trade-offs in sequence length, vocabulary efficiency, and model generalization. Long input sequences increase computational cost and memory requirements while also limiting the amount of contextual information available in fixed-length architectures. In this work, we experiment with a new technique that integrates multiple tokenization strategies to explore potential benefits in reducing sequence length and semantic preservation. Our goal is to assess whether this combination can improve the efficiency and performance of large language models in practice.

Digital Object Identifier 10.1109/MCE.2025.Doi Number

Date of publication 30 06 2025; date of current version 30 06 2025

Related works

Earlier studies tried to increase tokenization efficiency using different tricks. One of the ideas was to make tokenization learn along with the downstream model [4], allowing for improved sequence representations and better overall performance. This method integrates tokenization into the training process, demonstrating that adaptive subword segmentation can lead to shorter sequences and improved model efficiency.

Another notable approach is the AMBERT model [8], which introduces *multigrain tokenization* by combining both fine-grain and coarse-grain tokenization strategies. AMBERT uses two parallel encoders: one operating at the word level to capture syntactic details, and another at the phrase level to capture broader semantic structures. By sharing parameters between these encoders, the model integrates both granular and high-level linguistic information. This method leads to enhanced language representations, improved performance on benchmarks such as CLUE, GLUE, SQuAD, and RACE, and increased robustness compared to models using a single tokenization strategy.

Building on the idea of using existing tokeniza-

tion knowledge, the Fast Vocabulary Transfer (FVT) method [2] proposes a simplified and efficient technique to adapt tokenizers across domains. Instead of training a new tokenizer from scratch, FVT allows an in-domain tokenizer to partially inherit embeddings from a general-domain tokenizer. Specifically, if a token from the in-domain tokenizer appears in the general tokenizer's vocabulary, it is embedded directly using the general-domain embedder. Otherwise, the token is decomposed into subtokens of the general tokenizer, and its representation is computed as the sum of their embeddings. This strategy not only reduces the sequence length but also yields performance improvements on tasks such as GLUE. Importantly, the approach of combining embeddings for out-of-vocabulary tokens offers valuable insights for hybrid tokenization frameworks like ours, where multiple tokenization strategies coexist.

Although previous studies explore dynamic or multigrain tokenization and cross-domain adaptation, none have directly implemented a technique that merges the outputs of different tokenizers in a unified preprocessing pipeline. Our work explores this field by experimentally combining BPE, Unigram, and WordPiece tokenizers, aiming to leverage their complementary strengths.

Methodology

Model Selection

For our experiments, we use **Tiny-Llama-110M** - this is the 110M parameter Llama 2 architecture model trained on the TinyStories dataset. These are converted from karpathy/tinyllamas. Tiny-LLaMA-100M is a suitable choice for our experiment. Its lightweight nature allows for faster training and evaluation cycles on limited hardware, while still capturing essential behaviors of transformer-based language models.

Tokenization Strategies

We employ three different tokenization approaches in our study:

- **Tiny Llama Default Tokenizer** – The standard BPE-based tokenizer used in Llama models.
- **WordPiece Tokenizer** – A subword-based tokenizer trained specifically for our task.
- **Unigram Tokenizer** – A probabilistic subword tokenizer optimized for efficient vocabulary representation.

To train the WordPiece and Unigram tokenizers, we utilize 2M rows of text in the **OpenWebText** [3] dataset, a large-scale corpus designed to resemble OpenAI's WebText dataset. This ensures a diverse and high-quality linguistic distribution for learning subword segmentation.

Combined Tokenization Approach

To implement our hybrid tokenization strategy, we tokenized the dataset three separate times using distinct tokenizers: BPE, WordPiece, and Unigram. Each version produced a different sequence of token IDs for the same text sample. We then merged the three tokenized representations into a single input sequence by concatenating them with a special separator token ([200000]), which does not appear in any of the tokenizers' vocabularies. This separator allows the model to identify and correctly route each segment to the appropriate embedding layer.

Then we modified the architecture of the Tiny-LLaMA-100M model by adding two additional embedding layers alongside the original one. These new embedding layers correspond to the WordPiece and Unigram tokenizers. We froze all parameters of the base model, allowing only the embedding layers to be updated during training. This ensures that the model learns to align and integrate the different tokenization strategies without altering its pre-trained core.

The forward function of the model was overridden to handle the composite tokenized input. Specifically, when a tokenized sequence is received, the function splits it into three segments using the [200000] separator. Each segment is then embedded using its respective embedding layer (original BPE, WordPiece, or Unigram). To ensure dimensional consistency, the embeddings are truncated to the minimum sequence length among the three segments before being summed element-wise. The resulting unified embedding is passed into the model in place of standard token embeddings.

To prepare the dataset for this setup, we mapped the original dataset three times using the three tokenizers. The tokenized outputs were then combined by concatenating the input_ids and attention_mask fields with separators. This unified format was used as the training input for the modified model.

Embedding combining strategies

After splitting the input into three parts and passing each through its corresponding embedding layer, we

obtained three embedding matrices of equal shape. To integrate these representations into a single input for the model, we experimented with the following strategies:

Mean of embeddings (*mean_emb*): All three embeddings were summed in element-wise order and then divided by three.

Weighted Average (*w_avg*): A learnable set of scalar weights was introduced for each type of embedding. These weights were applied to the respective embeddings before summing, allowing the model to prioritize tokenizers differently depending on the task.

Summarization (*summarization*): All three embeddings were summed in element-wise order. This simple aggregation treats each tokenizer equally and merges their representations directly.

Evaluation

To comprehensively assess the impact of combining multiple tokenization strategies, we evaluated the models along three key axes: instruction-following accuracy, memory consumption, and generation speed on the [1] dataset, a widely used benchmark for instruction-following language models. We use a **TinyLlama-110M** as the base causal language model.

Evaluation Metrics

We employed the following metrics:

- **ROUGE-L** [**lin2004rouge**]: Measures the longest common subsequence between generated and reference texts, useful for capturing sequence-level overlap.
- **BERTScore** [**zhang2019bertscore**]: Computes semantic similarity using contextual embeddings from a pretrained BERT model, better capturing paraphrased or semantically close predictions.
- **Memory Usage (MB)**: Captures the peak memory used during text generation to assess the computational efficiency.
- **Generation Speed (Tokens Per Second)**: Reflects how quickly tokens are generated, critical for real-time applications.
- **Latency (ms)**: The time required for a single sample generation.
- **Throughput (Samples Per Second)**: The number of full generations the model can perform per second.

Experimental Setup

We conducted the evaluations using 100 samples from a standard instruction-following dataset [1]. Each sample was assessed using the combined tokenization approach with new embedding weights, differing only in the preprocessing tokenization strategy: *mean_emb*, *summarization*, and *w_avg*. The results were compared against the baseline model using the original TinyLLaMA tokenizer.

Results

Table 1 summarizes the performance of all evaluated methods. We observed that the *w_avg* strategy achieved the highest ROUGE-L among the combined-tokenization approaches, suggesting better n-gram and sequence-level overlap. Meanwhile, *mean_emb* produced the highest BERTScore, indicating better semantic alignment with the reference outputs.

Discussion

Despite testing several tokenization strategies, our results show that all models, including the baseline, struggle to generate high-quality outputs. ROUGE-L score were very low in all methods, indicating a poor overlap with the reference text in terms of n-grams and sequences. Although the baseline model had a high BERTScore (0.802), this may only reflect superficial semantic similarity, not accurate instruction following.

Although the combined tokenization strategies (*mean_emb*, *summarization*, and *w_avg*) improved computational performance, achieving more than 27,000 tokens per second and up to 9 times higher throughput, there was no significant improvement in the quality of the generated text. These results highlight a key issue: our tokenization technique does not justify itself, and further improvements are needed.

Conclusion

In this study, we explored the impact of combining multiple tokenization strategies —*mean_emb*, *summarization*, and *w_avg* on the performance of a small language model, Tiny-LLaMA-100M. Our approach involved tokenizing the dataset using BPE, WordPiece, and Unigram tokenizers, concatenating their outputs with a special separator token, and modifying the model to include additional embedding layers specific to each tokenizer. Despite architectural modifications and the integration of three tokenized representations per input, our experiments showed that none of the strategies led to improvements in output quality, as

TABLE 1. Evaluation results of different tokenization strategies across generation quality and performance metrics.

Method	ROUGE-L	BERTScore	Mem (MB)	TPS (t/s)	Latency (ms)	Throughput (samp/s)
Original	0.089	**0.802**	112.36	79.58	628.27	1.59
mean_emb	0.105	0.729	2803.82	**29071.15**	**105.74**	**9.46**
summarization	0.007	0.707	2839.02	27586.28	111.43	8.97
w_avg	**0.111**	0.717	**2870.97**	27434.30	112.05	8.92

measured by ROUGE-L and BERTScore. Moreover, the model struggles to generate a meaningful output. These findings suggest that our approach did not prove its efficiency.

In future work, we plan to investigate training the model together with its language modeling head rather than keeping it frozen, and to incorporate regularization techniques into the embedding summarization method. We believe that these steps could help the model better integrate the information from different tokenizations and potentially improve performance on complex instruction-following tasks.

References

[1] M. Alpaca. *FineTome-Alpaca-100k: A Dataset for Instruction-Following Models*. https://huggingface.co/mlabonne/FineTome-Alpaca-100k. Accessed: 2023-04-29. 2023.

[2] Leonidas Gee et al. "Fast Vocabulary Transfer for Language Model Compression". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Association for Computational Linguistics, 2022, pp. 409–416. DOI: 10.18653/v1/2022.emnlp-industry.41. URL: http://dx.doi.org/10.18653/v1/2022.emnlp-industry.41.

[3] Aaron Gokaslan et al. *OpenWebText Corpus*. http://Skylion007.github.io/OpenWebTextCorpus. 2019.

[4] Tatsuya Hiraoka et al. "Joint optimization of tokenization and downstream model". In: *arXiv preprint arXiv:2105.12410* (2021).

[5] Taku Kudo. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2018, pp. 66–75. URL: https://aclanthology.org/P18-1007/.

[6] Mike Schuster and Kaisuke Nakajima. "Japanese and Korean Voice Search". In: *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

[7] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2016, pp. 1715–1725. URL: https://aclanthology.org/P16-1162/.

[8] Xinsong Zhang and Hang Li. "AMBERT: A Pre-trained Language Model with Multi-Grained Tokenization". In: *CoRR* abs/2008.11869 (2020).

arXiv: 2008.11869. URL: https://arxiv.org/abs/2008.11869.