Tokenization is not the problem

Danil Kutny
danil.kutny@gmail.com

Abstract

This paper introduces a modification to standard GPT-like models by incorporating character-level encoding. The model uses an LSTM to process individual characters within tokens, which are then embedded into the original token embedding space. This allows the model to maintain token-level processing while adding character-level information to each token. Trained on the BookCorpus dataset, the model was evaluated on tasks requiring character-level manipulation, such as counting letters and reversing words. Surprisingly, the modified model performed similarly to the baseline GPT model, with no significant improvements, suggesting that GPT-like models may inherently learn character-level representations from tokenized inputs. The code is available at [Github].

1. Introduction

Large Language Models (LLMs), such as GPT, have become the standard for a variety of NLP tasks [1], however, these models rely on tokenization schemes to process input text, which abstract away the finer details of character-level information. Tokenization, while efficient, involves splitting text into discrete tokens (e.g., words or subword units), which abstracts away individual character details [2]. This limitation has led to the common belief that LLMs are not well-suited to tasks requiring fine-grained manipulation of text at the character level.

In this study, we investigate the potential of enhancing LLMs by integrating a character-level encoding mechanism. The primary goal is to explore whether this additional encoding allows the model to process characters within tokens without sacrificing the benefits of token-based models. We hypothesize that by incorporating this new encoding, the model could perform better on tasks that involve manipulation at the character level, such as counting letters or reversing strings, which are typically challenging for standard LLMs.

2. Related Work

In recent years, several approaches have been proposed to improve LLMs' ability to handle character-level tasks. These approaches generally fall into two categories: (1) models that incorporate character-level information directly into their tokenization process [3], and (2) models that apply post-tokenization methods to reintroduce character-level understanding [4]. While tokenization remains the default method for LLMs, some research has shown that hybrid

models, which combine token-level and character-level processing, can yield improvements on specific tasks [5].

3. Base Model Architecture

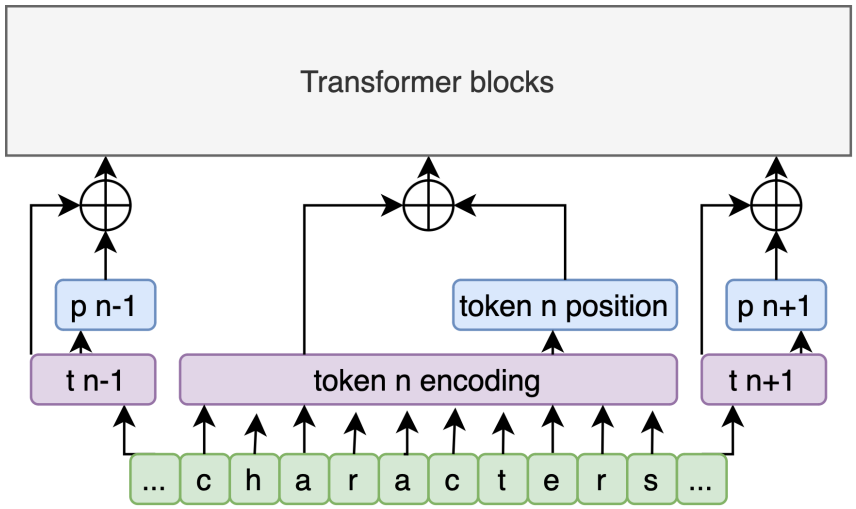

Figure 1. Traditional embeddings

The architecture proposed in this paper consists of a standard GPT-like transformer model with two key modifications: the introduction of a character-level encoding and an alternative token embedding mechanism. Figure 1 illustrates the standard process of tokenization and embedding.

3.1 Character-Level Encoding

To address the issue of tokenization, we introduce a character-level encoding that is incorporated into the model's embedding and positional encoding layers.

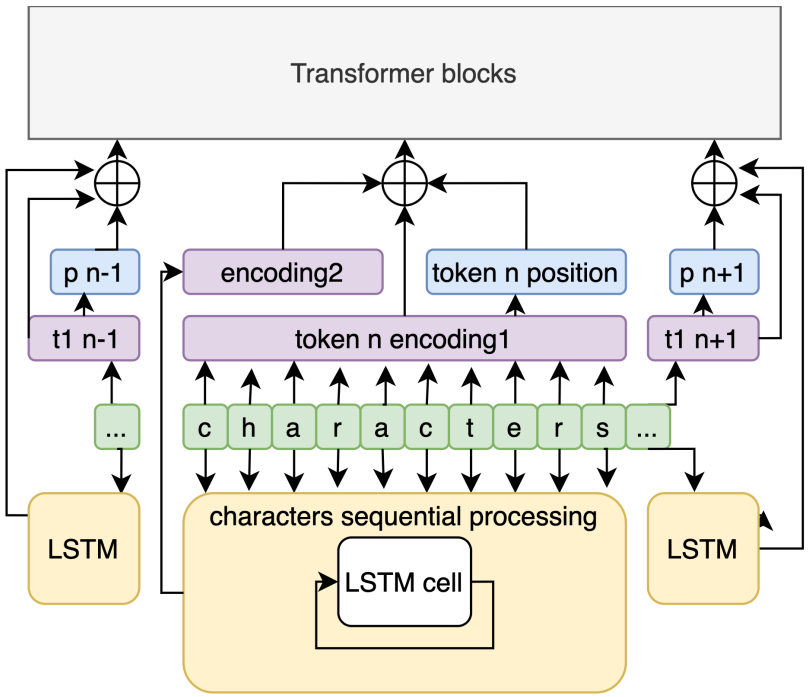

Figure 2. LSTM character-level embeddings

In this design (figure 2), each token is split into original characters and first passed through an LSTM [6] that processes the individual characters of the token in a sequential manner. The last hidden state of the LSTM, which represents the sequence of characters, is then embedded into a character encoding space with an additional linear layer. This character encoding is combined with the original token embeddings and positional embeddings, allowing the model to retain character-level information, along with token embedding and its position.

3.2 Additional Embeddings Model

As a baseline comparison, we also introduce a second model that incorporates additional parameters in the embedding layer (figure 3). Instead of using an LSTM, this model passes the token one-hot representation through a fully connected layer, which maps the tokens into a higher-dimensional vector space. The resulting representation is then embedded into the original transformer embedding space. Such a secondary embedding layer should not alter the behavior compared to the original model, but additional embedding is used to add parameters to a transformer in the embedding layers of neural network to match the parameter count in the LSTM version, while leaving the GPT transformer blocks architecture untouched. This ensures that if the LSTM model exhibits any improvement in metrics, it is not merely due to a higher parameter count but reflects genuine model enhancement, but done in an equal comprising.

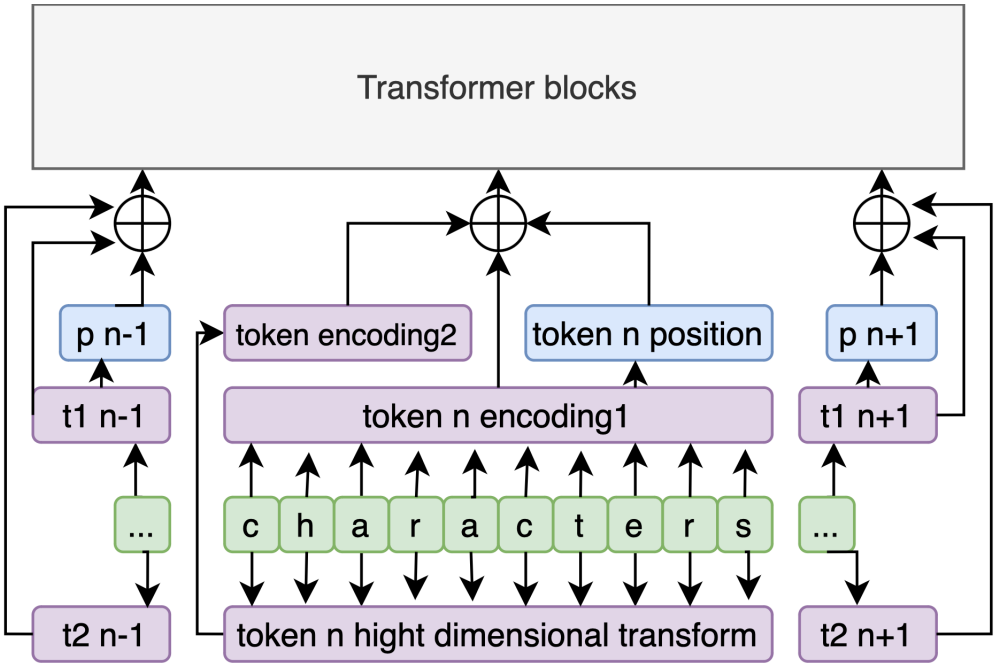

Figure 3. Additional standard embedding for parameters equality

4. Tokenization and Dataset

Given the limitations in computational resources, a custom tokenization scheme was developed for this experiment. The tokenizer is based on GPT-4's approach but is limited to a vocabulary of only 512 tokens [7]. This tokenization scheme was applied to the subset of BookCorpus dataset [8]. Full dataset also is available with the code in repository.

5. Experimental Setup

5.1 Training Procedure

Both models were trained on the custom tokenized lite version of the BookCorpus dataset. The training procedure followed standard practices for transformer models, including gradient descent optimization and the use of appropriate meta-parameters. Andrej Karpathy Nano-GPT was taken as a reference starting point [10]. For evaluation models were fine-tuned on a set of character-level tasks, which included:

- Counting the number of specific letters in a word
- Reversing the order of letters in a word
- Find the first index of exact letter in a word
- Swap a specific letter in a word with another one.

These tasks were selected because they require fine-grained control over individual characters and should making them particularly challenging for standard token-based models. All questions were answered in natural language.

6. Results

6.1 Performance Comparison

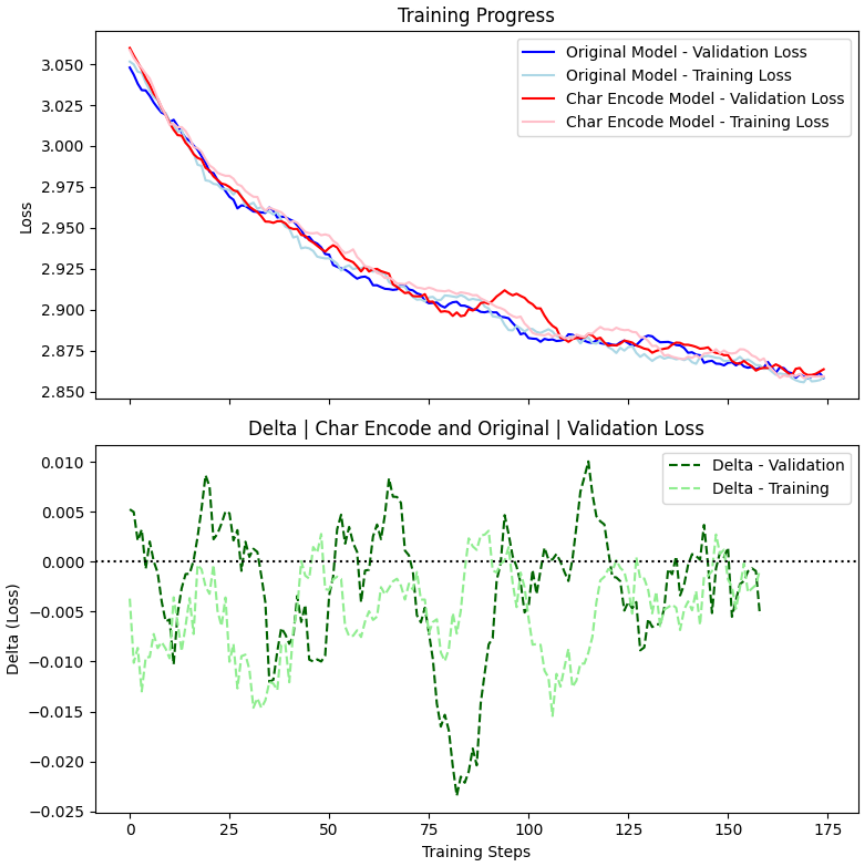

Figure 4. Language modeling training

Surprisingly, both the LSTM-based, or Char Encode model and the baseline, or Original model performed nearly identically on the core language tasks.(figure 4). Despite the addition of character-level encoding in the newly proposed model, the improvements on standard tasks were negligible, suggesting that the baseline transformer model was capable of learning standard language-modeling tasks without explicit character encoding [1].

6.2 Character-Level Task Performance

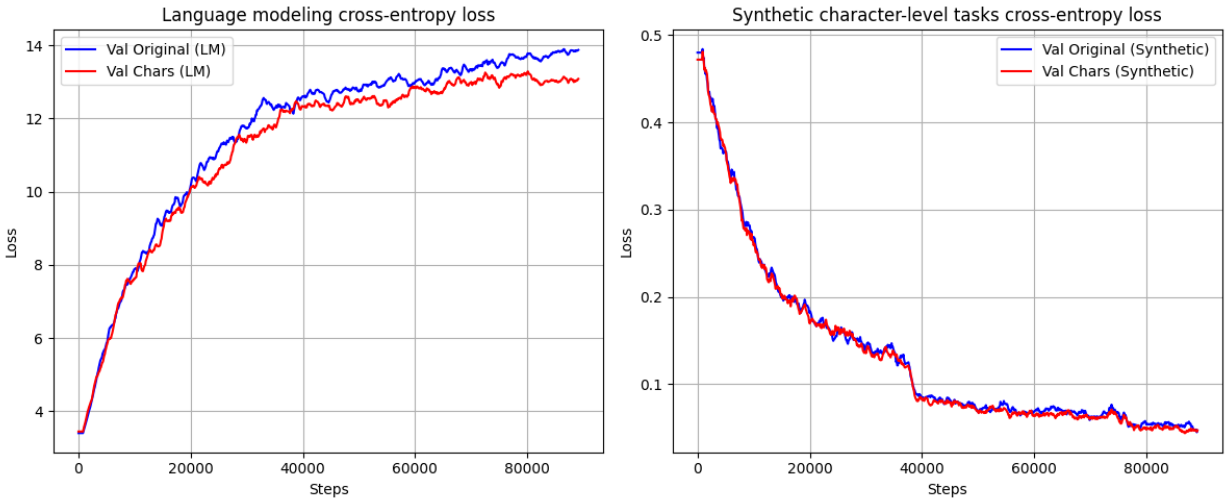

Figure 4. Synthetic character-level tasks fine-tuning

When fine-tuned on the character-level tasks, the results were unexpected (figure 5). Although we anticipated that the LSTM-based model, which processes individual characters, would outperform the baseline transformer model, it did not. In fact, the baseline transformer slightly outperformed the character-aware models in all tasks though by an insignificant margin. This suggests that the basic GPT model, through its tokenization and internal representations, is already capable of learning the necessary character-level information to perform well on these tasks. Character encoding offered no significant advantage; in some cases, the additional complexity introduced by the LSTM marginally hindered performance.

7. Discussion

The results of this study provide an intriguing insight into the capabilities of GPT-like models. Despite their reliance on tokenization, these models appear to have an inherent ability to learn the necessary character-level information when trained on a sufficiently large corpus. This challenges the traditional view that LLMs are fundamentally limited by their inability to process individual characters. The simplicity of the standard embedding layer suggests that character-level representations can be learned through the model's existing architecture without the need for complex modifications like LSTM networks.

8. Conclusion

In this paper, we explored the hypothesis that large language models could benefit from a character-level encoding mechanism, designed to allow the model to "see" individual characters. While the results did not show a clear improvement over the baseline GPT-like transformer model, the findings suggest that hypothesis is incorrect and LLMs can learn the necessary character-level representations on their own, even in tokenized form. This insight challenges the need for explicit character-level processing in many cases, opening up new directions for future research and model optimization.

References

1. Vaswani, A. "Attention is all you need." Advances in Neural Information Processing Systems (2017).
2. Sennrich, Rico. "Neural machine translation of rare words with subword units." *arXiv preprint arXiv:1508.07909* (2015).
3. Zhang, Xiang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." *Advances in neural information processing systems* 28 (2015).
4. Dai, Zihang. "Transformer-xl: Attentive language models beyond a fixed-length context." *arXiv preprint arXiv:1901.02860* (2019).
5. Kim, Yoon, et al. "Character-aware neural language models." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.
6. Hochreiter, S. "Long Short-term Memory." *Neural Computation MIT-Press* (1997).
7. OpenAI. *GPT-4 Technical Report*. OpenAI, 2023. Available at: https://openai.com/research/gpt-4
8. Bejan, Matei. *15000 Gutenberg Books Dataset*. 2021, Kaggle, https://www.kaggle.com/datasets/mateibejan/15000-gutenberg-books.
9. Karpathy, A. (2023). *nanoGPT: A PyTorch implementation of GPT from scratch*. GitHub repository. Retrieved from https://github.com/karpathy/nanoGPT