Can Perplexity Predict Fine-Tuning Performance? An Investigation of Tokenization Effects on Sequential Language Models for Nepali

Nishant Luitel, Nirajan Bekoju, Anand Kumar Sah and Subarna Shakya
Dept. of Electronics and Computer Engineering,
Pulchowk Campus, Tribhuwan University,
Lalitpur, Nepal
{076bct041.nishant, 076bct039.nirajan, anand.sah}@pcampus.edu.np, drss@ioe.edu.np

Abstract

Recent language models use subwording mechanisms to handle Out-of-Vocabulary(OOV) words seen during test time and, their generation capacity is generally measured using perplexity, an intrinsic metric. It is known that increasing the subword granularity results in a decrease of perplexity value. However, the study of how subwording affects the understanding capacity of language models has been very few and only limited to a handful of languages. To reduce this gap we used 6 different tokenization schemes to pretrain relatively small language models in Nepali and used the representations learned to finetune on several downstream tasks. Although byte-level BPE algorithm has been used in recent models like GPT, RoBERTa we show that on average they are sub-optimal in comparison to algorithms such as SentencePiece in finetuning performances for Nepali. Additionally, similar recent studies have focused on the Bert-based language model. We, however, pretrain and finetune sequential transformer-based language models.

1 Introduction

Nepali language, written in Devanagari script, is an Indo-Aryan language and the official language of Nepal. As per the NPHC 2021 report in Nepal, 44.9% (about 13 million) speak the Nepali language as their mother tongue, while 46.2%(nearly 13.5 million) speak it as their second language. Nepali is also spoken in neighboring parts of India, Bhutan, Brunei, and Myanmar. The structure of Nepali sentences differs from that of many other languages. The dominant sentence structure in the Nepali language is subject-object-verb. Due to the lack of quality data resources and computational resources, there has not been a significant advancement in the NLP domain of the Nepali language. A rich set of vocabulary and morphology has made it difficult to write clear, concise, and correct Nepali content. Hence, investigating the applicability of various recent NLP technologies in Nepali can be beneficial to both researchers and Nepali Speakers. Moreover, the findings on the Nepali language can also be generalized to several other languages with Devanagari script like Hindi, Sanskrit, Maithili, Bhojpuri, etc.

Tokenization is a process in NLP that involves breaking down textual data into smaller units, typically words or subwords, referred to as tokens. This fundamental technique serves as a cornerstone in various NLP tasks, including machine translation, sentiment analysis, and named entity recognition. By segmenting text into tokens, the complexity of language is reduced to manageable units, facilitating computational analysis and understanding. Moreover, tokenization plays a crucial role in preprocessing raw text data for machine learning algorithms, enabling efficient feature extraction and model training. In essence, tokenization is the first step in unlocking the power of language data for analysis and understanding.

Most language models today generate human-like text by leveraging deep learning models and huge datasets. Researchers have developed a wide range of transformer-based language models for text generation or representation learning. These language models are pre-trained using either a Masked language model approach where the model learns to predict the masked tokens based on the surrounding context like BERT (Devlin et al., 2018) or an auto-regressive language model approach where the model is trained to predict the next word given the previous context words like GPT-X (Radford et al., 2019; Brown et al., 2020) or PaLM (Chowdhery et al., 2022). The strength of the auto-regressive language models is that they only need previous time contextual information to predict the next token which is essential when generating sequences. The strength of the masked language is that it takes into account the surrounding context in both directions to learn powerful representations that can used for finetuning. In this paper, instead of using a BERT-based Nepali language model, we use sequential language models trained with various tokenization methods to finetune on downstream tasks. We have used the term sequential LM to mean an autoregressive LM which we use interchangeably in this paper.

The major contributions of our paper are as follows:

1. We pretrained 7 different sequential language models with following tokenizers: Word Tokenizer (30,000 and 60,507 vocabs), SentencePiece tokenizer, WordPiece tokenizer, BPE tokenizer, Morpheme tokenizer, combination of Morpheme and BPE tokenizers (all with 30,000 vocabs).

2. Comparison of performance language models based on perplexity was done during pretraining with different tokenization methods.

3. Comparison of performance of the pretrained language models was done based on fine-tuning over several Nepali Natural Language Understanding(NLU) tasks. We have made all our code and models open via github.

2 Related Works

The goal of the language model is to predict the next word given the context words.(Bengio et al., 2000) presented the Neural Probabilistic Language Model(NPLM) that can learn distributed representation for each word along with the probability function for the word sequences. Before the introduction of RNNs, various approaches based on parse trees, and n-gram statistics were used. (Mikolov et al., 2010) introduced an RNN-based language model (RNN LM) with application to speech recognition which proved the superiority of connectionist language models to the standard n-gram techniques, except for their high computational complexity. (Sutskever et al., 2011) showed the development of character-level modeling for text generation by training the RNNs with the Hessian-Free optimizer. The introduction of Transformer in (Vaswani et al., 2017) revolutionalized the field of natural language processing. (Vaswani et al., 2017) implemented the attention mechanism to develop the SOTA machine translation model, i.e. to generate the text in one language, given the context in another language. The parallelization ability in the transformer model solves the issue of the high computational and training complexity of previous sequential models. Various transformer-based models such as BERT (Devlin et al., 2018) and GPT(Brown et al., 2020) were developed which are the basis of many NLP tasks present today.

In recent years there have been some research towards pretraining and finetuning of NLP tasks in low-resource language like Nepali. (Maskey, 2023) pre-trained text-generation model with the same configuration as the (Sanh et al., 2019) on a combined dataset: Oscar, cc100, and a set of scraped Nepali Articles on Wikipedia by using the SentencePiece Model as a tokenizer, with vocabulary size 24, 576. (Maskey et al., 2022) trained three distinct transformer-based masked language models (distilbert-base, deberta-base, and XLM-ROBERTa) for Nepali text sequences which were evaluated and compared with other Transformer-based models on a downstream classification task. Similarly, (Niraula and Chapagain, 2022) fine-tuned Multilingual Bert for NER task. (Timilsina et al., 2022) trained another Bert-based language model for Nepali using the WordPiece vocabulary of 30,522 sub-word tokens and showed that their model performed better than other Bert-based LMs (Rajan, 2021; Devlin et al., 2018; Conneau et al., 2020) when fine-tuned on four distinct tasks namely, Content Classification, Named Entity Recognition, Part of Speech Tagging, and Categorical Pair Similarity. Although, several pretraining and finetuning have been performed in Nepali, a comparative study on the performance of language model on downstream tasks due to various tokenization haven't been performed.

However, there have been some similar research in other languages. (Toraman et al., 2022) analyzed the efficiency(in terms of training time, carbon emissions) and effectiveness(in terms of performance) of various tokenization techniques by finetuning a Turkish Bert-based LM on various downstream NLP tasks. They found that for similar and smaller vocab sizes BPE(char-level) and WordPiece are better than other tokenization techniques like word-based. (Park et al., 2020) found that for Korean, morpheme tokenization followed by BPE(char-level) achieved best performance. This type of tokenization schemes forces BPE to not consider the sequence of bytes that span across multiple morphemes. Similar result was obtained by (Alrefaie et al., 2024) for Arabic where BPE combined with Morphemes was optimal. Finally, (Alyafeai et al., 2021) also evaluated performance of different tokenization methods on three Arabic NLP classifications task but didn't use transformer based architecture. Our method is different from these studies because we finetune on sequential language model rather than Bert-based language models. Secondly, we also analyze the performance of byte-level BPE tokenization algorithm which hasn't been performed in these studies. Finally, we try to reason with emperical evidence that the intrinsic metric that is often used during pretraining of a language model i.e. perplexity has no prediction ability about the finetuning performances.

3 Methodology

3.1 Tokenization Techniques

We have trained 6 different tokenizers keeping the vocabulary size at the constant of 30000. We intend to perform a comparison of LMs(perplexity and finetuning performance) but the perplexity scores tend to decrease with decreasing vocabulary size. Hence comparison through constant vocab size across models makes more sense. The table 1 shows encoded text for the same input by every tokenizer. Below are the specifics of how we trained these tokenizers.

1. **Word-based**: In the word-based tokenization scheme we used top 30k vocab based on frequency. We include a <unk>token to represent all the OOV words during training and evaluation. Further, we also include <num>token in the vocab to encode all the number strings in Nepali. We used torchtext from PyTorch to build the vocabulary.

2. **Morphemes**: Morphemes are the smallest subdivisions of a word that contain meaning. We used the morfessor 2.0 library to train a model that learns to break a compound word into morphemes using MAP estimate (Smit et al., 2014). We used this morfessor model to prepare a morpheme-level training corpus by using around a third of the OSCAR corpus. As suggested in (Park et al., 2020), we add a '*' token to represent the presence of space between words in the corpus so that during decoding we know when to add space and when to combine the morphemes to make compound words. Following the mentioned scheme, the text 'AB C' would be segmented as 'A B * C'.

3. **WordPiece**: The concept behind WordPiece is to divide a given word into multiple subwords that are frequently used to form a compound word. The algorithm works by first dividing a word into characters with the addition of '##' in the beginning to non-starting tokens. For example 'जीवन' would be initially divided as '(ज, ##ी, ##व, ##न)'. Then the word pieces are combined in order given by score in equation 1 where 'f' means frequency.

$$score = \frac{f_{pair}}{f_{1st} * f_{2nd}} \quad (1)$$

   This score prioritizes the frequent combination of infrequent subtokens. The encoding in WordPiece works by finding the largest subtoken that is present in the vocabulary made during training. We used the Bert WordPiece algorithm to train this tokenizer using the 'Tokenizers' Python package. This didn't support some of the modifier tokens(diacritics) in the characters hence we replaced all these tokens in the corpus with English letters during the preprocessing phase. The reverse of this preprocessing was done during decoding.

4. **SentencePiece(with BPE)**: In this tokenizer, we used character level BPE compatible with SentencePiece. Unlike WordPiece, the BPE algorithm sequentially merges the characters/subtokens based on the frequency of the merged token directly, and during encoding, the rules that are learned in training are applied sequentially (Sennrich et al., 2016). Our approach is similar but with the feature of white space handling as introduced in (Kudo and Richardson, 2018). We used the 'Tokenizers' Python package to train this tokenizer.

Tokenization Method	**Tokens**
Word	['महानायक', 'राजेश', 'हमाल', 'अहिले', 'चलचित्र', 'क्षेत्रमा', 'पातलिए', '।']
Morpheme	['महानायक', '*', 'राजेश', '*', 'हमाल', '*', 'अहिले', '*', 'चलचित्र', '*', 'क्षेत्रमा', '*', 'पातलिए', '*', '।']
WordPiece	['महान', '##ा', '##यक', 'राज□श', 'हमाल', 'अहिल□', 'चलचित□र', 'क□ष□त□रमा', 'पात', '##लिए', '।']
SentencePiece	['_मह', 'ानायक', '_राजेश', '_हमाल', '_अहिले', '_चलचित्र', '_क्षेत्रमा', '_पात', 'लिए', '_।']
BPE	**[‘à¤®à¤¹’, ‘à¤¾’, ‘à¤¨’, ‘à¤¾’, ‘à¤¯à¤ķ’, ... 37 gibberish tokens]**
Mprpheme+ BPE	**[‘à¤®à¤¹’, ‘à¤¾’, ‘à¤¨’, ‘à¤¾’, ‘à¤¯à¤ķ’, ... 37 gibberish tokens]**

Table 1: Comparison of tokenization methods for encoding the Nepali sentence ‘महानायक राजेश हमाल अहिले चलचित्र क्षेत्रमा पातलिए ।’. The □ symbols in WordPiece tokenization represents an English letter used in place of one of the modifier character(diacritic).

5. **Byte Level BPE**: Byte level BPE works like character level BPE as described in SentencePiece but the merging takes place with individual bytes instead of characters. Byte-level BPE provides more guarantee of preventing OOV words than character-level BPE as it works in a lower level of abstraction. However, using the byte-level algorithm is less efficient than the character-level algorithm as there are larger subdivisions in the former than in the latter for the same sequence of words.

6. **Morphemes and BPE**: Finally, we appliyed two tokenization algorithms Morphemes and byte-level BPE sequentially. This has the effect that the tokens that are finally trained won't span multiple morphemes. We applied the byte-level BPE to the same corpus we made using morfessor library as described already.

3.2 Model Architecture

For every tokenization technique, the same model architecture was used for pretraining the language model. A simple architecture consisting of 6 layers of transformer encoder blocks with 6 attention heads each was modeled. The size of input embedding layer used for tokens was 300 and the dimension used for feedforeward network was 1024. To regularize, we used a dropout rate of 20%. Finally, both the batch size and the sequence length used were 64. The parameters used are summarized in the table 2. The total number of parameters in the 30k vocab LMs was 24M.

For finetuning, we added a hidden layer and an output layer feedforward network on top of the representation learned on the final layer of the last transformer block. The dimension of the hidden layer used was again 1024 with ReLU activation function, and the output layer's dimension was equal to the number of classes for the particular task.

Parameter	Value
emsize	300
dim_feedforeward	1024
nlayers	6
nhead	6
dropout	0.2
batch size	64
seq. length	64

Table 2: Transformer Model Parameters

4 Experiment

4.1 Dataset for LM Pre-training

We used Oscar corpus for the Nepali language (Ortiz Suárez et al., 2019) with the removal of duplicated sentences. The total data that became available from this corpus was 1.2GB. From this dataset, four versions of LMs were trained i.e. word-based, SentencePiece, WordPiece and BPE-tokenized LMs on 300k paragraphs while morphemes and morphemes with BPE-tokenized LMs were trained on 100k paragraphs. Before training the sentences were preprocessed, tokenized, encoded(given id), and then batched. After batching i.e. grouping 64 training examples, we get 16791 unique batches of training data when word-based tokenization is used. Using any other preprocessing and tokenization scheme led to larger number of batches as shown in Figure 1. The morpheme-based models were only trained on a third of the dataset hence the percentage was calculated relative to the batches calculated using word-based tokenization on this dataset.

4.2 Pre-Training

We trained 6 transformer-based language models using tokenizers of 3.1 with the architecture as de-

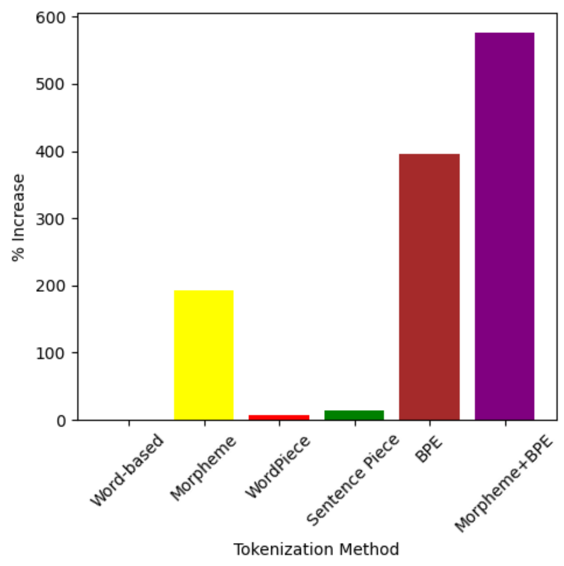


Figure 1: Percentage increase in number of batches with different tokenization methods relative to word-based tokenization.

scribed in 3.2. Additionally, we also trained a word-based language model with 60k vocabulary but the same model architecture. This provided us with some insights into performance based on vocabulary size. The model evaluation during the pertaining is based on the perplexity score which can be calculated using the eq. 2 where we have replaced $P(x_i|context)$ with $P(x_i)$.

$$\text{Perplexity} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(x_i)\right) \quad (2)$$

4.3 Finetuning

The pre-trained language models were finetuned on Nep-gLUE benchmark datasets (Timilsina et al., 2022) which consists of four Natural Language Understanding tasks. The details on the finetuning approach and the datasets are briefly mentioned below:

4.3.1 Categorical Pair Similarity(CPS)

Categorical Pair Similarity(CPS) is a pair-wise sequence classification task where the job is to find whether the given two sequences belong to the same category. CPS dataset was created (Timilsina et al., 2022) by extracting 2.5k of similar sequence pair for each of the 9 categories(total = 22.5k) and a 22.5k of different category sequence pair through random sampling accross dissimilar pair formed by pairing 2.5k sentences in each category with sentences from different category, resulting in a balanced dataset of 45k paired samples. Both of the sentences were passed through the pre-trained model and the finetuning was performed on the concatenation of the representations from both the sequences. The prediction category was 1 for a similar pair and 0 for a dissimilar pair and, truncation was used whenever the sequence length limit was reached.

4.3.2 Part of Speech Tagging(POS)

Part of Speech Tagging(POS) is a sequence labeling task where every word in the sequence of text has to be classified to one of tags such as noun, verb etc. This dataset was taken from a publicly available repository (Nepali Bhasa, 2020) which consists of 4251 sentences with more than 110k labels accross 39 tags. For preprocessing, multiple sequences for a same sentence was created and label was generted for each sequence. For example: Sentence ABC with words A(Tag: La),B(Tag: Lb) and C(Tag: Lc) can be decomposed into sentences A, AB, ABC. Then the label for sequence A is La , AB is Lb and ABC is Lc. Finally, the finetuning was performed using the representation of the last token. Hence to categorize the tag of B in sequence AB, we take the representation of B by passing AB into the pretrained model. Also, the truncation is performed from the beginning whenever the maximum sequence length is reached meaning that if the length limit is 2 then the sequence ABC would be trucated to BC.

4.3.3 Named Entity Recognization(NER)

Similar to the POS task, Named Entity Recognition (NER) is also a sequence labeling task but here the job is to find the type of named entity like person, location or organization. The dataset used in the benchmark (Singh et al., 2019), consists around 3289 sentences with labels that belong to one of 7 classes including the other token '**O**'. Similar approach to POS tagging task was used as mentioned in sec. 4.3.2 in preprocessing, truncation and finetuning.

4.3.4 Content Classification(CC)

Content classification is a task where the natural language content or sequence has to be classified in one of the categories. CC dataset was created (Timilsina et al., 2022) by scraping news articles from 9 different categories consisting of around 45k data points. The finetuning was performed on the sequence with truncation from the end.

5 Result and Discussion

5.1 Perplexity Trend

Table 3 shows the perplexity values at the end of training and validation. The training and validation perplexity is lowest for Morpheme with BPE followed by only BPE. Similarly, the score is highest for SentencePiece followed by WordPiece. Surprisingly, the performance of word-based tokenization is better than both WordPiece and SentencePiece. From the figure 2, we can see the trend of both the training and validation perplexity score(in log scale) from the start of training to the end. The training trend of all the versions show us that the perplexity decreases during the initial steps and then the curve becomes flat. The validation trend of WordPiece, SentencePiece, Word-level and Morpheme is similar with an initially large decrease in perplexity before becoming constant. However, the byte-level BPE-based algorithms have flat curve from the beginning reflecting the large number of training steps already performed during 1^{st} epoch due to the large number of batches.

Tokenization	**Training**	**Validation**
BPE	6.328	5.863
Morpheme+BPE	3.854	3.677
SentencePiece	134	120.6
WordPiece	125.6	116.3
Morpheme	14.09	13.71
Word based(30k)	106.8	97.08

Table 3: Perplexity values during training and validation

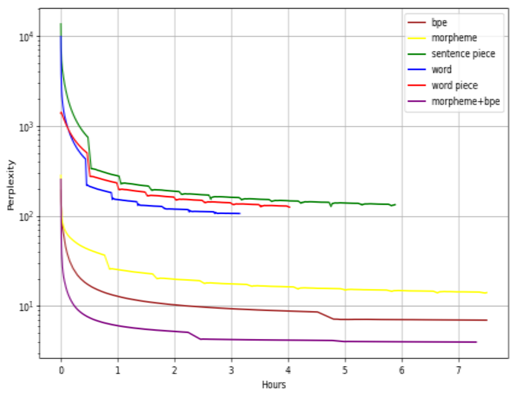
Figure 3 shows the comparison of the perplexity trend during training and validation for word-based tokenization with 30k tokens and 60k tokens. The perplexity score for 30k is less than for 60k during every phase of training and validation suggesting that an increase in vocab size in this region also tends to increase in perplexity.

5.2 Understanding Perplexity

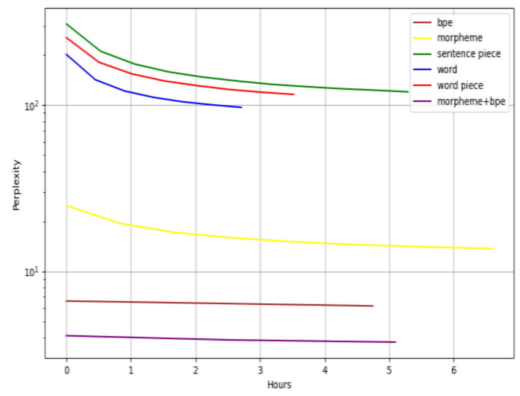
The perplexity scores for the tokenization method involving Morpheme or BPE are much lower than other tokenization methods. Does it mean that this tokenization leads to a better language model than the other tokenization methods? We conduct frequency analysis(to explain the perplexity trend) on both the training and evaluation corpora using the tokenizers trained in the training corpus, as illustrated in Figure 4. The frequency analysis of all vocab reveals that the SentencePiece algorithm has larger frequency of the all the middle tokens(up to 25k token shown). In fact, larger the perplexity score at the end of the evaluation higher is the frequency curve of that tokenization method. However, if we look at the most frequent tokens as shown in the frequency analysis of the top 15 vocabs, we find that the worst-performing SentencePiece algorithm starts from the lowest normalized frequency. This means that the SentencePiece algorithm produces token frequencies that are more uniformly distributed(only relatively) than other algorithms compared here. Hence, this relatively higher uniformity means that the probability of the most probable prediction from token output distribution is likely to be less than other algorithms. In other words, the algorithm less confidently predicts frequent tokens and more confidently predicts lower frequency tokens relatively. Table 4 shows that the most frequent token in SentencePiece covers only 4.7% of the corpus while the most frequent token('*') in morpheme covers 47.9% of the corpus. Hence, the morpheme makes almost half its prediction very confidently and that is why it has very low perplexity compared to other subword and word-based algorithms.

Another way to look at why BPE performs better(in perplexity score) than other algorithms is that working with bytes can produce larger number of high-frequency tokens than other methods. For instance, the BPE tokenization method has a much higher normalized frequency for around 1st hundred most frequent tokens. Hence BPE captures this repetitive pattern easily in text, working at the byte level than working at the character level and hence is more confident about its prediction. But does this apparent good performance in perplexity mean a higher understanding capacity?

On the contrary, we found that on average the worst-performing algorithm in perplexity i.e. SentencePiece is better than all other algorithms when finetuned for NLU tasks. Also, though byte-level tokenizations have led to very confident language models, given a training corpus the total training time is much higher. This is because division into smaller sequences of tokens leads to generating a large number of tokens overall during encoding. Moreover, generating a large number of tokens

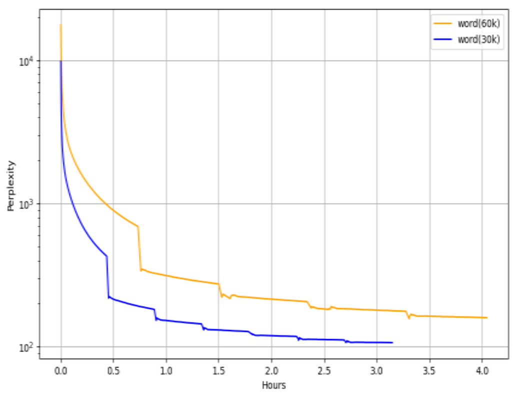


(a) Training Perplexity

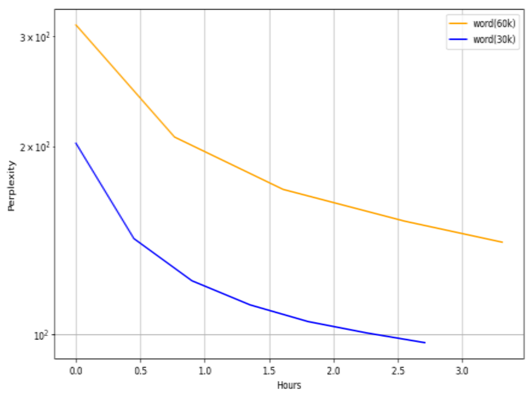


(b) Validation Perplexity

Figure 2: Comparison of tokenization methods for perplexity

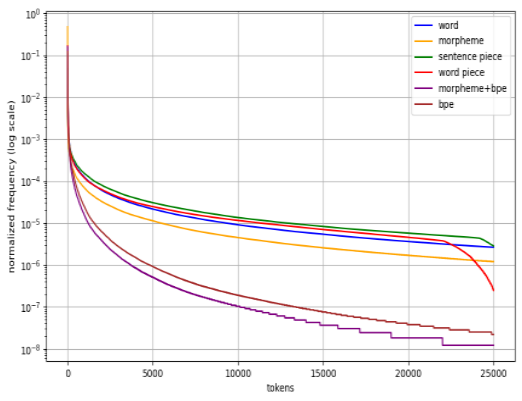


(a) Training Perplexity

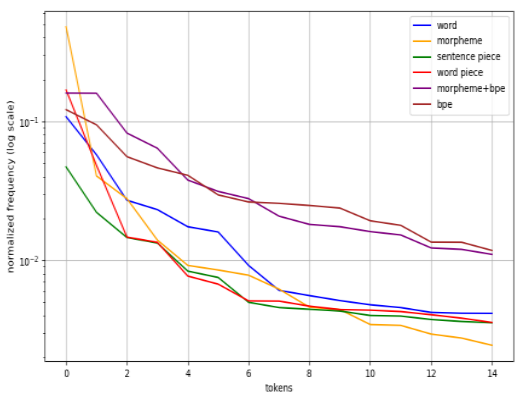


(b) Validation Perplexity

Figure 3: Comparison of vocabulary size for perplexity



(a) Frequency of first 25k vocabs(descending)



(b) Frequency of top 15 vocabs(descending)

Figure 4: Comparison of normalized frequency of tokens in the corpus

leads to less contextual understanding considering the same sequence length.

Tokenization	% of most freq. token
Morpheme+BPE	0.160
Bpe	0.121
SentencePiece	0.047
WordPiece	0.168
Morpheme	0.479
Word	0.108

Table 4: Tokenization Methods and normalized frequency of the most frequent token

5.3 Finetuning Performance

The result of finetuning on 4 tasks of NepgLUE benchmark datasets are displayed in Table 5. The best-performing model/models are boldfaced across all the tasks and the GLUE scores. We observe that for the CPS task, the worst-performing tokenization method in perplexity i.e. SentencePiece has the best performance in the macro-f1 score along with 30k and 60k versions of word-based tokenization, and the best-performing tokenization method in perplexity i.e. Morpheme+BPE has the worst performance in the macro-f1 score. For POS tagging, Morpheme+BPE performs best in the macro-f1 score. However, though SentencePiece has the largest perplexity, it outperforms all other tokenization schemes including using BPE(with morpheme). This again tells us that perplexity is very poor in determining the representation learning capability of Language models. In the NER task, the Morpheme algorithm performs best. All other algorithms also have similar performance except the worst-performing algorithm WordPiece. Finally, for CC SentencePiece Algorithm prevails followed by word-based and Morpheme-based tokenization schemes. The byte-based algorithms perform very poorly comparatively.

The NepGLUE score calculated by averaging the performance across all tasks reveals that SentencePiece is the best tokenization method with a score of 0.88 and WordPiece is the worst with a score of 0.78 followed by Morpheme+BPE with a score of 0.81. Similarly, (Liu et al., 2019) also hinted that byte-level BPE algorithms performed worse than character-level BPE. Finally, if the comparison between the word-based algorithms with 30k and 60k vocabulary sizes is done on finetuning performance, we can see that larger vocab size has led to greater(although marginally) or equal performance across all tasks. However, drastic performance improvement wasn't seen. Unlike (Toraman et al., 2022), we didn't change the model size between two different vocab sizes which could be the reason of neglegant performance improvement, as pointed in (Alrefaie et al., 2024).

6 Future Work

Although we have compared the model performance for 6 different tokenization schemes there are other tokenization techniques that we didn't consider like n-gram characters, Unigram LM (Kudo, 2018), and SentencePiece with sampling(to increase robustness) (Kudo and Richardson, 2018). Including these algorithms in a future study would provide a more comprehensive comparison. Further, a comparison of perplexity and downstream performance due to variable vocabulary size has been limited to word-based tokenization with only two different vocabulary sizes. We leave this as future work. Also exploring the performance of larger-sized models with multiple languages can be an interesting future direction.

7 Conclusion

In this paper, we made a comparison of perplexity scores with different tokenization methods using autoregressive language models. We found that normally a granular tokenization method leads to a lower number of high-frequency tokens which corresponds to a smaller value of perplexity. Moreover, we also made a comparison of perplexity scores on word-based tokenization with two different vocabulary sizes which shows an increase in vocabulary size leads to an increase in perplexity. Finally, we finetuned the language models on various NLU tasks which revealed that best performing tokenization algorithms in pertaining of LMs(through perplexity) such byte-level BPE(with and without Morphemes) doesn't lead to best performance on the understanding tasks. And, we found that on average, SentencePiece algorithm is better than other tokenization methods.

8 Limitations

Despite the best efforts, there remain several limitations in our study. Firstly, the language models that we trained have only 24M parameters(30k

Tokenization	CPS	POS	NER	CC	NepGLUE
Morpheme+BPE	0.86	**0.90**	0.72	0.77	0.81
BPE	0.89	0.87	0.75	0.81	0.83
SentencePiece	**0.96**	0.89	0.74	**0.91**	**0.88**
WordPiece	0.93	0.71	0.64	0.85	0.78
Morpheme	0.94	0.74	**0.76**	0.88	0.83
Word (30k)	**0.96**	0.75	0.72	0.90	0.83
Word (60k)	**0.96**	0.76	0.74	**0.91**	0.84

Table 5: Finetuning performance(Macro-F1 score) of language models with different tokenization schemes on four different NLU tasks Categorical Pair Similarity(CPS), Parts Of Speech Tagging(POS), Named Entity Recognition(NER) and Content Classification(CC) from Nep-gLUE benchmark. The final NepGLUE score represents the average performance across all tasks.

versions). It is larger than only the smallest of the Bert models(14M) and is nowhere near the range of large sequential language models. Hence, the applicability of the result to LLMs cannot be confidently stated. Secondly, our pre-trained models use a maximum sequence length of only 64 hence the comparison between tokenization algorithms such as byte-level BPE and word-based could be considered unfair in terms of the number of context words seen(though it is a fair comparison computationally). Lastly, the benchmark datasets that have been used don't contain sequence generation tasks such as text summarization, machine translation, or question answering. Hence, it would be wrong to generalize the results to generation tasks unless further research is done in that area.

References

Mohamed Taher Alrefaie, Nour Eldin Morsy, and Nada Samir. 2024. Exploring tokenization strategies and vocabulary sizes for enhanced arabic language models.

Zaid Alyafeai, Maged S. Al-Shaibani, Mustafa Ghaleb, and Irfan Ahmad. 2021. Evaluating various tokenizers for arabic text classification. *Neural Processing Letters*, 55:2911–2933.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and

Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Utsav Maskey. 2023. distilgpt2-nepali. `https://huggingface.co/Sakonii/distilgpt2-nepali`.

Utsav Maskey, Manish Bhatta, Shiva Bhatt, Sanket Dhungel, and Bal Krishna Bal. 2022. Nepali encoder transformers: An analysis of auto encoding transformer language models for nepali text classification. In *Proceedings of the 1st Annual Meeting of the ELRA/ISCA Special Interest Group on Under-Resourced Languages*, pages 106–111.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari.

Nepali Bhasa. 2020. pos-tagger: Part of speech tagging in nepali. `https://github.com/nepali-bhasa/pos-tagger`.

Nobal Niraula and Jeevan Chapagain. 2022. Named entity recognition for nepali: Data sets and algorithms. *The International FLAIRS Conference Proceedings*, 35.

Pedro Javier Ortiz Suárez, Benoît Sagot, and Laurent Romary. 2019. Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures. In *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*, Cardiff, United Kingdom. Leibniz-Institut für Deutsche Sprache.

Kyubyong Park, Joohong Lee, Seongbo Jang, and Dawoon Jung. 2020. An empirical study of tokenization strategies for various Korean NLP tasks. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 133–142, Suzhou, China. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Rajan. 2021. Nepalibert. `https://huggingface.co/Rajan/NepaliBERT`.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units.

Oyesh Mann Singh, Ankur Padia, and Anupam Joshi. 2019. Named entity recognition for nepali language. In *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, pages 184–190.

Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. 2014. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 21–24, Gothenburg, Sweden. Association for Computational Linguistics.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1017–1024.

Sulav Timilsina, Milan Gautam, and Binod Bhattarai. 2022. Nepberta: Nepali language model trained in a large corpus. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing*, pages 273–284.

Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. 2022. Impact of tokenization on language models: An analysis for turkish. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 22:1 – 21.

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