`Tokenization Matters`: Improving Low-Resource Language Modeling for Yakut with Custom Tokenizer

Mariia Eremeeva* emariia
Abu Bakr Rahman Shaik* ashaik
Rada Kamysheva* rkamysheva
Nishant Kumar Singh* singhni

Abstract

Recent advances in Natural Language Processing have primarily benefited high-resource languages, leaving low-resource languages underrepresented due to limited annotated data and linguistic resources. Tokenization is a critical challenge for morphologically rich low-resource languages, where standard approaches often fail to capture linguistic structure. In this work, we address these limitations for Yakut (Sakha), a Turkic language with Cyrillic orthography, by engineering a Byte Pair Encoding (BPE) tokenizer with Yakut-specific pre- and postprocessing rules and tailored special tokens. The tokenizer is integrated into the Llama 3.2 architecture, with a reconstructed input embedding layer to align with the reduced vocabulary and the use of weight tying for output layer consistency. Fine-tuning is performed in multiple phases on a curated Yakut corpus, employing RoPE scaling for extended context and loss masking for answer supervision in Q&A fine-tuning. The resulting model achieves substantial improvements in exact match accuracy and output coherence over the baseline, demonstrating the effectiveness of language-specific tokenization and adaptation for low-resource NLP.

1 Introduction

Recent advances in Natural Language Processing have primarily benefited high-resource languages such as English, Chinese, and Spanish, driven by extensive datasets. In contrast, numerous low-resource languages remain significantly underrepresented due to the scarcity of annotated corpora and linguistic resources. This disparity hinders the development and deployment of effective NLP applications for such languages (Pakray et al., 2025).

Effective tokenization critically influences the accuracy of subsequent NLP tasks (Pattnayak et al., 2025), particularly for morphologically rich languages, where word formation often involves extensive use of prefixes, suffixes, and complex morphological patterns. Standard tokenizers, which are mainly designed based on data from high-resource languages with relatively simpler morphological structures, typically fail to adequately segment or represent text in low-resource languages (Toraman et al., 2023; Arnett and Bergen, 2024). These tokenizers tend to overlook subtle morphological distinctions and nuances inherent to low-resource languages, thereby generating suboptimal token sequences that negatively impact the overall performance of downstream NLP tasks.

In this paper, we focus on Yakut (Sakha), an agglutinative Turkic language predominantly spoken in the Sakha Republic, Russia, to illustrate the linguistic complexities and resource constraints typical of low-resource languages. Known by approximately 500,000 Siberians, the Yakut language represents a unique linguistic blend: it uses the Cyrillic alphabet with additional characters, reflecting Slavic influence, while its grammatical structure is rooted in its Turkic origins (Wikipedia, 2025). This hybrid nature, combined with limited digital resources and the absence of dedicated language models, presents significant challenges for conventional tokenization methods. Enhancing tokenization is critical for enabling effective language technology applications in Yakut and other typologically similar low-resource languages.

To address the limitations of ineffective tokenization in low-resource languages, we develop a BPE-based Yakut tokenizer specifically optimized for Yakut's Cyrillic script and agglutinative morphology, reducing the standard LLM vocabulary size by 75% to 32,000 tokens and improving subword segmentation coherence. We incorporate Yakut-specific pre-tokenization rules based on whites-

*These authors contributed equally to this work.

pace and punctuation splitting, as well as post-processing to ensure Cyrillic script integrity. Furthermore, we address the limitations of existing multilingual tokenizers by incorporating Yakut-specific pre-tokenization rules based on whitespace and punctuation splitting, as well as post-processing to ensure script integrity. We introduce language identification tokens and integrate the tokenizer into Llama 3.2 by constructing a new embedding matrix that copies weights for shared tokens and randomly initializes Yakut-specific embeddings, with the output layer synchronized via weight tying. For evaluation, we assess performance using translated XQuAD and synthetic QA datasets, employing both exact match accuracy and qualitative analysis. Our results show that the Yakut-specific tokenizer yields substantial improvements in exact match accuracy, grammatical correctness, and output coherence, while eliminating language switching and segmentation artifacts observed with standard multilingual models.

2 Related Work

The development of NLP tools for low-resource languages has gained increasing attention in recent years, driven by the broader goal of achieving linguistic inclusivity and accessibility in NLP applications. Several studies have highlighted the unique challenges and explored solutions specifically tailored for languages with limited resources. There is a significant disparity in NLP advancements between high-resource and low-resource languages, highlighting the need for dedicated linguistic resources and datasets (Pakray et al., 2025).

For instance, morphologically rich low-resource languages pose a significant problem for tokenization. In agglutinative languages (e.g. Turkic languages like Turkish, or Uralic languages like Finnish and Hungarian), a single orthographic word can contain many morphemes (prefixes, suffixes, infixes), leading to extremely large vocabularies of distinct word forms, making naive approaches to tokenization impractical, especially for low-resource languages (Parra, 2024).

2.1 Subword Segmentation Algorithms

Byte-Pair Encoding (Sennrich et al., 2016) and WordPiece (Wu et al., 2016) introduced frequency-based subword merging to address open-vocabulary issues in Neural Machine Translation (NMT) systems. SentencePiece (Kudo and Richardson, 2018) unified BPE and Unigram Language Model into a language-agnostic framework that eliminates language-dependent pre-tokenization. Although these methods perform well on high-resource data, they often result in suboptimal vocabularies for morphologically rich, low-resource languages.

2.2 Morphological and Unsupervised Segmentation

Unsupervised morphology induction tools, such as Morfessor, segment words into morphemes by optimizing a minimum description length objective (Creutz and Lagus, 2005; Smit et al., 2014). Subword regularization further improves robustness by sampling multiple segmentations during training (Kudo, 2018). However, these approaches have predominantly been evaluated on Indo-European languages and degrade markedly when confronted with the extensive agglutinative morphology found in Turkic languages like Yakut.

2.3 Tokenization in Multilingual Pretrained Models

Pretrained multilingual models (mBERT (Devlin et al., 2019), XLM-R (Conneau et al., 2020), mT5 (Xue et al., 2020)) use a shared subword vocabulary across dozens of languages, facilitating transfer but often under-allocating tokens to low-resource languages. Empirical analyses show that these shared vocabularies produce highly fragmented, less semantically coherent tokens for underrepresented languages, impairing zero- and few-shot performance (Pires et al., 2019; Arnett and Bergen, 2024; Toraman et al., 2023).

3 Data

3.1 Corpora

The project utilizes three primary data corpora. The first corpus consists of parallel text data sourced from the OPUS collection (Tiedemann, 2012), specifically the Wikimedia and Tatoeba corpora containing Yakut-Russian sentence pairs, with the Wikimedia corpus providing 8,256 sentence pairs and the Tatoeba corpus contributing 996 sentence pairs. The second corpus comprises monolingual Yakut text extracted from the OSCAR (Open Super-large Crawled Aggregated coRpus) dataset (Ortiz Suàrez et al., 2019; Ortiz Suárez et al., 2020), which contains web-crawled text data in Yakut. And the third corpus is a set of raw

Yakut text files used in the Sakha-NLP project on GitHub (Pukhov et al.). In addition to that, we also utilize the instruction-output pairs available on HuggingFace in the "MURI-IT" dataset (Köksal et al., 2024).[*]

3.2 Pre-processing

The preprocessing pipeline implements a comprehensive text cleaning and normalization framework that performs URL removal, HTML tag stripping, Unicode normalization, and whitespace regularization while applying quality filters based on minimum character counts and word requirements. For the parallel corpus, the preprocessing incorporates a sophisticated neural machine translation component using the Helsinki-NLP model from HuggingFace (Tiedemann and Thottingal, 2020) to translate Russian text into English, thus creating trilingual Yakut-Russian-English datasets that we aim to utilize for training. The parallel data cleaning process includes length ratio filtering (0.3 to 3.0 ratio bounds) to ensure translation quality and alignment consistency, with batch processing capabilities for efficient neural translation.

3.3 Instructions Back-engineering

We implemented the Multilingual Reverse Instructions (MURI) methodology (Köksal et al., 2024) to generate high-quality instruction tuning datasets for the Yakut and English languages without requiring human annotators. The MURI approach inverts the traditional instruction dataset creation process by starting with high-quality human-written texts in the target language and generating appropriate instructions that would naturally lead to those texts as responses, thereby preserving cultural relevance and linguistic authenticity while avoiding translationese artifacts. After translating the initial Russian data into English, as described in Section 3.2, we continued with the core instruction generation step. We used Google's Gemini models, specifically Gemini-2.0-flash-lite (Google, b), accessed through the Generative AI API to generate contextually appropriate English instructions using carefully crafted prompting that demonstrated the relationship between potential instructions and their corresponding outputs. Finally, we translated the generated English instructions back into Russian and then into Yakut, using Google Cloud translation API (Google, a), which was then sample-wise verified by a Yakut native speaker. We designed this middle-man architecture, since translating from Russian into Yakut demonstrates significantly better results. This approach enabled us to create instruction-output pairs entirely in the target language while maintaining the semantic relationships established during the English generation phase.

[*] https://huggingface.co/datasets/akoksal/muri-it

4 Tokenizer Engineering

The performance of Large Language Models in multilingual or low-resource settings is highly influenced by the effectiveness of the tokenizer. The main goal of this project was to make a good tokeniser for Yakut (Sakha), which, as mentioned in Section 1, is a Turkic language written in the Cyrillic alphabet and is not very well-known in the mainstream NLP. The default tokeniser that Llama 3.2 uses underperforms on Yakut since it is not tailored to the morphology and character distribution that are common in this language.

This section documents the methodology used to engineer a custom tokenizer tailored for Yakut, integrated into the Llama 3.2 pipeline for subsequent fine-tuning.

4.1 Tokenizer Limitations in Multilingual Contexts

Initial testing revealed that Llama's built-in tokenizer was unable to tokenize Yakut words into intuitive subwords. For instance, see Figure 1.

Incomprehensible tokenization leads to a total loss of semantic integrity of words, reducing training efficiency and performance in downstream tasks. This limitation prompted the design of a dedicated BPE-based tokenizer.

4.2 Implementation Details

As seen in Figure 2, the tokenizer implementation was performed in 6 phases.

4.2.1 Pre-tokenization Design

As seen in the figure, the original Llama tokenizer tokenizes the text by mapping it to some random strings of special characters. This approach leads to semantic unawareness. Hence, given Yakut's Cyrillic script and morphological richness, our method employs *WhitespaceSplit* to preserve word boundaries and *Punctuation* to ensure that punctuation is treated as distinct tokens.

Language	Text	Llama-3.2-1B Tokenizer	Our Trained Tokenizer
Yakut	Мин аатым Кэскил	'ÐĮ', 'Ð¸Ð½', 'ĠÐ°', 'Ð°ÑĤ', 'ÑĭÐ¼', 'ĠÐļ', 'Ñį', 'ÑģÐºÐ¸', 'Ð»', '.'	'<sah>', 'Мин', 'аа', 'тым', 'Кэскил', '.'
Russian	Привет, как дела?	'ÐŁÑĢ', 'Ð¸Ð²', 'ÐµÑĤ', ',', 'ĠÐºÐ°Ðº', 'ĠÐ´ÐµÐ»Ð°', '?'	'<ru>', 'При', 'вет', ',', 'как', 'дела', '?'
English	To fix this issue	'To', 'Ġfix', 'Ġthis', 'Ġissue'	'<en>', 'To', 'fix', 'this', 'issue'

Figure 1: Tokenization Comparison: Default Llama 3.2 vs. Custom-Trained Tokenizer

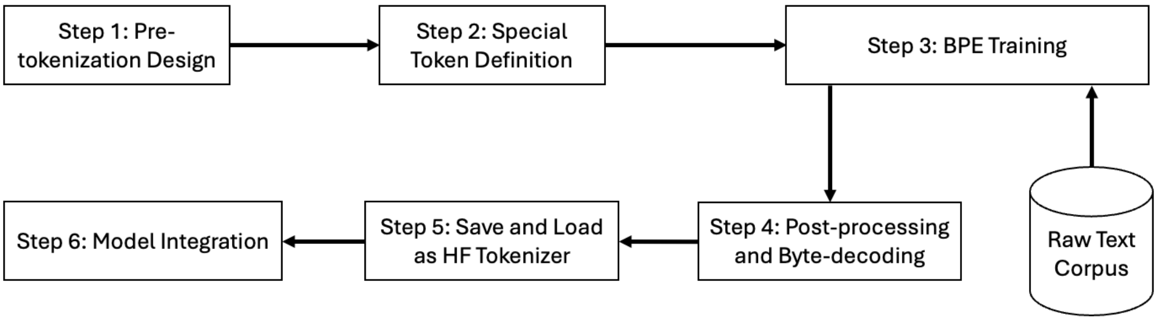

Figure 2: Tokenizer Training Workflow

4.2.2 Special Token Definition

To facilitate multilingual identification, specific markers were included in alignment with Llama 3's existing special token format.

```
special_tokens = [
    "<|begin_of_text|>",
    "<|end_of_text|>",
    "<unk>",
    "<pad>",
    "<en>",
    "<ru>",
    "<sah>"
]
```

These allow prefix-based language conditioning during training and inference.

4.2.3 BPE Training

The tokenizer was trained using BpeTrainer. BPE is a subword tokenization method introduced to NLP by Sennrich et al. (Sennrich et al., 2016). It involves iteratively merging the most frequent adjacent token pairs (starting from characters) to form a vocabulary of subwords.

The vocabulary size of the original Llama tokeniser is 128,000, as it fully supports 8 languages and partially supports several other languages. However, since our focus is only on 3 languages, the vocabulary size was reduced by 75% to mere 32,000.

4.2.4 Post-processing and Byte-decoding

The decoder sequence minimally postprocesses the text by replacing underscores with whitespace and decodes the subword tokens into words, ensuring accurate detokenization and script integrity in Cyrillic.

4.2.5 Save and Load as HF Tokenizer

The trained tokenizer file is saved in the format of an HF tokenizer with the contents named `tokenizer.json`, `tokenizer_config.json` and `special_tokens_map.json`.

4.2.6 Model Integration

Embedding Layer Adjustment

Adapting the Llama 3.2 model to the new tokenizer requires an updated input embedding layer. The input embedding layer of the model is essentially a lookup table that maps each token ID to an embedding that represents the semantic meaning of that token in a high-dimensional space.

Since the Llama model was trained with a large vocabulary of 128,000 tokens, and our new tokenizer has a smaller vocabulary of 32,000 and a different token set, this results in incompatibility with the existing embedding layer.

The solution is to construct a new embedding matrix that matches the new tokenizer's vocabulary.

This matrix must be initialized carefully to retain knowledge for tokens shared between the original and new vocabularies, and sensibly initialize new tokens (such as those representing Yakut-specific subwords).

Steps:

1. **Create a new embedding matrix:**
   Use `nn.Embedding`, setting dimensions to match the new vocabulary size and the hidden size of the original model.
   - Original embedding: $[128{,}000 \times 4096]$
   - New embedding matrix: $[32{,}000 \times 4096]$
2. **Copy embeddings for overlapping tokens:**
   For each token in the new vocabulary that also exists in the original Llama vocabulary, the corresponding embedding is copied from the original model.
3. **Initialize embeddings for new tokens:**
   New embeddings are randomly initialized using a normal (Gaussian) distribution, consistent with Llama's original initialization strategy.

Output Layer Synchronization

This step ensures that the language modelling head (output layer) is consistent with the updated embedding space.

Since Llama 3.2 employs *weight tying*, where the input embedding matrix and the output projection layer share the same weight tensor, we assign:

```
model.lm_head.weight =
            new_embeddings.weight
```

For non-tied architectures, we would reconstruct a new output layer and explicitly copy weights. However, in our case, this assignment resolves both the input embedding and output layer requirements while maintaining the mathematical relationship:

$$logits = hidden\_states \cdot E^{\top}$$

where E is the unified embedding matrix.

This process creates a hybrid embedding matrix that preserves pretrained knowledge for familiar tokens and reasonably initializes new tokens. It is a crucial step that ensures the adapted Llama model can accept input from the new Yakut tokenizer without architectural mismatches or degraded performance.

5 Training Methodology

The training of the model was divided into two phases so that the model was able to learn and comprehend the Yakut (sah) language in a step-by-step manner. The two phases span across slightly different techniques to fine-tune the model weights for distinct tasks and thereby improve it's performance. The entire training took place on an NVIDIA A100 GPU (40GB) using Google Colab Pro.

Phase 1: Initial Language Adaptation

The first phase of training began by adapting the pre-trained Llama 3.2 and our custom tokenizer's integration with the utilisation of 340.7 megabytes of data containing 366,093 text samples of raw Yakut text extracted from the sources mentioned in Section 3.1, comprising approximately 1.2 million tokens after rigorous preprocessing.

With a learning rate of 3e-5 and a weight decay of 0.01, the model training lasted approximately 11 hours and 45 minutes. It had an effective batch size of 256 achieved through gradient accumulation (32 steps with a per-device batch size of 8). For the training procedure, the sequence length was fixed at 512 tokens, and the model completed 6,795 training steps over 5 epochs.

The results for manual testing demonstrated a limited semantic depth for the model's ability to process basic Yakut (sah) syntax, as per its response to the input "once" (in Yakut) generating output translating to "one day you have to achieve a lot. I think,". This indicates that there are emerging pattern recognition capabilities but lack contextual sophistication. In addition, the training loss decreased from 5.57 to 0.66, while the validation loss improved from 5.36 to 1.66 during this phase.

Phase 2: Training and Data Preparation

Data Preparation

The trained model supports special tokens such as `<|begin_of_text|>`, `<|end_of_text|>` and language delimiters: `<en>`, `<ru>`, and `<sah>` to identify language-specific inputs.

The translated XQuAD dataset, originally provided in JSON format with a triplet structure of *context*, *question*, and *answer*, was preprocessed to incorporate these tokens. Each sample was converted into a prompt with the following format:

```
<|begin_of_text|><sah>
context: {context}
question: {question}
answer: {answer}<|end_of_text|>
```

The preprocessed data preserves clear separation of components (context, question, answer), while maintaining the complete structure for efficient fine-tuning.

Extended Context Handling with RoPE Scaling

Initially, the fine-tuned Llama model encountered difficulties when handling longer input sequences, often replicating the input instead of generating meaningful output. To resolve this, RoPE (Rotary Position Embedding) positional encodings were employed. RoPE integrates relative positional information directly into the self-attention mechanism using a rotation matrix (Su et al., 2023).

Dynamic NTK-aware scaling was applied to extend the context length capability of the model. This approach adjusts the base frequency of the positional embeddings based on the input sequence length. The following configuration was used:

```
model.config.rope_scaling = {
    "type": "dynamic",
    "factor": 2.0
}
```

This enhancement increased the model's effective context window to 4096 tokens, enabling better handling of long passages while preserving positional accuracy.

Custom Data Collation with Loss Masking

A custom data collator, `QADataCollator`, was implemented by extending HuggingFace's `DataCollatorForLanguageModeling`. This collator applied a selective loss mask to guide the model's attention primarily to the answer segment of each input.

Specifically, loss computation was enabled only for the tokens that followed the answer prefix. If the position of the answer could not be accurately located, the loss was applied across the entire sequence as a fallback.

Training Configuration

Training was performed using HuggingFace's `Trainer` API with the hyperparameters mentioned in the Appendix.

These settings were selected to ensure a balance between computational efficiency and training stability.

Inference

After training, the model was used to generate answers for both unseen samples from the XQuAD test set and a synthetic dataset. The generation process was driven by prompts containing the *context* and *question*, and answers were generated using sampling-based decoding using *top_k*, *top_p* and *temperature*. These parameters promote diversity in responses while maintaining coherence.

The phase-by-phase training approach cautiously addressed the learning and challenges of Yakut language processing, with each phase building on previous successes and also taking specific constraints into account. Although quantitative metrics demonstrate consistent progress across phases, the manual evaluation areas imply that the model cannot be generalized for Yakut yet. Persistent challenges with managing long context texts and out-of-vocabulary (OOV) words indicate areas that could use more research and possible fine-tuning adjustments.

6 Evaluation

6.1 Evaluation Datasets

To thoroughly evaluate our model performance in Yakut, we created three complementary evaluation datasets, each targeting different aspects of language understanding.

The first dataset is a Yakut-translated version of the original *XQuAD benchmark released by DeepMind* (Artetxe et al., 2020), which includes 1190 question-answer pairs based on 240 short context paragraphs, see an example below.

XQuAD Example

Context: The Panthers defense gave up just 308 points, ranking sixth in the league, while also leading the NFL in interceptions with 24 and boasting four Pro Bowl selections.
Question: How many points did the Panthers defense surrender?
Answer: 308

We translated the original XQuAD dataset from English to Yakut using the Google Cloud Translation API. The translation got validated by a native speaker. The dataset was first converted to SQuAD-compatible format to facilitate structured processing. This approach enabled efficient gener-

ation of a Yakut version of the benchmark while maintaining the original semantic structure.

In parallel, we also translated the more recent *Hugging Face version of XQuAD* into Yakut, using the same approach. This version provides a standardized and updated set of QA pairs and helps validate the consistency of the model performance across different versions of the same benchmark.

Both of these translated datasets test the model ability to understand natural language questions and locate correct answers within context, which is a key measure of reading comprehension.

Lastly, we develop a *synthetic QA dataset* by translating the English source material into Yakut using an automated translation API, followed by a thorough review and confirmation by a native Yakut speaker to ensure linguistic accuracy and cultural appropriateness. The dataset consists of context passages, simple everyday questions, four answer options, and a designated correct answer. This combination of automated generation with human validation allows us to create realistic, natural language examples that challenge the model's ability to perform complex reasoning and comprehension tasks in Yakut. The dataset contains 100 examples emphasizing both linguistic correctness and relevance to real-world usage, see an example in English below.

Synthetic Dataset Example

Context: Cats sleep for many hours each day, usually in warm spots.
Question: Which animal sleeps a lot in warm places?
Options: Dog, Cat, Fish, Bird
Answer: Cat

Together, these resources enable systematic evaluation of reading comprehension, language understanding, and reasoning abilities in Yakut, ensuring both linguistic accuracy and practical relevance.

6.2 Results

6.2.1 XQuAD Evaluation

A translated version of the XQuAD benchmark in Yakut was used to compare our model with the base Llama 3.2 using Exact Match (EM) evaluation metric, which measures the percentage of predictions that match any one of the ground truth answers exactly.

Our adapted model achieved an EM score of 5.88%, while the baseline Llama 3.2-1B model achieved 0.00%, failing to produce any exact matches across 238 question-answer pairs.

Key observations from the predictions:

1. **Numerical Reasoning:** In numerical and date-based questions, our model frequently approximated the correct year (e.g., predicting 1754 vs. ground truth 1760), while Llama produced off-topic narratives.

2. **Named Entity Recognition and Fact Recall:** Both models faced challenges with culturally specific terms, but our model demonstrated marginal improvements in recognizing contextually appropriate answers. The base model often defaulted to generic or unrelated responses, indicating poor comprehension of Yakut-specific queries.

6.2.2 Synthetic Dataset Evaluation

The performance of our model was also evaluated against the Llama 3.2-1B baseline model using a synthetic dataset of 100 question-answer pairs. The evaluation metric used was once again Exact Match (EM).

Our adapted model achieved an EM score of 13.00%, correctly answering 13 out of 100 questions. In contrast, the baseline Llama 3.2-1B model scored 1.00%, with only one correct answer.

Key observations from the predictions:

1. **Output Coherence:** The base Llama model frequently generated nonsensical outputs, often including partial words, unrelated text, or placeholder-like responses. Although some responses were still incorrect, our model produced more coherent and contextually relevant answers.

2. **Grammatical Coherence:** In the base model, Yakut's agglutinative structure was not preserved, leading to illogical affixation and grammatical word formations. Our model showed better alignment with Yakut linguistic structures.

3. **Factual Accuracy:** While the dataset primarily tested general knowledge, our model occasionally provided plausible answers, whereas the base model failed to produce meaningful responses in such cases.

Input Text	Llama-3.2-1B - Zero Shot	Our Phase 1 Fine-tuned Model - Zero Shot
Аан дойдутааҕы	Аан дойдутааҕы биир оҥоһулларынан Саха сиригэр дьаралыы көрэҕээх	Аан дойдутааҕы биир кэлим научнай - техническэй стандар тын (Г Ч П) оҥоруу уонна сайыннарыы
Мин аатым	Мин аатымызда күнөөлэр жўмҮлҪёлд¥рҒҔҲЎлГһ	Мин аа тым күн сиригэр . * — Саха Өрөспүүбүлүкэтин Ил Түмэ нин Бэрэссэдэ этэлэ Александр Жирков
Өскөтүн Эн	Өскөтүн Энэрхээс Ööрўмдәр In 1998, a team of archaeologists from the Institute of Archaeology and Eth	Өскөтүн Эн циклопе дия ны 19 20 - с сыллар бүтү үлэр иттэн саҕ алы ахха наада

Figure 3: Comparison of Text Generation: Base Llama-3.2-1B vs. Phase 1 Trained Model (with Special Tokens)

6.2.3 Evaluating Llama 3.2 and Our Early-Stage Model

Moreover, a comparative evaluation of Llama 3.2 1B model with our model after only being integrated with the custom tokenizer and the first phase of training, on Yakut text generation revealed stern differences in output quality, linguistic consistency, and adherence to the input language. Considering Figure 3, for the Yakut input text, we have the following observations:

1. **Language Switching:** Llama 3.2 generated a few coherent Yakut tokens, but at some point defaulted to English output, disregarding the input language context. On the other hand, our model continuously generated output exclusively in Yakut.

2. **Semantic Fragmentation:** Llama 3.2 generated responses that frequently had disparate text passages and diverged from the content of the input. However, responses from our model extended the conversation without meaning drift while remaining logically aligned with the input prompt.

3. **Tokenization Artifacts:** Improper subword segmentation from Llama 3.2 was evident from the appearance of special characters in the output. However, the absence of special characters and segmentation errors confirmed the efficacy of the custom tokenizer.

Therefore, our model from the initial stages itself showed improved coherence and retention of the Yakut language. The observed performance gap highlights the advantages of our language-specific adaptations, such as including tokenizer optimization and targeted fine-tuning. The significance of customized approaches for low-resource languages like Yakut is highlighted by our model's superior performance over the base Llama model, despite its modest accuracy. Further refinements in training data and instruction tuning are expected to enhance accuracy and contextual understanding. Our findings suggest that tailored linguistic adaptations can yield meaningful improvements even within resource-constrained environments.

7 Limitations and Future work

One of the study limitations is the use of the Google translator, which shows noticeably low accuracy when handling Yakut. As a low-resource language, Yakut lacks the large-scale data needed to support high-quality machine translation, leading to issues such as mistranslations and loss of nuance. Although this is a known and somewhat unavoidable challenge inherent in working with low-resource languages, it nonetheless affects the reliability of the translated content. Future work should consider exploring more specialized translation tools or leveraging community-driven resources to improve translation quality in this context.

8 Acknowledgements

We would like to thank Artem Vasilev (saryalovich@kaist.ac.kr), who is a native speaker of Yakut, for his valuable assistance with translation verification and synthetic dataset generation.

References

Catherine Arnett and Benjamin K. Bergen. 2024. Why do language models perform worse for morphologically complex languages?

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. 2020. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Alexis Conneau, Kartik 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.

Mathias Creutz and Krista Lagus. 2005. Unsupervised morpheme segmentation and morphology induction from text corpora using morfessor 1.0. Technical report a81, Helsinki University of Technology, Espoo, Finland.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Google. a. Cloud translation api - google cloud. Accessed: 2025-06-15.

Google AI Google. b. Gemini-2.0-flash. Accessed: 2025-06-15.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Abdullatif Köksal, Marion Thaler, Ayyoob Imani, Ahmet Üstün, Anna Korhonen, and Hinrich Schütze. 2024. Muri: High-quality instruction tuning datasets for low-resource languages via reverse instructions.

Pedro Javier Ortiz Suárez, Laurent Romary, and Benoît Sagot. 2020. A monolingual approach to contextualized word embeddings for mid-resource languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1703–1714, Online. Association for Computational Linguistics.

Pedro Javier Ortiz Suàrez, Benoit Sagot, and Laurent Romary. 2019. Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures. Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019. Cardiff, 22nd July 2019, pages 9 – 16, Mannheim.

Partha Pakray, Alexander Gelbukh, and Sivaji Bandyopadhyay. 2025. Natural language processing applications for low-resource languages. *Natural Language Processing*, 31(2):183–197.

Iñigo Parra. 2024. Morphological typology in bpe subword productivity and language modeling.

Priyaranjan Pattnayak, Hitesh Laxmichand Patel, and Amit Agarwal. 2025. Tokenization matters: Improving zero-shot ner for indic languages.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Dima Pukhov, Andrey Bugaev, and domotova. Github - nlp-sakha/sakha-embeddings. `https://github.com/nlp-sakha/sakha-embeddings?tab=readme-ov-file`. [Accessed 23-Jun-2025].

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

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.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC 12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.

Cagri Toraman, Eyup Halit Yilmaz, Furkan Sahinu, and Oguzhan Ozcelik. 2023. Impact of tokenization on language models: An analysis for turkish. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 22(4).

Wikipedia. 2025. Yakut language — Wikipedia, the free encyclopedia. [Online; accessed 15-June-2025].

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2020. mt5: A massively multilingual pre-trained text-to-text transformer.

A Training Configuration

Training Configuration Parameters

Epochs: 10
Batch Size: 4 per device
Gradient Accumulation: 8 steps
Learning Rate: 3e-5
Warmup Steps: 200
Weight Decay: 0.01
Precision: FP16 (for memory and speed optimization)
Gradient Checkpointing: Enabled
Evaluation: Disabled during training for speed
Logging: Every 50 steps
Model Checkpoint Saving: Every 500 steps
Data Loader Workers: 4 threads