Final Report: Fine-tuning Large Language Models with Romanized Text

Group 35

Anusha Pant Nav Sanya Anand Paul Kurian Sebastian Escalante Shreyas Malewar

University of Southern California

Abstract

Our primary objective has been to extend upon the insights presented in the research paper titled "ROMANSETU: Efficiently Unlocking Multilingual Capabilities of Large Language Models via Romanization" authored by Husain et al [7]. In this project, we aim to take forward the ideas presented in the paper. We tested the efficacy of fine-tuning LLMs with romanized forms of non-Latin languages like Hindi and Korean. The LLM tasks we examined were translation to English and Sentiment Analysis. We observed that romanized fine-tuning had a positive impact on the performance of the LLMs and thus serves as an effective means to combat the lack of pretraining of LLMs on non-Latin, low-resource languages.

Introduction

Large Language Models (LLMs) have significantly transformed Natural Language Processing (NLP) tasks, showcasing impressive capabilities in numerous English applications. However, significant challenges arise when it comes to extending these capabilities to non-English languages, especially those with non-Latin scripts. The inherent complexity of non-Latin scripts poses hurdles in effectively using LLMs for these languages. Our project aims to address this gap by evaluating the applicability and effectiveness of fine-tuning LLMs with romanized text. Romanization refers to the transliteration of non-Latin text to the Roman script. By doing so, we seek to facilitate the seamless integration of these languages into LLMs, thereby enhancing their performance and applicability for diverse languages. In our experimentation, we focused on Hindi and Korean, as they are both non-Latin languages.

Related Work

Of the numerous papers that we read in the space of romanization and LLMs, here are a few which introduce topics directly related to our own experiments.

1. ROMANSETU: Efficiently Unlocking Multilingual Capabilities of Large Language Models via Romanization

In this paper by Husain et al [7], the authors demonstrated the benefits of fine-tuning LLMs with romanized Hindi. Focusing on Hindi, the study demonstrates that romanized text improves inference efficiency due to its lower fertility. Hindi text had a fertility score of 7.36 whereas romanized Hindi had 2.98. Fertility scores will be discussed in the next section. Interestingly, even with limited pretraining on romanized data, the LLM achieved competitive results in tasks like translation and sentiment analysis. The paper also explores combining romanized and native script prompts when fine-tuning, suggesting potential for further improvement. The paper experimented only with Hindi, leaving room for further exploration of other languages and tasks.

2. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models

Fertility scores measure the average number of subwords per tokenized word. Higher scores indicate aggressive word splitting. Lower scores suggest that the tokenizer aligns better with the language's natural word structure, increasing the model's performance on various tasks [16].

3. On Romanization for Model Transfer Between Scripts in Neural Machine Translation

Amrhein and Sennrich discuss that while romanization can involve information loss, it can also facilitate transfer learning by allowing low-resource languages to share a vocabulary with a high-resource parent model[2].

Problem Description

Large Language Models have reduced effectiveness in tasks for non-English languages. Most English-heavy LLMs like Llama [17] and Mistral [9] may be able to perform adequately on other languages that use the same Latin script. However, for non-Latin scripts, we see issues with a lack of shared vocabulary with the pretraining data and a higher inference latency . As demonstrated in ROMANSETU [7], our project aims to improve the performance of LLMs on non-Latin languages for the following:

1. Translation to English
2. Sentiment analysis

Methodology

1. Datasets

1.1. Language Selection

For our project, we selected Hindi and Korean as the languages to experiment on. Hindi's Devanagari script poses challenges due to its complex character structure and ambiguous vowel representation [11]. Additionally, selecting Hindi would allow us to verify the results obtained in the ROMANSETU project.

We selected Korean as the other non-Latin language to experiment on as it is a low-resource language with a very different script. Korean's Hangul alphabet simplifies romanization with its phonemic nature, although issues remain with representing specific Korean sounds [13]. This highlights the need for tailored NLP techniques depending on the writing system's characteristics.

1.2. Datasets used

For Hindi Translation, we use an English-Hindi Dataset[19] containing 19239 sentences. For Hindi sentiment analysis, we use a Hindi Language sentiment dataset[8] containing 9077 reviews with negative, positive, and neutral ratings.
For Korean Translation, we use a Korean-English Parallel Corpus [15] containing 4564 sentences. For Korean sentiment analysis, we use KR3: Korean Restaurant Reviews with Ratings [12] containing 1236 reviews with negative, positive, and neutral ratings.

2. Romanization Scheme

2.1. Hindi

IndiXlit [1] is a toolkit designed for transliterating Indic scripts, including Hindi, into other writing systems. It iterates through each Hindi sentence, using the `XlitEngine` to generate its romanized version

2.2. Korean

Revised Romanization of Korean [10] is a system for converting Hangul (the Korean alphabet) into the Roman script. This romanization system is officially adopted by the Republic of Korea (South Korea) to romanize Korean names. This system aims to provide a more consistent and phonemically accurate representation of Korean sounds compared to earlier romanization methods. The existence of such a romanization scheme served as an additional benefit for selecting Korean for our project.

3. Model Details

We considered a number of LLMs for our experiments. We worked with Microsoft's Phi-2, [5], Llama 2 and TinyLlama. We eventually settled on using TinyLlama (via Hugging Face) for the final implementation of our project.

3.1. LLama2 and TinyLlama

Llama 2 is an auto-regressive language model based on an optimized transformed architecture. It was pretrained on 2 trillion tokens of public data. [18]. It is worth noting that the base Llama 2 7B is listed to have 0.06% Korean in its pretraining data, while the percentage for Hindi is not mentioned. Due to resource constraints, it was not feasible to fine-tune or perform inference with the Llama 2 model for our purposes. Thus, we decided to use a smaller model TinyLlama [20], which has the same architecture and tokenizer but only has 1.1 billion parameters and is pretrained on 1 trillion tokens.

3.2. Low Rank Adapter (LoRA)

Due to the size and cost of training the LLMs, Low-Rank Adapters have been proposed as an alternative to full parameter fine-tuning [6]. This approach reduces the trainable parameters by 10000 and the GPU memory requirements by a factor of 3, thereby making our fine-tuning experiments feasible.

4. Evaluation Metrics

4.1. BLEU

BLEU (Bilingual Evaluation Understudy) is a metric to evaluate machine translation quality. It assesses the similarity between a machine translation and human reference translations by counting

matching n-grams (sequences of n words). While simple and widely used, BLEU has been criticized for not reflecting semantic equivalence and favoring brevity [14].

4.2. BERTScore

BERTScore uses BERT's contextual embeddings [3] to compare candidate and reference sentences, aligning with human assessments at both sentence and system levels. Moreover, it calculates precision, recall, and F1 score, offering insights for evaluating language generation tasks [4].

5. Pipeline

Our final pipeline involves fine-tuning Tiny-Llama using Low-Rank Adaptors by prompting it multiple times with different samples and their corresponding outputs (translations or sentiment). At test time, we prompt the model without the desired output to test how well the fine-tuned model can complete the sequence. For a given task and a given language, we fine-tuned three different copies of the TinyLlama model, each with a different input used for fine-tuning: Native, Romanized, and Combined. The model fine-tuned with the Native script is the baseline since it measures how it would perform without romanized fine-tuning.

Experimental Results

1. Translation

1.1. Experimental Setup

For the translation task, we used one-shot prompting in the following format:
Translate the following sentences from X to English. The output should be in English and no other language.
X: ***Text***
English: ***Translation***

Detailed examples of these prompts can be found in our appendix. We ran our experiments over *50 steps*, with a learning rate of *2e-4*, and used two evaluation metrics, *BLEU and BERTscores*. We split our dataset into training, validation, and test sets with an 80-10-10 split.
To evaluate the model, we computed the BLEU score for each English sentence that the model gave as output on the test set. We gave equal importance to unigrams, bigrams, and trigrams, with a weight of 0.33 for each. These scores are then averaged over the test samples.

1.2. Results and Discussion

	Native	Romanized	Combined
Hindi	0.185	**0.224**	0.18
Korean	0.275	**0.453**	0.254

Table 1: Translation Results - BLEU scores

Romanized Hindi outperforms Hindi by roughly 0.04. Similarly, romanized Korean performs much better than native Korean, with an improvement of 0.18. The combination of the two scripts does not fare as well in comparison.
We also computed BERTscores for the models.

	Native	Romanized	Combined
Hindi	0.8852	0.869	**0.8853**
Korean	**0.907**	0.870	0.902

Table 2: Translation Results - BERTscores (F1 score)

The BERTscores are fairly even for both Hindi and Korean, with the Native and Combined fine-tuned models performing the best.

Inferences
Looking at the BLEU scores, we believe that the romanized versions of these two languages perform better than their corresponding native scripts (the baseline) because the fertility scores of Hindi and Korean are higher than those of English. A higher fertility score makes the tokenization more complicated, bringing the model's performance down when it has been trained for the same number of steps on a given dataset [21]. This also explains the more drastic jump in the score from native to romanized for Korean because Korean has a higher fertility score, making the tokenization even more complicated[21]. The combined input of the native and romanized scripts may require a more complex model or more training to be able to get similar results due to its fundamentally more complex nature.

Looking at the BERT F1 scores, we see that the baseline models perform either equal to or marginally better than the models fine-tuned with romanized input. BERTscores consider the similarity between the contextualized embeddings of the MT output and their references, thereby capturing the semantical correctness of the translation output. These scores show us that the semantical correctness of the three different strategies is comparable. We believe that there is merit in the

fact that the BERTscores obtained are comparable with the baseline, especially considering the improvement in inference efficiency which we get with romanized fine-tuning. As explained in the ROMANSETU paper [7], this improvement in efficiency arises due to the lower fertility of romanized text, which means that it gets tokenized faster.

2. Sentiment Analysis

2.1. Experimental Setup

Similar to the pipeline used for the translation experiments, we took three copies of the base TinyLlama model (per language) and performed one-shot prompting to fine-tune the models:
Classify the sentiment of the below X (Hindi or Korean) **sentence into Positive, Neutral, or Negative. The output should be either 'Positive' or 'Neutral' or 'Negative'.**
X: ***Text***
Output: ***Sentiment***

2.2. Results and Discussion

Here are the accuracies of the model outputs on the test set after fine-tuning it using the training and validation sets:

	Native	Romanized	Combined
Hindi	37%	**46.65%**	36.89%
Korean	80.64%	**81.73%**	81.45%

Table 3: Sentiment analysis Results

For sentiment analysis in Hindi, we achieve the highest accuracy of 45.65% when fine-tuning with Romanized Hindi.

In Korean, we noticed only a marginal improvement in accuracy when fine-tuning with Romanized Korean, unlike Hindi. We believe this is because of the smaller size of the dataset we used for this experiment. The gap in performance between the Hindi and Korean models may be attributed to the fact that Llama was pretrained on more Korean text than Hindi [18]. This disparity in pretraining data is presumably present in TinyLlama by extension since it was trained on SlimPajama which is a corpus based on Llama's pretraining data [20].

Conclusion

1. Challenges faced

Our primary challenge with this project was the limitation in resources required to fine-tune 12 copies of TinyLlama, since we had three different fine-tuning strategies, two different languages, and two different tasks. Using LoRA certainly made the fine-tuning of the models on the non-Latin language's text possible. Despite that, none of our team members had the necessary computing or memory resources available on our local machines to fine-tune an LLM adequately.

We decided to use Google Colab's T4 GPU resources to proceed with our project. We noticed that using Colab allowed us to fine-tune each TinyLlama model for an adequate number of steps (50) without running out of memory. We were able to load up to 7000 rows of data from our datasets without issues.

2. Final takeaways and Future Work

Even with limited computational resources, we have established that romanization does have a positive impact on the performance of LLMs for non-Latin languages. In the case of the translation task, we see that romanized fine-tuning helps improve BLEU scores. Additionally, the reduced fertility of the romanized text makes the model perform faster at inference time. In the case of sentiment analysis, the model with romanized fine-tuning performed the best.

Here are the ways we believe that our work can be taken forward:

- Extending this practice of romanized fine-tuning to other non-Latin, low-resource languages certainly seems promising in light of the improvement we've seen with Hindi and Korean, which are significantly different scripts.
- We believe the performance of the combined input of a native script with its romanized version should yield better results with an increase in training time and a larger LLM.
- We believe that fine-tuning with romanized input gives the LLM a better grasp of non-Latin languages due to the shared vocabulary introduced between the languages. As we studied in our course, the encoding of a shared vocabulary through methods like Byte-pair encoding improves the performance of Machine translation. Further work can be done to study whether this improvement applies to the performance of zero-shot transfer translation, for instance, Hindi to Korean.

Division of Work

- Anusha Pant: Research on fine-tuning LLMs like Llama, Phi-2, implementation of fine-tuning experiments with Tiny Llama for translation (Hindi and Korean). Experimented with BLEU scores and different hyper-parameters to boost results.
- Nav Sanya Anand: Data Research and Curation, Romanization pipeline, Result Analysis and Research
- Paul Kurian: Research on fine-tuning LLMs like Llama, Phi-2. Design of the fine-tuning pipeline of TinyLlama. Experiments with translation evaluation metrics like BERTscores.
- Sebastian Escalante: Research on common benchmarks to evaluate performance and smaller LLMs for our use with limited compute resources.
- Shreyas Malewar: Literature review, research on Low-Rank Adapters to finetune LLMs, implementing sentiment analysis for Korean, and evaluating results.

References

[1] AI4Bharat. 2022. Indicxlit: Transliteration models for 21 indic languages. https://github.com/AI4Bharat/IndicXlit/tree/master. Accessed: 2024-04-22.

[2] Chantal Amrhein and Rico Sennrich. 2020. On romanization for model transfer between scripts in neural machine translation. *arXiv preprint arXiv:2009.14824*.

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

[4] Hugging Face. Bertscore. https://huggingface.co/spaces/evaluate-metric/bertscore. Retrieved April 20, 2024.

[5] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. 2023. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*.

[6] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

[7] Jaavid Aktar Husain, Raj Dabre, Aswanth Kumar, Jay Gala, Thanmay Jayakumar, Ratish Puduppully, and Anoop Kunchukuttan. 2024. Romansetu: Efficiently unlocking multilingual capabilities of large language models models via romanization. *Preprint*, arXiv:2401.14280.

[8] Mahesh M J. 2023. Hindi language sentiment dataset. https://www.kaggle.com/datasets/maheshmj007/hindi-language-sentiment-dataset. Retrieved April 20, 2024.

[9] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

[10] V. Mair. 2021. Language log: Korean romanization. https://languagelog.ldc.upenn.edu/nll/?p=51316. Accessed: 2024-04-22.

[11] Musa. 2006. A simple script for devanagari. *Language in India*, 6.

[12] ninetyninenewton. 2023. Kr3 - korean restaurant reviews with ratings. https://www.kaggle.com/datasets/ninetyninenewton/kr3-korean-restaurant-reviews-with-ratings. Retrieved April 20, 2024.

[13] H. K. Pae. n.d. *Analyzing the Korean Alphabet*. Springer. Retrieved April 20, 2024.

[14] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. n.d. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. Retrieved April 20, 2024.

[15] rareloto. 2023. Naver dictionary conversation of the day. https://www.kaggle.com/datasets/rareloto/naver-dictionary-conversation-of-the-day. Retrieved April 20, 2024.

[16] Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

[17] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[18] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

[19] Preet Viradiya. 2023. English hindi dataset. https://www.kaggle.com/datasets/preetviradiya/english-hindi-dataset. Retrieved April 20, 2024.

[20] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *Preprint*, arXiv:2401.02385.

[21] Judit Ács. 2020. Exploring bert's vocabulary. http://juditacs.github.io/2019/02/19/bert-tokenization-stats.html. Accessed: 2024-04-22.

Appendix

1. Examples of one-shot prompting used for translation

Note the examples are provided for Korean but the same can be applied to Hindi. Note that at the time of inference we perform prefix prompting where we remove the test set's reference output (English translation or sentiment depending on the task) and allow the LLM to predict the output given this prefix.

1.1. Fine-tuning with native Korean text (Baseline model)

Translate the following sentences from Korean to English. The output should be in English and no other language.

`Korean:`지구상 바다의 대부분은

English: *Most of the planet is ocean water.*

1.2. Fine-tuning with romanized Korean text

Translate the following sentences from Korean to English. The output should be in English and no other language.

`Korean:jigusang badaui daebubuneun`

English: *Most of the planet is ocean water.*

1.3. Fine-tuning with native and romanized Korean text

Translate the following sentences from Korean to English. The output should be in English and no other language.

`Korean:` 지구상 바다의 대부분은

`jigusang badaui daebubuneun`

English: *Most of the planet is ocean water.*

2. Examples of one-shot prompting used for sentiment analysis

2.1. Fine-tuning with native Korean text (Baseline model)

Classify the sentiment of the below X sentence into Positive, Neutral, or Negative. The output should be either 'Positive' or 'Neutral' or 'Negative'.

`Korean:`또 가고 싶다 맛있어

Output: *Positive*

2.2. Fine-tuning with romanized Korean text

Classify the sentiment of the below X sentence into Positive, Neutral, or Negative. The output should be either 'Positive' or 'Neutral' or 'Negative'.

`Korean:tto gago sipda masisseo`

Output: *Positive*

2.3. Fine-tuning with native and romanized Korean text

Classify the sentiment of the below X sentence into Positive, Neutral, or Negative. The output should be either 'Positive' or 'Neutral' or 'Negative'.

`Korean:`또 가고 싶다 맛있어

`tto gago sipda masisseo`

Output: *Positive*