Machine Transliteration : A deep learning model

Maneesha Reddy mmaddula@umass.edu
Disha Chiplonker dchiplonker@umass.edu
Shruthi Mysore Narayanaswamy smysorenaray@umass.edu
Sahana Patte Keshava spattekeshava@umass.edu

1 Problem statement

Transliteration is a process that accurately and unambiguously maps the source content to the target content while preserving phonetic aspects and pronunciation. Transliteration is the process by which named entities or proper nouns are converted from one orthographic system to another. Many machine transliteration algorithms have been developed in recent years based on the phonetics of the source and target languages, as well as statistical and language-specific methods. Given the prevalence of various languages and the growing number of multilingual speakers, automated, machine learning-based transliteration systems are in high demand. Transliteration is used as part of many multilingual applications such as corpus alignment and also as component of machine translation system. With the advent of deep learning techniques, few research attempts have been made using deep learning. The deep learning frame-works used are similar to that of the Sequence to Sequence machine translation. Since there is a lot of research related to machine translation but such research limits to machine transliteration by a lot. Our goal with this project was to implement machine transliteration with Aksharantar dataset. We came across this dataset and IndicXlit which is a multilingual model for transliteration from romanized input to native language script. Our goal was to implement a model which could work with the Aksharantar dataset or multiple language pairs, and also implement a system of models based on RNN, ConvS2S, Transformer models.

2 What you proposed vs. what you accomplished

Our transliteration system was proposed to adapt different neural machine translation (NMT) frameworks: recurrent neural network and convolutional sequence to sequence based NMT, Transformer based encoder and decoder models. the proposed architecture for transliteration was using RNN and ConvS2S NMT, Transformer framework like IndicXlit and ensemble technique using character and byte-level representations for generating accurate transliterations. Additionally, we proposed implementing a frequency based ensemble technique using the results of these systems.The ensemble technique involves taking the weighted average of the outputs of the models. The weights are determined based on the performance of the individual models on the validation data. During the entire course, we were successfully able to accomplish implementing the ConvS2S NMT and Transformer models and using ensemble technique to combine the output from both the model.

- ~~Collect and preprocess dataset~~
- Building Individual Convolutional and Transformer Models on dataset and examine its performance : Could not make the Recurrent NMT model work
- ~~Using the prepared data, train and verify each individual model~~
- ~~Ensemble Techniques Implementation : Use ensemble techniques to combine the results of the separate models' predictions~~
- ~~Utilize suitable metrics to assess the performance of the ensemble models. Adjust the hyperparameters to fine-tune the group models.~~
- ~~Perform in-depth error analysis to figure out what kinds of examples our approach struggles with~~

3 Related work

Recent research in transliteration has explored various techniques to improve accuracy and handle multilingual scenarios. One common approach is the use of deep learning models, particularly sequence-to-sequence architectures, inspired by neural machine translation (NMT) frameworks. Few models (Soumyadeep Kundu1, 2018; Mohd Zeeshan Ansari, 2022) propose a convolutional neural network (CNN)-based sequence-to-sequence model that adapts the NMT framework, incorporating variations like multi-hop attention mechanisms.In another work, (Madhani et al., 2022), researchers compare attention-based sequence-to-sequence models with epsilon insertion models and CTC alignment. The study reveals that attention-based models generally perform better on transliteration benchmarks.To address transliteration challenges across languages, multilingual models have been explored. "Aksharantar: Towards building open transliteration tools for the next billion users" (Soumyadeep Kundu1, 2022) introduces IndicXlit, a transformer-based multilingual transliteration model for converting Roman script to Indic scripts. The study demonstrates the advantages of multilingual models, especially for low-resource languages, while simplifying deployment by using a single model for multiple languages.Efforts have also been made to incorporate phonetic information into transliteration models. In another study (Yuan H, 2020), researchers introduce a phonetic auxiliary task that enhances transliteration accuracy. The model, based on a sequence-to-multiple-sequence architecture, shares sound information through a shared encoder and dual decoders.Addressing the source language origins, another work (Masato Hagiwara, 2011) presents a latent class transliteration method that models language origins as latent classes. This approach improves accuracy compared to conventional models without latent classes, particularly when dealing with words from multiple language origins.In summary, recent research in transliteration has explored deep learning models, multilingual approaches, attention mechanisms, and the incorporation of phonetic information. The papers (Soumyadeep Kundu1, 2018; Madhani et al., 2022; Soumyadeep Kundu1, 2022; Masato Hagiwara, 2011)"A Deep Learning Based Approach to Transliteration" (2018), "Sequence-to-sequence neural network models for transliteration" (2016), "Aksharantar: Towards building open transliteration tools for the next billion users" (2022), and the 2011 paper on latent class transliteration highlight these advancements. These techniques aim to improve accuracy, handle multilingual scenarios, and capture contextual and phonetic aspects of transliteration, contributing to the development of effective transliteration tools for communication across different scripts and languages.Ensemble techniques involve combining multiple models or predictions to obtain a final result. This can be achieved through various strategies such as model averaging, voting, stacking, or boosting. Ensemble methods are effective because they leverage the diversity and complementary strengths of individual models, thereby reducing errors and increasing overall accuracy. According to (Roman Grundkiewicz, 2018), An ensemble of independent models usually outperforms an ensemble of different model checkpoints from a single training run as it results in more diverse models in the ensemble and in (Soumyadeep Kundu1, 2018), we implement an ensemble technique based on the frequency of occurrence of the output words. Corresponding to each input word, we calculate the most occurring output word from all the generated results, which inspired us to include an ensemble technique to combine results from all the three models in our implementation.

4 Dataset

Aksharantar is the largest publicly available transliteration dataset for 19 Indic languages. The corpus has 26M Indic language-English transliteration pairs. It is the largest available transliteration corpora in 20 Indic languages from existing parallel translation corpora (Ramesh et al., 2022), monolingual corpora (Kakwani et al., 2020) and manual annotations from human annotators. It is an Indic language transliteration set containing 103k words pairs spanning 19 languages that enables fine-grained analysis of transliteration models. The task at hand involves building models for transliteration of romanized inputs into native scripts for Indic languages. Additionally, manually transliterated data was compiled to complement the mined datasets and create a comprehensive transliteration corpora. The dataset statistics are as follows:-

- Size: 26 million word pairs

- Languages: 21 languages (Indic languages)
- Source: Mined from Wikidata, Samanantar parallel translation corpora and IndicCorp monolingual corpora
- Additional data: Manually transliterated data
- Evaluation benchmark dataset: Aksharantar testset
  - Size: 103,000 word pairs
  - Languages: 19 languages
  - Characteristics: Contains native language words with diverse n-gram characteristics and named entities of Indic and foreign sources spanning different entity categories

Few of the properties of the data that make the task challenging are the diversity and scale of the dataset. The dataset covers a wide range of Indic languages and scripts, including the Indo-Aryan, Dravidian, Austro-Asiatic, and Tibeto-Burman language families. These languages are written in different scripts, such as Brahmi family abugida scripts, Arabic-derived abjad scripts, and Alphabetic Roman script. The dataset's scale and diversity pose challenges in developing models that can handle the various languages and scripts effectively. Some scripts are used by multiple languages. For example, the Devanagari script is used to write Hindi, Marathi, Konkani, Maithili, and Sanskrit, while the Bengali script is used for Bengali, Assamese, and Santali.

Handling multiple scripts for the same language adds complexity to the transliteration task. Many users in the Indian subcontinent are comfortable with the Roman keyboard layout, and learning multiple keyboard layouts for different scripts would be cumbersome. This highlights the need for an optimal solution that automatically transliterates romanized input into the native script.

The prominent task for our project is, for each language input/output pair, producing the transliterated output in the target language. For example, a transliteration pair (te, en), would have an input word in language te, and output word in language en. In this way, we have tried to implement our models on the kannada-english and telugu-english subset datasets of the Akshantara dataset. Few of the input-output pairs are as shown in the figures 1 4 2 3

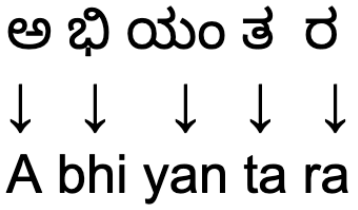

Figure 1: Kannada Transliteration Example

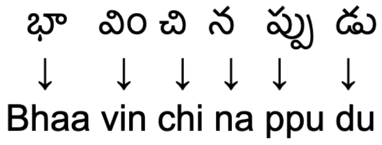

Figure 2: Telugu Transliteration Example

4.1 Data preprocessing

The following are the data preprocessing steps which we adapted for our model implementations.

- Tokenization: The input and target sequences are split into individual tokens. Each token represents a meaningful unit, such as a word or a subword. Tokenization helps in standardizing the input and target representations for further processing.
- Vocabulary Creation: A vocabulary is built by collecting all unique tokens from the input and target sequences. Each token is assigned a unique index in the vocabulary, which is later used for encoding and decoding sequences.
- Padding: To ensure that all input and target sequences have the same length, padding is applied. Padding involves adding special padding tokens (e.g., ¡PAD¿) to the sequences so that they match the length of the longest sequence in the dataset.
- Special Tokens: Special tokens such as <BOS>(beginning of sequence),

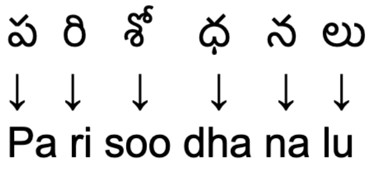

Figure 3: Telugu Transliteration Example

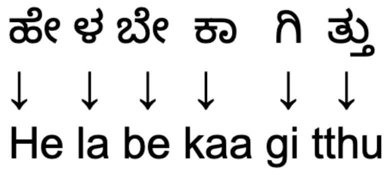

Figure 4: Kannada Transliteration Example

<EOS>(end of sequence), and <UNK>(unknown token) may be added to the input and target sequences. These tokens help the model learn the start and end points of the sequences and handle out-of-vocabulary (OOV) words.

- Numericalization: Each token in the input and target sequences is replaced with its corresponding index from the vocabulary. This step converts the sequences from a string/textual format to a numerical representation that can be processed by the model.

- Positional Encoding: Positional encoding is applied to the input sequences to provide positional information to the model. This allows the transformer to capture the order and relative positions of the tokens in the sequences.

- Batching: The processed sequences are organized into batches for efficient training and inference. Batching involves grouping sequences of the same length together to minimize the amount of padding required.

- Masking: Masking is applied to the target sequences during training to prevent the model from attending to future tokens that it should not have access to. This is typically done using a masking token (e.g., <MASK>) or by setting the corresponding positions to zero in the attention masks.

  These data processing steps ensure that the input and target sequences are properly formatted, encoded, and preprocessed to be fed into the seq2seq transformer model. By following these steps, the model can effectively learn the mapping between the input and target sequences and generate accurate predictions or translations.

5 Baselines

5.1 Transformers

From the study and paper (Raj and Laddagiri, 2022), previously, Recurrent Neural Networks (especially LSTMs) dominated the field of Natural Language Processing. This was because text data follows a sequential structure, where the context for predicting the next words or characters depends on the preceding ones. LSTMs were popular because they could process words sequentially and retain the contextual information for making accurate predictions. Unlike traditional Deep Neural Nets, LSTMs could handle inputs of varying lengths. However, when dealing with longer sequences, the performance of LSTM models suffered. To address this issue, the attention mechanism was introduced, which helped focus on relevant parts of sentences and improved the performance of neural machine translation systems. Nonetheless, recurrent architectures still faced fundamental problems such as vanishing or exploding gradients, slow training due to sequential input feeding, and difficulty in remembering long text sequences.

To overcome these challenges, Transformers were introduced in 2017. Unlike previous models, Transformers relied solely on attention without any recurrence or biases towards the sequential nature of language. Transformers have now become the standard approach for natural language processing. Typically, Transformers are pre-trained on large text corpora and then fine-tuned for specific downstream tasks. Due to their architecture's generality and lack of biases, Transformers have also been successfully applied in other domains such as computer vision, reinforcement learning, and speech recognition. Undoubtedly, Transformers have had a profound impact on the entire field of machine learning.

Originally, Transformers were designed with an encoder-decoder structure. In this structure, the input sequence is encoded into a feature vector, which is then used by the decoder to generate the target sequence in an auto-regressive manner. The encoder stack consists of two main sub-layers: multi-headed self-attention and fully-connected layers, both incorporating residual connections. The input sequence undergoes embedding and positional embedding before being fed into the encoder. The decoder stack adds a third sub-layer, performing multi-head attention on the encoder's outputs. Additionally, the first self-attention layer in the decoder is modified to prevent tokens from attending to future tokens in the output sequence during training. This is achieved by applying a triangular attention mask, ensuring that a token at position "i" can only depend on positions preceding "i."

5.2 ConvS2S Model

From (Soumyadeep Kundu1, 2018), a convolutional neural network (CNN)-based model is employed for sequence-to-sequence neural machine translation (NMT) with a multi-hop attention mechanism connecting the encoder and decoder. Our CNN architecture computes the encoder state (represented as "z") and the decoder state (represented as "h"). Input units and their respective absolute positions are embedded as a combined input element representation (referred to as "f"). Similarly, we utilize a similar CNN architecture to construct the output element representation (referred to as "e") for the decoder network. Our approach incorporates a multi-step attention mechanism, enabling the network to repeatedly refer back to the combined input representation (f) in order to generate the output representation (e).

To create a vector representation of the input units (f), the encoder employs a CNN that performs computations on all input units simultaneously. Simultaneously, the CNN decoder produces output units (e) one at a time in each step, utilizing the multi-step attention mechanism. Here's how the multi-step attention layer operates:

The first layer identifies relevant context information from the combined input representation (f) and passes it to the second layer. The second layer utilizes this information during attention weight computation and then passes it on to the subsequent layer, continuing this process iteratively. Additionally, the decoder has immediate access to the attention history from previous time steps, enabling it to incorporate the knowledge of past attention weights.

6 Your approach

The Universal Transformer is a development of the Transformer model for sequence-to-sequence tasks. Recurrence and adaptive computation are added to the model to help it better represent long-term dependencies in sequences.The Universal

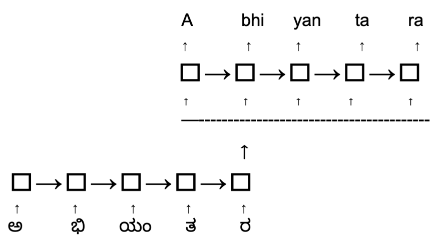

Figure 5: Encoder Decoder Seq-2-Seq

Transformer updates each position in the sequence iteratively as opposed to the original Transformer, which processes inputs in parallel. It repeatedly pays attention to each position's previous states, enabling a more sophisticated representation that takes both the local and the global context into account.The Adaptive Computation Time (ACT) feature of the Universal Transformer also determines the number of repetition steps for each position on a dynamic basis.The Universal Transformer enhances its understanding of context and captures long-range dependencies by incorporating recurrence and adaptive computation.

Gathered a large dataset of Telugu and Kannada languages, preprocessed the data, and trained two different models: a Seq2Seq Transformer and a CNN Transformer.

Here are the accuracy results we obtained:

Universal Transformer gave the following accuracies:

Dataset	**Accuracy (%)**
Telugu Test	63.5
Telugu-Kannada Test	65
Telugu Dev	72
Telugu-Kannada Dev	75.1278

Table 1: Universal Transformer Accuracy Results

The CNN Seq2Seq model a variant of the sequence-to-sequence architecture that incorporates convolutional neural networks (CNNs) into the encoder-decoder framework is the second model used it utilizes convolutional layers to capture local patterns and extract meaningful features from the input sequence. The encoder component employs CNN layers to encode the input sequence, while the decoder generates the output sequence using recurrent neural networks (RNNs).

CNN Seq2Seq gave the following accuracies:

Hidden Parameters used are as stated below:

Embedding Dimension: 100

Dataset	Accuracy (%)
Telugu Test	29.2571
Telugu Dev	35.2

Table 2: CNN Seq2Seq Accuracy Results

Dataset	Train Size	Valid Size	Test Size
Kannada	2,906,728	7,025	11,380
Telugu	2,429,562	7,681	10,260

Table 3: Dataset Sizes

Number of Heads: 4
Encoder Layers: 6
Decoder Layers: 6
Encoder Hidden Size: 200
Decoder Hidden Size: 200
Learning Rate: 1e-3
Dropout: 0.1
Dimension Feed Forward: 2048

These hyperparameters determine various aspects of the model, such as the size of the embedding vectors, the number of attention heads, the depth of the encoder and decoder layers, the hidden sizes, and the learning rate. They are tuned based on experimentation to achieve optimal performance on the given task and dataset.

Once both the Seq2Seq CNN model and the Transformer model are trained ,we use ensemble technique to give the final output predicted. For each input sequence, we have generated the transliteration predictions using both models and combined the predictions using distance based ensemble technique. The distance-based ensemble technique involves calculating the distance between the predicted word and the target word and selecting the output with the least distance as the final prediction. By considering the distance between the predicted and target words, this ensemble technique aims to combine the strengths of different models to improve the overall prediction accuracy.

7 Error analysis

Overall, both the CNN model and Transformer model made errors in generating the correct outputs for the given inputs. The errors include missing letters, adding extra characters, and replacing letters with incorrect ones. The ensemble model showed similar errors to the CNN model in three out of four cases, indicating that the CNN model's errors may have had a more significant influence on the ensemble's output. The output file containing outputs from all 3 models: CNN, Transformer and Ensemble technique can be found here to open the file.

Target	CNN Output	Transformer Output	Ensemble Output
dhairyaaniki	dhainaaryaardhiki	dhairyaniki	dhairyaniki
votti	votti	wotty	votti
jvaramthoo	jarvaramtho	jwaranto	jarvaramtho
goutamugaani	gautamuni	gauthamuni	gautamuni
aakruthulaloo	aalakrutulalo	akrutulalo	akrutulalo
krushnarjunulu	krishnaarnaajulu	krishnaarjunulu	krishnaarjunulu
ardhamavuthundi	adharmavutundi	ardhamavutundi	ardhamavutundi
maantrika	maantraki	mantrika	mantrika
uparithalaaniki	uparitaalaaniki	uparitalaaniki	uparitalaaniki
pondutaam	pondutam	pondutam	pondutam
siddhaantamu	siddhaamantamu	siddhaantamu	siddhaantamu
positive	pajipativ	pagitive	pagitive
award	adwar	award	award
madonnaa	madonna	madonna	madonna
okanoka	okokana	okaanoka	okaanoka

Table 4: Output Comparison

7.1 CNN Model

- For the word ”dhairyaaniki,” the CNN model outputted ”dhainaaryaardhiki.” It seems like the CNN model added extra characters in the middle of the word.

- For the word ”votti,” the CNN model correctly generated the same word.

- For the word ”jvaramthoo,” the CNN model outputted ”jarvaramtho.” It seems like the CNN model missed the first letter ”j” and replaced it with ”ja.”

- For the word ”goutamugaani,” the CNN model generated ”gautamuni.” It incorrectly replaced the letter ”o” with ”au.”

- For the word ”aakruthulaloo,” the CNN model generated ”aalakrutulalo.” It added an extra ”a” at the beginning and replaced ”kru” with ”krut.”

- For the word ”krushnarjunulu,” the CNN model outputted ”krishnaarnaajulu.” It replaced ”ushna” with ”ishnaarna.”

- For the word ”ardhamavuthundi,” the CNN model outputted ”adharmavutundi.” It replaced ”rdh” with ”dh” at the beginning and added an extra ‘r’ in the sentence.

- For the word ”maantrika,” the CNN model outputted ”maantraki.” It replaced the last letter ”a” with ”i.”

- For the word ”uparithalaaniki,” the CNN model outputted ”uparitaalaaniki.” It replaced ”h” with continuous ‘a’s

7.2 Transformer Model

- For the word ”aakruthulaloo,” the Transformer model outputted ”akrutulalo.” It missed the letter ”a” at the beginning and letter ‘o’ at the end.
- For the word ”krushnarjunulu,” the Transformer model outputted ”krishnaarjunulu,” which is the correct output.
- For the word ”ardhamavuthundi,” the Transformer model outputted ”ardhamavutundi,” which is the correct output.
- For the word ”maantrika,” the Transformer model outputted ”maantrika.” , which is correct output.
- For the word ”uparithalaaniki,” the Transformer model outputted ”uparitalaaniki.” It replaced “tha” with “ta”

7.3 Ensemble Model (Combination of CNN and Transformer):

- For the word ”aakruthulaloo,” the ensemble model outputted ”akrutulalo,” which is the same as the Transformer model’s output.
- For the word ”krushnarjunulu,” the ensemble model outputted ”krishnaarjunulu,” which is the same as the Transformer model’s output.
- For the word ”ardhamavuthundi,” the ensemble model outputted ”ardhamavutundi,” which is the same as the Transformer model’s output.
- For the word ”maantrika,” the ensemble model outputted ”maantrika,” which is the same as the Transformer model’s output.
- For the word ”uparithalaaniki,” the ensemble model outputted ”uparitalaaniki,” which is the same as the CNN model’s output.

Additional cases:

- For the word ”pondutaam,” all models correctly generated ”pondutam.”
- For the word ”siddhaantamu,” all models correctly generated ”siddhaantamu.”
- For the word ”positive,” the CNN model outputted ”pajipativ,” the Transformer model outputted ”pagitive,” and the ensemble model outputted ”pagitive.” All models made errors by replacing ”s” with ”p” and ”t” with ”g.”
- For the word ”award,” the CNN model outputted ”adwar,” the Transformer model outputted ”award,” and the ensemble model outputted ”award.” The CNN model and ensemble model made errors by adding ”d” at the beginning.
- For the word ”madonnaa,” the CNN model outputted ”madonna,” the Transformer model outputted ”madonna,” and the ensemble model outputted ”madonna.”

7.4 Future Improvements

Based on the error analysis, here are some potential improvements that can be made to the models:

CNN Model: The model accuracy has to be improved by building the models in much more efficient way and tune the hyper-parameters much accurately. Explore different architectures or configurations for the CNN model. The current architecture may not be effectively capturing the necessary patterns in the data. Increase the amount of training data for the CNN model to improve its ability to generalize and reduce overfitting. Augment the training data with variations of the words, such as different spellings or phonetic representations, to make the model more robust to variations in input.

Transformer Model: The model accuracy has to be improved by building the models in much more efficient way and tune the hyper-parameters much accurately. Adjust the input representation of the Transformer model. It seems to struggle with consistently capturing the first letter of the words, so modifying the input format or adding positional encoding may help address this issue. Fine-tune the Transformer model with a larger and more diverse dataset to improve its ability to handle different word structures and variations. Experiment with different hyper-parameters, such as the learning rate or the number of layers and attention heads, to optimize the performance of the Transformer model.

Ensemble Model: Consider using different method of ensemble technique by use averaging of weights so that any model which is not performing at par can be assigned lesser weight. If the CNN model consistently performs worse than the Transformer model, reducing its influence on the final output may lead to better results. Explore other ensemble techniques, such as stacking or boosting, to further improve the performance of the combined models. Investigate alternative ways to combine the outputs of the CNN and Transformer models, such as using a weighted average or a learnable gating mechanism, to allow for more flexible and adaptive blending of their predictions. In addition to these model-specific improvements, collecting more diverse and representative training data is crucial for enhancing the models' performance and generalization. It can help expose the models to a wider range of word variations and improve their ability to handle different linguistic patterns.

8 Contributions of group members

List what each member of the group contributed to this project here. For example:

- Maneesha Reddy: built and trained model using ConvS2S and ensemble methods
- Shruthi Narayanaswamy: trained and tested Transformer and literature survey, tried implementing RNN
- Disha Chiplonker: data preprocessing, report writing with error analysis
- Sahana Patte Keshava: trained and tested Transformer and literature survey, tried implementing RNN

9 Conclusion

The Transformer model, with its self-attention mechanism, can capture long-range dependencies and perform well on sequence-to-sequence tasks like transliteration. Convolutional Seq2Seq models, on the other hand, excel at capturing local patterns and can be effective for shorter sequences.Selecting appropriate hyperparameters for the Transformer and Convolutional Seq2Seq models is essential for achieving good results. Parameters like embedding dimension, number of layers, learning rate, dropout, and hidden sizes can significantly impact the model's performance.Determining the optimal model complexity can be challenging. Transformer models with a larger number of layers or parameters may require more computational resources and longer training times. Finding the right trade-off between model complexity and available resources is crucial. Accuracies of the models can be further improved. The implementation is amateur and needs advancements to be made in both the transformer and CNN model. Further work is mainly to reattempt working on RNN model and increasing overall accuracy.

10 AI Disclosure

- Did you use any AI assistance to complete this proposal? If so, please also specify what AI you used.
  - Yes

If you answered yes to the above question, please complete the following as well:

- If you used a large language model to assist you, please paste *all* of the prompts that you used below. Add a separate bullet for each prompt, and specify which part of the proposal is associated with which prompt.
  - Prompts for help with coding doubts for CNN and Transformer.
- **Free response:** For each section or paragraph for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good output, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to generate new text, check your own ideas, or rewrite text?
  - your response here

References

Madhani, Y., Parthan, S., Bedekar, P., Khapra, R., Kunchukuttan, A., Kumar, P., and Khapra, M. S. (2022). Sequence-to-sequence neural network models for transliteration. pages 1–5.

Masato Hagiwara, S. S. (2011). Latent class transliteration based on source language origin. pages 1–5.

Mohd Zeeshan Ansari, Tanvir Ahmad, M. M. S. B. F. A. (2022). Hindi to english transliteration using multilayer gated recurrent units. pages 1–8.

Raj, Y. and Laddagiri, B. (2022). Hindi to english transliteration using multilayer gated recurrent units. pages 1–8.

Roman Grundkiewicz, K. H. (2018). Neural machine translation techniques for named entity transliteration. pages 1–6.

Soumyadeep Kundu1, Sayantan Paul1, S. P. (2018). A deep learning based approach to transliteration. pages 1–5.

Soumyadeep Kundu1, Sayantan Paul1, S. P. (2022). Aksharantar: Towards building open transliteration tools for the next billion users. pages 10–19.

Yuan H, S. B. C. (2020). English-to-chinese transliteration with a phonetic auxiliary task. pages 10–19.