**Khmer-English Transliteration**

**1. Abstract**

This project aims to develop a **transliteration system** that converts Romanized Khmer words, often referred to as “sing Khmer,” into their corresponding Khmer script. To achieve this, we train a lightweight deep learning model on a dataset of Romanized–Khmer word pairs. The model is designed to be small enough to ensure low latency, making it practical for real-time use. To improve accuracy, the system incorporates a dictionary check that validates model outputs against existing Khmer words. Instead of producing only one result, the system generates **five candidate suggestions**, giving users the flexibility to choose the correct word if the top prediction is inaccurate.

**2. Dataset**

The dataset, provided by **Chrea Chanchhunneng**, contains **28,576 Romanized–Khmer text pairs**. Each entry consists of a Romanized Khmer word alongside its correct Khmer equivalent. Example pairs include:

* ប្រដែ: brodae
* អសង្ខៃយ: aasangkheyy
* ឆាតកភ័យ: chhatkophey
* ទព្វសម្ភារៈ: topvosompheareak
* ទោចក្រយាន: touchkroyean
* នារី: neari

In addition to this dataset, I will also make use of the **Khmer Dictionary 44k** dataset available on Hugging Face ([Khmer Dictionary 44k](https://huggingface.co/datasets/seanghay/khmer-dictionary-44k)). This dictionary contains over 44,000 Khmer words and their Romanized forms. It will be utilized as a reference resource for cross-checking and validating the model’s output, ensuring that the generated transliterations correspond to valid Khmer words

**3. Methodology**

**3.1 Data Preprocessing**

The preprocessing pipeline begins with cleaning the Romanized text by removing special characters (e.g., !@#$%^&\*;’...). Next, Khmer texts are normalized using the **Khmer Spell** Python package via the **khnormal** function. This step ensures that text encoding issues are resolved and that combining characters are ordered correctly. After normalization, both the Romanized Khmer and Khmer text are tokenized using TensorFlow’s tokenization method. We apply character-level tokenization rather than word-level, since the task requires mapping sequences of characters from one script to another rather than translating whole words. This approach also enables the model to generate words that were not seen during training, such as transliterating **Facebook** into “**ហ្វេសប៊ុក**”

**3.2 Model Training, and Selection**

Several architectures will be explored to identify the best-performing model for the transliteration task. We focused on sequence-to-sequence (seq2seq) encoder–decoder models, specifically **Recurrent Neural Networks** (RNNs**), Long Short-Term Memory** (LSTM) networks, and **Gated Recurrent Units** (GRUs). After training and evaluating all three, the best-performing model was selected. We then enhanced this model by incorporating additional components such as **bidirectional layers**, which capture context from both directions, and an **attention mechanism**, which allows the decoder to focus on the most relevant parts of the input sequence during prediction.

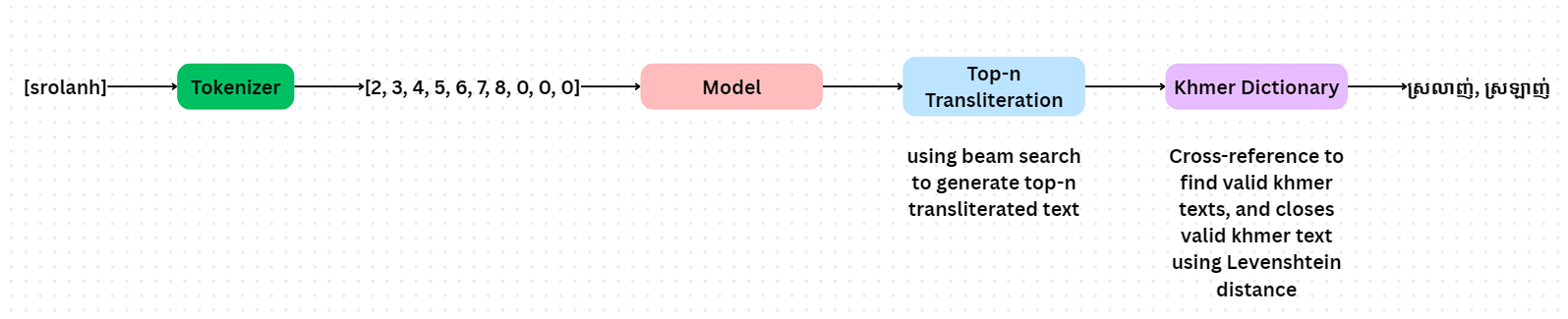
**3.3 Evaluation Metrices**

To evaluate the model performance, we use Token-Level Accuracy, Character Error Rate (CER), and BiLingual Evaluation Understudy (BLEU).

**3.4 Inference**

During inference, the trained encoder–decoder model generates candidate transliterations for a given Romanized Khmer input. To improve prediction quality, we apply **beam search decoding**, which maintains multiple candidate sequences at each step instead of committing to the single most likely character. In our system, we use a beam width of five, meaning the model produces the top five candidate transliterations ranked by probability.

Since the model may sometimes generate sequences that are not valid Khmer words, we further refine these outputs using a **dictionary alignment step**. First, the system checks if any of the beam search candidates exactly match entries in a Khmer dictionary. If no exact match is found, we apply **Levenshtein distance** to compute the edit distance between each candidate and dictionary words. The closest valid dictionary entry within a predefined maximum edit distance is then selected as the final prediction.



**4. Web Interface**

To demonstrates the system, a user web interface will be created using Flask. The interface provides two main functionalities:

* **Suggested Transliteration**: This tab displays the top transliteration suggestion. Users can confirm if the output is correct, and press **SPACE** to accept the suggestion.
* **Switch Transliteration Suggestion**: Each suggestion can be cycle through by using the **TAB keys**.