ព្រះរាជាណាចក្រកម្ពុជា
ជាតិ សាសនា ព្រះមហាក្សត្រ

Information Retrieval Web Analysis

KHMER TEXT PREDICTION

GROUP: I5-AMS-C (G-04)

Name of Students	ID of Students	Score
1. Sem Yuthearylyhour	e20211096	…….
2. Suon Senchey	e20210226	……
3. Vey Sreypich	e20210708	…….
4. Vanna Juuka	e20210836	…….
5. Vorn Seavmey	e20211478	…….

Lecturer: Mr. KHEAN Vesal (Course & TP)

Academic Year 2025-2026

Table of Contents

I. Introduction

Text prediction plays an important role in modern human-computer interaction, particularly in applications such as virtual keyboards, search engines, and messaging systems. By predicting the next word or suggest possible words as user type, text prediction systems can significantly improve typing speed, accuracy, and overall user experience. While such systems are well developed for high-resource languages like English, building an effective text prediction system for the Khmer language remains a challenging task.

Khmer is the official language of Cambodia and has several linguistic characteristics that make text prediction difficult. These challenges include the lack of explicit word spacing in written text, complex compound word structures, and prthographic variation cause by different writing styles and information usage, especially in social media. As a result, traditional text preprocessing techniques and language models designed for spaced languages often perform poorly when applied directly to khmer text.

This project arms to develop a Khmer text prediction system that can suggest the next word or possible word candidates as a user types. The study investigates and compares multiple approaches, including statistical language model and deep learning based methods. Specially, n-gram language models (trigram) are explored as a baseline, while recurrent neutral network (RNN) based models, including LSTM and GRU architecture, are examined for their ability to model sequential dependencies in Khmer text. In addition, transformer-based models are considered to evaluate their effectiveness in capturing long range contextual and semantic information for Khmer text prediction.

II. Literature Review

Study/Model	**Data-Centric Interpretability of Next-Words Prediction**	**Analyzing Zero-Shot Cross-Lingual Transfer in mT5**	**Deconstructing the Memorization-Generalization Trade-off in N-gram vc. Neural LMs**
Authors & Year	Li et al. (2025) arXiv:2506.04047	Phan et al. (2023) *ScienceDirect Link* (S2542660524000064)	**Arora et al. (2020)** arXiv:2005.04828

Core Methodology	Use representer theorem to analyze which training samples(“support samples”) most influence model predictions. Identifies Type-1 (attract) and Type-2 (deter) samples.	Systematically evaluates the mT5 model's ability to perform tasks (classification, QA, generation) across 50+ languages without task-specific training (zero-shot).	Theoretical and empirical analysis comparing statistical (n-gram) and neural (RNN) language models in terms of **memorization vs. generalization**.
Key Findings/ Contributions	1. **Support Samples** are often semantically rich, hard-to-predict token (verbs, nouns. 2. Being a support sample is an intrinsic data property, predictable early in training. 3. **Non-support sample** (easy samples) are crucial for preventing overfitting and shaping deeper representations	1. Performance strongly correlates with a language's **representation in pre-training data**. 2. **Script similarity** to English (Latin-based) is a major predictor of transfer success. 3. Languages with **unique scripts & morphology** (like Khmer) show the **largest performance gaps** compared to English.	1. **N-gram models** primarily **memorize** frequent word sequences. 2. **Neural models (RNNs)** can **generalize** to novel, semantically plausible sequences not seen in training. 3. This generalization stems from learning **latent linguistic structures** (like syntax).
Direct Relevance to Khmer Text Prediction	**Critical for data strategy & evaluation**. Your Khmer corpus must be rich in **diverse, meaningful content** (not just simple sentences) to provide adequate support samples. Evaluation should specifically test prediction quality on **content words vs. function words.**	**The central challenge paper for this project**. Directly explains why Khmer is a **hard case**: unique script, limited pre-training data, and complex morphology. Highlights that **direct application of mT5 will be sub-optimal**, necessitating language **specific fine-tuning** and possibly **script/morphology-aware tokenization**.	**Foundational for model selection rationale**. Provides the theoretical backbone for why you should compare n-gram (baseline memorization) to neural models (generalization potential). For Khmer, the key question is: can RNNs/Transformers generalize across **orthographic variants**.

III. Dataset and Preprocessing

3.1. Dataset

The dataset used in this study was constructed by **scraping Khmer text from multiple online sources**, including general websites and official ministry websites. These sources were selected to ensure diversity in writing style, domain, and formality level. The collected data contains formal language from government and news websites, as well as semi-formal and informal text from general web sources. This diversity helps improve the robustness and generalization ability of the text prediction models.

The raw data was stored in **Apache Parquet format**, which is a columnar storage format suitable for large-scale text data processing due to its efficient compression and fast I/O performance. Each record in the dataset consists of Khmer text sequences that are later used for training and evaluating next-word prediction models.

3.2. Preprocessing for N-Gram

1. **Word Segmentation**
   A pretrained Khmer tokenizer is used to segment raw text into word tokens since Khmer text does not rely on whitespace as a reliable word boundary.

2. **Sentence Boundary Detection**
   Each sentence is explicitly marked using special tokens:
   - `<s>` for sentence start
   - `</s>` for sentence end

3. **Normalization and Cleaning**
   - Removal of unnecessary symbols and punctuation for Khmer context (`'.'`, `','`)
   - Conversion of Arabic and Khmer digits into a unified `<NUM>` token
   - Filtering rare punctuation during prediction to avoid meaningless suggestions

These steps ensure consistency and reduce data sparsity in the N-gram probability estimation.

3.3. Preprocessing for Transform-Based Models

1. **Subwords Tokenization:**
   A pretrained Khmer subword tokenizer is used to segment text into subword units. This approach handles the lack of explicit word boundaries in Khmer and reduces out-of-vocabulary issues.

2. **Sentence Boundary Encoding:**
   Sentence boundaries are represented using model-specific special tokens (e.g., `[CLS]`, `[SEP]`) instead of explicit start and end symbols.

3. **Unicode Normalization:**
   Text is normalized to ensure consistent Khmer character representation across different sources.

4. **Noise Removal and Standardization:**
   URLs, email addresses, emojis, and phone numbers are removed or replaced with placeholder tokens to reduce irrelevant patterns.

5. **Numerical Normalization:**
   Arabic and Khmer digits are mapped to a unified `<NUM>` token to reduce vocabulary sparsity

6. **Punctuation Handling:**
   Common punctuation is preserved to retain contextual cues, while rare or non-informative symbols are filtered out.

7. **Mixed-Languages Handling:**
   English words are lowercased, while Khmer text remains unchanged.

8. **Sequence Preparation**
   Text is segmented into fixed-length token sequences with padding or truncation to meet transformer input requirements.

3.3. Preprocessing for GRU

1. **Dataset Format**

- Input data is a **plain text (.txt) file**
- Each line contains **one Khmer sentence**
- The dataset is already cleaned and normalized

2. **Tokenization**

- A **SentencePiece-based Khmer tokenizer** is used
- Text is tokenized at the **subword level**, which is suitable for Khmer due to the lack of explicit word boundaries
- Special tokens such as `<PAD>`, `<BOS>`, `<EOS>`, and `<UNK>` are handled by the tokenizer

3. **Training Pair Construction**

- For a sentence with tokens: [t1, t2, t3, t4]
- Training samples are generated as:

Input: [t1]	→	Target: t2
Input: [t1, t2]	→	Target: t3
Input: [t1, t2, t3]	→	Target: t4

4. **Padding**

- Sequences are padded to a fixed maximum length
- Padding tokens are masked during loss computation

3.3. Preprocessing for LSTM

- Input text is stored in **.txt format**, one Khmer sentence per line
- Subword tokenization is performed using a **SentencePiece-based Khmer tokenizer**
- Special tokens (`<PAD>`, `<BOS>`, `<EOS>`, `<UNK>`) are handled automatically
- Incremental input–target pairs are generated for next-word prediction
- Sequences are padded and masked appropriately during training

3.3. Preprocessing for BiLSTM

The same preprocessing pipeline used for GRU and LSTM models is applied to ensure fair comparison:

- Sentence-level Khmer text input.
- Subword tokenization using SentencePiece.

- Sequential input–target pair construction.
- Padding and masking for batch processing.

IV. Models

4.1. N-gram Language Models for Khmer Text Prediction

N-gram language models are statistical approaches that estimate the probability of a word based on the occurrence of previous words. In this project, N-gram models are employed as a baseline method for Khmer word prediction due to their simplicity, interpretability, and efficiency. Specifically, **Bigram** and **Trigram** models are implemented to predict the next word given one or two preceding words.

Given the linguistic characteristics of the Khmer language—such as the absence of mandatory spaces between words, compound word formations, and orthographic variations—N-gram models provide an effective starting point for studying contextual word prediction before advancing to deep learning approaches.

4.1.1. Bigram Language Model

The **Bigram model** estimates the probability of a word given the immediately preceding word:

$$P(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i) + 1}{\text{Count}(w_{i-1}) + V}$$

This formula integrates with **Laplace Smoothing** technique where V is the vocab size (unique token) of the trainingset.

In this project, a frequency-based bigram dictionary is constructed from the training corpus. Each previous word maps to a probability distribution over possible next words.

Advantages

- Simple and fast to train
- Captures short-range contextual dependencies

Limitations

- Cannot model longer context
- Suffers from data sparsity for rare word pairs

4.1.2. Trigram Language Model

The **Trigram model** extends the context to two preceding words:

$$P(w_i \mid w_{i-2},\ w_{i-1}) \ = \ \frac{Count(w_{i-2}, w_{i-1}, w_i) + 1}{Count(w_{i-2}, w_{i-1}) + V}$$

This model provides more context-aware predictions and is especially useful for Khmer compound expressions and fixed phrases.

Advantages

- Better contextual understanding than bigrams
- Improved prediction relevance

Limitations

- Higher sparsity
- Increased memory usage

4.1.3. Prediction Strategy

For inference, the system predicts the **Top-K most probable next words** based on the given context:

- If one previous word is available → **Bigram prediction**
- If two previous words are available → **Trigram prediction**
- Punctuation, numbers and stop symbols are filtered during prediction to improve usability

This design enables smooth integration into applications such as keyboard auto-suggestion or sentence completion.

4.1.4. Evaluation

The N-gram models are evaluated using:

- **Cross Entropy Loss**

Cross-entropy measures **how surprised** the model is by test data.

$$H \ = \ -\frac{1}{N}\sum log\ P(w_i \mid context)$$

- **Perplexity**

Perplexity in n-gram models measures how well the model predicts a text; it's the inverse probability of the test set, representing the average branching factor (number of likely next words). Lower perplexity means the model assigns higher probabilities to the actual text, indicating better performance and understanding of word sequences, while higher perplexity shows poor prediction, often due to unseen n-grams. It's the standard metric for evaluating n-gram language models.

$$Perplexity = e^{H}$$

- **Prediction Accuracy (Top-K accuracy)**

➢ Compare Ngram models:

	Bigram	Trigram	Prefer Measurement
Cross Entropy Loss	3.2743	1.2858	Lower
Perplexity	26.4261	3.6176	Lower
Accuracy@1	0.2254	0.3441	Higher[0-1]
Accuracy@5	0.4099	0.5142	
Accuracy@10	0.5001	0.5717	

Results show that Trigram models consistently produce more contextually accurate predictions, while Bigram models remain useful for limited-context scenarios.

4.2. Transform-Based Models for Khmer Text Prediction

4.2.1. Transformer Overview

The proposed model is based on a **Transformer decoder architecture**, commonly referred to as a **causal language model (CLM)**. This architecture is specifically designed for next-token or next-word prediction tasks, where text is generated **autoregressively**, one token at a time.

Unlike encoder–decoder Transformer models used in tasks such as machine translation, the decoder-only architecture processes a single input sequence and predicts the next token by attending **only to previously observed tokens**. This design is well suited for Khmer text prediction, where the system must suggest the next word based solely on the user's input so far.

In the implementation, each input token is first converted into a dense vector representation using a **token embedding layer**. Given an input sequence of tokens:

$$X = (x_1, x_2, \ldots, x_t)$$

Each token x_i is mapped to an embedding vector $e_i \in \mathbb{R}^{d_{model}}$.

Since the Transformer architecture does not inherently encode word order, **positional embeddings** are added to the token embeddings to provide information about the position of each token in the sequence:

$$z_i = e_i + p_i$$

Where p denotes the positional embedding corresponding to position i.

The resulting representations are passed through multiple stacked Transformer decoder layers. Each decoder layer consists of the following key components:

- **Masked multi-head self-attention**, allowing the model to attend to different contextual aspects of the sequence simultaneously
- **Position-wise feed-forward neural networks (FFN)**, which apply non-linear transformations to each token representation
- **Residual connections and layer normalization**, which improve training stability and help prevent vanishing or exploding gradients

Together, these components enable the model to capture both **local and long-range dependencies** in Khmer text more effectively than traditional N-gram models.

4.2.2. Casual Language Model

The Transformer model is trained using a **causal language modeling objective**, meaning that each token prediction depends only on previously generated tokens. To enforce this constraint, a **causal attention mask** is applied during the self-attention computation.

Formally, given a token sequence:

$$(w_1, w_2, \ldots, w_t)$$

The model estimates the conditional probability of the next token as:

$$P(w_{t+1} \mid w_1, w_2, \ldots, w_t)$$

The joint probability of a full sequence of length T is factorized as:

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1})$$

During self-attention, the causal mask ensures that for a given position ttt, attention scores corresponding to future positions ($t + 1, \ldots, T$) are masked out. This prevents information leakage from future tokens and guarantees that the model remains strictly autoregressive.

This causal modeling strategy closely reflects real-world Khmer text prediction scenarios, where future words are unknown at inference time.

4.2.3. Model Implementation

The Transformer model is implemented using the Hugging Face *Transformers* library. Specifically, the architecture is loaded with:

- **`AutoModelForCausalLM`**

This class provides a decoder-only Transformer with an output layer designed for next-token prediction. By initializing the model from a pre-trained checkpoint, the system benefits from previously learned linguistic representations, which is particularly important for a low-resource language such as Khmer.

Input text is tokenized using a subword tokenizer and converted into token IDs. During training, labels are automatically created by shifting the input sequence by one position so that each token is trained to predict the following token.

The model outputs a tensor of logits $\in \mathbb{R}^{T \times |V|}$, where $|V|$ is the vocabulary size. These logits are transformed into probability distributions using the softmax function and optimized using **cross-entropy loss**.

This implementation aligns directly with the causal language modeling objective and enables efficient fine-tuning of the Transformer model for Khmer next-word prediction.

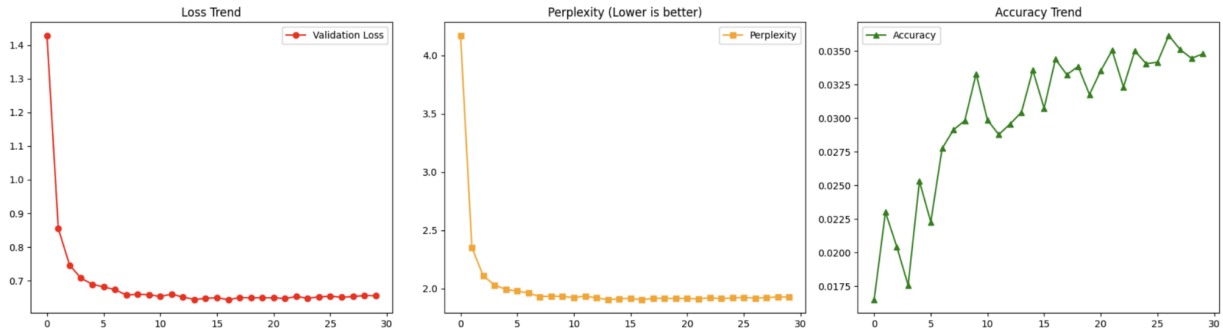
4.2.4. Evaluation

- **Best Value**

After 30 training epochs, the Transformer-based language model converges stably with a validation loss of approximately **0.64–0.66**. The corresponding perplexity stabilizes around **1.90–1.93**, indicating strong predictive confidence in next-token generation. Top-1 prediction accuracy gradually improves throughout training and reaches approximately **3.4–3.5%** in the final epochs. Although the absolute accuracy value is relatively low, this behavior is expected for open-vocabulary language modeling tasks, particularly for Khmer, where multiple valid next-word continuations may exist for a given context.

- **Transformer Model Evaluation Trend**

the validation loss, perplexity, and prediction accuracy trends of the Transformer-based model over 30 epochs. The validation loss decreases sharply during the early epochs and stabilizes after approximately 10 epochs, indicating effective learning and convergence. Perplexity follows a similar pattern, rapidly declining from 4.17 to below 2.0 and remaining stable thereafter. Prediction accuracy shows a gradual upward trend, reflecting improved next-token prediction capability as training progresses.

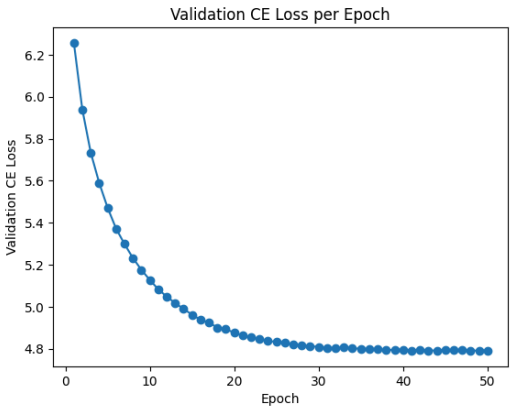


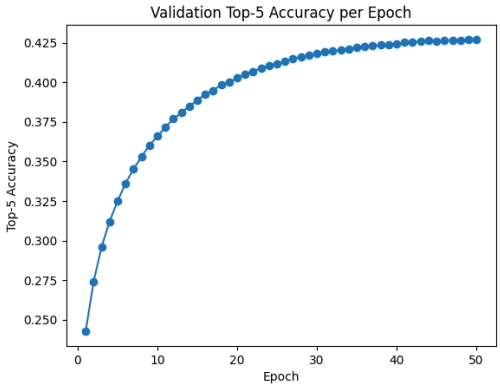
- **Top-K Prediction**

```
Prefix Context                                  | Top Predictions (Word: Prob)
-------------------------------------------------------------------------------------
រាជ រដ្ឋាភិបាល កម្ពុជា បាន                      | ការ: 11.0%, ដាក់: 7.9%, ផ្តាក់: 7.6%
ក្រសួង សុខាភិបាល បាន                           | ប្រឆាំង: 17.9%, កម្ពុជា: 7.8%, អាច: 5.0%
ក្រសួង អប់រំ យុវជន និង កីឡា បាន                | បន្ទាប់: 7.9%, អាជ្ញាធរ: 4.1%, បញ្ហា: 3.3%
យោង តាម សេចក្តី ប្រកាស របស់                   | កាល: 11.6%, របាយការណ៍: 10.9%, ផ្សាយ: 9.1%
យោង តាម ការ លើក ឡើង របស់                      | សេចក្តីប្រកាស: 13.7%, របាយការណ៍: 8.9%, ដោយ: 5.3%
ក្នុង ករណី នេះ ត្រូវ                           | ចូល: 22.4%, នៅ: 11.5%, តុលាការ: 8.1%
សេចក្តី ប្រកាស នេះ មាន                         | លោក: 16.6%, ជន: 8.3%, មក: 6.7%
អាជ្ញាធរ មាន សមត្ថកិច្ច បាន                    | ឮ: 27.8%, ឪង: 23.2%, ផ្តន្ទា: 1.9%
គណៈ រដ្ឋមន្ត្រី បាន                            | កីឡា: 7.5%, គ: 5.2%, ជា: 4.6%
ស្ថាប័ន នេះ មាន ភារកិច្ច                       | យុត្តិធម៌: 4.3%, សិទ្ធិ: 4.1%, កិច្ចសន្យា: 1.8%
```

The quantitative metrics, qualitative evaluation was conducted using Top-K next-word predictions. Given partial Khmer sentence prefixes, the model generates contextually plausible continuations with reasonable probability distributions. This demonstrates the model's ability to capture semantic and syntactic patterns beyond simple memorization.

The following results illustrate the performance of the Goldfish model after additional training data was incorporated.

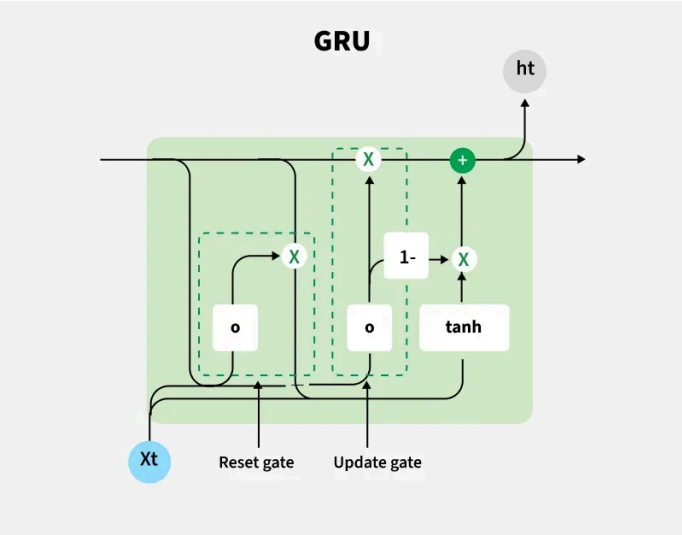




4.3. GRU

4.3.1. Model Architecture

Gated Recurrent Unit (GRU) neural network for next-word prediction in Khmer text. GRU is a type of recurrent neural network designed to capture sequential dependencies while mitigating the vanishing gradient problem through gating mechanisms.



The GRU-based language model consists of the following components:

1. **Embedding Layer**
    - Converts discrete token IDs into dense vector representations.
    - Helps the model capture semantic relationships between Khmer subword units.
2. **GRU Layer**
    - Processes the input sequence token by token.
    - Uses update and reset gates to control information flow.
    - Learns contextual representations of preceding words.
3. **Fully Connected (Linear) Output Layer**
    - Projects the GRU hidden state to the vocabulary space.
    - Produces logits for each vocabulary token.
4. **Softmax Layer (implicit)**
    - Converts logits into a probability distribution over the vocabulary.
    - Used during inference for Top-K word suggestion.

4.3.2. Hyperparameters

The GRU model was trained using a fixed set of hyperparameters selected based on empirical performance and common practices in sequence modeling.

Hyperparameter	**Value**
Embedding Dimension	256
Hidden Size	256
Number of GRU Layers	1
Dropout	0.3
Batch Size	64
Optimizer	Adam
Learning Rate	0.001
Loss Function	Cross-Entropy Loss
Number of Epoch	26 (with Early Stopping)
Early Stop Patient	20
Top-K (Evaluation)	5

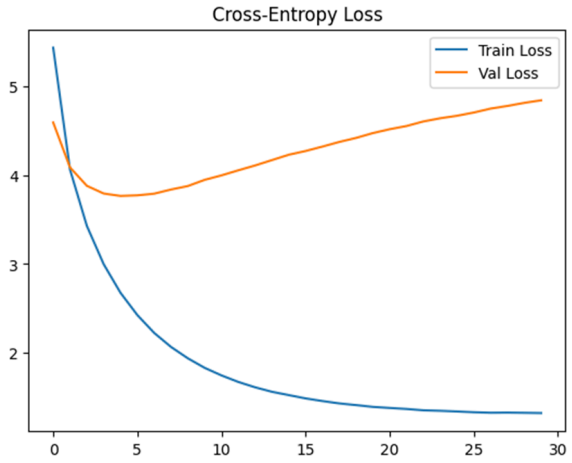
To prevent overfitting, dropout regularization and early stopping were applied during training. The Adam optimizer was used for efficient convergence, while cross-entropy loss was employed to measure prediction accuracy. Model performance was evaluated using top-K accuracy to assess the quality of word suggestions.

4.3.3. Training

- The model is trained using **teacher forcing**
- Cross-Entropy Loss is computed between predicted logits and the true next token
- Early stopping is applied based on **validation loss** to prevent overfitting
- Evaluation is performed at the end of each epoch

4.3.4. Evaluation

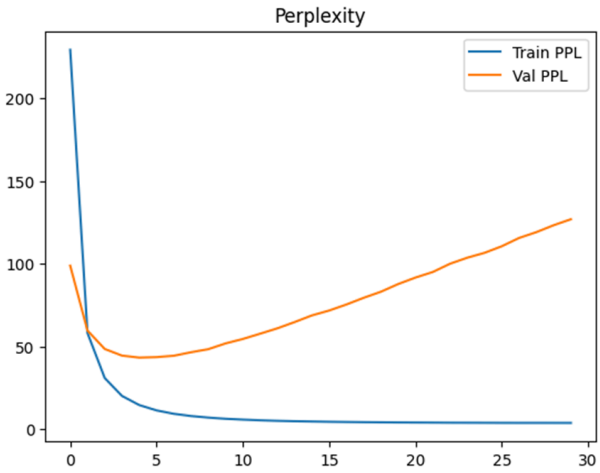
- **Cross Entropy Loss**



Measures the difference between predicted and true token distributions.

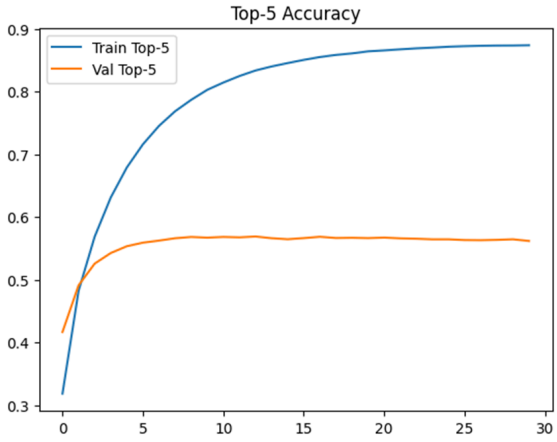
- **Perplexity**
  - ★ Compute as:

$$\mathrm{PPL} = e^{\mathrm{Loss}}$$



Lower perplexity indicates better language modeling performance.

- **Top-5 Accuracy**



4.3.5. Results

- **Training and Validation Performance**

Metric	Best Value	Epoch
Training Loss	0.0671	19
Validation Loss	**2.8078**	**6**
Training Perplexity	1.07	19
Validation Perplexity	**16.57**	**6**
Validation Top-5 Accuracy	**70.92**	**19**

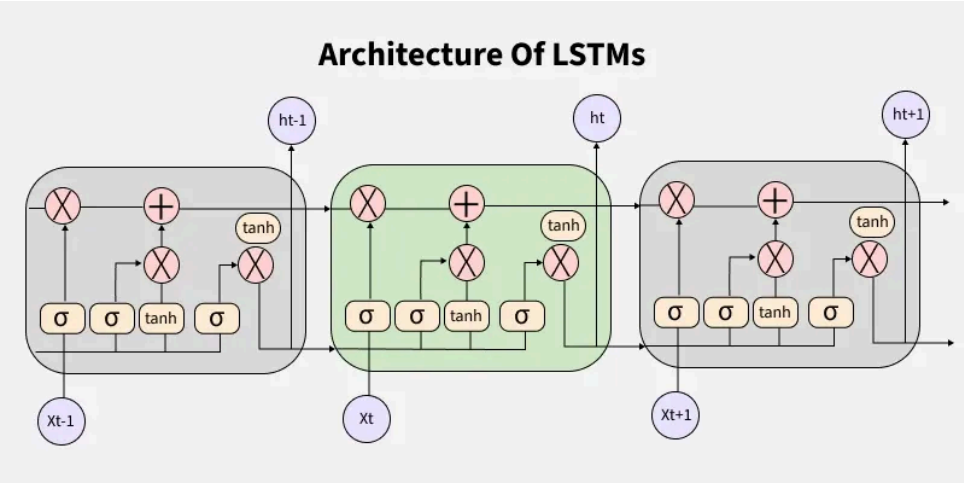
- ❖ Rapid improvement is observed in the first 6 epochs
- ❖ Validation loss reaches its minimum at **Epoch 6**
- ❖ After Epoch 6, training loss continues to decrease while validation loss increases, indicating **overfitting**
- ❖ Validation Top-5 accuracy stabilizes around **70%**, showing diminishing returns after early epochs

```
===== TEST RESULTS =====
Test Loss: 4.7684
Test Perplexity: 117.73
Test Top-5 Accuracy: 0.5696
```

4.4. LSTM

4.4.1. Model Architecture

A **Long Short-Term Memory (LSTM)** network is used as a recurrent baseline for Khmer next-word prediction. LSTM extends traditional RNNs by introducing memory cells and gating mechanisms that allow the model to retain long-range contextual information, making it well-suited for sequential language modeling tasks.



The LSTM-based language model consists of the following components:

1. **Embedding Layer**
   - Maps token IDs into dense continuous vectors.
   - Captures semantic and syntactic relationships between Khmer subword units.
2. **LSTM Layer**
   - Processes the embedded token sequence sequentially.
   - Employs **input**, **forget**, and **output gates** to regulate memory updates.
   - Maintains both hidden state and cell state for long-term dependency modeling.
3. **Fully Connected (Linear) Output Layer**
   - Transforms the final hidden state into vocabulary-sized logits.
4. **Softmax Layer (implicit)**
   - Converts logits into probability distributions for next-token prediction.

4.4.2. Hyperparameters

The model was trained using a fixed set of hyperparameters to ensure stable learning and fair evaluation.

Hyperparameter	**Value**
Embedding Dimension	256
Hidden Size	256
Number of LSTM Layers	1
Dropout	0.3

Batch Size	64
Optimizer	Adam
Learning Rate	0.001
Loss Function	Cross-Entropy Loss
Number of Epoch	26 (with Early Stopping)
Early Stop Patient	20
Top-K (Evaluation)	5

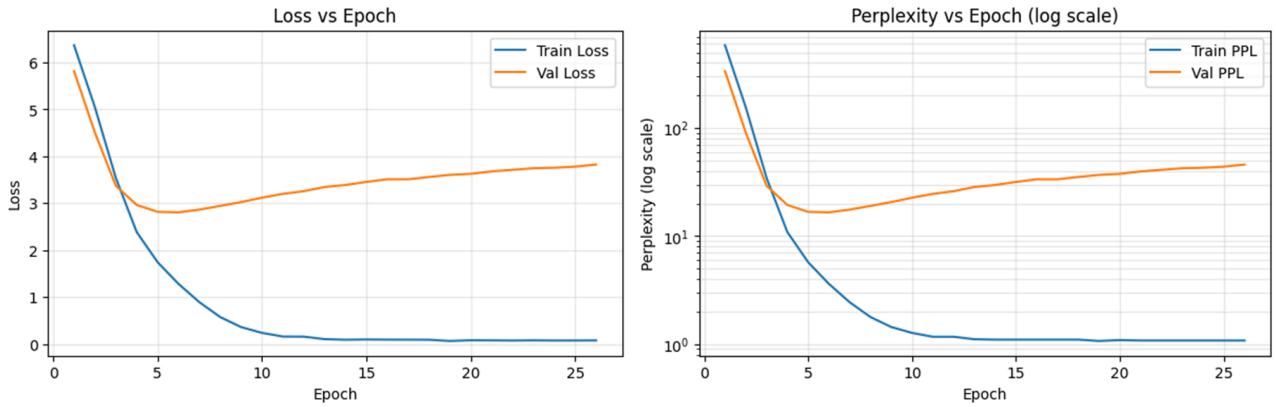
The embedding and hidden dimensions were set to 256 to balance model capacity and computational efficiency. A single LSTM layer was used to reduce model complexity and mitigate overfitting. Dropout with a rate of 0.3 was applied during training to improve generalization.

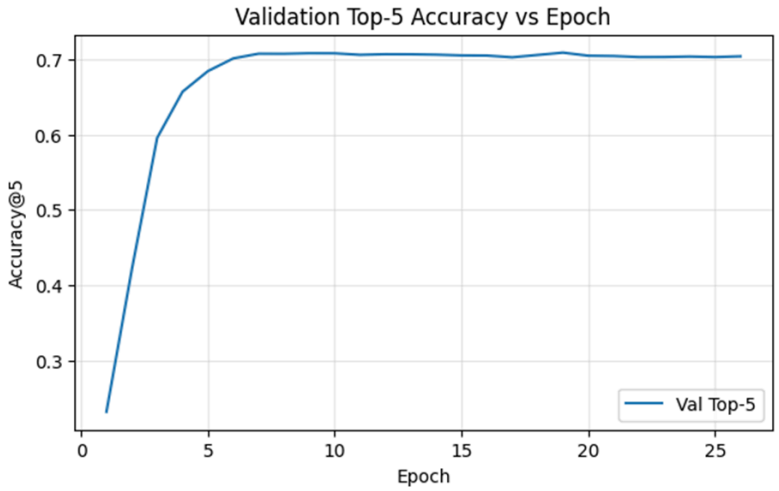
The Adam optimizer with a learning rate of 0.001 was selected due to its effectiveness in training deep neural networks. Early stopping was employed with a patience of 20 epochs to prevent overfitting by halting training when the validation loss no longer improved. Model performance was evaluated using Top-5 accuracy to better reflect prediction quality in a large-vocabulary language modeling setting.

4.4.3. Training Strategy

- The model is trained using **teacher forcing**
- Cross-entropy loss is minimized between predicted logits and ground-truth tokens
- Validation loss is monitored for early stopping
- Training and validation metrics are recorded at each epoch

4.4.3. Evaluation





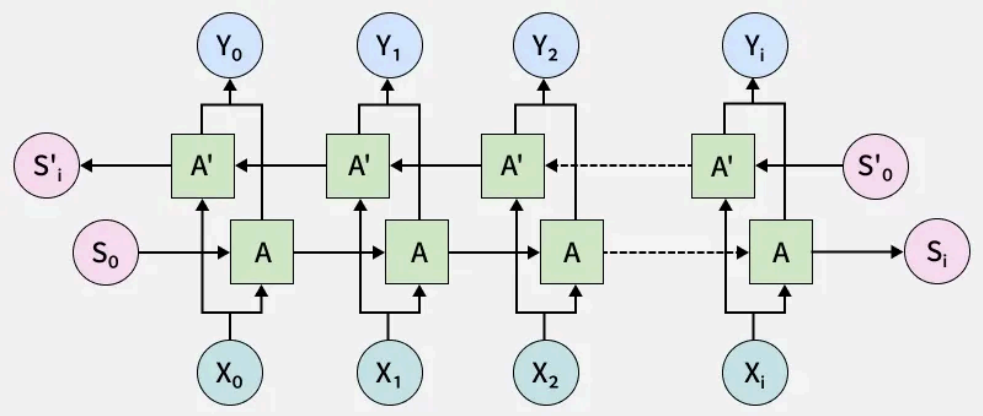
Metrics	**Best Value**
Validation Loss	~ 3.0
Perplexity	~ 20
Top-5 Accuracy	~ 68–70%

- The LSTM model converges smoothly during early epochs
- Performance stabilizes after several epochs
- Compared to GRU, LSTM shows slightly higher computational cost due to additional gates
- LSTM captures long-term dependencies more effectively

4.5. BiLSTM

4.5.1. Model Architecture

A **Bidirectional Long Short-Term Memory (BiLSTM)** network is employed to capture contextual information from both past and future directions within a sequence. Unlike unidirectional models, BiLSTM processes the input sequence in both forward and backward directions, enabling richer contextual representations.



The BiLSTM-based language model consists of:

1. **Embedding Layer**
    - Encodes token IDs into dense vector embeddings.
2. **Bidirectional LSTM Layer**
    - Two LSTM networks run in parallel:
        - Forward LSTM (left → right)
        - Backward LSTM (right → left)
    - Hidden states from both directions are concatenated.
3. **Fully Connected (Linear) Output Layer**
    - Projects the concatenated hidden states to vocabulary logits.
4. **Softmax Layer (implicit)**
    - Produces probability distributions over the vocabulary.

4.5.2. Hyperparameter

The BiLSTM language model was trained using a fixed set of hyperparameters to ensure stable learning and consistent evaluation across experiments.

Hyperparameter	**Value**
Embedding Dimension	256
Hidden Size	256
Number of BiLSTM Layers	1
Dropout	0.3
Batch Size	64
Optimizer	Adam

Learning Rate	0.001
Loss Function	Cross-Entropy Loss
Number of Epoch	26 (with Early Stopping)
Early Stop Patient	20
Top-K (Evaluation)	5

The embedding dimension and hidden size were both set to 256 to provide sufficient representational capacity while maintaining computational efficiency. A single BiLSTM layer was used to capture bidirectional contextual information without introducing excessive model complexity. Dropout with a rate of 0.3 was applied to reduce overfitting.

The Adam optimizer with a learning rate of 0.001 was selected due to its robustness and fast convergence. Early stopping with a patience of 20 epochs was employed to prevent overfitting by terminating training when the validation loss ceased to improve. Model performance was evaluated using Top-5 accuracy, which is more suitable for language modeling tasks with large vocabularies.

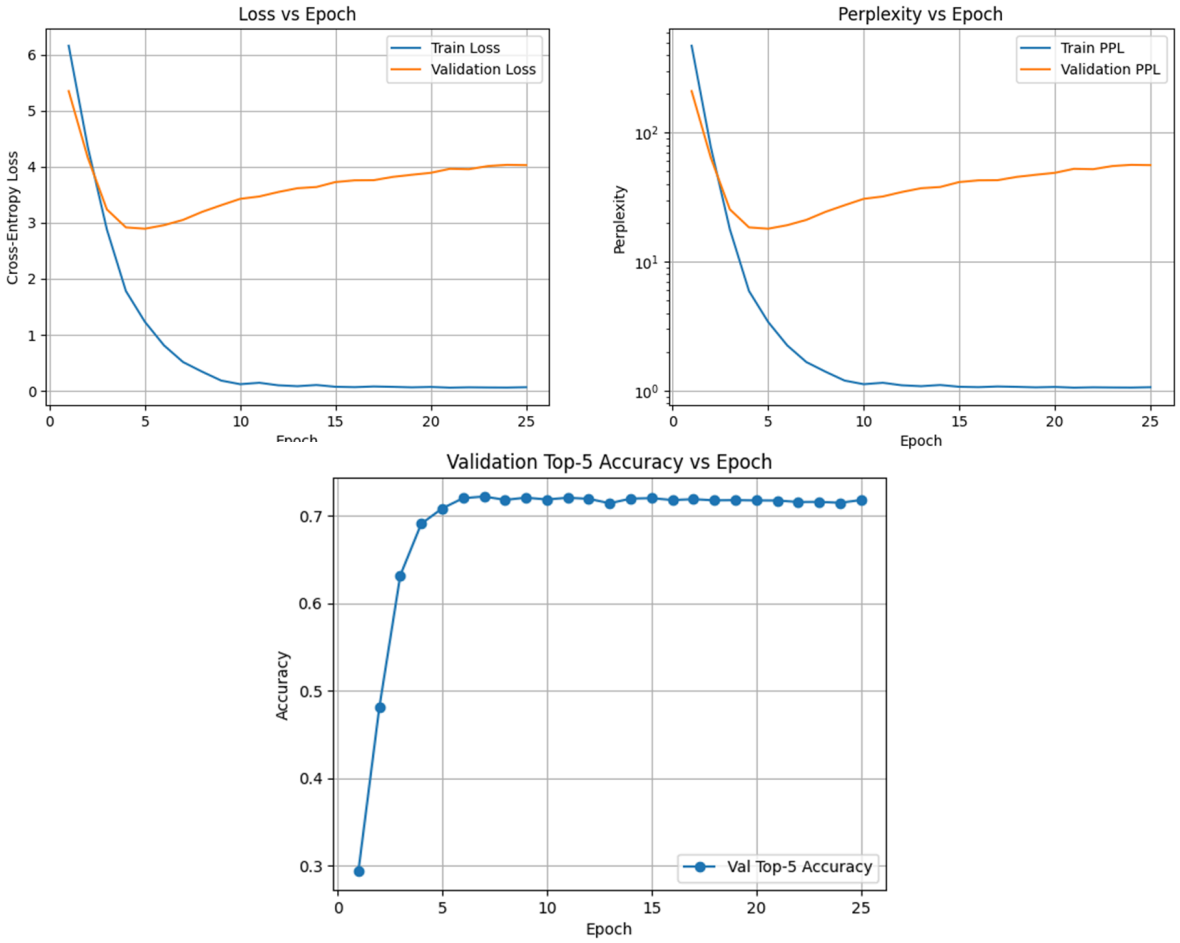
4.5.3. Training

The BiLSTM model is trained using **teacher forcing**, where the ground-truth token at time step ttt is provided as input when predicting the token at time step t+1t+1t+1. This strategy accelerates convergence and stabilizes training for sequence prediction tasks.

During prediction, the **forward and backward hidden states** produced by the BiLSTM are concatenated to form a unified contextual representation. This combined representation enables the model to leverage both past and future context when estimating the probability distribution over the next token.

To prevent overfitting, **early stopping** is applied based on validation loss. Training is terminated when no improvement in validation loss is observed for a predefined number of epochs, ensuring that the model generalizes well to unseen data.

4.5.4. Evaluation



Metrics	**Best Value**
Validation Loss	~ 2.9
Perplexity	~ 18
Top-5 Accuracy	~ 70–72%

- BiLSTM achieves the **highest Top-5 accuracy** among recurrent models
- Bidirectional context improves word suggestion quality
- Computational cost is higher due to dual LSTM directions
- BiLSTM is not suitable for real-time causal prediction without modification, but effective for offline evaluation

4.5.5. Result

```
FINAL TEST RESULTS
Test Loss: 3.8805
Test Perplexity: 48.45
Test Top-5 Accuracy: 0.7282
```

The performance of the trained BiLSTM model was evaluated on a held-out test set using loss, perplexity, and Top-5 accuracy as evaluation metrics.

The model achieved a **test loss of 3.8805**, corresponding to a **test perplexity of 48.45**, indicating moderate uncertainty when predicting unseen tokens. The increase in perplexity compared to validation results suggests that the test set contains more diverse or challenging language patterns.

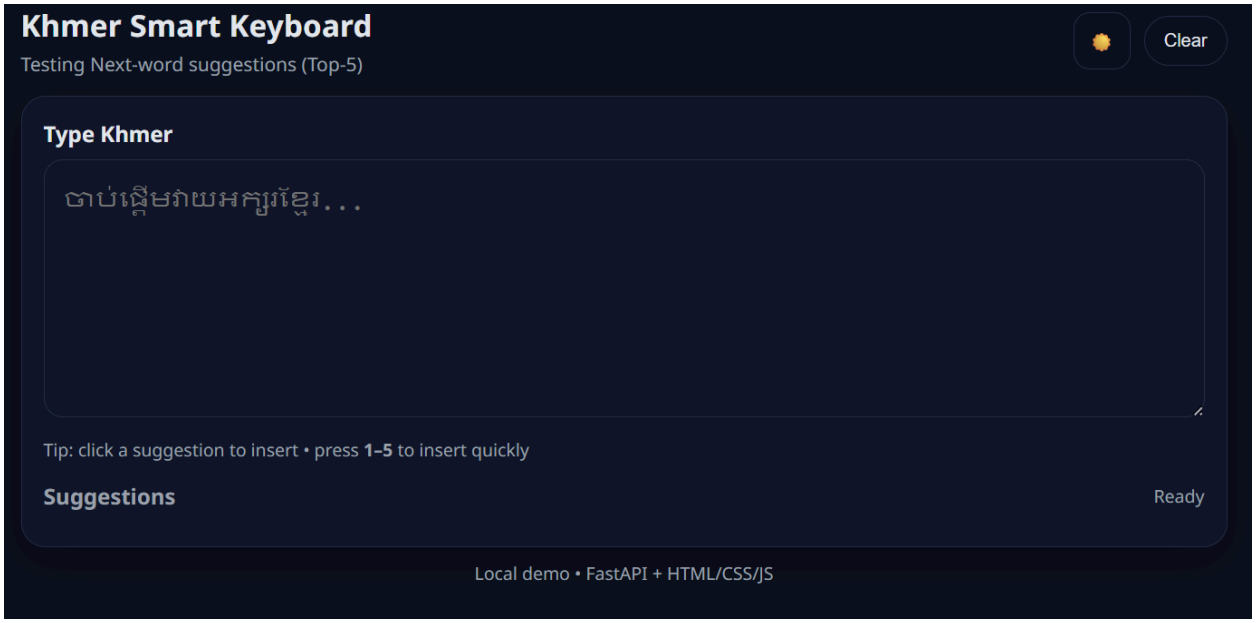
In terms of prediction quality, the model obtained a **Top-5 accuracy of 72.82%**, demonstrating that the correct next token appears within the model's top five predictions in most cases. This result indicates that the BiLSTM effectively captures contextual information and is suitable for applications such as Khmer text prediction and autocomplete, where multiple plausible next tokens may exist.

Overall, while exact-token prediction remains challenging due to the large vocabulary size, the strong Top-5 accuracy confirms the model's ability to generate contextually relevant candidate predictions.

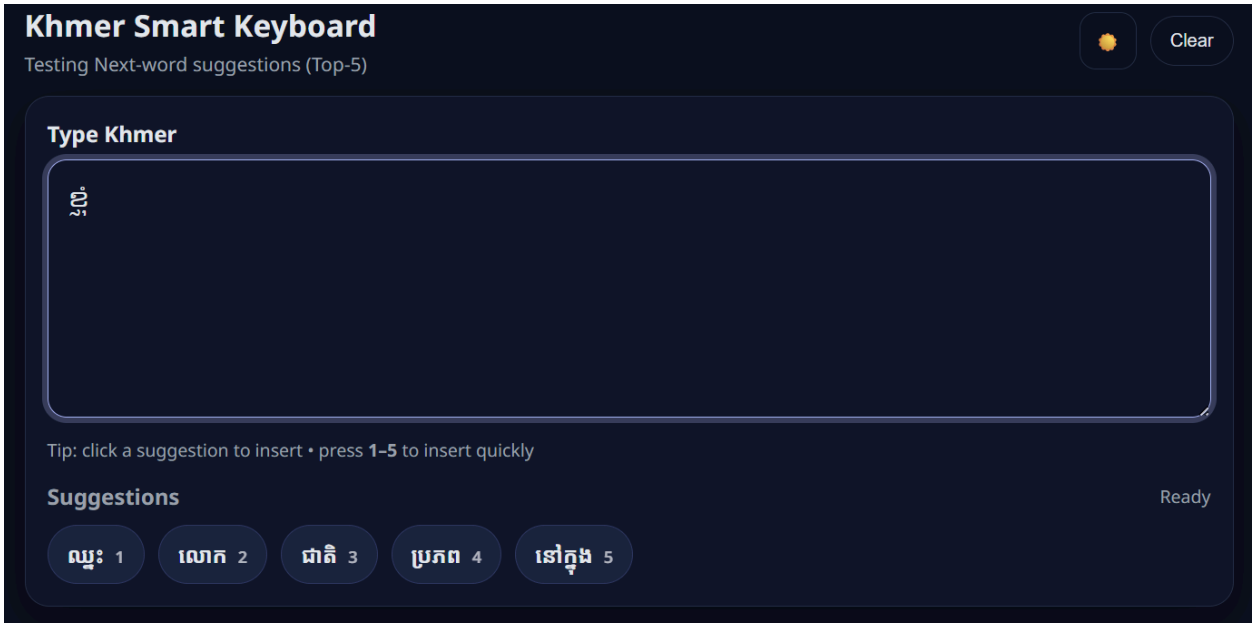
V. Keyboard Auto-Suggestion Demo

To demonstrate the practical applicability of the proposed Khmer text prediction models, a **keyboard auto-suggestion prototype** was developed. The prototype simulates a real-time text input environment where word suggestions are generated dynamically as the user types in Khmer. The system was implemented using a **web-based front-end** combined with a **FastAPI backend**, which serves the trained language model for inference. This architecture allows a clear separation between the user interface and the prediction logic, making the system modular and extensible.

Here is the overall UI:



Users can types and it suggest the possible 5 words.



VI. Conclusion

This project explored the development of a **Khmer text prediction system** by investigating and comparing multiple language modeling approaches, ranging from traditional statistical models to modern deep learning architectures. Given the linguistic challenges of the Khmer language, such as the lack of explicit word boundaries, orthographic variation, and limited language resources, the study placed strong emphasis on appropriate data preprocessing and model design.

As a baseline, **N-gram language models** (Bigram and Trigram) were implemented to analyze short-range contextual dependencies. Experimental results showed that the **Trigram model consistently outperformed the Bigram model**, achieving lower cross-entropy loss and perplexity, as well as higher Top-K accuracy. This confirms that incorporating a longer context significantly improves prediction quality for Khmer text, although N-gram models remain limited by data sparsity and their inability to generalize beyond observed word sequences.

To address these limitations, **neural network–based models** were explored, including **GRU, LSTM, and BiLSTM** architectures. These recurrent models demonstrated a substantial improvement over N-gram models, particularly in terms of **Top-5 prediction accuracy**, indicating their ability to capture sequential and contextual patterns more effectively. Among them, the **BiLSTM model achieved the best overall performance**, benefiting from bidirectional context by leveraging information from both past and future tokens during training. The BiLSTM achieved a **Top-5 test accuracy of 72.82%**, showing strong practical potential for next-word suggestion tasks despite increased computational cost.

In addition, a **Transformer-based causal language model** was implemented to evaluate the effectiveness of attention mechanisms for Khmer text prediction. The Transformer model showed stable convergence with low validation loss and perplexity, demonstrating strong capability in modeling long-range dependencies. Qualitative Top-K predictions further confirmed that the model produces contextually meaningful word suggestions beyond simple memorization.

Finally, to demonstrate real-world applicability, a **keyboard auto-suggestion prototype** was developed using a web-based interface and FastAPI backend. This demo illustrates how the trained language models can be integrated into interactive systems to provide real-time Khmer word suggestions, validating the practical relevance of the proposed approaches.

Overall, this project highlights that while **N-gram models provide a useful and interpretable baseline**, **deep learning models—particularly BiLSTM and Transformer-based approaches—are more effective for Khmer text prediction**. The results emphasize the importance of language-specific preprocessing and contextual modeling for low-resource languages. Future work may focus on larger-scale datasets, causal bidirectional architectures suitable for real-time deployment, and fine-tuning pretrained multilingual models specifically for Khmer.

VII. Reference

- [1] "What is LSTM - Long Short Term Memory?," GeeksforGeeks, Dec. 23, 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/.
- [2] "RNN vs LSTM vs GRU vs Transformers," GeeksforGeeks, Jul. 23, 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/rnn-vs-lstm-vs-gru-vs-transformers/ .
- [3] "LSTM Diagram," GeeksforGeeks, Oct. 4, 2025. [Online]. Available: https://media.geeksforgeeks.org/wp-content/uploads/20251004115807507088/1-.webp .