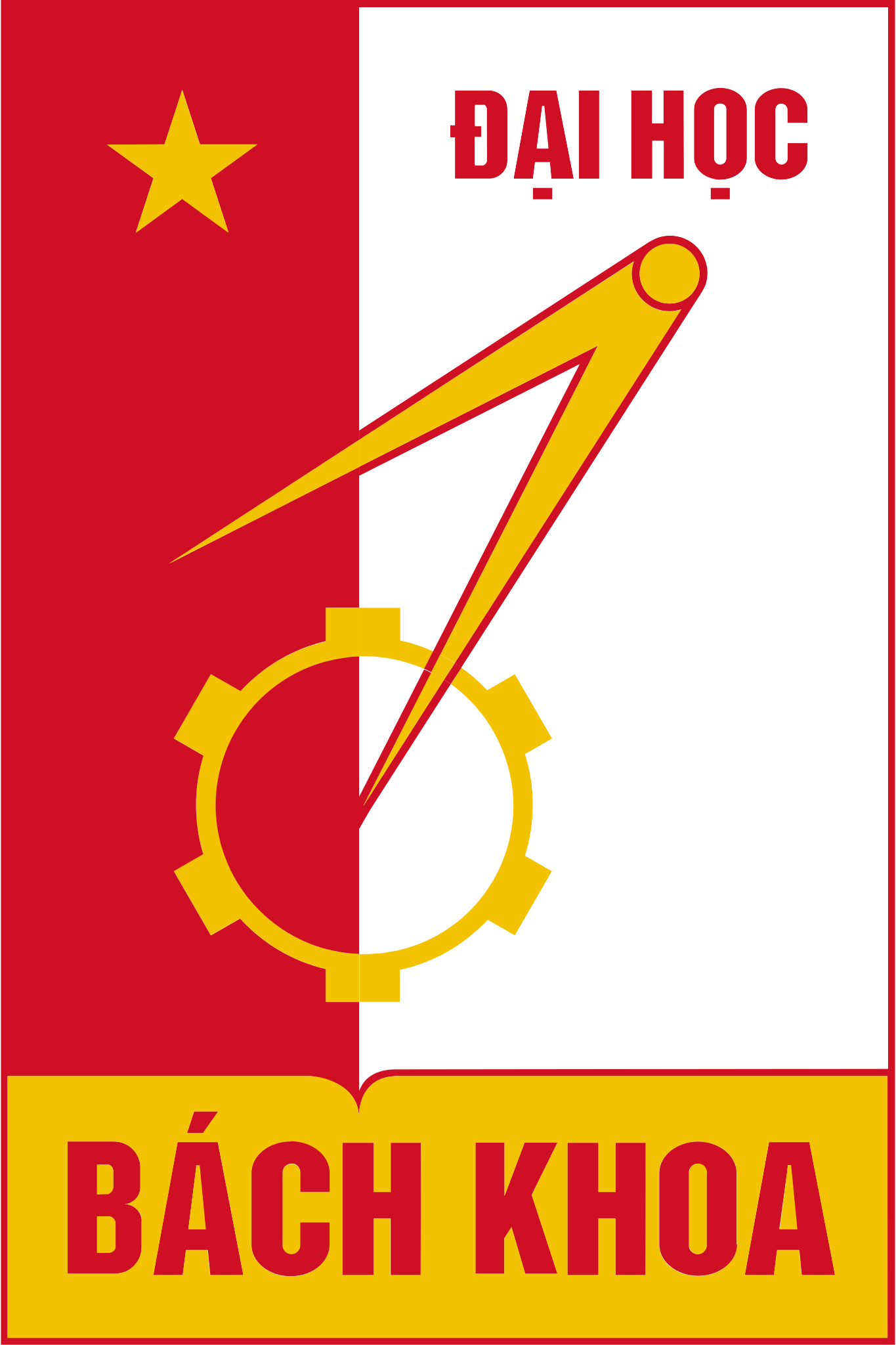
Hanoi University of Science and Technology

School of Electrical and Electronic Engineering



REPORT

NATURAL LANGUAGE PROCESSING

Project: Detection & Classification of Slang and Abbreviations

Instructor: Ph.D. Do Thi Ngoc Diep

Students:

Trương Tuấn Hưng - 20224284

Hanoi, January 2025

**Table of contents**

[**Abstract 3**](#_n3twys60rnmh)

[**1. Introduction 3**](#_2d4rydavjpp5)

[**2. Three approaches 4**](#_140u10xw8lmi)

[**2.1. Machine Learning Approaches for Detecting Abusive Bangla Comments 4**](#_y5b2on1m0dbt)

[**2.2. Machine Learning Approaches for Bengali Slang Detection 4**](#_cvltr8hp8cli)

[**2.3. Sentiment Analysis on Bengali Facebook Comments Using Machine Learning 4**](#_3u8203edew6i)

[**3. Methodology 4**](#_uee53uxswxib)

[**3.1. Dataset 4**](#_qrotw2aj525u)

[**3.1.1. Dataset collection 4**](#_erl49iu0rolc)

[**3.1.2. Ground truth 4**](#_si9s5y308tvy)

[**3.1.3. Data preprocessing 4**](#_pck741kzndom)

[**3.2. Model design 6**](#_gzkjmtbf4ihw)

[**3.2.1. Model architecture 6**](#_o6ndubjal1n0)

[**3.2.2. Optimization 7**](#_yw0jlb33s2ml)

[**3.2.3. Loss function 8**](#_81htoeblsv22)

[**3.3. Model training 8**](#_jwtik6yxe03)

[**4. Experimental Results 8**](#_rsmxhzxrydr2)

[**5. Evaluate 9**](#_1cjreh700ud)

[**5.1. Training and Validation Performance Metrics 9**](#_ly0t3d5h5k0x)

[**5.2. Precision, recall and F1-score 10**](#_ore3yvakr7m1)

[**6. Conclusion 11**](#_c1daezmtph0v)

[**7. References 11**](#_vydom5ieavcv)

**Report**

**Natural Language Processing**

**Detection & Classification of Slang and Abbreviations**

# Abstract

Vietnamese slang has become a vital aspect of everyday communication in Vietnam. Predominantly used on social media and messaging platforms, this informal style includes features such as abbreviations, slang expressions, and intentional misspellings. Its increasing prevalence, particularly among young users, poses significant challenges for automated text analysis systems due to its complex linguistic nature.

To tackle this issue, we have developed a model that integrates both character-level and word-level features to effectively capture the distinct characteristics of Vietnamese slang. The preprocessing phase involves removing special characters, converting text to lowercase, and filtering out stop words. Tokenization is then applied using padded sequences to prepare the data for classification.

For training and evaluation, we employed the dataset, which includes roughly 1,017 social media comments on Threads – a new social media of Meta. The experimental results highlight the model’s effectiveness in identifying Vietnamese slang. This approach has promising real-world applications, such as content moderation, social media analysis, and language learning tools.

# 1. Introduction

Social media platforms have seen a sharp rise in usage in recent years, giving people a means of text based communication as an expressive medium. Even while most social media users speak to each other in formal language, using slang and informal language is becoming more common, especially among younger generations. There were 5.3 billion internet users globally as of October 2023. Of them, 4.95 billion people, or 61.4% of the global population, used social media. The growing number of offensive information, cyberbullying, and online harassment are increasing day by day.

It is becoming more difficult to locate people since they are employing phony identities and anonymous accounts. But it’s best to stop them before they start and in order to achieve it, we must identify hateful remarks and speech. With the primary goal of “Vietnamese Slang Detection with AI,” there have been some earlier investigations towards identifying various forms of hate speech in Vietnamese. Some of them focused on general derogatory texts, combative remarks, and abusive Vietnamese texts on social media.

This study introduces a hybrid Bi-LSTM model to detect Vietnamese slang in transliterated text. Using cross-validation ensures reliable performance on new data. The model captures both forward and backward context, improving accuracy on the Vietnamese Slang Dataset. Results show it outperforms existing methods, advancing natural language processing and content moderation.

# 2. Three approaches

## 2.1. Machine Learning Approaches for Detecting Abusive Bangla Comments

Faruqe et al. (2023) have proposed an automatic Bangla hate speech detection system using natural language processing (NLP) and deep learning approaches. BERT, Bi-LSTM, GRU, and attention techniques have been applied to detect Bangla hate speech. The GRU and attention techniques performed best with 98.87% and 98% accuracy, respectively [5].

## 2.2. Machine Learning Approaches for Bengali Slang Detection

Hamid et al. (2023) have applied Naive Bayes, logistic regression, KNN, random forest, decision tree, SVM, and XG-Boost classification-based research to detect slang phrases automatically from given Bengali text. Logistic regression recognized slang terms with 70% accuracy. The accuracy result is comparatively low [4].

## 2.3. Sentiment Analysis on Bengali Facebook Comments Using Machine Learning

Khanet et al. (2021) used supervised learning, specifically the Support Vector Machine (SVM) classier, for sentiment analysis on Bengali Facebook comments. Other ma- chine learning techniques were also employed for comparison, including Random Forest, K-Nearest Neighbors, Naive Bayes, and neural networks. The paper achieved an accuracy of 62% for support vector machines (SVM) . Their dataset was not fully labeled. Their dataset is also small and imbalanced, which impacts their accuracy [3].

# 3. Methodology

After conducting research, we have decided to implement the method using the BiLSTM model for the project “Detection & Classification of Slang and Abbreviations”.

## 3.1. Dataset

### **3.1.1. Dataset collection**

To create the dataset for model training, we begin by collecting 1,017 comments and statuses on various topics from Thread, a Meta application similar to X. Using the *dataExtract.py* script, we extract the data from Thread and store it in a file named *Data.csv*. To enhance clarity and minimize errors during processing, the raw data in *Data.csv* is cleaned by removing icons, special characters, and other unnecessary elements. The resulting file is named *DataClean.csv*.

### **3.1.2. Ground truth**

We create the ground truth for each sentence in *Data.csv*. This involves identifying slang and abbreviations in the sentences and rewriting them in a formal way. The processed data is saved in a file named *DataTraining.csv*. To ensure accuracy and consistency, we perform additional cleaning on *DataTraining.csv* to remove any remaining special characters. The resulting file is named *DataTrainClean.csv*.

### **3.1.3. Data preprocessing**

**Tokenization**

To process Vietnamese data, we utilize Pyvi for tokenization. Tokenization is applied not only to the raw text in the “Text” column of the *DataTrainClean.csv* file but also to the content in the “Slang” and “Abbreviation” columns.

**Label**

Our dataset is annotated with four distinct labels: “slang,” “abbreviation,” “slang, abbreviation,” and “none.” The label “slang” is assigned to slang words, while “abbreviation” is used for abbreviations. For words that are both slang and written in an abbreviated form, the label “slang, abbreviation” is applied. All other words that do not fall into these categories are labeled as “none”.

We create a list containing pairs of token, label, and root sentence. Token refers to words or phrases tokenized using Pyvi. Label is determined based on the ground truth provided in *DataTrainClean.csv*. If a token matches an entry in the slang or abbreviation columns, the corresponding label is assigned as either “slang,” “abbreviation,” or “slang, abbreviation” if both categories apply. Root sentence is included to preserve the context in which each token appears, facilitating better analysis and debugging.

**Mapping words to numbers**

Mapping words to numbers is a crucial step in natural language processing. This is necessary because computers cannot directly process words or text in their raw, character-based form. In this project, the transformation also helps to standardize and streamline the analysis, ensuring that even nuanced elements like slang tokens are represented effectively for computational models.

We build a dictionary mapping words to numbers by iterating through a list of tokenized sentences, checking, and assigning a unique index to each word not already in the dictionary. The input data is combined from the columns “Text”, “Slang”, and “Abbreviations”, then tokenized using the ViTokenizer library. The result is a dictionary *{word: index}*, where words are represented by integers, and this dictionary is saved to a *word\_index.json* file for reuse, facilitating data preparation for subsequent natural language processing steps.

**Mapping labels to numbers**

Just like converting words into numbers, mapping labels to numbers is equally important in machine learning and deep learning. Models cannot process textual labels like "none", "slang", "abbreviation" or “slang, abbreviation” directly, so converting them into integers is essential for training. This transformation allows loss functions such as cross-entropy to work effectively and ensures that the data is in a consistent numerical format. Numeric labels also simplify the calculation of evaluation metrics like Precision, Recall, and F1-Score by enabling direct comparison with predicted outputs.

In step maps labels to numbers, we use a label\_map dictionary, where each label such as "none," "slang," "abbreviation," and "slang, abbreviation" is assigned a unique integer. It then iterates through the list of token-label pairs, mapping tokens to indices using the *word\_index* dictionary and converting labels to numbers using *label\_map*. The list of token IDs is padded to ensure all sequences have the same length, while the list of numeric labels is stored for training. This process prepares the data in integer format, ensuring compatibility with deep learning models.

**Label One-Hot Encoding**

We convert numeric labels into one-hot encoding by using the *to\_categorical* function, creating binary vectors to represent each label. Each vector has a length equal to the number of classes, with a value of 1 at the position corresponding to the label and 0 elsewhere. This process prepares the output data for compatibility with multi-class classification models and loss functions like *categorical\_crossentropy*.

## 3.2. Model design

The objective of the model is to process text sequences and classify them into one of four categories (“slang”, “abbreviation”, “slang, abbreviation”, “none”). The model is designed with a combination of embedding, convolutional, LSTM, and fully connected layers to learn both spatial and contextual features from the input data.

### **3.2.1. Model architecture**

**Layer 1: Embedding layer**

This is the first layer in the model, its purpose is to transform words or tokens in the data into fixed-size feature vectors. These vectors encapsulate semantic relationships, such as the similarity between words.

**Input:** The data consists of integer sequences (each number representing a word in the vocabulary).

**Output:** The data consists of embedding vectors in the form of a 2D matrix.

**Role:** Generate semantic representations of words so that subsequent layers can learn the relationships between words.

**Layer 2: Conv1D (256 filters, kernel\_size=3)**

This layer applies 1D convolutional filters to extract local features from the text sequence. Its role is to detect local features within the sequence, such as important phrases or word combinations with contextual meaning.

**Input:** The matrix from the Embedding layer.

**Output:** A new feature matrix.

**Role:** Extract local feature patterns from the sequence.

**Layer 3: Bidirectional LSTM Layer**

The Bidirectional LSTM layer captures long-term dependencies in sequences by processing data in both forward and backward directions. This layer can capture context from both past and future tokens, enhancing the model’s ability to understand the full context of the text.

**Input:** The feature matrix from the Conv1D layer.

**Output:** If *return\_sequences=True*, this layer returns the hidden sequence.

**Role:** Learn sequential context from both directions, helping capture the overall meaning of the text.

**Layer 4: Dropout Layer**

The Dropout layer randomly disables a fraction of neurons during training to prevent overfitting. This helps to ensure that the model generalizes well to new, unseen data by reducing reliance on any specific feature during the learning process.

**Input:** Data from the Bidirectional LSTM layer.

**Output:** Data with some units (neurons) randomly dropped (45% rate).

**Role:** Reduce overfitting and improve generalization.

**Layer 5: Conv1D (512 filters, kernel\_size=3)**

This convolutional layer performs additional feature extraction by applying filters that focus on different aspects of the input. The larger number of filters (512) enables the network to learn a diverse set of high-level features, allowing it to capture more complex patterns within the data.

**Input:** Data from the Dropout layer.

**Output:** A new feature matrix with 512 dimensions for each sub-sequence.

**Role:** Extract higher-level features from sequential data.

**Layer 6: Bidirectional LSTM**

The second Bidirectional LSTM layer further refines the learned representations by considering both local and global context. By processing the feature matrix from the previous Conv1D layer, it enhances the model’s understanding of the text by combining sequential information from both directions.

**Input:** The feature matrix from the Conv1D layer.

**Output:** The final hidden sequence representing the contextual understanding of the entire sequence.

**Role:** Combine local and global context to enhance the model’s semantic understanding.

**Layer 7-10: Dense and Flatten**

The Dense and Flatten layers process the final feature sequence from the LSTM layer to produce the model's output. The Flatten layer ensures that the data is transformed into a one-dimensional vector, while the Dense layer learns generalized features. The final Dense output layer classifies the data into predefined categories based on the learned features.

**Input:** The feature sequence from the final LSTM layer.

**Role:**

Flatten: Convert the output from a multi-dimensional matrix to a 1D vector.

Dense: Learn generalized features from the extracted data.

Output Dense Layer: Classify the text into 4 classes using the softmax activation function.

### **3.2.2. Optimization**

**Optimization**

Optimization in machine learning involves adjusting the parameters of a model to minimize the error or loss function during training. It is a mathematical process used to find the best configuration of the model that aligns with the training data. The optimizer defines how the model learns from data by updating parameters based on the computed gradients of the loss function. In our model, the *Adam* Optimizer is used. It is an adaptive gradient-based optimization algorithm.

**Parameters in Adam:**

Learning Rate (lr): Controls the size of the step taken in the parameter space during optimization (set to 0.0005).

Beta1 & Beta2: Coefficients used for computing running averages of gradient and its square (set to 0.9 and 0.999).

Epsilon: A small constant to prevent division by zero (set to 1e-07).

### **3.2.3. Loss function**

The loss function quantifies the difference between the model's predictions and the actual target values. It provides feedback to the optimizer about how well the model is performing and drives the weight updates. The model uses *Categorical Crossentropy* because it is a multi-class classification problem where the target is one-hot encoded. It encourages the model to output probabilities that match the true class labels closely and works well with the softmax activation function in the output layer, which ensures that the predicted probabilities sum to 1.

Formula: Loss = . log(

Where: N: Number of samples.

C: Number of classes.

: True label (0 or 1) for the i-th sample and j-th class.

: Predicted probability for the i-th sample and j-th class.

## 3.3. Model training

**Data Splitting**

Training Set (x\_train, y\_train): Represents 80% of the data, used to train the model.

Test Set (x\_test, y\_test): Represents 20% of the data, used to evaluate the model after training.

Input data:

padded\_sequences: Preprocessed text sequences, padded and converted to integer indices.

labels\_one\_hot: Labels encoded in one-hot format, suitable for multi-class classification tasks.

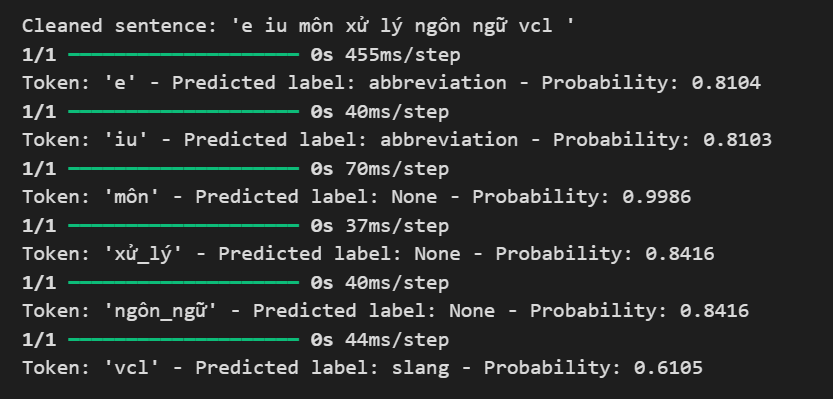
**Model training**

The training is conducted over 25 epochs with a batch size of 512, optimizing the *categorical\_crossentropy* loss using the *Adam* optimizer. During each epoch, the model learns patterns from the training data (x\_train, y\_train), evaluates performance on the test set (x\_test, y\_test), and adjusts weights through backpropagation to minimize the loss. The training process is tracked using the *history* object, which logs metrics like accuracy and loss to monitor progress and assess model performance.

# 4. Experimental Results

Input: “E iu môn xử lý ngôn ngữ vcl :)))”

Output:

****

# 5. Evaluate

## 5.1. Training and Validation Performance Metrics

**Loss Chart (Model Loss):**

The training loss steadily decreases and gradually converges to a value below 0.1, indicating that the model is learning well from the training data.

The validation loss initially decreases but fluctuates after around 5-6 epochs without significant further reduction.

Comment: Although the training loss decreases effectively, the validation loss fluctuates considerably, which may suggest that the model begins to overfit after a certain number of epochs.

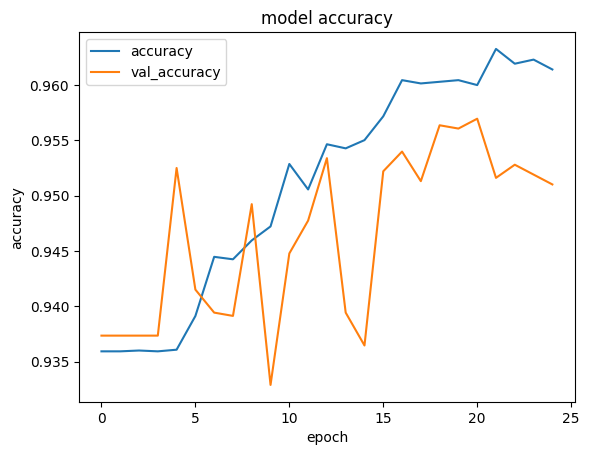
****

**Accuracy Chart (Model Accuracy):**

The training accuracy gradually increases and reaches nearly 0.96 after 25 epochs.

The validation accuracy shows significant fluctuations, does not increase consistently, and seems to peak around the middle (approximately 0.955) before slightly declining in later epochs.

Comment: The fluctuations in validation accuracy and its lack of consistent improvement indicate that the model may face generalization issues on the test dataset.

****

**Overall Evaluation:**

The model performs well on the training data, but its effectiveness on the test data is unstable due to overfitting.

## 5.2. Precision, recall and F1-score

**Precision (0.62):**

Precision =

Precision is at a moderate level, indicating that the model performs relatively well in correctly predicting positive samples. However, there is still a significant proportion of false positives.

**Recall (0.69):**

Recall =

Recall is higher than Precision, meaning that the model has detected most of the actual positive samples. However, it still misses some positive instances.

**F1-Score (0.63):**

F1-Score = 2

F1-Score, which is the harmonic mean of Precision and Recall, is 0.63, reflecting a reasonable balance between the two metrics. This shows that the model performs adequately, but there is room for improvement.

**Overall Assessment:**

The model has a fairly good performance but could be further improved to achieve a better balance between Precision and Recall.

Precision < Recall indicates that the model tends to prioritize detecting more positive samples over avoiding false positives, resulting in a higher false positive rate

# 6. Conclusion

In this study, we have developed a Bi-LSTM model to detect Vietnamese slang and abbreviations in transliterated text, addressing the complex and informal nature of social media communication. Our approach integrates both character-level and word-level features, showing improved performance in capturing the unique characteristics of Vietnamese slang. The dataset, collected from Threads – a social media platform by Meta, provided a challenging environment with 1,017 comments.

While the model demonstrated promising performance, achieving a reasonable F1-score of 0.63, there are areas for improvement, particularly in balancing precision and recall. The validation accuracy exhibited some fluctuations, suggesting overfitting, which could impact generalization on unseen data. Despite this, the results highlight the model’s potential for real-world applications such as content moderation, social media analysis, and language learning tools.

# 7. References

[1] [Github Bangla\_Text\_Classification\_Utilizing\_CNN\_with\_Bi-LSTM](https://github.com/OviSarkar62/Bangla_Text_Classification_Utilizing_CNN_with_Bi-LSTM/blob/main/text-classification-with-cnn-lstm.ipynb)

[2] [Github banglish-slang-detection](https://github.com/Fiazul/banglish-slang-detection/blob/main/slangbanglish.ipynb)

[3] [Article Detection of Slang Words in e-Data using semi-Supervised Learning](https://www.researchgate.net/publication/273011760_Detection_of_Slang_Words_in_e-Data_using_semi-Supervised_Learning)

[4] [Article Slang detection and identification](https://aclanthology.org/K19-1082.pdf)

[5] [Article Detection of Banglish Slang in Social Media Comments Using a Hybrid Bidirectional Long ShortTerm Memory (Bi-LSTM) Model](https://assets-eu.researchsquare.com/files/rs-4109962/v1/32522d08-b1b3-4eef-bf49-027371ceaca0.pdf?c=1710895952)

[6] [Article Exploring Deep Learning for Chittagonian SlangDetection in Social Media Texts](https://www.researchgate.net/publication/384738184_Exploring_Deep_Learning_for_Chittagonian_Slang_Detection_in_Social_Media_Texts#fullTextFileContent)