VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

GRADUATION THESIS

CONTEXTUAL VIETNAMESE SPELLING CORRECTION

Major: COMPUTER SCIENCE

THESIS COMMITTEE:	COMPUTER SCIENCE 1
SUPERVISOR:	ASSOC PROF. QUẢN THÀNH THƠ
	MS. BĂNG NGỌC BẢO TÂM
REVIEWER:	DR. NGUYỄN HỨA PHÙNG

—o0o—

STUDENT 1:	ĐỖ ANH KHOA	1852471
STUDENT 2:	NGUYỄN TRẦN HIẾU	1852370

HO CHI MINH CITY, 05/2022

Declaration Of Authenticity

We declare that this research is our own work, conducted under the supervision and guidance of Assoc. Prof. Quan Thanh Tho. The result of our research is legitimate and has not been published in any form prior to this. All materials used in this research are collected from various sources and are appropriately listed in the references section.

In addition, within this research, we also used the results of several other authors and organizations. They have all been aptly referenced.

In any case of plagiarism, I stand by my actions and will be responsible for it. Ho Chi Minh City University of Technology, therefore, is not responsible for any copyright infringements conducted within our research.

Ho Chi Minh City, May, 2022
Author
Do Anh Khoa
Nguyen Tran Hieu

Acknowledgment

We are using this opportunity to express our gratitude to everyone who supported us during the making of our thesis. We are thankful for the invaluable knowledge, experience sharing received that had helped us improve.

We would like to show our gratitude to our mentor and supervisor throughout our thesis, Associate Professor Ph.D. Quan Thanh Tho. Throughout the making of this thesis, he has guided us with full support and constructive criticism.

We thank the Ho Chi Minh University of Technology for providing us the many great invaluable knowledge and lessons within four years of attending the school. Without attending the school, we would not have the chance to be friends, and this thesis would not be possible.

Finally, our motivation to pursue this thesis comes from the support of our families and friends. We would like to show our deepest and most sincere gratitude to our parents, who sacrifice and support, and were with us in our life journey; our friends and siblings, whose encouragement and advice we took to heart along the way.

Abstract

Spelling errors are common in everyday life, from writing emails to editing documents. The task consists of detecting the error and correcting it to suggest a suitable replacement. There are two characteristics of spelling errors: non-word and real-word errors. Non-word error is when the error itself is not in the dictionary and is not a known word; for example, ”tôi” is mistyped to ”toooi”. Real-word error is due to misplacing a word to another word in the dictionary; for example, ”học hành” is mistyped to ”học hanh”, a diacritic error. Errors with real-word characteristics are harder to detect, as we have to rely on the surrounding context to perform reasoning and deduce the correct token. Moreover, spelling errors are increasingly difficult when multiple tokens are merged; for instance, ”học hành” when removing space becomes ”họchanfh”, or when a token is split into sub-tokens like ”tôi” to ”t ôi”.

Historically, rule-based methods are used to correct misspellings, which do not capture misplaced tokens and handled more complex misspelling cases, leading to studies of statistical models like N-gram. These models are trained on a large corpus and can capture the surrounding text’s information. Nonetheless, a major setback is when two (or more) spelling mistakes stand near each other, the model will not be able to understand and capture the context correctly, which has been studied in [1]. Recently, two research groups from Vietnam National University [1] and Zalo Corporation [2] proposed contextual spelling correction using a deep-learning-based approach. In this paper, we study the effect of leveraging a pre-trained Transformer Encoder such as PhoBERT for Contextual Vietnamese Spelling Correction. We also adapted a tokenization repair module for Vietnamese, which handle merged syllables such as ”ănuoongs” and split syllable such as ”t ôi”, as our main model can only correct on syllable level. Finally, we perform a comparative study with different baselines, including state-of-the-art models for Chinese Spelling Correction for reference.

Our approach shows some promising results for the spelling correction task and can handle error cases we included, although we notice a few setbacks in real-world trials. We also tested our model against two public tests set by the Vietnamese National University and Zalo Corporations achieve better results on the former and worse outcomes on the latter, accordingly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

We as humans will inevitably make mistakes during typing, whether it is accidental keystrokes or misunderstanding of its meaning. As we need our emails and document to be error-free and presentable, spelling correction is vital and has a broad application value to our life. Although this problem has been extensively researched in English [3, 4, 5] and Chinese [6, 7], it is still relatively new to Vietnamese.

Previous studies approach the problem using rule-based methods, which work for simple structured cased. However, it often fails to correct more complicated misspelling cases like close-proximity keystrokes and cannot detect real-word cases where words have no visible error due to the method not utilizing context. A statistical model approach like n-gram [8, 9] yields a more promising result, capturing the surrounding information to deduce the correct answer. Nonetheless, it is fragile to consecutive spelling errors, which can break the context and reduce accuracy. In November 2021, the Zalo research team [2] released a paper on spelling correction tasks by proposing a hierarchical transformer encoder, combining character and word-level representation and promising results. At the same time, the Vietnamese National University [1] released a different approach based on the machine translation approach, translating an erroneous sentence into an error-free sentence.

Spelling correction has not been widely studied in Vietnamese compared to Chinese and English, although its application is essential. Search engines can utilize spelling correction systems to improve user experience in real applications. Many word processing software is working on building spelling correction systems for day-to-day usages, such as Microsoft Word, Google Docs, or third-party integration like Grammarly for English. It can also be generalized to error correction tasks, which can be helpful in other fields such as Optical Character Recognition, which studies the textual extraction of images, or Automatic Speech Recognition.

1.2 Problem Statement

Given an input text sentence, where each word may or may not contain spelling errors, our goal is to predict the correct sentence that minimizes the number of spelling errors. Spelling errors may fall into two main characteristics: non-word and real-word. Non-word errors, also known as a misspelling ("tôi" to "toooi", "học" to "họv"), are errors where the resulting tokens are out-of-vocabulary. Real-word errors, also known as misplaced words (”Dang rộng” to ”Giang rộng”), are errors where the resulting tokens occur in the vocabulary; however, they are used in the wrong context.

1.3 Goals

In this thesis, we aim to achieve several outcomes:

- Performing a survey on Vietnamese spelling errors and previous study that tackles this problem.
- Study the effectiveness of leveraging a pre-trained Transformer Encoder such as PhoBERT for the Contextual Vietnamese Spelling Correction task.
- Enhance our correction model with a tokenization repair module to handle merged syllables and split tokens cases.
- Perform a comparative study between different deep-learning approaches, including state-of-the-art Soft-masked BERT[6] from Chinese Spelling Correction.
- Evaluate our approach against published testing sets from other researches.

1.4 Scope

Per the scope of our thesis, we will focus only on Vietnamese spelling errors. This excludes foreign language correction, shortcuts, and teen-codes which we believed to be an intentional input.

1.5 Thesis Structure

There are totally seven chapters in this thesis:

Chapter	Content
1	A brief introduction about plan and objectives of thesis
2	Introduction of theoretical background as foundation knowledge that are applied in the project
3	Related works have been done on this problem
4	Solution and design approach for problem statement of project
5	Results and test set evaluation
6	Discussion and future works
7	Thesis summary

Table 1.1: Thesis structure

Chapter 2

Theoretical Background

2.1 N-Grams

An n-gram is a sequence of n words or tokens: a 2-gram (bi-gram) is a two-word sequence, and a 3-gram (tri-gram) is a three-word sequence, and so forth. A probabilistic model could assign a probability of the last word of n-gram given the previous words and assign probabilities to the whole sequence, which is called the **language model**.

To compute the probability of an entire sequence with m words $P(w_1, w_2, ..., w_m)$, traditionally, the chain rule is applied:

$$\begin{aligned} P(w_1, w_2, ..., w_m) &= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)...P(w_m|w_1...w_m) \\ &= \prod_{i=1}^{m} P(w_i|w_1...w_{i-1}) \end{aligned}$$

The intuition of the n-gram model is that the probability of a word is calculated as a conditional probability given its last n words rather than its whole history. For instance, a bi-gram model could approximate the probability of a word given its previous word.

$$P(w_m|w_1...w_m) \approx P(w_m|w_{m-1})$$

This approximate assumption is called a **Markov** assumption. From that assumption, we could generalize n-gram by looking at n - 1 previous word. Thus, the general equation for n-gram approximation to the conditional probability of the next word in a sequence is:

$$P(w_m|w_1...w_{m-1}) \approx P(w_m|w_{m-n+1}...w_{m-1})$$

Hence, the probability of the entire word sequence is for bi-gram is:

$$P(w_1...w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-1})$$

Maximum likelihood estimation (MLE) is used for estimating these probabilities. We get an MLE estimate for parameters of the n-gram model by getting counts from a corpus and then normalizing those counts to lie between 0 and 1.

$$P(w_m|w_{m-n+1}...w_{n-1}) = \frac{C(w_{m-n+1}...w_{m-1}w_n)}{C(w_{m-n+1}...w_m)}$$

The estimation of n-gram probability is computed by dividing the observed frequency of a particular sequence by the observed frequency of a prefix; this ratio is called relative frequency.

2.2 Minimum Edit distance

In computational linguistics and computer science, edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other. There are several algorithms that have been used to solve this problem, though Levenshtein Distance (V. Levenshstein, 1965) is more widely used in the field for error correction.

Levenshtein distance is a number that tells you how different two strings are, the higher the number, the more different the two strings are. The distance is measured by a set of three operations on single-character edits: insertion, deletion and substitution. For instance, the string ”kitten” and ”sitting” is 3 edit distance away due to:

- ”$\mathbf{k}itten$” $\rightarrow$ ”$\mathbf{s}itten$”: substitution of ”s” for ”k”
- ”$sitt\mathbf{e}n$ $\rightarrow$ ”$sitt\mathbf{i}n$”: substitution of ”e” for ”i”
- ”$sittin$ $\rightarrow$ ”$sittin\mathbf{g}$”: insertion of ”g”

Given two string $\mathbf{a}, \mathbf{b}$ of length $|a|$ and $|b|$, then $\mathbf{lev}(\mathbf{a}, \mathbf{b})$ is calculated by the following function:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

where i, j are character position of string a and b, $1_{(a_i \neq b_j)}$ denotes 0 when $|a| = |b|$ and 1 otherwise.

Although using minimum edit distances would eliminate several options for choosing the appropriate word, there could still be a possibility where there are multiple words with the same edit distance values. Therefore, using the Levenshtein similarity ratio could be helpful in such cases:

$$ratio = \frac{(|a|+|b|) - lev_{a,b}(i,j)}{|a|+|b|}$$

where $|a|$ and $|b|$ are the length of sequences a and b, respectively.

2.3 Neural networks

Neural network is the basic concept of the field of Deep learning in general. Originally inspired by biological neural networks, this mathematical model mimics the interconnected network of neurons transmitting elaborate patterns of signals. The image below illustrates a simple neural network.

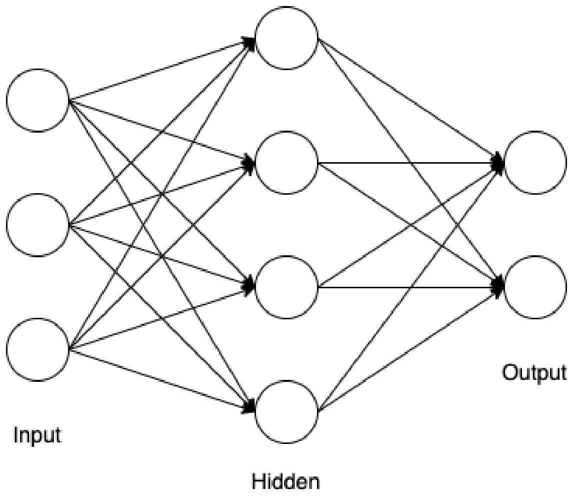


Figure 2.1: A simple neural network

A neural network is composed of a large number of different types of neurons. The image illustrates three types: input layers consist of 3 neurons, hidden layers consist of 4 neurons, and output layer consists of 2 neurons. The hidden neurons, or nodes, are where the computation is done and transfer the signal to the output layer. Neurons transmit their signal to others through a link called connection. In a sense, a neuron *i* ”transmits” its signal to another neuron *j* through the connection, and each connection holds a weight w_{ij} value that decides how much of the signal is transferred.

In the deep learning field, neural networks have been applied to various tasks. Natural language processing tasks such as text classification and tokens classification (part-of-speech tagging, name-entity recognition) witness a boost in performance by training different neural architecture. Training in deep learning, in a sense, is adjusting the weight and bias values to produce the desired output.

2.3.1 Feed-forward neural networks

A feed-forward network is an artificial network where the connections within the network do not form a cycle. Therefore, this type of network allows information to move in only one forward direction, from the input node and ends at the output nodes.

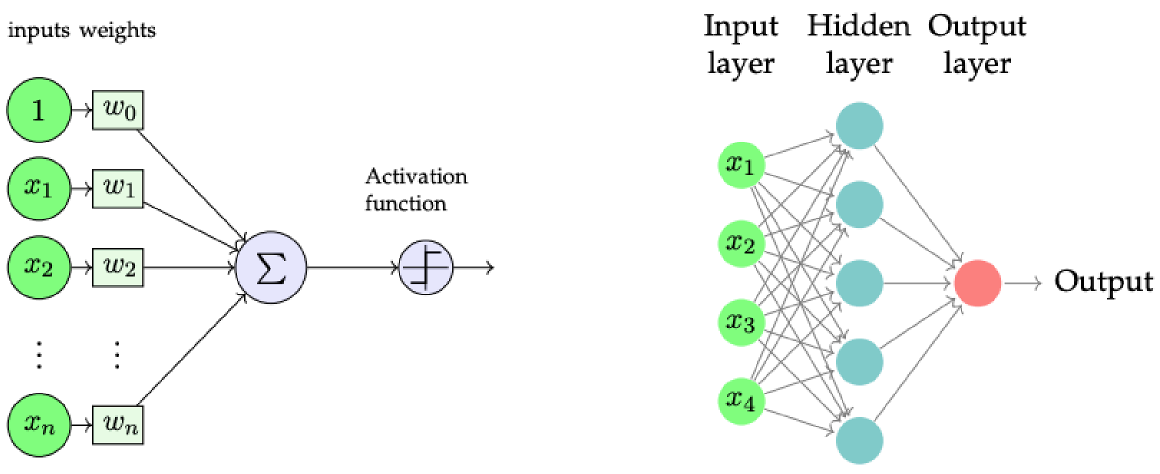


Figure 2.2: Single-layer Perceptron vs Multi-layer Perceptron

Single-layer Perception
The single-layer network does not contain any hidden layers. It consists of an input layer and an output layer of nodes. In neural nets, we do not account for the input layer as it only receives input fed from the outside, while no computation is done. Let us assume $\mathbf{x} = (x_1, x_2, .., x_n)$ as a vector of dimension n to be fed into the network. Then, output z, the sum weight of inputs have the formula:

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \mathbf{x} + \mathbf{b}$$

where $\mathbf{w} = (w_1, w_2, .., w_n)$ is a vector containing weight values. Hence, $\mathbf{w^T}$ is the transpose of $\mathbf{w}$, $\mathbf{b}$ is the bias value. This linear transformation z is then passed through an activation function to produce output.

Multi-layer perceptron
An extension to single-layer perceptron, this class of network contains more than one layer for more complex computation, still in a direct way. Each node in a layer has at least one connection to a node in the subsequent layer. These networks may contain more than one hidden layer, which allows them to learn the non-linear representation of the data presented.

2.3.2 Recurrent neural networks

Sequential modeling is the study of generating a sequence of values by analyzing a series of input values. The ability to handle and process sequences are important in the field of Natural Language Processing; for example, given a text (sequence of words or characters), we want to analyze the sentiment values associated with it. With the traditional feed-forward model, we cannot represent data with various timestamps to learn effectively.

Definition
A recurrent neural network (RNN) is a type of neural network that can handle sequential data effectively. An extension of the traditional neural network possessed by RNN is the cyclic connection that each unit can have with each other; while a simple ANN network learns to produce an output from a given input, RNN can utilize the inputs and outputs of all previous units, thus

providing memory-like internal states for the network.

Let us define a sequence of inputs $\mathbf{X} = (x_1, x_2, ..., x_n)$ that is compressed into a fixed dimension vector. This allows us to deal with variable-length input and outputs. At time step t, we have a vector h_{t-1} that contains the hidden state value up to $t-1$ time step. The recurrent neural network will compute the new internal state h_t from the previous internal state and current input as follow:

$$h_t = activation(W_h x_t + U_h h_{t-1} + b_h)$$

$$\hat{y}_t = softmax(W_y h_t + b_y)$$

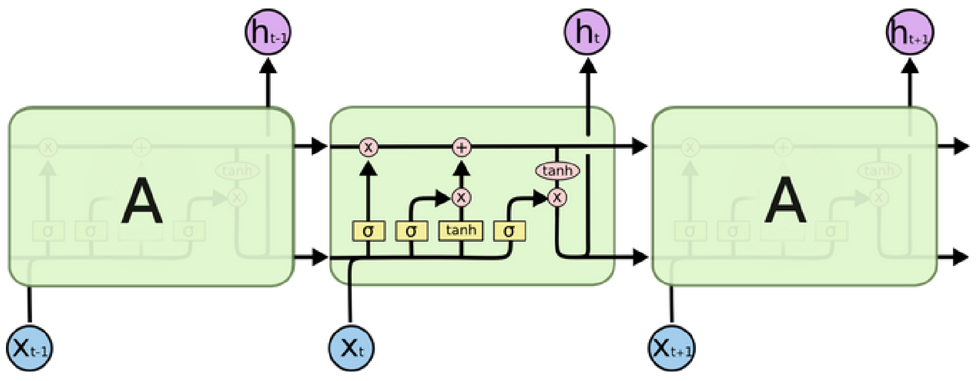
where W_h is the weight matrix of input, U_h is the recurrent weight matrix, and b_h is the bias values. After we obtain the hidden value for the current time step, we calculate the output y_t. W_y refers the the weight matrix of output layer, and b_y is the bias value to the output layer. After that, we can take the softmax function to retrieve the output. Training a recurrent neural network refers to adjusting the parameters in these equations.

In RNNs, we encounter a phenomenon called *vanishing gradient*. The multiplicative gradient can be exponentially increasing or decreasing, hence RNNs will suffer from capturing long-term dependencies. This is a major problem as long-term dependencies are one of the main characteristics of language.

Efforts have been made to confront the vanishing gradient problem. Most notably, to architecture called Gated Recurrent Unit (GRU) (Cho et al.) [10] or Long Short-term memory (LSTM) (Hochreiter et al.) [11] architecture. Within the scope of the thesis, we chose the latter to implement, which we will discuss in the next section.

Long short-term memory

Long Short-Term Memory (LSTM) (Horeichter et al.) [11] network is an architecture of Recurrent Neural Network, as it is used to resolve long-term dependencies as stated above. Today, it is one of the most widely used models in Deep Learning for NLP.



The repeating module in an LSTM contains four interacting layers.

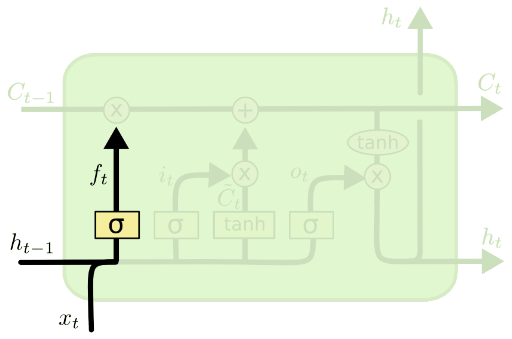
Figure 2.3: LSTM architecture

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The heart of LSTM network is the cell state and gate units that control the internal states. The gate units consist of input gate, forget gate, and output gate.

Forget gate

The forget gate mechanism allows LSTM to decide the information that will be kept or discarded when passed through this time step. It uses a sigmoid function layer called the ”forget gate layer”. The input is a combination of current time step x_t and previously hidden state h_{t-1}, and the sigmoid function returns a value ranging from 0 to 1, which resembles how much of the information will be retained. W_f refers to the weight matrix for the forget gate, and b_f refers to the bias for the forget gate. In this case, 1 stands for completely retaining information, while 0 stands for the opposite.



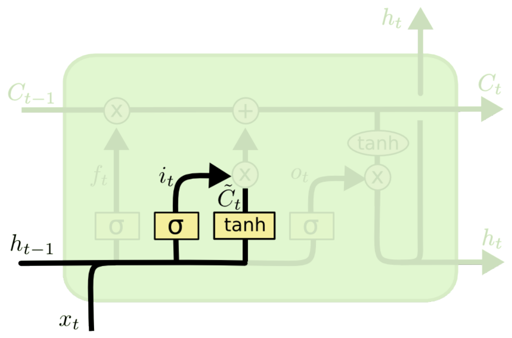
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 2.4: LSTM forget gate

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Input gate

To make the decision on what information will be stored in the cell state, the input gate passes input through two different activation functions at each timestep x_t. Firstly, a sigmoid layer decide which value will be updated, which also outputs a value from 0 to 1. In this input layer, W_i is the weight matrix, b_i is the bias value. At the same time, the tanh layer creates a vector of new candidate values to be added to the state, where W_C and b_C stand for the weight matrix and bias, respectively.

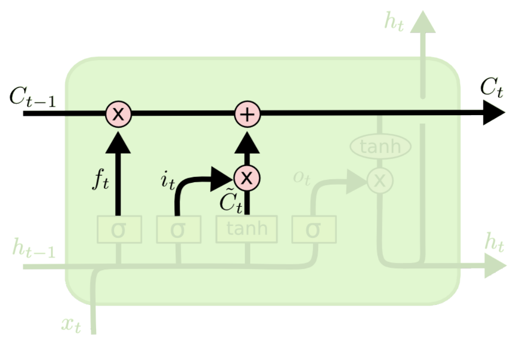


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 2.5: LSTM input gate

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Cell state
To transform the old cell state C_{t-1} to the new cell state, we firstly perform "forgetting" the input we decided at the forget gate step f_t, through a point-wise multiplication operation. Then, we add on the new information from the input gate i_t to achieve the current cell state $\tilde{C}_t$ through a point-wise addition operation.

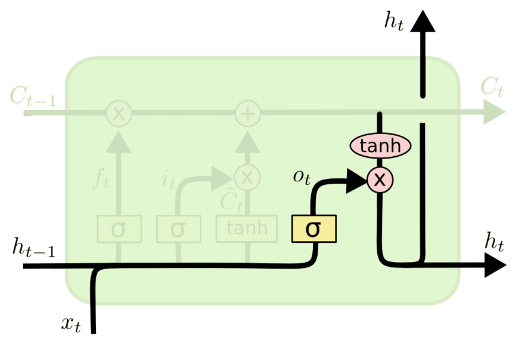


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.6: LSTM cell state

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Output gate
Finally, we achieve the output through the output gate, which takes in the previous hidden state h_{t-1} and current time step input x_t, and the cell state. Through the sigmoid layer with W_o as weight matrix and b_o as bias value, LSTM decides the part of the cell state to be the output o_t. Then combine it with the cell state C_t that has been through tanh activation and produce hidden representation h_t through point-wise multiplication.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.7: LSTM output gate

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bidirectional Recurrent Neural network
The previous discussion focuses on RNNs, that are able to observe past input and hidden states in order to learn and predict the next symbol in a data sequence. However, for many sequence

labeling tasks, we would also like to have access to future context. To achieve this, we can use two RNN models that read the input sequence in two ways: right-to-left propagation and left-to-right propagation. This concept is commonly referred to as Bidirectional Recurrent Neural networks (BRNNs) (Schuster et al.) [12].

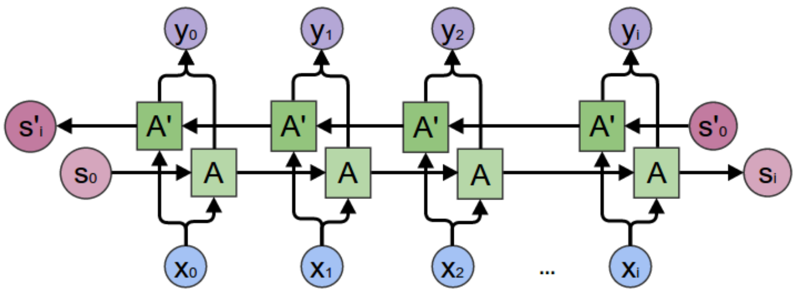


Figure 2.8: Bidirectional Recurrent Neural Network

For Bi-RNN, the right-to-left propagation is the same as the traditional RNN, while the left-to-right propagation runs in the opposite direction. As a result, the input sequence is forwarded to 2 different hidden layers for computation.

$$\overrightarrow{h_t} = tanh(\overrightarrow{W}x_t + \overrightarrow{U}\overrightarrow{h_{t-1}}) + \overrightarrow{b}$$

$$\overleftarrow{h_t} = tanh(\overleftarrow{W}x_t + \overleftarrow{U}\overleftarrow{h_{t+1}}) + \overleftarrow{b}$$

Note how the calculation of right-to-left propagation hidden state depends on the hidden state of future time step. We will concatenate the hidden state of both right-to-left $\overleftarrow{h_t}$ and left-to-right $\overrightarrow{h_t}$ to obtain the hidden state h_t that will forward to the output layer.

Bidirectional recurrent neural networks have improved results in various domains [13, 14].

2.3.3 Sequence-To-Sequence Architecture

Sequence-to-Sequence (Sutskever et al.) [15] is a Deep Neural Networks model to generate an output sequence from an input sequence whose length may be different from that of output. Although the model was initially proposed to tackle Machine Translation problem, Sequence-to-Sequence has also been applied in numerous other systems such as Speech recognition, Text summarization, etc.

The model comprises 2 main parts: Encoder and Decoder, both are recurrent neural networks that are connected through a context vector.

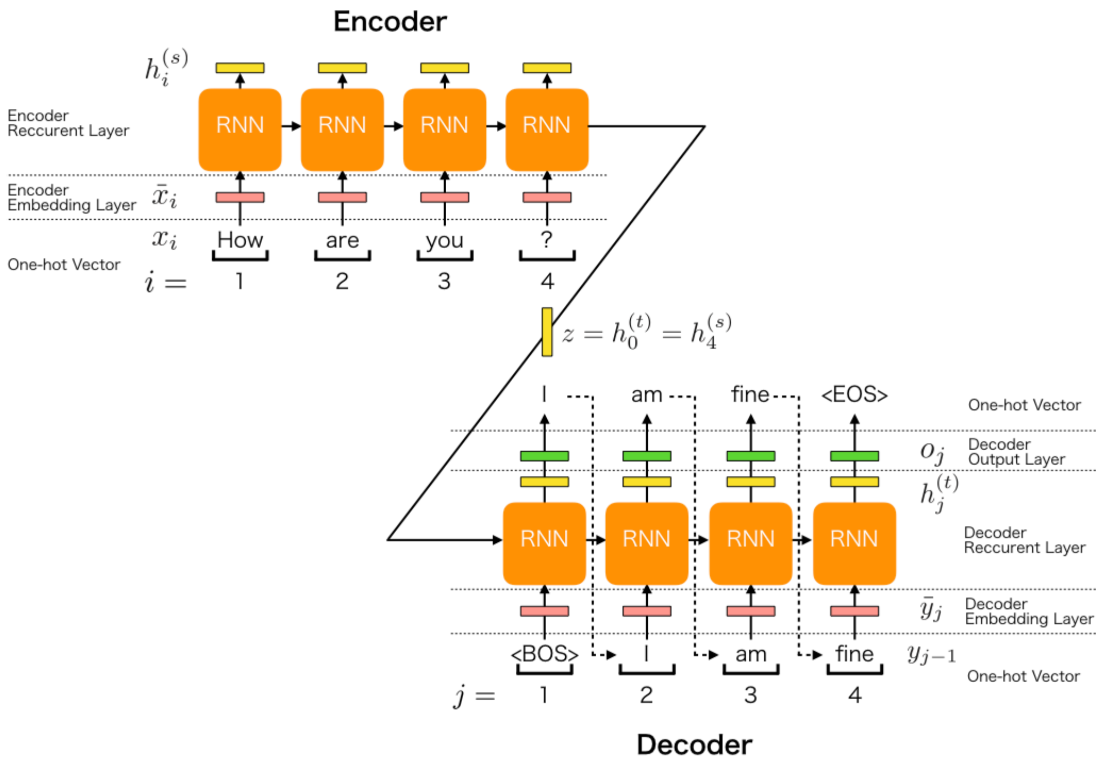


Figure 2.9: An overview of Sequence to Sequence architecture (Tokui et al.) [16]

Encoder
This encoder is a stack of recurrent units (which could be LSTM (Long short-term memory) or GRU (Gated recurrent unit) to avoid vanishing gradient). It is fed in overtime an input sequence $\mathbf{x} =< x_1, x_2, ..., x_n >$ with x_t is the t^{th} token of the sequence, processes each token of the input and tries to encapsulate all the information from the input sentence and store them in its final internal state, which is hidden state h_t calculated by the following formula:

$$h_t = f(x_t, h_{t-1}) = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

These final hidden states, called *context vector*, are then passed onto the decoder part and act as an initial hidden state for the decoder block to produce the target sentence.

In order to boost the performance, undirectional RNNs are replaced with the bidirectional due to their capabilities of capturing both the forward and backward direction of the sequence.

Decoder
While the encoder packs all the information and encoded the sequence into a context vector, the decoder is used to decode that vector and generate the output sequence. The same structure is applied to the decoder block. A stack of several recurrent units that produce an output y_t for each time step t. Each recurrent unit accepts a hidden state from the previous unit, and also produces and outputs its own hidden state. These hidden states are calculated by:

$$h_t = f(h_t) = f(W^{(hh)}h_{t-1})$$

A fully-connected layer is added as the output layer with the purpose of finding the probability vector to determine the final output along with the time steps by softmax.

$$y_t = softmax(W^s h_t)$$

Finally, the loss is calculated on the predicted outputs and then the errors are back-propagated to update the parameters. And through training over a period of time with sufficiently large data, the model could provide predictions with good performance.

2.3.4 Tokenization

Tokenization is a process of dividing a sequence into smaller units, called tokens, which then can be fed into a model. By tokenizing an unstructured stream of text, we achieve meaningful representations that are more useful for our model to learn. Choosing the right tokenization method is important to the spelling correction task. We present some tokenization methods commonly used in natural language processing tasks.

- **Word level tokenization**: In this method, sentences are split up into lists of tokens separated by white space. Each token will be monosyllabic in Vietnamese per the nature of the language. This technique is commonly used in word embedding methods like GloVe, and word2vec.

- **Multi-word level tokenization**: Language like Vietnamese uses a composite number of monosyllabic tokens to convey an actual word. For example, ”xác định” is composed of 2 syllables ”xác” and ”định”, which have no relation to the composite word in meaning. Multi-word levels are useful to convey the meaning of the word, hence used in NLP tasks such as part-of-speech tagging, and named entity recognition. Some open-source libraries provide support for multi-word tokenization such as underthesea, pyvivn,...

- **Character level tokenization**: Word and above usually suffer from a large vocabulary size, which is especially true for spelling correction tasks where one syllable can have multiple different types of errors. Two problems arise: a large vocabulary size causes model to train and converge slower, and out-of-vocabulary problems for words that the model has never seen. Character tokens are atomic as characters are the basic unit that composes a word, hence a smaller set of tokens in vocabulary can fully cover a wide range of words, reducing out-of-vocabulary tokens. A word embedding method known as fasttext utilize character tokenization.

- **Subword tokenization**: Although character tokenization addresses the problem of word-level tokenization, it is not without limitations. Dividing a sentence to character level greatly increase sequence length, which leads to slow training and inference time. Another problem is character level method cannot convey word context as well as word level. Subword tokenization inherits both word and character level embedding. The idea is to observe the common patterns of multiple characters from words to form the subword units. For example, ”faster” may be decomposed to ”fast”, ”er”.

  Subword tokenization is commonly referred to as inbetween word and character level and is used in many modern models in transformer-based architecture like BERT (Devlin et al.) [17], or its variations. A popular algorithm used to train subword tokenizer is Byte pair encoding [18, 19]. The byte pair encoding algorithm works as follows:

- Step 1: Sentences are split into word level. Each word is appended a special token $\backslash w$ to mark the end of a word.
- Step 2: Split all words into their character components and add them to vocabulary. At this initial step, vocab only contains character tokens.
- Step 3: Count all occurrence of token pair that appears in the corpus, and add frequent pairs to the vocabulary as a new token.
- Step 4: Repeat step 3 until desired vocab size is reached or several merge operation is performed.

Training BPE is unsupervised and the only hyperparameter we can tune is vocab size or the number of merge operations.

2.4 Transformer Architecture

The paper ”Attention is all you need” (Vaswani et al) [20], one of the most impactful paper in 2017, introduce the transformer architecture, which fully relies on attention mechanism, instead of the recurrent mechanism of traditional method like Recurrent Neural Network, Long Short-Term Memory,... To understand the proposed model, we first discuss attention.

2.4.1 Attention

Sequence-to-sequence models work great for machine translation tasks. However, its disadvantage is based on the fact that the neural network will compress all information from the sentence to a fixed-length vector. As Cho et al. [21] pointed out that the performance of a basic sequence-to-sequence deteriorates proportionally to the sequence length. Bahdanau et al. [22] proposed the attention mechanism, an extension of encoder-decoder architecture to address this problem.

At an abstract level, attention extension does not build a single context vector out of the encoder’s last hidden state. Instead, what makes the attention mechanism special is that it creates shortcuts between the context vectors and the entire source input. After that, the decoder can soft-search for the set of positions in the source sentence for the most relevant information. This is done by assigning customizable weighted sum of context vectors.

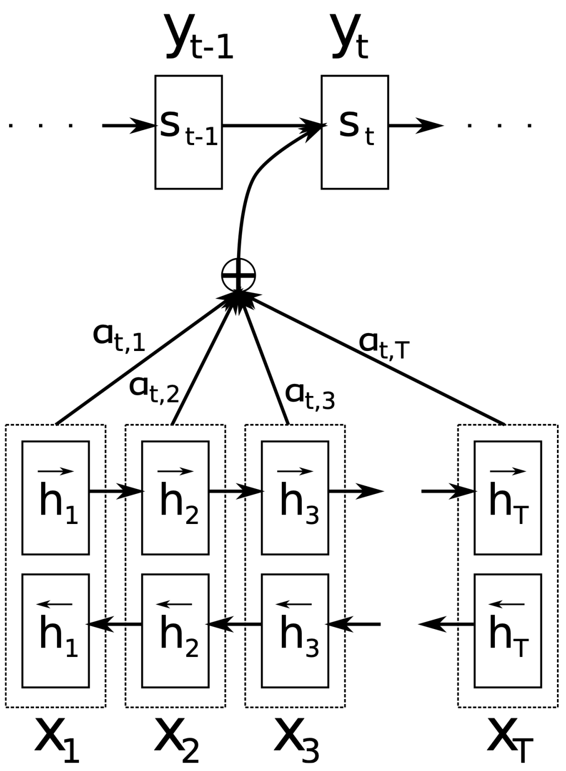


Figure 2.10: Sequence to sequence model with Bahdanau attention mechanism.
Image source: Bahdanau et al. [22]

Lets define the problem of machine translation with input source sequence $\mathbf{x} = (x_1, x_2, ..., x_n)$ with length n, and output target sequence $\mathbf{y} = (y_1, y_2, ..., y_m)$ with length m.
The encoder (which is a bidirectional RNN), contains the forward hidden states $\overrightarrow{\mathbf{h_i}}$ and backward hidden state $\overleftarrow{\mathbf{h_i}}$. The overall representation is denoted as

$$\mathbf{h_i} = [\overrightarrow{\mathbf{h_i}}^T, \overleftarrow{\mathbf{h_i}}^T]^T, i = 1, 2, .., n$$

In the proposed model, decoder has hidden state $\mathbf{s}_i = f(\mathbf{s}_{i-1}, y_{i-1}\mathbf{c}_i)$. $\mathbf{c}_i$ denotes for the context vector at position i, which is calculated by the weighted sum of hidden state of the input sequence x. The weight is an alignment score, calculate as below:

$$\mathbf{c}_t = \sum_{j=1}^{n} \alpha_{t,j}\mathbf{h}_j$$

$$\alpha_{t,j} = \frac{exp(score(s_{t-1}, h_t))}{\sum_{k=1}^{n} exp(score(s_{t-1}, h_k))}$$

Here, the alignment model will score how matching the input at position j and output at position t match. In the original paper, the alignment model will be parameterized as a feed-forward neural network which will be jointly learned with the model.

$$score(\mathbf{s}_t, \mathbf{h}_j) = \mathbf{v}_a^T tanh(\mathbf{W}_a[\mathbf{s}_t; \mathbf{h}_j])$$

where $\mathbf{v}$ and $\mathbf{W}$ are learnable weight matrices. Since then, multiple studies have extended the attention mechanism [23, 20]. Sequence-to-sequence models with attention improve the performance of traditional models in machine translation tasks and have achieved an increase in translation performance.

2.4.2 Self-attention

Self-attention, also known as intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence. For each word, this mechanism allow the model to find different positions in sequence for relevancy. Self-attention has been adopted by some papers [24]. The term ”self-attention” was coined when the paper ”Attention is all you need” [20] is published.

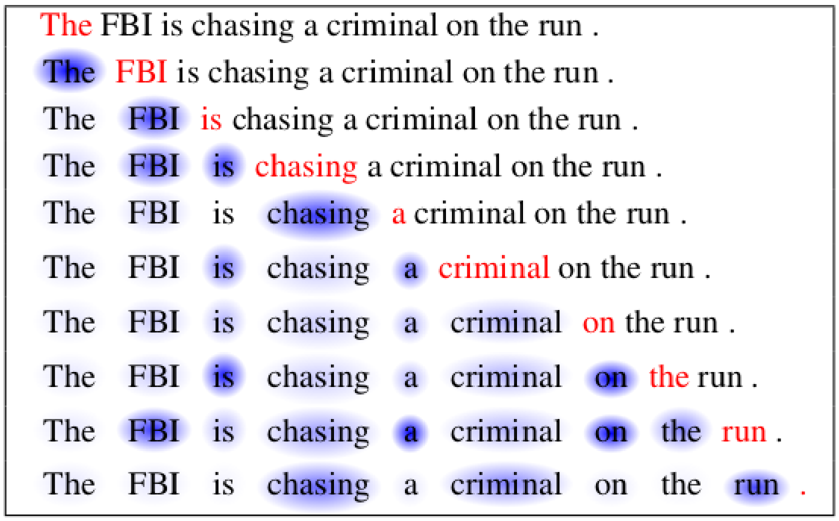

The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .

Figure 2.11: Self-attention: Each word is associated with other words in different positions
Image source: Cheng et al. [22]

The procedure for self attention is as follows:

- Step 1: Perform input embeddings.
- Step 2: Derive from weight matrices a set of **Key, Value, Query** for each embedding.
- Step 3: Calculate attention scores for each input by multiplying each **Query** vector to all **Key** vector
- Step 4: Calculate softmax of attention scores
- Step 5: Multiply attention scores with **Values**
- Step 6: Calculate the sum weighted for each output.
  To better understand the mechanism, lets perform a simple example.

First, lets initialize 3 input embeddings with dimension 4, and weight matrices corresponding to Key, Value, Query. Assuming Key, Value, Query are all of dimension 3, then these weight matrix will have dimension of 4x3.

```
## Initialize input embedding
Input1 = [1,0,0,1]
Input2 = [0,2,2,0]
Input3 = [2,1,1,2]

## Initialize key weight matrix
Weight_key = [[0, 0, 1],
             [1, 1, 0],
             [0, 1, 0],
             [1, 1, 0]]

## Initialize query weight matrix
Weight_query = [[1, 0, 1],
               [1, 0, 0],
               [0, 0, 1],
               [0, 1, 1]]

## Initialize value weight matrix
Weight_value = [[0, 2, 0],
                [0, 3, 0],
                [1, 0, 3],
                [1, 1, 0]]
```

Step 2: Calculate, **key**, **value** and **query** representation for input embedding.

```
# Input 1's key, value, query calculation
### Key
            [[0, 0, 1],
[1,0,0,1]  x [1, 1, 0], = [1,1,1]
             [0, 1, 0],
             [1, 1, 0]]

### Value
            [[0, 2, 0],
[1,0,0,1]  x [0, 3, 0], = [1,3,0]
             [1, 0, 3],
             [1, 1, 0]]
### Query
            [[1, 0, 1],
[1,0,0,1]  x [1, 0, 0], = [1,1,2]
             [0, 0, 1],
             [0, 1, 1]]

#** We can see that these calculations can operate with multiple embedding
   at the same time.
### Key
             [[0, 0, 1],
[[1,0,0,1], x [1, 1, 0], = [[1,1,1],
[0,2,2,0],    [0, 1, 0],    [2,4,0],
[2,1,1,2]]    [1, 1, 0]]    [3,4,2]]

### Value
             [[0, 2, 0],
[[1,0,0,1], x [0, 3, 0], = [[1,3,0],
[0,2,2,0],    [1, 0, 3],    [0,6,0],
[2,1,1,2]]    [1, 1, 0]]    [3,9,3]]

### Query
             [[1, 0, 1],
[[1,0,0,1], x [1, 0, 0], = [[1,1,2],
[0,2,2,0],    [0, 0, 1],    [2,0,2],
[2,1,1,2]]    [0, 1, 1]]    [3,2,5]]
```

Next, let's calculate attention scores for each input and apply softmax function

```
### Input1
                  T
          [[1,1,1],
[1,1,2] x [2,4,0],    = [4, 6, 18]
          [3,4,2]]

softmax([4,6,18]) = [0.0, 0.1, 0.9]
```

Next, we multiply scores with values and get the sum weighted as attention representation for input 1

```
### multiply score with value
0.0 x [1, 3, 0] = [0, 0, 0]
0.1 x [0, 6, 0] = [0, 0.6, 0]
0.9 x [3, 9, 3] = [2.7, 8.1, 2.7]

### attention representation for Input1
    [0  , 0  , 0  ]
  + [0  , 0.6, 0  ]
  + [2.7, 8.1, 2.7]
=   [2.7, 8.7, 2.7]
```

We can repeat this process to achieve attention representation for the other 2 embeddings.

2.4.3 Multi-head Attention

What makes the transformer model special is its multi-head self-attention mechanism. We take a deep dive to see what the attention is all about.

Scaled dot-product attention

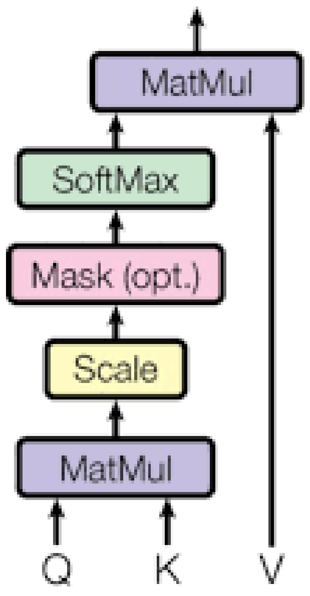


Figure 2.12: Scale dot product attention
Image source: Vaswani et al. [20]

The scaled dot-product attention mechanism adopts the general attention. Firstly, we calculate the dot product between query Q and key K. The result is then scaled by dividing by a factor of $\sqrt{d_k}$ and proceed to apply the softmax function. The result is the weight used to scale the value V.

$$attention(Q, K, V) = \frac{QK^T}{\sqrt{d_k}}V$$

Multi-head Attention

The multi-head mechanism allows us to run multiple scaled dot product attention heads in parallel. After the attention representation has been calculated, it is concatenated and sip through an affine layer to "allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this." [20].

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

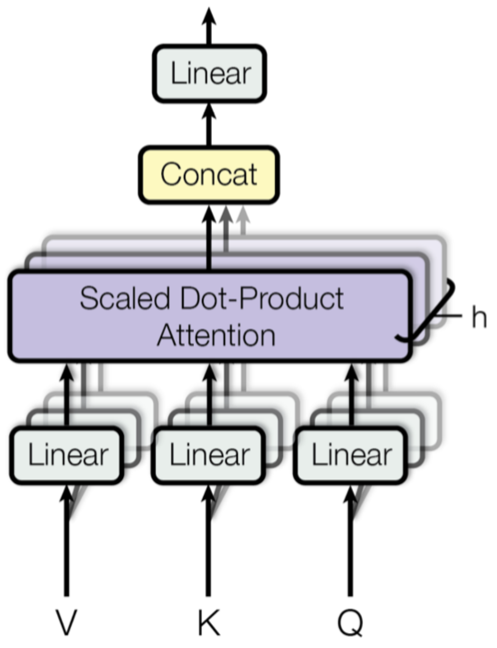


Figure 2.13: Multi-head self-attention
**Image source:* Vaswani et al. [20]

2.4.4 Model Description

The architecture of transformer is as follow:

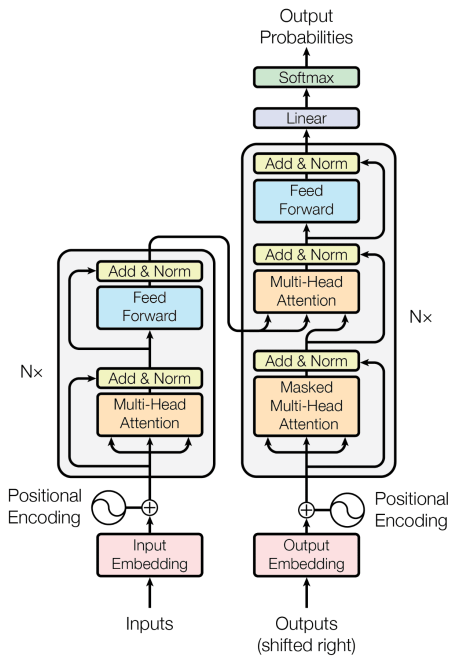


Figure 2.14: Transformer architecture

Encoder
Input embedding
The tokenized input will be fed into the input embedding layer to extract the vector representation.

Positional encoding
The input embedding encodes the context and meaning of a word. However, at different positions, words may convey a different meaning. This architecture adds a layer to encode the positional information relating to each word.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

where pos is the relative position of each word, and PE is the positional encoding at position i. The sum of positional encoding and input embedding will be fed to the actual encoder layers.

Encoder layers
The encoder generates an attention-based representation with the capability to locate a specific piece of information from a potentially infinitely-large context. The original paper used 6 layers of encoding, each comprised of multi-head attention and a feed-forward layer. The first layer receives input $+$ positional embedding, while others receive the output of the previous layers.

Decoder
Similar to the encoder, the decoder comprises N layers. The first layer receives the input and positional embedding of the target sentence, while the other layer receives output from the previous layer. An important difference is an introduction of masking multi-head attention. By introducing masking, the decoder only attends to input embedding that precedes it, whereas the encoder can attend to both directions. The masking makes the decoder unidirectional.

Encoder-Decoder attention
In addition, another multi-head attention is put on the decoder. In these layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This enables the decoder to attend to all positions of the source input.

2.5 BERT Model

There have been many state-of-the-art NPL proposals since the introduction of pre-trained language models such as ELMo (Peters et al.) [25], Generative Pre-trained Transformer (OpenAI GPT) (Radford et al.) [26], ULMFit (Howard, Ruder et al.) [27]. Downstream tasks could be solved using the two following approaches for applying pre-trained language representations:

- Feature-based: this approach address the usage of task-specific architectures that are based on pre-trained representations as additional features (e.g., ELMo)
- Fine-tuning: task-specific parameters are introduced and trained on downstream tasks by fine-tuning all pre-trained parameters (e.g., OpenAI GPT, ULMFit)

These models refrain from reaching the actual power of the pre-trained model due to its unidirectional nature, which limits the option of architectures that could be used during pre-training.

BERT is proposed to tackle this problem.

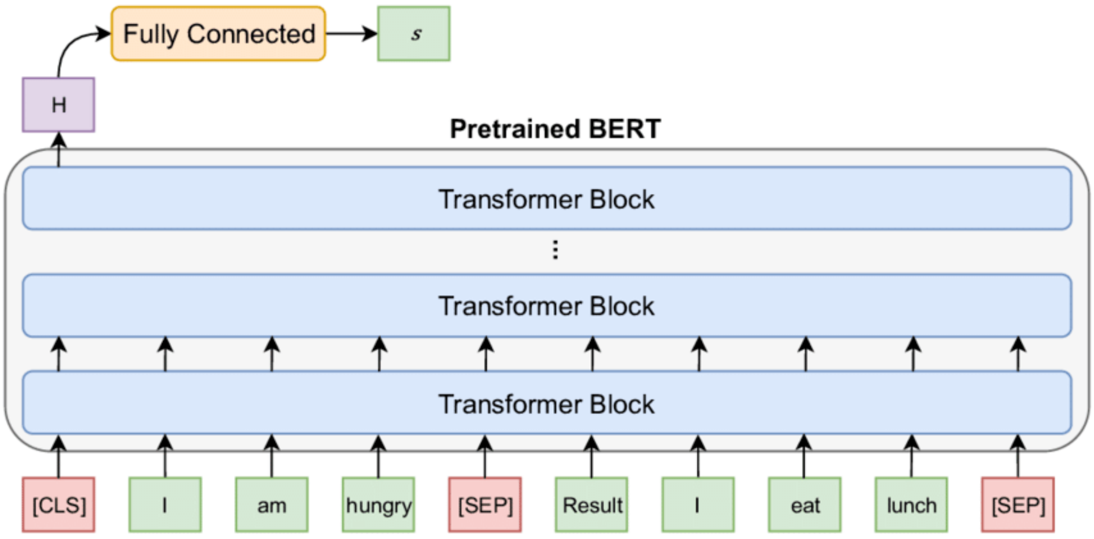


Figure 2.15: A sample of BERT architecture with multiple layer of Transformer block

2.5.1 Definition

BERT (Bidirectional Encoder Representations from Transformers) (Jabob Devlin et al.) [17] is a transformer-based machine learning model developed with fine-tuning approach by researchers at Google AI Language. BERT is designed to overcome the limitations that persist in already pre-trained models such as OpenAI GPT and ELMo while also enhancing the ability to encapsulate the context of sentences. Another distinguishing feature of BERT is that there are minimal changes in the pre-trained architecture and final downstream architecture.

2.5.2 Model Architecture

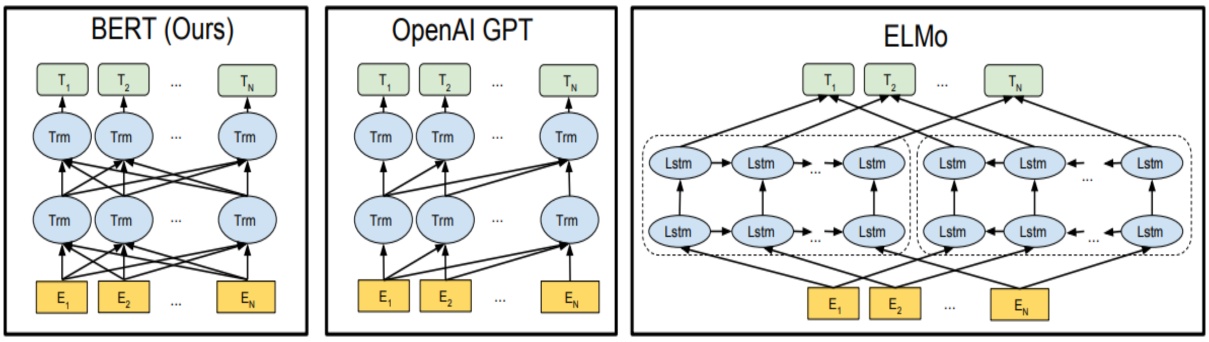


Figure 2.16: Differences between BERT, OpenAI GPT and ELMo architecture

BERT's model architecture is nearly identical to that of Transformers, specifically, a stacked bidirectional Transformer encoder without the decoder of Transformer. The differences between

BERT, ELMo, and OpenAI GPT could be seen in figure 8. BERT leverage the power of a bidirectional Transformer while OpenAI GPT uses a left-to-right Transformer and ELMo concatenates trained left-to-right and right-to-left LSTM.

Let L, H, and A be denoted for the number of layers (i.e. Transformer blocks), the hidden size, and the number of self-attention heads, respectively. There are two versions of BERT in different sizes:

- $BERT_{BASE}$ (L=12, H=768, A=12, Total Parameters=110M)
- $BERT_{LARGE}$ (L=24, H=1024, A=16, Total Parameters=340M)

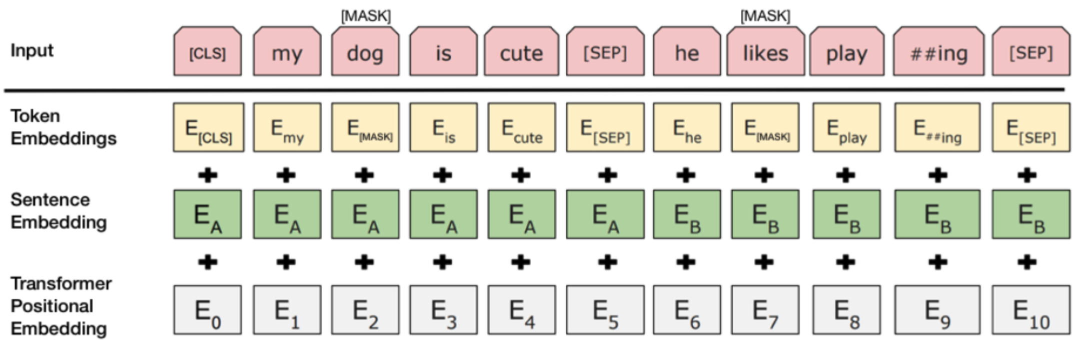

Input
[MASK]
[MASK]
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} $E_{[MASK]}$ E_{is} E_{cute} $E_{[SEP]}$ E_{he} $E_{[MASK]}$ E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Sentence Embedding
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Transformer Positional Embedding
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2.17: BERT's input representation

In the vanilla BERT configuration, the input representation could be one single sentence or a pair of sentences, which means the input sequence not necessarily to be an actual linguistic sentence but rather an arbitrary span of continuous text. This enables various downstream tasks (e.g., Question and Answer) to perform well with little modifications.

The initial approach of embedding is a 30000 token vocabulary WordPiece (Wu et al., 2016) [28]. The author also appends several special tokens for better sequence representation shown in figures 2.17 and 2.15.

- $[CLS]$: A special classification token stays at the beginning of the sequence. The final hidden state that maps to this token is used as a combined sequence representation for classification tasks. Therefore, this token is ignored in non-classification tasks.
- $[SEP]$: A special separate token to differentiate sentences in the input sequence. A learned embedding is also added for specifying whether each token belongs to which sentence.

Each token representation is the sum of its corresponding token, segment and positional embeddings.

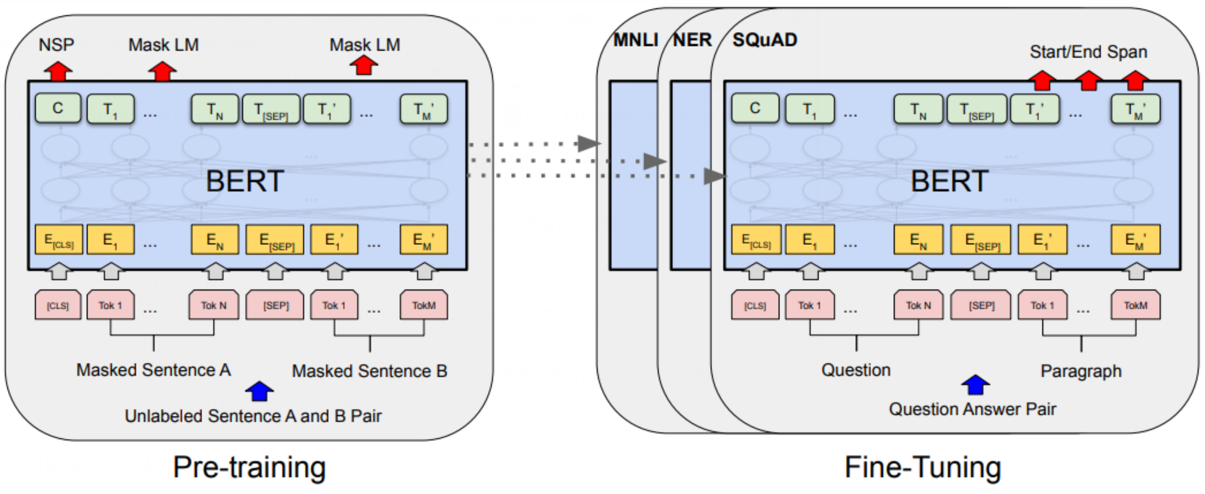


Figure 2.18: Overall pre-training and fine-tuning procedures for BERT.

2.5.3 Pre-Training BERT

Masked Language Modeling (MLM)
Language Modeling is the task of predicting the next word given a sequence of words. In masked language modeling, instead of predicting every next token, only the masked tokens are predicted. BERT is pre-trained with this task by randomly masking 15% of all tokens in a sequence and only predicting those masked words. To avoid mismatching in fine-tuning phase due to $[MASK]$ token do not appear during fine-tuning, the author provides an approach that: if a token is chosen to be replaced, it will have an 80% chance to be replaced with $[MASK]$ token, 10% with a random token or left unchanged for the last 10%.

Next Sentence Prediction (NSP)
Next sentence prediction task is a binary classification task in which, given a pair of sentences, it is predicted if the second sentence is the actual next sentence of the first sentence. This task is helpful for many downstream tasks such as question answering or natural language inference where they require the understanding of the relationship between two sentences. To pre-trained BERT, say choosing sentences A and B, then B would have a 50% probability to actually be the next sentence that follows A (labeled as *IsNext*) while the other 50% of the time is a random sentence from the corpus (labeled *NotText*).

2.5.4 Fine-tuning BERT

The self-attention mechanism of the Transformer enables BERT to be applied in numerous downstream tasks with little modification. The input of the pre-trained BERT is similar to sentence pairs in paraphrasing, question-passage pairs in question answering, and a degenerated text-$\emptyset$ in text classification or sequence tagging. While at the output, sequence tagging or question answering tasks would be fed into the output layer their output token representations, and $[CLS]$ token is fed into the output layer for classification tasks.

2.5.5 BERT Variants

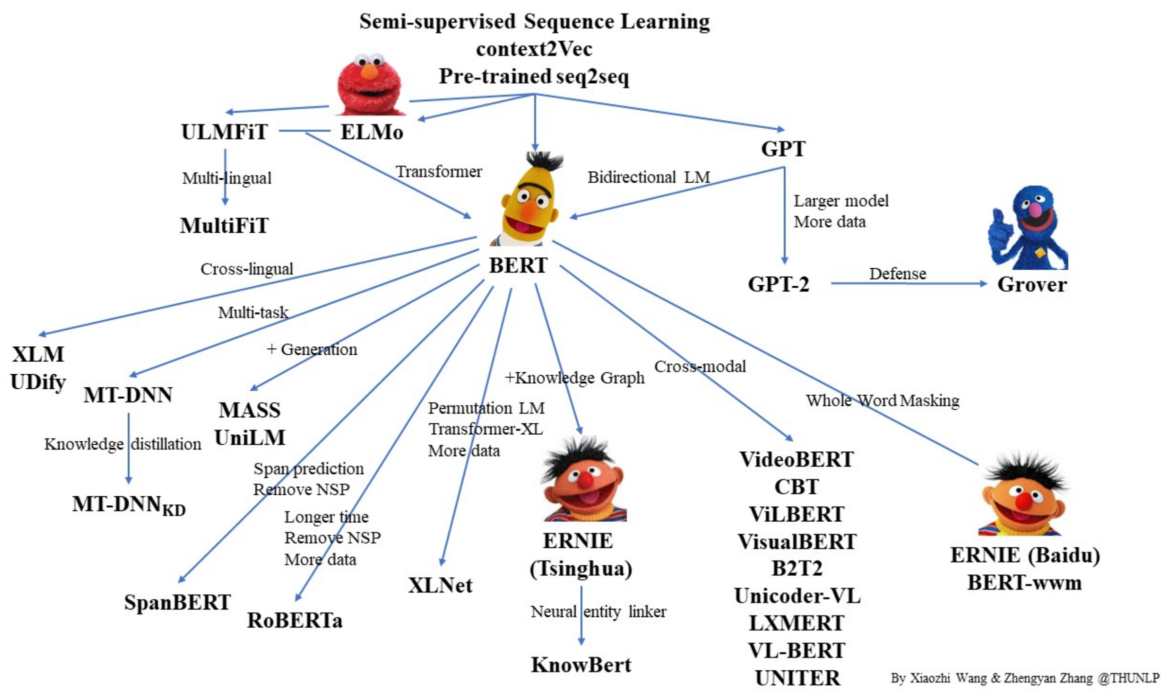


Figure 2.19: The many spawned variants of BERT

Since its release, BERT brought significant changes and has achieved state-of-the-art in multiple NLP problems, and also in other fields such as computer vision. Despite its great achievement, there are still many limitations due to its architecture and pre-trained data. Therefore, many alternative models that take BERT as its core has been proposed for many domain-specific tasks such as BioBERT (biomedical text), SciBERT (scientific publications), etc. Below are several variants that have been proved to outperform the initial BERT in many tasks:

- RoBERTa (Liu et al.) [29]: The author of this model showed that the original BERT was undertrained. RoBERTa is trained for a longer period, on a bigger dataset and in bigger batches with longer sequences. Especially, it also removes the task of NSP out of its pre-trained procedure and uses a dynamic masking scheme.

- XLNet (Yang et al.) [30]: This model uses permutation to capture bidirectional context rather than do token masking, also combines the best of denoising autoencoding of BERT and autoregressive language modeling of Transformer-XL (Dai et al.) [31].

- ALBERT (Lan et al.) [32]: Known as ’A lite version of BERT’, the model was proposed to enhance the training and results of BERT architecture by using parameter sharing and factorizing techniques. It reduces the number of parameters needed to train and uses the task of Sentence Order Prediction rather than NSP.

Chapter 3

Related Works

3.1 Tokenization Repair

When fixing text errors, one particular problem arises related to the unintended white spaces that can mess up the tokenization process. Characters that belong to another word may confuse the model as the character information is crucial to identifying the correct word. Traditionally, this is a word segmentation problem, which can be solved with algorithms such as Best first search or Peter Norvig algorithm [33], segment words in a fixed vocabulary. When considering a spelling correction problem, we cannot create a vocabulary for misspelling, since it is impractical. Bast et al. [34] called this the tokenization repair problem: Given a natural language text with any combination of missing or spurious spaces, reverse it to the correct token. Fixing this type of problem is extremely hard, so the paper utilizes a character language model, optionally coupled with a sequence labeling model to improve accuracy. The model calculates probabilities that whitespace occurs after a sequence of characters which we talk about below.

3.1.1 Unidirectional Language Models

The first component is a character language model, which estimates the probability of a string to occur in some language based on individual characters. The component consists of a Long short-term memory (LSTM) network that learns the context of the character stream. This is then passed through a dense layer and a softmax output layer for character classification. The model predicts $\vec{p}(s|c)$ where p is the probability that a character s comes after a context c.

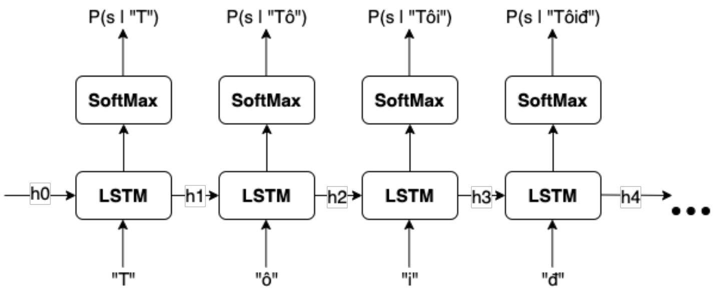


Figure 3.1: Unidirectional Language Model with LSTM

Model is trained using the categorical cross entropy loss, and sum up to 6,287,563 trainable parameters. To calculate the probability whether the space exist after a word p(␣s|c), we define the formula:

p(␣s|c) = p(␣|c) * p(s|c␣)

A backward language model can easily be implemented as well, as the formula remains unchanged.
Beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a set limited by a parameter called **beam** or beam-width. At each step of the search, the algorithm keeps track of the score and the partial solution string in the search state.

At each time step i, the search state is determined by whether a space is added before adding the character T_i or not. Without adding space, the search state is:

$$S_i = S_{i-1} - log(\vec{p}(T_i|R_{i-1})) + P_{del}$$

$$R_i = R_{i-1}T_i$$

With adding space, search state is:

$$S_i = S_{i-1} - log(\vec{p}(␣T_i|R_{i-1})) + P_{ins}$$

$$R_i = R_{i-1}␣T_i$$

Notice that we add 2 penalty scores for inserting and deleting a space. However, this penalty is only added if the change does not exist in the original sentence, otherwise, it is zero.

3.1.2 Bidirectional Sequence Labeling Model

The second component is a sequence labeling model. Given a sequence of non-space strings of character from the front and back, the model learns to predict the probability $\overleftrightarrow{p}$ of space occur in between. Here we again use the same RNN model as the language model, only now utilizing a bidirectional version. The model is trained with binary cross entropy loss.

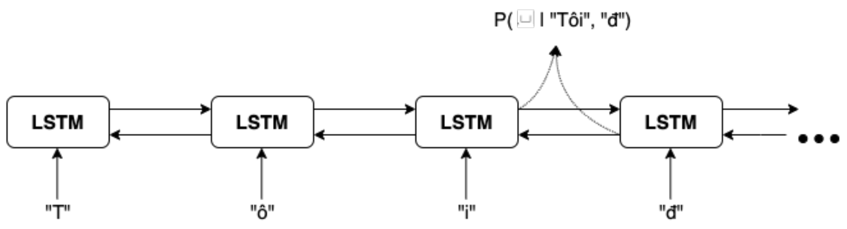


Figure 3.2: Bidirectional Sequence Labelling with BiLSTM

To combine both models, we can slightly modify beam search. At each time step i, the search state is determined by whether a space is added before adding the character T_i or not. Without adding space, the search state is:

$$S_i = S_{i-1} - log(\vec{p}(T_i|R_{i-1}) * (1 - \overleftrightarrow{p})) + P_{del}$$

$$R_i = R_{i-1}T_i$$

With adding space, search state is:

$$S_i = S_{i-1} - log(\vec{p}(\llcorner T_i|R_{i-1}) * \overleftrightarrow{p}) + P_{ins}$$

$$R_i = R_{i-1i}$$

3.2 Transformer-based Model for Vietnamese Spelling Correction

Research team from Vietnam National University, Hanoi, Do et al. [1] proposed a deep-learning approach on the Spelling Correction task in November 2021. Their results include a correction pipeline based on transformer architecture, and a public testing set that for future papers comparison.

3.2.1 Model

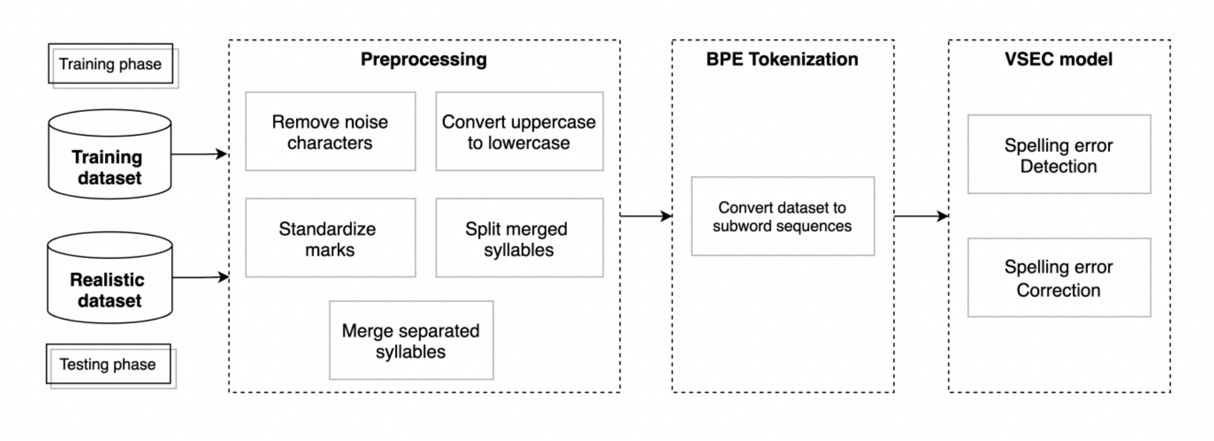


Figure 3.3: VSEC model

Preprocessing
The preprocessing phase consists of five steps:

- Step 1: Remove noise characters
- Step 2: Convert uppercase to lowercase
- Step 3: Standardize Vietnamese marks
- Step 4: Split merged syllables: Either due to user mistakes or hardware error, sometimes whitespace character is not registered, which leads to merging syllables. For example: ”xin lỗi” -> ”xinlooix”. They utilize Peter Norvig’s word segmentation algorithm in conjunction with transforming into telex form to create the word.

- Step 5: Merge separated syllables: Conversely, sometimes whitespace characters are misclicked, such as ”chuyển” -> ”ch uyển”. They employ the Trie structure [21], which has demonstrated its ability to browse prefixes.

Tokenization
The paper opts for subword tokenization instead of character or word. The reasoning for this decision is that spelling errors come in multiple sizes and shapes, therefore having a syllable level will inevitably increase vocabulary size. Character level, on the other hand, increases sequence length and leads to slower training and convergence. Sub-word tokenization, therefore, is the perfect blend that jointly brings both contextual information of word level and a fixed vocabulary at the character level.

Model
The idea is to treat Vietnamese spelling error correction as a machine translation problem. In this paper, they utilize the seq-2-seq architecture provided in the paper by Vaswani et al. [20] Given an errorful sentence, we translate it into an error-free sentence. The Transformer encodes a misspelling sentence to a context hidden state using a stack of 3 encoder blocks with multi-head self-attention layers and feed-forward networks. The decoder uses the encoder’s hidden states and the sequence of previous target tokens to generate the target hidden state by applying a stack of 3 decoder blocks.

3.2.2 Testing set

In addition, the research team published public realistic testing set for comparison. The testing set was sampled from real-world sources, and hand-labeled by three different people. Contents include 618 documents at Tailieu, an educational material website. All in all, the dataset includes 9,341 sentences, which contain 11,202 spelling errors in 4,582 different types.

3.2.3 Result and Evaluation

In general, the paper achieves positive results while using a deep-learning approach to the spelling correction task.

Through their research, they have concluded that their approach is more effective than using a statistical-based method such as N-gram. N-gram calculates the probability of a word occurring given the context, which means that it will have a hard time choosing the right word if the nearby word contains more errors. Both non-word and real-word errors can be identified and corrected, however, some are limited due to domain-specific terminology.

3.3 Hierarchical Transformer Encoders for Vietnamese Spelling Correction

In May 2021, researchers from Zalo Group, Hieu et al. [2] published a paper on Vietnamese Spelling Correction. Their paper includes a hierarchical transformer architecture that combines both word and character level information, which we will include later, and a.public testing set for comparison.

3.3.1 Model

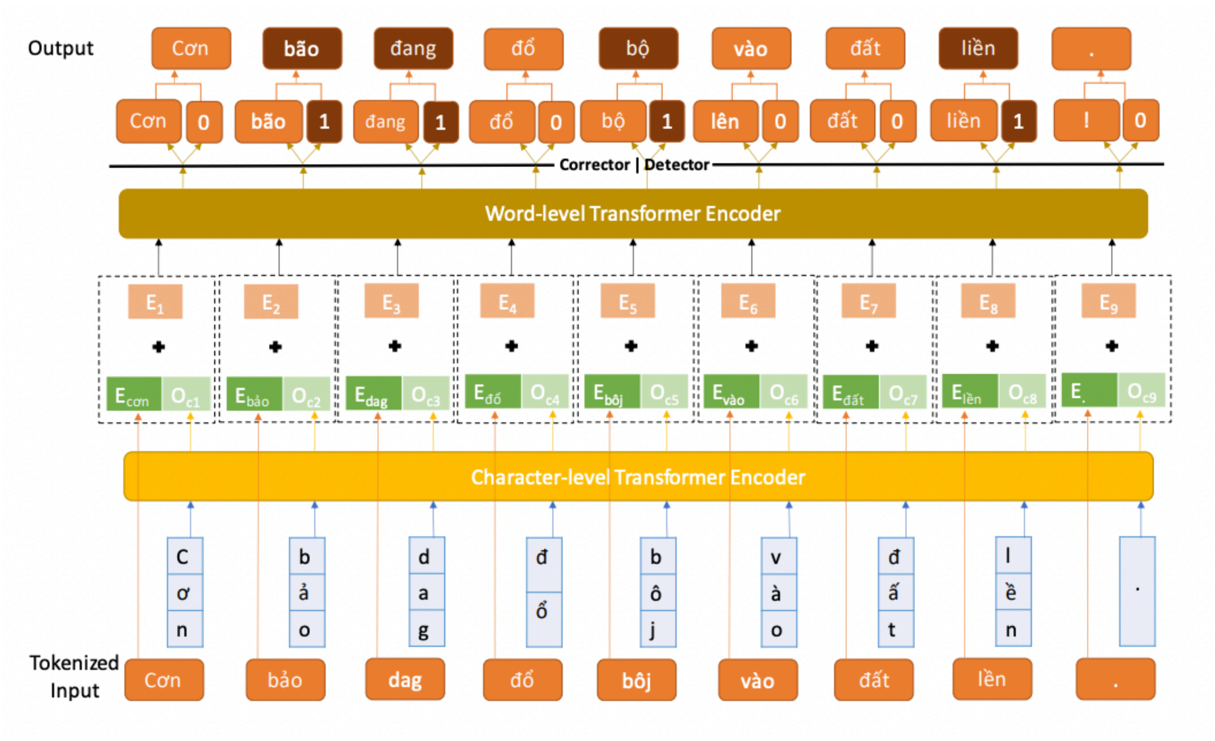


Figure 3.4: Hierarchical Transformer Model

Tokenization
Instead of opting for a hybrid subword tokenization method, the paper argues that sub-words only stand for phonetic features rather than the meaning and that Vietnamese has a fairly small amount of tokens. Therefore, the word around is to combine both character information and word information, which contains both knowledge of the word itself and the context around it.

Model
Both character and word levels adopt Transformer encoder architectures to encode information. The character level encoder is composed of 4 self-attention layers of hidden size 256 while the word-level one is bigger, with 12 self-attention layers of hidden size 786, which is the same size as conventional BASE-size BERT. The outputs are then forwarded to two classifiers, one for detection and the other for correction.

3.3.2 Testing set

For performance measurement of the Vietnamese spelling correction problem, they collected Wikipedia drafts from varied domains. We select ones that have a substantial number of spelling errors and then manually detect the errors and suggest reasonable substitute words. In total, the test set provides more than 100 articles with about 1,500 spelling errors.

3.3.3 Result and Evaluation

The paper proposes a novel architecture based on transformer that combines both word-level information and character-level information to handle the correction task, which achieved incredible results, trained on 3Gb of clean text. In general, the paper achieves positive results while using a deep-learning approach towards the spelling correction task, while also including a public test set for the Natural Language Processing community, which we discussed in our discussion.

3.4 Soft-masked BERT

The idea of integrating BERT into machine translation and grammatical error correction tasks is the uprising approach [7, 35, 36]. Different approaches have been implemented: Using BERT as an encoder for the encoder-decoder model [35], using BERT as input embedding for a sequence to sequence model [36, 7] or using BERT as a language model.

Soft-masked BERT (Zhang et al.) [6] achieve impressive results for Chinese spelling correction. We would like to implement this model in Vietnamese spelling correction tasks. The model specifically tackles different sub-tasks: detecting misspelled words and correcting them based on context. The model is decomposed into two different networks specifically for those sub-tasks: a detection network and a correction network.

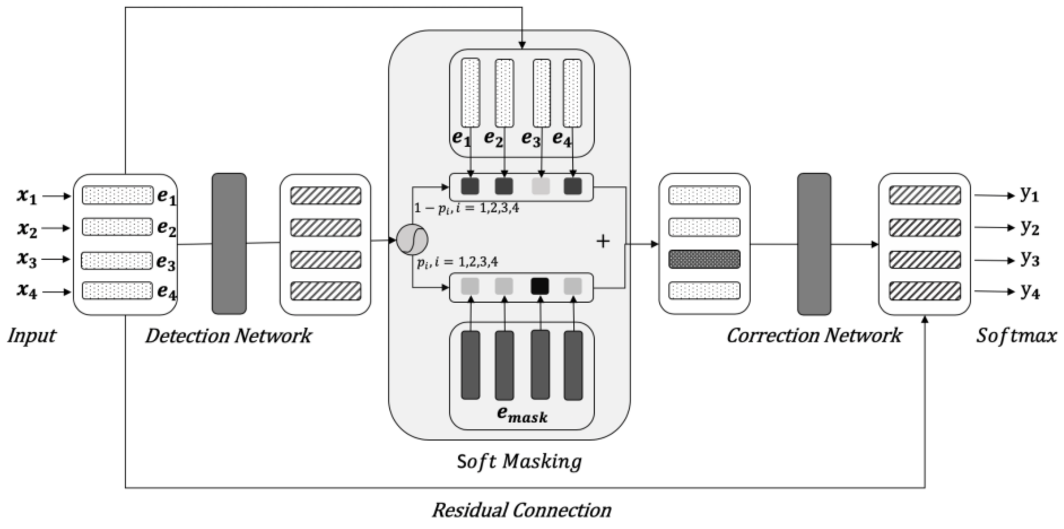


Figure 3.5: Soft-masked BERT.

The idea behind Soft-masked BERT is to re-use the already optimized pre-trained BERT model which was trained with the Masked Language Modelling task. Suppose an input X is a sequence of tokens $(x_1, x_2, ...x_n)$ and an output sequence $Y = (y_1, y_2, ..., y_n)$, both sequence have the same length. If a token in X is incorrect, the model will replace that token with the correct word to obtain the correct Y. The usage of Detector and Corrector are as followed:

3.4.1 Detector

The detection network is a sequential binary labeling model, implemented as a bi-directional GRU network, where it is used to predict whether there is any token in the input sequence that is incorrect. The network input is the embedding $E = (e_1, e_2, ...e_n)$ from BERT embedding layer, and the output is a sequence of label $G = (g_1, g_2, ..., g_n)$, with e_i, g_i is the embedding and label of token i^{th}, respectively. The label value of 1 indicates that token i^{th} is incorrect, and 0 if it is correct. For each token, there is a probability p_i indicates how likely that token is to be incorrect. p_i is calculated by the following equation:

$$p_i = P_d(g_i = 1|X) = \sigma(W_d h_i^d + b_d)$$

where P_d denotes the conditional probability of the detection network, σ is sigmoid function, W_d and h_d are parameters, h_i^d is the hidden state of Bi-GRU.

A soft-masked embedding e'_{mask} is created by calculating the weighted sum of input embeddings e_i and mask embeddings e_{mask} with error probabilities p_i as weights.

$$e'_{mask} = p_i \cdot e_{mask} + (1 - p_i) \cdot e_i$$

The soft-masked embedding e'_{mask} would be closer to the mask embedding e_{mask} if the probability p_i is high or else it is close to the input embedding e_i.

3.4.2 Corrector

The corrector is a BERT-based sequential model that assign probability to each of the mask embeddings $E' = (e'_1, e'_2, ..., e'_n)$ tokens and output a sequence of tokens $Y = (y_1, y_2, ..., y_n)$. The author takes the output of the final hidden layer h_i^c and computes the probability:

$$P_c(y_i = j|X) = softmax(W(h_i^c + e_i) + b)[j]$$

where $P_c(y_i = j|X)$ is the conditional probability that token x_i is corrected as token j from the candidate list, W and b are the parameters, h_i^c is the hidden state from the last hidden layer and e_i is the input embedding of token x_i.

The training of the detector and corrector is defined by the two objective functions, which are then be combined into one overall objective for the learning process:

$$\mathcal{L}_d = -\Sigma_{i=1}^{n} log P_d(g_i|X)$$

$$\mathcal{L}_c = -\Sigma_{i=1}^{n} log P_c(y_i|X)$$

$$\mathcal{L} = \lambda \cdot \mathcal{L}_c + (1 - \lambda) \cdot \mathcal{L}_d$$

where $\mathcal{L}_d$ and $\mathcal{L}_c$ is the objective for training the detector and corrector, respectively, $\mathcal{L}$ is the overall objective, and $\lambda \in [0, 1]$ is coefficient of user choice.

Chapter 4

Our Works

4.1 Errors and Augmenters

Firstly, we have to understand different types of spelling errors. In the spelling correction task, spelling mistakes can have either of the two characteristic: non-word and real-word [37]. In layman's terms, errors with non-word characteristic are misspelled words that create an out-of-vocab word; for example, ”trừ” may be misspelled to ”truwf”, due to a telex mechanism in the Vietnamese keyboard. Accidental insertion and deletion of characters also fall into this category, which we will discuss later.

Errors with real-word characteristic , on the other hand, generates other words inside the vocabulary, which makes it harder to detect and the model must rely on context to correct it. Vietnamese spoken language is diverse and varies from region to region, an instance of this would be ”vui vẻ” and ”dui dẻ”, where both words phonetically sound the same but are spelled differently. Shortcuts and teen-codes are not considered real-world errors.

4.1.1 Error analysis

To observe on the frequency of error types, we crawl several small corpus from informal news websites such as kenh14.vn, tintinonline.com.vn, etc.

In total, we achieve the following statistics:

	Counts
Number of sentences	1067
Total tokens	48816
Total error tokens	573

Table 4.1: Real world data counts

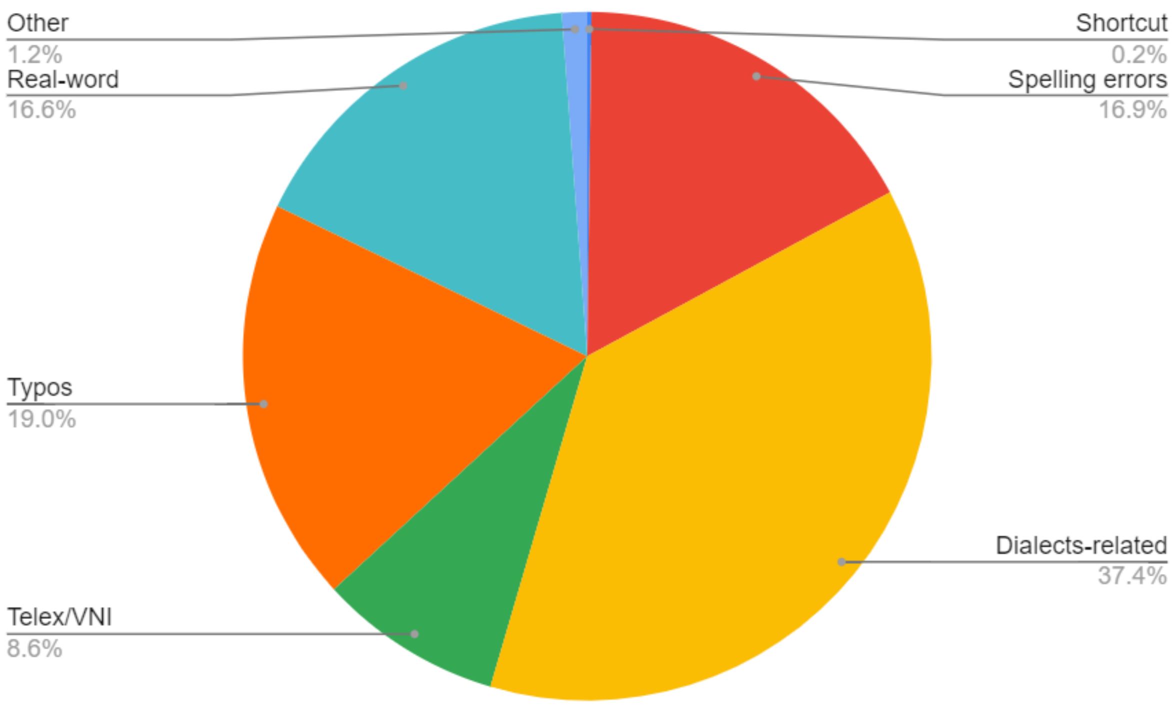


Figure 4.1: Errors type percentages

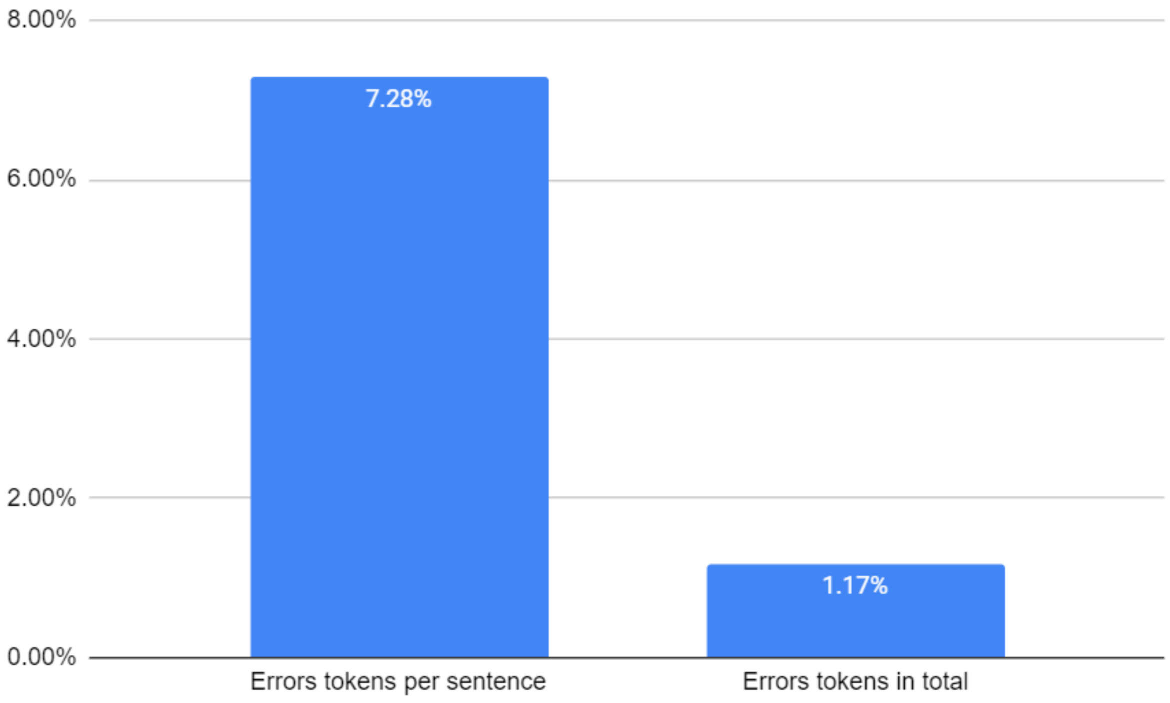


Figure 4.2: Percentages of error tokens

While the number of error tokens only takes up just above 1% of total tokens, on sentences that have errors, the percentage of error tokens amounts to over 7%. With most of the errors are related to dialect marks, such as missing or wrong dialects. Typos is another common error where having keyboard misclicked often occurred. Not to mention, spelling errors also have a high rate of occurrence, words with missing characters are the most common type.

4.1.2 Defining Errors

Through observation of real-life errors, we can separate different errors categories to analyze for augmentation.

Typos
Typos errors are mostly related to keyboard typing mechanisms. For this category, we further divide it into more specific type:

- VNI, Telex: Vietnamese typists mainly use Telex or VNI typing mechanisms for pure characters (á, ă, â). These characters are compounded by multiple characters, for example, á is a combination of a and s in telex. User might forgot to turn on these mechanism, hence resulting these errors.
- Keyboard-related error: While typing, we might misclicked on a close-proximity characters on the keyboard, which produce words with non-word or real-word characteristics.
- Random characters: User might press a key multiple times mistakenly, or the keyboard does not register the response, resulting in words with missing or duplicated characters.
- Merged tokens: Typing while space bar not registered or the user forgot to enter space could result in merged words. This error could also lead to VNI, Telex errors.
- Split tokens: In contrast to white-space omission, user might click on space bar accidentally.

Spelling Errors
In different regions, people may use unofficial or regional dialects. Not to mention, misused knowledge or using spoken languages in text is quite also common in daily life.

- Starts with d/gi/v, tr/ch, s/x, or ends with c/t: People often confuse these letters due to its similar pronunciation.
- Duplicate copy: While typing, user might forgot that they already typed some certain words. Hence, words are repeated unnecessarily.
- Others: These errors are those with no certain rules, and mostly due to the wrong knowledge of the user.

Diacritic-related error
Diacritics are an important part of the Vietnamese alphabet. In the written language, vowels (a, e, i, o, u) can be comprised of more than just the base but also diacritics to form a full word. When a sentence is full of missing diacritics, it is harder to deduce the meaning of a word. Numerous scholars have studied the field of Vietnamese Diacritics Restoration problem [38, 39, 40] with positive results. Non-word errors in this category comprise fully missing diacritics or partially

missing diacritics, which Pham et al. [39] have classified as type I error. These errors can be predictable and can be easily regenerated.

Categories	**Error**	**Example**
Typos	Telex Error	”uống” => ”uoosng”, ”uoongs”,.. ”trăng” => ”trawng”, ”trangw”,..
	VNI Error	”uống” => ”uo16ng”, ”uong61”,.. ”trăng” => ”tra7ng”
	Keyboard-related Error	”học” => ”họv”, ”học” => ”hock”
	Random Character	”tôi” => ”tôu”, ”sáng” => ”ság”, ”răng” => ”răngg”
	Merge tokens	”tôi đi học” => ”tooiddi học”
	Split token	”tôi” => ”tô i”
Spelling Errors	Start with d/gi/v	”dang rộng” => ”vang rộng”, ”giang rộng”
	Start with tr/ch	”chấn thương” => ”trấn thương”
	Start with s/x	”sẻ” => ”xẻ”
	Ends with c/t	”cá cược” => ”cá cượt”
	Duplicate copy	”tôi đi học” => ”tôi đi đi học”
	Others	”không” => ”khôn”, ”khổng”, ”hông”,.. ”thôi” => ”thô”, ”hôi”, ”thi”,..
Diacritics-related	No diacritic marks	”ăn uống” => ”an uong”
	Missing diacritic marks	”một ngày” => ”môt ngay”
	Wrong diacritic marks	”bác sĩ” => ”bác sỉ”

Table 4.2: Example of some misspelling errors

4.1.3 Out-of-scope error types

We will discuss why we leave out several common errors such as teen-code, shortcut terms, and also the problem of dealing with foreign words.

Teen-code
While teen-codes is quite commonly used in daily text and messages, it is an intentional attempt by the user to stylize the word, hence do not fall into the spelling correction category. Also, we could not recreate this type of error without spending hours labeling them. Due to its uniqueness and rule-less, dealing with them is painful.

For instance, someone may put in a number “3” to replace the letter “e”, randomly drop some letters, or add random punctuation in-between letters and words. A particular teen-code sentence would be as follow: “b4.nh dzạo n3‘ thja’ nà0 r04j‘” which translates to “Bạn dạo này thế nào rồi”.

Moreover, the teen-code “dictionary” tends to increase drastically, with different ways of creating them being “invented” almost daily, not to mention, the emergence of trending words. Humans may not understand every word written with teen-codes, even given the whole context of the passage. Hence, correcting them is not always possible.

Shortcut terms

As for shortcut terms, we could simply replicate this error by only taking the first letter of each word and then concatenating all those letters. However, not every word could be shortcuted. These shortcut terms tend to be relevant to one single particular domain. While several terms could be used for general purposes, these terms might be abbreviated to various meanings depending on the context. Moreover, several shortcut terms required additional tokens for correcting them, which only suffice with generation model.

A simple example of the term “bn” could stand for ”bạn” or “bao nhiêu”. In the context of clothing shop, one may text “Bộ này bn vậy bn?”, which could get us confused about the word order.

In order to solve this, building a specific dictionary for each field is a must if we want to augment this type of error. Again, time and resource limitations restrict us from doing so.

Foreign words
Although foreign words may still appear in the text and convey meaning, we decided to exclude them from our correct scope for a few reasons. First of all, we want to focus on Vietnamese correction, instead of general text correction and learn to correct text based on context. Secondly, adding correction for English tremendously increase vocabulary size, which may lead to a slower convergence time.

4.2 Dataset Augmentation

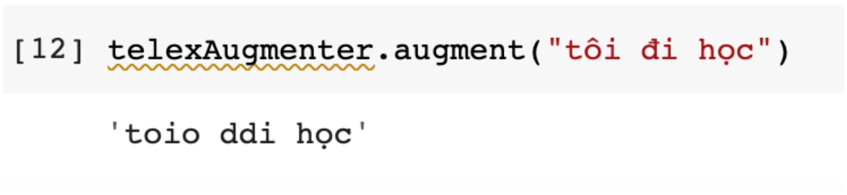
For the spelling correction task, there are no common data for us to train and compare with other models. Fortunately, Vietnamese spelling error follows some common rules that we can implement rule-based random augmenter. Through some research, we determined the rules to augment an error-free text into an errorful text.

4.2.1 Implementing Augmentation Method

Now that we have analysed error types thoroughly, we can implement rule-based random augmentation methods for each error types. In total, we have .. number of augmenter:

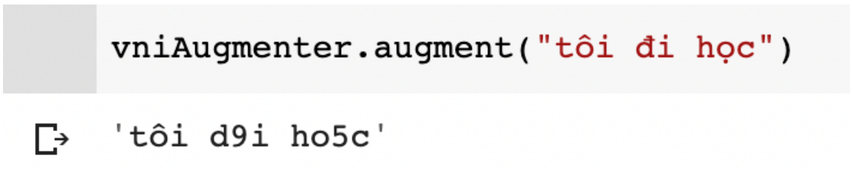
- Telex Augmenter
- VNI Augmenter
- Keyboard Augmenter
- Random Augmenter
- Split Token Augmenter
- Merged tokens Augmenter
- No Diacritic Marks Augmenter
- Wrong Diacritic Marks Augmenter
- Missing Diacritic Marks Augmenter

Below is some example related to different augmentation method Below we include some example of our augmenter output



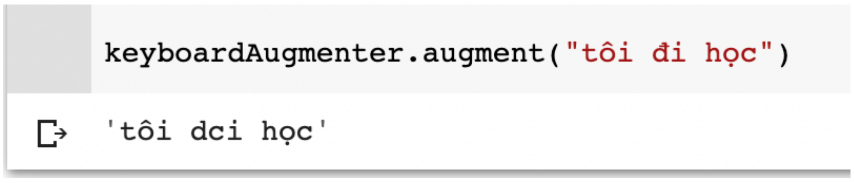
```
[12] telexAugmenter.augment("tôi đi học")

     'toio ddi học'
```

Figure 4.3: Telex Augmenter



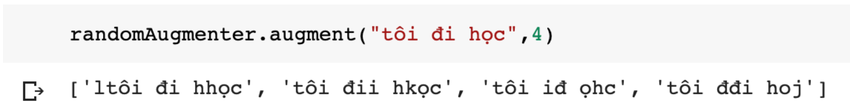
```
vniAugmenter.augment("tôi đi học")

'tôi d9i ho5c'
```

Figure 4.4: VNI Augmenter



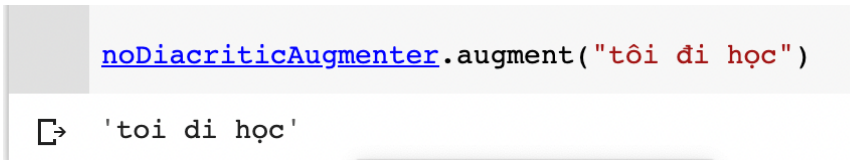
```
keyboardAugmenter.augment("tôi đi học")

'tôi dci học'
```

Figure 4.5: Keyboard Augmenter



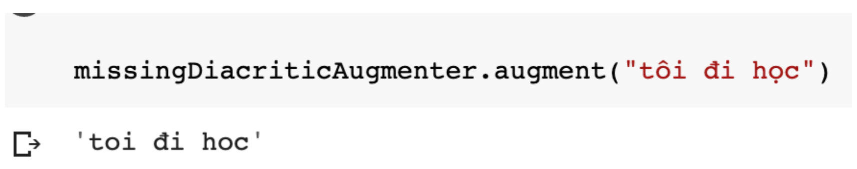
```
randomAugmenter.augment("tôi đi học",4)

['ltôi đi hhọc', 'tôi đii hkọc', 'tôi iđ ọhc', 'tôi đđi hoj']
```

Figure 4.6: Random Augmenter



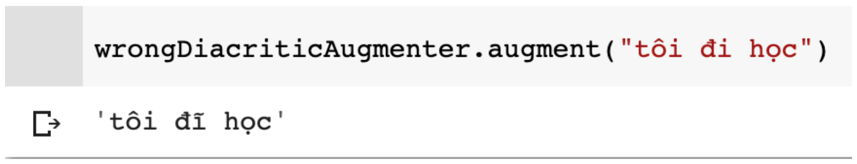
```
noDiacriticAugmenter.augment("tôi đi học")

'toi di học'
```

Figure 4.7: No Diacritic Augmenter



```
missingDiacriticAugmenter.augment("tôi đi học")
'toi đi hoc'
```

Figure 4.8: Missing Diacritic Augmenter



```
wrongDiacriticAugmenter.augment("tôi đi học")
'tôi đĩ học'
```

Figure 4.9: Wrong Diacritic Augmenter

4.2.2 Preprocessing Data

Our pre-processing pipeline goes through the steps below:

- Step 1: Remove noise characters: We clean the following illegal characters before augmentation:
  - Compound unicode to standard unicode
  - Remove links
  - Remove HTML tags from crawling
  - Remove emoji
  - Remove trailing space and newlines
- Step 2: Convert all characters to lowercase: Keeping it uppercase will drastically increase vocabulary size, which decreases the model performance.
- Step 3: Separate punctuation mark: Like uppercase, having an additional punctuation mark to the word does not add any information.

4.2.3 Augmentation Pipeline

Our augment process guarantees a more uniform distribution of errors across all tokens in the vocabulary. First, we build a vocabulary from the corpus and mark the sentence that contains the word within it. For each word, we sample a specific number of marked sentences for each type of error and replace the error token based on the augmenter output. We also maintain the error percentage of a sentence to be lower than 30%, so the marked sentence list has to be re-evaluated before sampling. This process is repeated until all word in the corpus has been augmented.

In our augmentation pipeline in the thesis proposal phase, we randomly augmented sentences with a random augmenter. Each sentence has a 30% error, and the augmenter is randomly chosen. Compared to this method, our current method guarantees that each word will equally be augmented with the same amount of error percentage, all the while maintaining the error percentage of a sentence to a certain percent. We also have finer control over which words will be augmented, specifically Vietnamese words.

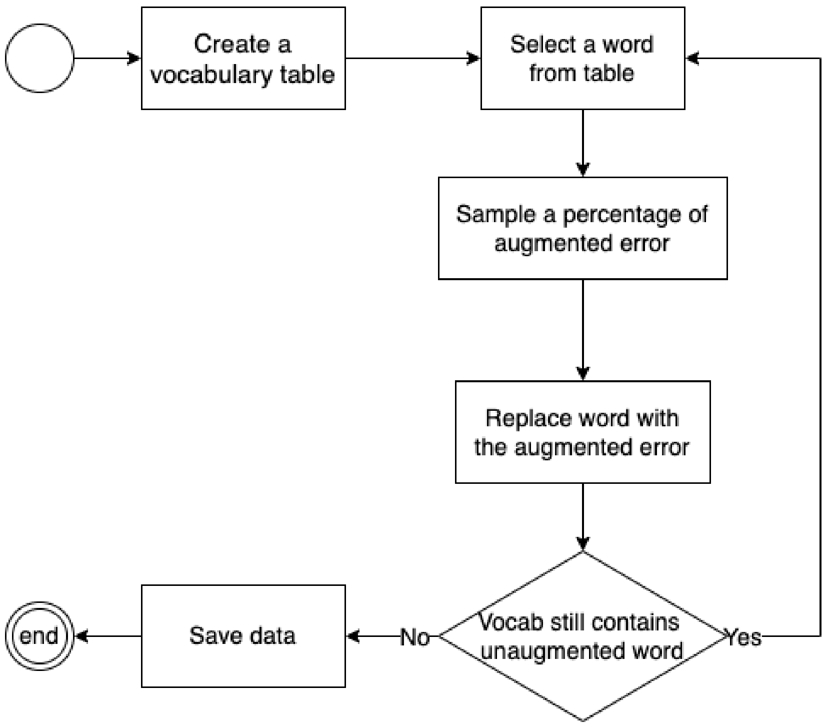

Create a vocabulary table
Select a word from table
Sample a percentage of augmented error
Replace word with the augmented error
Vocab still contains unaugmented word
Yes
No
Save data
end

Figure 4.10: Data augmentation pipeline

4.2.4 Overall Data

In total, we augmented the following:

Augmented Dataset	Pair of sentences
Training set	2.518.213
Validation set	171.910
Testing set	5.000

Table 4.3: Data for spelling correction

We realize our resource only allow for approximately 2.5 million pair of sentence, hence thats why we did not augmented more.

In addition to the augmented data, both Vietnam National University group and Zalo Corporation research group both publish a hand-labeled testing set for future study to compare.

Real-world Testing Dataset	Pair of sentences
VSEC testing set	9.341
Viwiki testing set	14.831

Table 4.4: Data for spelling correction

4.3 Correction Models

4.3.1 Baselines

4.3.1.1 Transformer

The Transformer model (Vaswani et al) [20] was originally proposed for the task of machine translation, which translate text from a language to another language. Noticing the potential of this model in the task of correction, we decided to use the Transformer model for this problem.

We chose the base implementation by Fairseq [41] for this model for the best optimization. And, to compare the effectiveness of different types of tokenizer, we decide to train the model with three types of tokenization: word, character and subword. For the subword tokenization, we limit the vocabulary size to 16000.

Configuration setting
For all three different type of tokenization of transformes, the configuration is as below:

- Encoder layer: 3
- Decoder layer: 3
- Encoder attention head: 8
- Decoder attention head: 8
- Encoder embedding dimension: 512
- Decoder embedding dimension: 512
- Encoder feed-forward dimension: 2048
- Decoder feed-forward dimension: 2048
- Drop out: 0.1
- Learning rate: $1 * 10^{-4}$
- Learning rate scheduler: inverse square root
- Batch size: 128

4.3.1.2 Soft-masked BERT

Soft-masked BERT is a model proposed in Chinese Spelling Correction. The model reaches State-of-the-art results for Chinese, so we decide to include the model for Vietnamese.

Configuration setting

- Learning rate: $1 * 10^{-5}$
- RNN layer: 2
- Feed forward dimension: 512
- Batch size: 32

4.3.1.3 Hard-masked XLMR

In the process of training the Soft-masked BERT model, by trial and error, we find out that in order to make this model perform well, an extensive amount of training time is needed due to its heavy architecture and large number of parameters (550M parameters).

With the same approach as Soft-masked BERT, we re-use the architecture of the model, called Hard-masked XLMR, with the only difference being that instead of using BERT to learn and correct the error, we use a pre-trained model that was trained with Masked Language Modelling for the corrector.

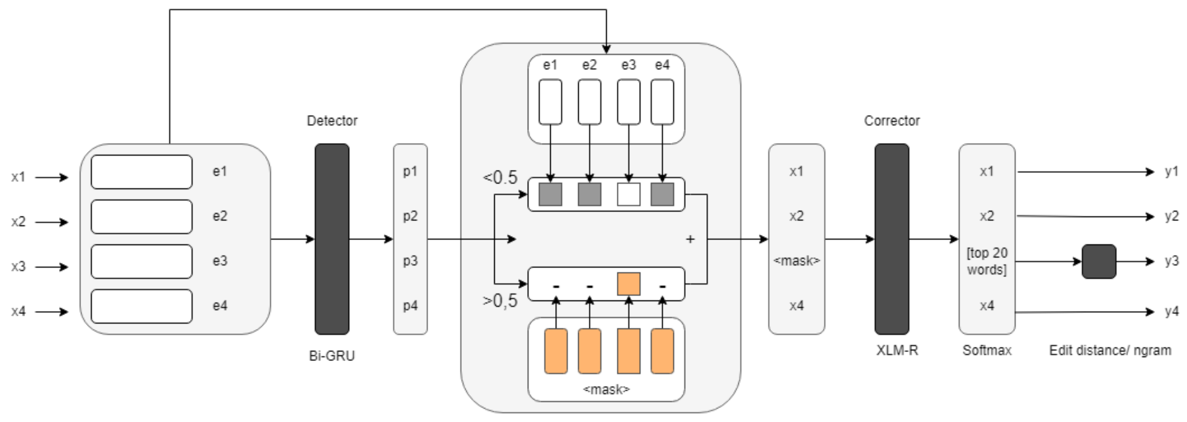


Figure 4.11: Hard-masked XLMR.

Detector
From the figure above, the detector of Hard-masked XLMR is the same as the soft-masked version, a Bi-GRU network. While the input of the detection network in soft-masked is the embeddings from BERT, we instead use a subword tokenizer trained with the SentencePiece package for ease of use. The output is also a sequence of labels where we use a sigmoid function to calculate the probability of the error tokens and use a threshold of 0.5 to manually mask the tokens.

Corrector
We chose pre-trained XLM-Roberta (Conneau et al.) [42] as our corrector. It is a cross-lingual masked language model based on RoBERTa and has been pre-trained on 100 different languages, including Vietnamese.

From the detector, a sentence with mask tokens is then processed to align with the model input settings, then pass these input into the model. A softmax function is used to find the probabilities for the candidate tokens so that we could choose 20 tokens with the highest probabilities.

For the task of choosing the appropriate token, we compute the Levenshtein similarity ratio of the incorrect token with each of the candidate tokens and eliminate all the tokens with a ratio less than 0.4. By doing this, we could assure that the candidate token is not completely different from that of the error token. Although, in many cases, several candidate tokens will have the same edit distance. Hence, a tri-gram model is used to score which token is suitable in the context of the input sequence and then pick the token with the highest score. Another version in which post-processing is not applied but instead uses the best candidate token via probabilities from the corrector will also be tested for comparison.

This approach, however, will not always correct the spelling mistake but rather choose the word that best suits the context based on its learning phase.

Configuration setting

- Learning rate: $1 * 10^{-5}$
- RNN layer: 2
- Feed forward dimension: 512
- Batch size: 32

4.3.2 Tokenization Repair

Of all the defined errors, the split token error (”tôi” -> ”tô i”) and the merged tokens error (”tôi đi học” -> ”tôiddi học”) are especially hard to correct. Keeping it can mess with the character information of each token, leading to false classifications. That is why we choose to correct these errors separately.

As mention from earlier, our approach is adapting a tokenization repair module to the Vietnamese language. The tokenization repair module purpose is, given a sequence of character with missing and spurious spacing, the model tries to predict a sequence with correct spacing.

We reused the same dataset for correction to train our tokenization repair module. Half of the dataset are errorful sentences, and the other half are error-less sentences. In total, the data consist of:

Augmented Dataset	Number of sentences
Training set	5.036.426
Validation set	343.820
Testing set	10.000

Table 4.5: Data for tokenization repair

4.3.3 BERT Fine-tuned

Inherits from the idea of Hieu et al. [2] paper, instead of training from scratch the whole transformer encoder, which takes more time and resources for the model to converge with large dataset, we use BERT as our encoder for extracting features vector from source sentence.

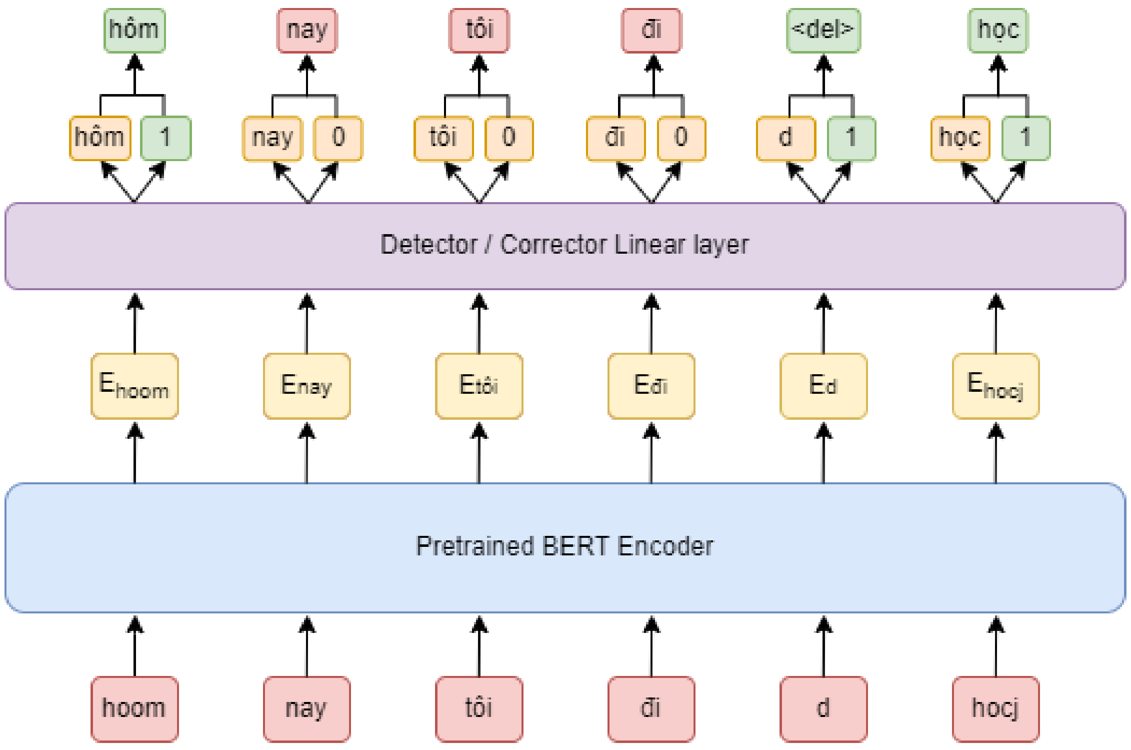


Figure 4.12: Our proposed BERT fine-tuned model

4.3.3.1 BERT embedding

We chose BERT as our encoder for extracting relevant information from the source sentence. To be more specific, PhoBERT base setting [43] was used due to its nature of already trained on large dataset of Vietnamese text, which already captures probably all the semantics and context of Vietnamese sentences. Hence, our job is to simply add fine-tuned layers for downstream tasks.

Given a source sequence $X = \{x_1, x_2, ..., x_n\}$, the input will go through a pretrained BERT subword tokenization to achieve source input ids $E^{src} = \{e_1, e_2, ..., e_n\}$ and attention mask $\{a_1, a_2, ..., a_n\}$. These inputs will then be feed into BERT encoder.

$$h^{src}_{1..n} = BertEncoder(E^{src}_{x_{1..n}})$$

The encoder learn to extract relevant information of the sentence and token and output hidden representation of each token. It is then forwarded to two classifiers, one for detection and the other for correction.

4.3.3.2 Detector

In the detection phase, our purpose is to detect which token has spelling errors. Hence, we cast the problem into a binary sequence classification task, a label **0** would mean that token should be kept, else a label **1** would indicate that the token is erroneous.

After getting the feature vectors for the BERT encoder, we take the last four hidden states of the output as our input vector for the detection phase by averaging the values in those four hidden states. A linear layer is then used for detection.

$$\mathcal{L}_d = -\frac{1}{N + 1e^{-5}}; \Sigma^{N-1}_{i=0} \, \Sigma^{1}_{j=0} \, y_{ij} \, logp_{ij}$$

4.3.3.3 Corrector

After getting the output of the detection phase, we forward that detection output vector into a linear layer with a softmax activation function. The output vector of that layer will then be concatenated with the mean feature vector of BERT encoder that was mentioned before. The intuition behind this is to incorporate the information from the detection phase, which we believe will enhance the result of correction. Another linear layer is added for classifying the word for each token.

Besides classifying the tokens with the predefined BERT vocabulary, we add a new special token called <*del*> to indicate the deletion of a token. Hence, if the corrector classifies a token as <*del*>, we would simply delete that token from the output sequence.

$$\mathcal{L}_c = -\frac{1}{N + 1e^{-5}}; \Sigma^{N-1}_{i=0} \, \Sigma^{V-1}_{j=0} \, y_{ij} \, logp_{ij}$$

4.3.3.4 Objective function

During the training phase, the cross-entropy loss is used for both the detector and corrector, with the addition is that the corrector loss will be computed using only the error token that is corrected, while the already non-error token is ignored. The final objective function would be the sum of both losses:

$$\mathcal{L} = \mathcal{L}_d + \mathcal{L}_c$$

where N is the total number of samples, V is the vocabulary size, y_{ij} and p_{ij} stand for ground-truth labels and prediction probability.

4.3.3.5 Configuration Setting

Here are the following configuration that we used for training the model.

- Leaning rate: $1 * 10^{-5}$
- Feed forward dimension: 512
- Batch size: 64
- Vocab size: 40000

4.4 Pipeline

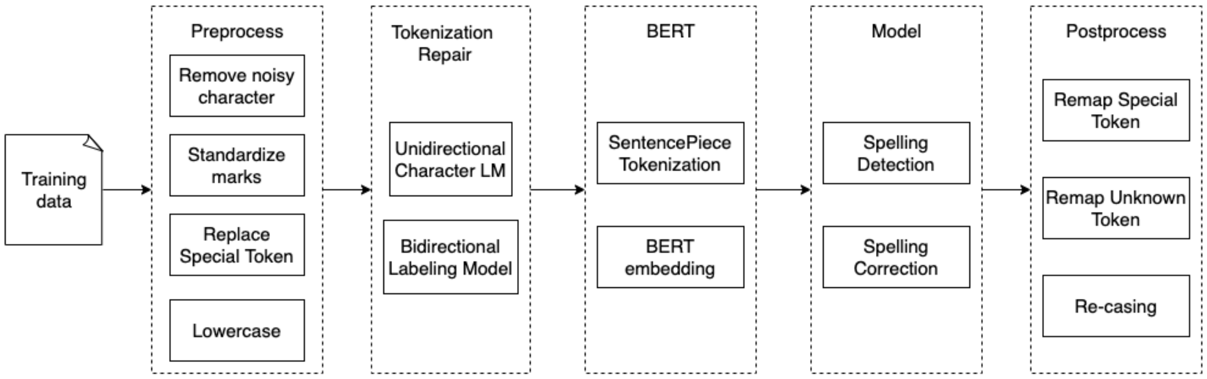


Figure 4.13: Spelling correction pipeline

Given a sentence with spelling errors, the pipeline is as follow: **Preproccess, Tokenization, Model inference, Postprocess**. For preprocess:

- **Remove noisy characters**: clean unnecessary characters such as emoji, leading spaces, etc.
- **Standardize mark**:the data is standardized using a mapping set between the telex syllable and the correct mark syllable. For example: “cuả” -> ”của”
- **Lowercase**: Keeping original casing with significantly increase vocabulary. Therefore, we lowercase tokens for a smaller and faster model.
- **Tokenization repair**: Sentences pass through the aforementioned tokenization repair model to fix whitespace errors, and fix merged tokens.
- **Replace Special Token**: some tokens like numbers and dates can increase vocabulary, all the while not adding context to the sentence, hence we can replace it with one special token like *date* or *number*.

In the tokenization phase, we use the default subword tokenizer used by PhoBert, which has a vocabulary size of 64000. After that, the tokens are fed to BERT model to retrieve embedding. average the sub-word representations to obtain the word representations. This is then fed to 2 linear layers, detection, and correction. The detection layer produces the probability of a token begin misspelled. Then, the correction layer will perform token classification to predict the correct word.

Finally, we perform post-process on the model output:

- **Remap Special Token**: replace special tokens like dates and numbers with the original tokens we have saved in preprocessing step.
- **Remap Unknown Token**: replace unknown token by aligning output with source sentence, then map all unknown tokens with the original tokens.
- **Recasing**: perform recasing for special cases like Capitalize, CAPLOCKS.

Let us take an example of a sequence-to-sequence model using sub-word tokenization:

Input Text	”Ngafy 14/3, học sinh toànquoosc được nghỉ học.”
Pre-processing	”ngafy **date** học sinh toàn quoosc được nghỉ học”
Tokenization	[”nga@@”,”f@@”,”y”, ”date”, ”học”, ”sinh”, ”toàn”, ”quoosc”, ”được”, ”nghỉ”, ”học”]
Model Inference	[”ngày”, ”date”,, ”học”, ”sinh”, ”toàn”, ”quốc”, ”được”, ”nghỉ”, ”học”]
Post-processing	[”Ngày”, ”14/3”,”,”, ”học”, ”sinh”, ”toàn”, ”quốc”, ”được”, ”nghỉ”, ”học”, ”.”]
Output Text	”Ngày 14/3, học sinh toàn quốc được nghỉ học.”

Table 4.6: Example of pipeline

Chapter 5

Result and Evaluation

5.1 Evaluation metrics

5.1.1 Detection and Correction

We discuss evaluation methods to measure how well the models perform. Spelling correction problem can be viewed as a binary classification of prediction, where the matching elements are considered as true predictions and the others, incorrect predictions.

In the past, studies for Vietnamese spelling correction often opt for word-level evaluation of precision, recall, and f-score [9, 8]. These metrics can be applied to both detection and correction tasks. In this work, we will use both the metrics that are provided by VSEC [1] and VIWIKI [2] paper.

$$DetectionPrecision = \frac{TrueDetections}{TotalErrorDetected} \tag{DP}$$

$$DetectionRecall = \frac{TrueDetections}{TotalActualErrors} \tag{DR}$$

$$DetectionF1 = \frac{2*DP*DR}{DP+DR} \tag{DF}$$

$$CorrectionPrecision = \frac{TrueCorrections}{TotalErrorDetected} \tag{CP}$$

$$CorrectionRecall = \frac{TrueCorrections}{TotalActualError} \tag{CR}$$

$$CorrectionF1 = \frac{2*CP*CR}{CP+CR} \tag{CF}$$

$$Accuracy_{\%detected} = \frac{TrueCorrections}{TrueCorrections+WrongCorrections} \tag{Acc$_d$}$$

$$Accuracy_{total} = \frac{TrueCorrections}{TrueCorrections+WrongCorrections+WrongDetections} \tag{Acc$_t$}$$

5.1.2 Tokenization Repair

Given a corrupt input text I, a ground truth text T, and a predicted text P, the algorithm predicts a set of space insertions and deletions that aims to transform C into T. The paper uses two metrics for the evaluation: F-score and sequence accuracy.

F-score: We define edits(A, B) as the space insertions and deletions that transform A into B. To achieve alignment on both sequences, we can use Levenstein dynamic alignment algorithm on the character level. If we let $C = edits(C, T)$ be the ground truth edit operations and $P = edits(C, P)$ the predicted edit operations, the number of true positives is $TP = |C \cap P|$, the number of false positives is $FP = |P \setminus C|$ and the number of false negatives is $FN = |C \setminus P|$. The F-score is computed as:

$$F - score = \frac{TP}{TP + \frac{1}{2} * (FP + FN)}$$

Sequence accuracy: sequence accuracy measures what fraction of exact correction (P = T) over the whole testing set.

5.2 Result And Evaluation

5.2.1 Result on our test set

We tested our models against three test sets. Firstly, we augmented a small test set of 5000 sentences that has not been seen in the training set. Our augmentation includes all errors that are included in the training set.

Models	Detect			Correct				
	DP	DR	DF	CP	CR	CF	Acc_t	Acc_d
Transformers-Char	96.22	73.24	83.17	71.73	54.59	62.00	71.73	74.54
Transformers-Word	71.89	84.23	77.57	40.00	46.86	43.16	40.00	55.63
Transformers-Subword	98.24	85.17	91.50	85.29	73.49	78.95	85.29	86.29
Hard-Masked XLMR	97.82	93.26	96.48	46.33	55.6	50.54	46.33	53.66
Soft-Masked BERT	96.36	93.92	95.12	72.35	68.73	70.49	72.35	75.12
Our Model	97.20	96.42	95.62	91.70	89.79	90.73	91.70	93.12
+ Tokenization Repair	**98.36**	**97.31**	**97.83**	**92.07**	**91.09**	**91.58**	**92.07**	**93.61**

Table 5.1: Augmented test set

Bold for highest score, underline for second highest

Out of the baseline transformers, we can see that subword tokenization clearly outperforms other character and word level models at 91.50% in terms of detection. We predicted the word-level model performs the worst since multiple non-word errors can be generated from one correct word, hence our dataset can not cover all cases of different errors. Character tokenization does perform better than word tokenization, although we note that the inference speed of both word and character models performs more poorly, especially the latter. In terms of correction, subword tokenization also achieves a notable result, where its accuracy surpasses other transformers baselines.

Soft-masked BERT model (which achieved SOTA in Chinese Spelling Correction) also performs poorly as it also uses word-level tokenization, which leads to a very large vocabulary and slower convergence rate. Its modified version, Hard-masked XLMR, does not achieve any higher result than the base model since it only corrects the mask tokens based on the surrounding context rather than the error token information. However, both models have improved the detection phase by incorporating a detection layer.

Our main model detection achieves an f-score around 95.62%, which outperforms other baselines. The results further improve as we add the tokenization repair module which increases to 97.83%. In terms of correction, we achieve 93.61% accuracy within the detected token and 92.07% accuracy in general. However, in real-life testing, our model still meets some limitations such as teen-codes, shortcut terms, and foreign words which we will touch on in the discussion.

We input our own test cases and observe how our model would perform. All in all, the model performs fairly accurate when there are one to two errors per sentence.

	Sentence
Incorrect	Cờ bạc, rượi chè là thối hư tật xấu cần bài trừ.
Correct	Cờ bạc, rượu chè là thói hư tật xấu cần bài trừ.
Predict	Cờ bạc, rượu chè là thói hư tật xấu cần bài trừ.
Incorrect	Cây bàng tto lớn như một vệ sĩ lặng lec âm thầm.
Correct	Cây bàng to lớn như một vệ sĩ lặng lẽ âm thầm.
Predict	Cây bàng to lớn như một vệ sĩ lặng lẽ âm thầm.

Table 5.2: Example of correction

However, for test cases that contain almost-to-none correct words, our model struggles to capture the context of the sentence. We will explain the reasons behind this result in the discussion section.

	Sentence
Incorrect	ngay trong khi ngu duoc gay ra boi su nghet khi quan
Correct	ngáy trong khi ngủ được gây ra bởi sự nghẹt khí quản
Predict	ngay trong khi ngũ được gây ra bởi sư nghẹt khi quân

Table 5.3: Example of correction (2)

5.2.2 Result on VSEC test set

Next, we evaluate our model on VSEC public test set by the Vietnam National University team.

Models	**Detect**			**Correct**				
	DP	DR	DF	CP	CR	CF	Acc_t	Acc_d
Transformers-Char	37.39	56.61	45.03	26.13	39.57	31.48	26.14	69.91
Transformers-Word	21.69	66.68	32.73	14.64	45.01	22.10	14.64	67.51
Transformers-Subword	82.00	76.95	79.39	71.82	67.40	69.56	71.82	87.59
Hard-Masked XLMR	85.03	78.11	82.24	53.91	42.44	47.49	53.91	68.78
Soft-Masked BERT	81.55	79.2	80.35	59.8	56.76	58.24	59.8	78.43
Our Model	92.20	85.23	88.58	83.70	77.36	80.40	83.70	90.77
+ Tokenization Repair	**93.59**	**85.49**	**89.36**	86.23	**78.76**	**82.33**	**86.23**	**92.76**
VSEC (2M)	89.1	76.9	82.6	82.6	71.3	76.5	-	-
VSEC (5M)	93.1	81.3	86.8	**87.4**	76.3	81.5	-	-

Table 5.4: VSEC test set

Bold for highest score, underline for second highest

Overall, each sentence in the VSEC test set were hand-labeled and contains one to three mistakes. The set includes all of our defined errors, plus some minor cases of adding a token, which our model fails to correct due to the fact that we only predict whether to keep or change the corresponding token.

The table above includes the VSEC model trained on a two million sentence dataset and a five million sentence dataset. In the detection task, our model performs better than the paper's best result. Our precision score is higher by 0.49%, while our recall score is higher by 4.19%. Overall, the f-score with our model increased by 2.56%. The VSEC model is trained on double the amount of data that we have; when comparing to the equivalent 2 million VSEC model, our model outperform for the detection task.

For the correction task, although our model performs worse on correction by 1.17%, we outperform on recall performance and f-score performance, by 2.46% and 0.83%, respectively. This indicates that within the correct detection, our model are less likely to suggest the right token; nonetheless, we are still able to capture and suggest more correct token. Once again our model outperforms all scores compared to the VSEC model on a two million dataset.

	Sentence
Incorrect	Thông qua công tác tuyên truyền, vận động này phụ huynh sẽ hiểu rõ hơn tầm quan trọng của việc giáo dục ý thức bảo vệ môi trường cho trẻ không phải chỉ ở phía nhà trường mà còn ở gia đình, góp phần vào việc gìn giữ môi trường sanh, sạch, đẹp.
Correct	Thông qua công tác tuyên truyền, vận động này phụ huynh sẽ hiểu rõ hơn tầm quan trọng của việc giáo dục ý thức bảo vệ môi trường cho trẻ không phải chỉ ở phía nhà trường mà còn ở gia đình, góp phần vào việc gìn giữ môi trường xanh, sạch, đẹp.
Predict	Thông qua công tác tuyên truyền, vận động này phụ huynh sẽ hiểu rõ hơn tầm quan trọng của việc giáo dục ý thức bảo vệ môi trường cho trẻ không phải chỉ ở phía nhà trường mà còn ở gia đình, góp phần vào việc gìn giữ môi trường xanh, sạch, đẹp.
Incorrect	Các bất cập đó đã làm nảy sinh ra việc sử dụng đất chưa hợp lí, ảnh hưởng từ quá trình công nghệp hóa làm suy giảm diện tích đất nông nghiệp.
Correct	Các bất cập đó đã làm nảy sinh ra việc sử dụng đất chưa hợp lí, ảnh hưởng từ quá trình công nghiệp hóa làm suy giảm diện tích đất nông nghiệp.
Predict	Các bất cập đó đã làm nảy sinh ra việc sử dụng đất chưa hợp lí, ảnh hưởng từ quá trình công nghiệp hóa làm suy giảm diện tích đất nông nghiệp.
Incorrect	Thế vào điều kiện dU < d < (4 – dU), ta thấy 1,735 < 1,739 < 2,265 thỏa điều kiện.
Correct	Thế vào điều kiện dU < d < (4 – dU), ta thấy 1,735 < 1,739 < 2,265 thỏa mãn điều kiện.
Predict	Thế vào điều kiện dU < d < (4 – dU), ta thấy 1,735 < 1,739 < 2,265 thỏa điều kiện.

Table 5.5: Example of correction from VSEC set.

5.2.3 Result on VIWIKI test set

Models	**Detect**			**Correct**	
	DP	DR	DF	Acc_t	Acc_d
Transformers-Char	1.33	49.08	2.6	0.65	48.93
Transformers-Word	1.08	67.17	2.14	0.46	42.81
Transformers-Subword	4.38	58.64	8.16	3.36	76.73
Hard-Masked XLMR	23.84	60.67	34.22	8.96	48.32
Soft-Masked BERT	21.72	58.76	31.72	12.38	60.54
Our Model	48.40	61.65	54.23	45.30	93.61
+ Tokenization Repair	<u>49.50</u>	<u>63.87</u>	<u>52,45</u>	<u>44.96</u>	<u>93.61</u>
Viwiki	**66.96**	**70.92**	**68.88**	**64.29**	**96.01**

Table 5.6: Viwiki test set

Bold for highest score, underline for second highest

Viwiki dataset comprised of all hand-labeled texts crawled from Vietnamese Wikipedia. Unfortunately, our model performs much worse compared to the paper model. In general, test cases in this test set not only includes Vietnamese spelling error, but also English spelling errors, which tremendously reduce our accuracy. All the models of our choices have high recall scores and low precision scores in the detection phase, which implies that false positive samples are significant to that of false negative. In other words, our model predicts a large portion of tokens as erroneous while in fact, they are not. These wrong detections also affect our accuracy of correction in total. Although the accuracy in the detected portion is 89.27%, which means, for every token that is an actual error, our model has an average accuracy of 89.27% of correcting them to true tokens.

Our model performance drop tremendously when tested against Viwiki dataset by Zalo. The research group crawled data from Wikipedia, a free-to-edit data source whose content is created by volunteers around the world, hence its contents are less monitored and checked for spelling. Nonetheless, our closer inspection reviews that this test set contains some interesting characteristics compared to VSEC.

Firstly, the research team gathered text from different various topics such as Vietnamese historical paragraphs, chemistry, technical site building, etc. which contains many domain-specific terms. For example, chemical formulas such as ”nitroglycerol” and ”trinitroglycerin” or English words can easily confuse both the tokenizer and correction model. In historical texts, our model does not perform well for ancient phrases and human names. We see this as room for improvements moving forward to improve the quality of our training set and expand to various domains, as well as looking for ways to incorporate entity knowledge, such as names, and organization, into our model.

Historical text	**Incorrect**: Dụ Tông bèn xuống chiếu cho An phủ sứ các lộ đem quân các đội phong đoàn đi bắt giặc cướp, đến năm 1360 thì Bệ bị giết, đồng đảng hơn 30 người đều bị xử tử. **Corrected**: Dụ Tông bèn xuống chiếu cho An phủ sứ các lộ đem quân các đội phong đoàn đi bắt giặc cướp, đến năm 1360 thì Bệ bị giết, đồng đảng hơn 30 người đều bị xử tử. **Predicted**: Dụ Tông bèn xuống chiếu cho An phủ sứ các lộ đem quân các đội binh đoàn đi bắt giặc cướp, đến năm 1360 thì Bệ bị giết, đồng đẳng hơn 30 người đều bị xử tử.
Chemical names	**Incorrect**: Các ”khoáng vật cacbonat ” bao gồm các khoáng vật chứa anion (CO3) 2 **Corrected**: Các ”khoáng vật cacbonat ” bao gồm các khoáng vật chứa anion (CO3) 2 **Predicted**: Các ”khoáng vật cacbonat ” bao gồm các khoáng vật chứa anion (CỎ) 2

Table 5.7: Example of domain-specific keyword

Secondly, when looking at the hand-labeled annotations, we notice that this test set contains error correction beyond our scope of Vietnamese error. For example, it also corrects English words and names such as ”Harrrry” to ”Harry”. Also, it includes a vague shortcut correction, for example: ”ko” is corrected to ”không”, while ”ts.”, which stands for ”tiến sĩ”, are not corrected. Another type of error is an insert token error.

English name	Incorrect: Kể từ khi đưa Harrry vào thế giới pháp thuật, Hagrid đã trở thành một trong những người bạn thân thiết nhất của Harry. Corrected: Kể từ khi đưa Harry vào thế giới pháp thuật, Hagrid đã trở thành một trong những người bạn thân thiết nhất của Harry. Predicted: Kể từ khi đưa Harrry vào thế giới pháp thuật, Hagrid đã trở thành một trong những người bạn thân thiết nhất của Harry.
Shortcuts	Incorrect: ... và bảo rằng cậu phải tìm cách trở về dòng thời gian Beta, nơi Mayuri sẽ ko chết. Corrected: ... và bảo rằng cậu phải tìm cách trở về dòng thời gian Beta, nơi Mayuri sẽ không chết. Predicted: ... và bảo rằng cậu phải tìm cách trở về dòng thời gian Beta, nơi Mayuri sẽ ko chết.

Table 5.8: Example of English word and Shortcut

Finally, we note that although the text was hand-labeled by separate teams, we still manage to find some diacritic and spelling errors during the process. This affects our false positive rate, which in terms decreases model performance. Ultimately, this test set does not reflect our true target. Nevertheless, we plan to expand our model scope to other types of errors and tackle this test set in a different paper.

5.2.4 Evaluation of tokenization repair module

We also measure our tokenization repair accuracy, which achieves a high score on our augmented test set. Unfortunately, we could not find a real test set in Vietnamese to evaluate.

Model	F-score	Sequence Accuracy
UNI	93.7	97.1
BID	97.5	98.1

Table 5.9: Tokenization repair result

UNI: Unidirectional Language Model, BID: Bidirectional Label

On its own, the tokenization repair module only handle restoring the tokens within a sentence.

	Sentence
Incorrect	Khôn gxar rác để giữ một môi rường xanh, xạch, đẹp
Predict	Không xar rác để giữ một môi rường xanh, xạch, đẹp
Incorrect	tôiddens đây v ới tu cách là người phán xử.
Predict	tôi ddens đây với tu cách là người phán xử.

Table 5.10: Example of tokenization repair standalone

In the first example, ”Khôn gxar rác” is actually ”Không xar rác”, where the token ”Không” was split to ”Khôn g”, and sub-token ”g” are merged with ”xả”, resulting in ”gxar” due to the

telex typing mechanism. This potentially enhance our results, as the character information ’g’ when merged with ”xar” may confuse our correction model. The second example show two errors, the first one being a merged syllables, and a second being a split token. In both cases, the tokenization repair module is able to correctly predict the space insertion and deletion of space in the presents of spelling mistakes.
This module is a crucial improvement for our fine-tuned model as it does not have the ability to separate merged syllables, such as ”tôiddens”, or merge split tokens such as ”v” and ”ới”. We take a few examples to demonstrate how our tokenization repair module enhances our correction module.

	Sentence
Original	qua công tác tuyêntryuen, phụ huynh cần phảin ắm rõ được tầm quan trọng trong việc nhận thức của con.
Without TR	qua công tác tuyêntryuen, phụ huynh cần phải nắm rõ được tầm quan trọng trong việc nhận thức của con.
With TR	qua công tác tuyên truyền, phụ huynh cần phải nắm rõ được tầm quan trọng trong việc nhận thức của con.
Original	tôi dduow c đi học
Without TR	tôi đưa c đi học
With TR	tôi được đi học

Table 5.11: Example of correction with tokenization repair

TR: tokenization repair

5.3 Effect of Hyperparameter

We also investigate further the effect of a larger dataset on the accuracy of the proposed method. Since we do have a limited resource, our training set max out at around 2.500.000 sentences. The result is displayed in the table below.

Training Set	**Detect**			**Correct**	
	DP	DR	DF	Acc_t	Acc_d
500k	90.37	89.98	90.17	85.70	90.60
1M	92.98	92.40	92.69	87.13	91.24
2M	94.50	93.76	94.13	89.91	92.08
2.5M	**98.36**	**97.31**	**97.83**	**92.07**	**93.61**

Table 5.12: Effect of training set size on accuracy

Chapter 6

Discussion and Future Work

6.1 Discussion on evaluation results

Failure in predictions
As mentioned above in the evaluation section, our model does not yield good results with sentences that contain too many error tokens. Several causes lead to this issue are as followed:

- Our augmentation strategy ensures each sentence will have less than 30% of error tokens. This affects our model drastically when encountering error-only sentences.
- In the detection phase, there is still a lot to improve with the imbalanced classification problem, where, most of the time, tokens do not contain errors. Though, this result is acceptable rather than detecting correct word as wrong.

Through observation, while the detector cannot recognize the errors, the corrector is still able to predict them with the correct tokens. We could solve this problem by setting a certain probability threshold. If the corrector prediction is overly confident, we will set that token to the predicted token. Though, choosing a particular threshold is often subjective and does not generalizes well.

Initial approach for improving Viwiki test set score
On several attempts, we try to incorporate a pre-trained named entity recognition (NER) system for increasing the detection score. Our approach is to ignore the person's name, geographical location, and organization. In particular, we use pre-trained model provided by NLTK [44] and VnCoreNLP [45]. However, in testing, both pre-trained model has several flaws while detecting entities.

- For NLTK package, we use the pre-trained English model for detecting foreign language entities. Since the model is trained in English only, it often classifies non-entities Vietnamese words as person names and organization.
- As for the VnCoreNLP package, the model works incredibly well since it is trained on the Vietnamese dataset. Though, in several cases where the names contain errors such as "Nguyeenx Xuân Phúc", ignoring these errors would have affected our model's overall performance. Another pre-trained Vietnamese language model was intentionally used for scoring the sentence fluency whether we ignore or fix these errors would be beneficial. Nonetheless, publicly available Vietnamese pre-trained language model is limited, we could not find any acceptable model that could be used. Not to mention, relying on too many models does not always yield good results.

6.2 Future works

Noticing our drawbacks, we have planned out several works that needed to be addressed for improving the performance.

- On the dataset level, expanding the size of the dataset with more topics while incorporating more types of errors is a must to apprehend better real-world data.
- Besides raw text features, integrating entity information or part of speech tags would be beneficial for the model to learn more knowledge. Hence, enhancing the detection and correction performance.
- The problem of imbalance classification remained in the detection phase, where most of the tokens are not erroneous. Applying more techniques is required to improve precision.
- Researching on new features such as detecting sentence boundaries, correct punctuation marks, and adding missing words that match the sentence context where possible.

Chapter 7

Summary

In this thesis, we tackle our goal for the thesis which is Vietnamese Contextual Spelling Correction using a deep-learning approach. Overall, we achieve some positive results as well as counter some setback.

7.1 Achievements

What we have achieved includes:

- Demonstrated that leveraging a pre-trained transformer encoder such as BERT can be effective for the correction task.
- Adapted a tokenization repair module that enhance correction model.
- The final model are able to fix errors within our scope.
- Achieve a higher detection and correction accuracy on VSEC test set by Vietnamese National University group.
- Build a demonstration app that showcase what we have achieved

7.2 Limitations

However, we encounter several setbacks that we need to address in the future:

- Tokenization repair can be misled when testing against English tokens, long chemical formula, etc.
- Errorful tokenization repair can mislead the correction model to wrongly predict token.
- Limited dataset size does not ensure a variety of topic coverage.
- Not capturing entity informations (name, location, etc) can leads to wrong prediction.
- Error scope does not contain english errors, leads to poor score on the Viwiki dataset.

References

[1] Dinh-Truong Do et al. *VSEC: Transformer-based Model for Vietnamese Spelling Correction*. 2021. arXiv: 2111.00640 [cs.CL].

[2] Hieu Tran et al. *Hierarchical Transformer Encoders for Vietnamese Spelling Correction*. 2021. doi: 10.48550/ARXIV.2105.13578. url: https://arxiv.org/abs/2105.13578.

[3] Sai Muralidhar Jayanthi, Danish Pruthi, and Graham Neubig. "NeuSpell: A Neural Spelling Correction Toolkit". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 158–164. doi: 10.18653/v1/2020.emnlp-demos.21. url: https://www.aclweb.org/anthology/2020.emnlp-demos.21.

[4] Keisuke Sakaguchi et al. *Robsut Wrod Reocginiton via semi-Character Recurrent Neural Network*. 2016. doi: 10.48550/ARXIV.1608.02214. url: https://arxiv.org/abs/1608.02214.

[5] Erik Jones et al. *Robust Encodings: A Framework for Combating Adversarial Typos*. 2020. doi: 10.48550/ARXIV.2005.01229. url: https://arxiv.org/abs/2005.01229.

[6] Shaohua Zhang et al. *Spelling Error Correction with Soft-Masked BERT*. 2020. arXiv: 2005.07421 [cs.CL].

[7] Jinhua Zhu et al. *Incorporating BERT into Neural Machine Translation*. 2020. arXiv: 2002.06823 [cs.CL].

[8] Phuong H. Nguyen et al. "Vietnamese spelling detection and correction using Bi-gram, Minimum Edit Distance, SoundEx algorithms with some additional heuristics". In: *2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*. 2008, pp. 96–102. doi: 10.1109/RIVF.2008.4586339.

[9] Huong Nguyen et al. "Using Large N-gram for Vietnamese Spell Checking". In: *Advances in Intelligent Systems and Computing* 326 (Jan. 2015), pp. 617–627. doi: 10.1007/978-3-319-11680-8_49.

[10] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[11] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. doi: 10.1162/neco.1997.9.8.1735.

[12] Mike Schuster and Kuldip Paliwal. "Bidirectional recurrent neural networks". In: *Signal Processing, IEEE Transactions on* 45 (Dec. 1997), pp. 2673–2681. doi: 10.1109/78.650093.

[13] Abdel-rahman Mohamed et al. "Deep bi-directional recurrent networks over spectral windows". In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. 2015, pp. 78–83. doi: `10.1109/ASRU.2015.7404777`.

[14] Zhiyong Cui et al. *Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction*. 2019. arXiv: `1801.02143 [cs.LG]`.

[15] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: `1409.3215 [cs.CL]`.

[16] Seiya Tokui et al. "Chainer: A Deep Learning Framework for Accelerating the Research Cycle". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2019, pp. 2002–2011.

[17] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: `1810.04805 [cs.CL]`.

[18] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. doi: `10.18653/v1/P16-1162`. url: `https://aclanthology.org/P16-1162`.

[19] Philip Gage. "A new algorithm for data compression". In: *The C Users Journal archive* 12 (1994), pp. 23–38.

[20] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: `1706.03762 [cs.CL]`.

[21] Kyunghyun Cho et al. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. doi: `10.3115/v1/W14-4012`. url: `https://aclanthology.org/W14-4012`.

[22] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: `1409.0473 [cs.CL]`.

[23] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: `1508.04025 [cs.CL]`.

[24] Jianpeng Cheng, Li Dong, and Mirella Lapata. *Long Short-Term Memory-Networks for Machine Reading*. 2016. arXiv: `1601.06733 [cs.CL]`.

[25] Matthew E. Peters et al. "Deep contextualized word representations". In: *CoRR* abs/1802.05365 (2018).

[26] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).

[27] Jeremy Howard and Sebastian Ruder. "Fine-tuned Language Models for Text Classification". In: *CoRR* abs/1801.06146 (2018).

[28] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016).

[29] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019).

[30] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *CoRR* abs/1906.08237 (2019).

[31] Zihang Dai et al. “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”. In: *CoRR* abs/1901.02860 (2019).

[32] Zhenzhong Lan et al. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In: *CoRR* abs/1909.11942 (2019).

[33] Peter Norvig. *How to Write a Spelling Corrector*. url: `http://norvig.com/spell-correct.html`.

[34] Hannah Bast, Matthias Hertel, and Mostafa M. Mohamed. “Tokenization Repair in the Presence of Spelling Errors”. In: *Proceedings of the 25th Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2021. doi: `10.18653/v1/2021.conll-1.22`. url: `https://doi.org/10.18653%2Fv1%2F2021.conll-1.22`.

[35] Kenji Imamura and Eiichiro Sumita. “Recycling a Pre-trained BERT Encoder for Neural Machine Translation”. In: *Proceedings of the 3rd Workshop on Neural Generation and Translation*. Hong Kong: Association for Computational Linguistics, Nov. 2019, pp. 23–31. doi: `10.18653/v1/D19-5603`. url: `https://aclanthology.org/D19-5603`.

[36] Masahiro Kaneko et al. *Encoder-Decoder Models Can Benefit from Pre-trained Masked Language Models in Grammatical Error Correction*. 2020. arXiv: `2005.00987 [cs.CL]`.

[37] Karen Kukich. *Techniques for automatically correcting words in text Share on*. 1992. doi: `146370.146380`.

[38] Cao Hong Nga et al. “Deep Learning Based Vietnamese Diacritics Restoration”. In: *2019 IEEE International Symposium on Multimedia (ISM)*. 2019, pp. 331–3313. doi: `10.1109/ISM46123.2019.00074`.

[39] Thai-Hoang Pham, Xuan-Khoai Pham, and Phuong Le-Hong. *On the Use of Machine Translation-Based Approaches for Vietnamese Diacritic Restoration*. 2017. arXiv: `1709.07104 [cs.CL]`.

[40] Hung Bui. “Vietnamese Diacritics Restoration Using Deep Learning Approach”. In: Nov. 2018, pp. 347–351. doi: `10.1109/KSE.2018.8573427`.

[41] Myle Ott et al. “fairseq: A Fast, Extensible Toolkit for Sequence Modeling”. In: *Proceedings of NAACL-HLT 2019: Demonstrations*. 2019.

[42] Alexis Conneau et al. “Unsupervised Cross-lingual Representation Learning at Scale”. In: *CoRR* abs/1911.02116 (2019).

[43] Dat Quoc Nguyen and Anh Tuan Nguyen. “PhoBERT: Pre-trained language models for Vietnamese”. In: *CoRR* abs/2003.00744 (2020). arXiv: `2003.00744`. url: `https://arxiv.org/abs/2003.00744`.

[44] Steven Bird and Edward Loper. “NLTK: The Natural Language Toolkit”. In: *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 214–217. url: `https://aclanthology.org/P04-3031`.

[45] Thanh Vu et al. “VnCoreNLP: A Vietnamese Natural Language Processing Toolkit”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 56–60. doi: `10.18653/v1/N18-5012`. url: `https://aclanthology.org/N18-5012`.