**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**SCHOOL OF INFORMATION TECHNOLOGY AND COMMUNICATION**

**********


# AI PROJECT REPORT

***PROJECT: Vietnamese Spelling Correction using Transformer with Tokenization Repair***

**Instructor: Assoc.Prof. Phạm Văn Hải**

**Students:** Group 2

| | |
|---|---|
| Nguyễn Đình Duy | 20190096 |
| Cao Tiến Trung | 20190098 |
| Trần Thế Thịnh | 20194853 |
| Bùi Thị Phương Anh | 20150022 |
| Vũ Thanh Hải | 20194756 |

**Hà Nội, tháng 12 năm 2022**

# Vietnamese Spelling Correction using Transformer with Tokenization Repair

Instructor: Assoc.Prof. Hai Pham Van

Students: Nguyen Dinh Duy, Cao Tien Trung, Tran The Thinh, Bui Thi Phuong Anh, Vu Thanh Hai

ICT K64, School of Information Technology and Communication, Hanoi University of Science and Technology, Hai Ba Trung, Hanoi, Vietnam.

**Abstract**: Spelling Correction is a challenging task in Natural Language Processing (NLP) which have crucial influence on downstream task. It also has many applications in real-world such as editors like Microsoft Word which improve user's productivity in typing documents, search engine or Post-OCR. In this project we will self-implement a deep learning model named Transformer with Tokenization Repair in master thesis of Sebastian Walter [2] and apply to Vietnamese language to deal with both word errors and white space errors.

**Keywords**: spelling correction, transformer, tokenization repair, spelling correction, post-ocr.

## 1. Introduction

Human's brains are good at languages, can easily understand a sequence of texts even if it is corrupted. However, when typing, there exists probability of misspelling words that in some case hard to find due to limitation of human eyes. Those cases need to thoroughly scan through the whole to find potential errors which are time-consuming. Moreover, some public search engine query logs show that large amount of query entered in are misspelling, which require enhance processing to enhance user's experience. Another problem that leads to the need of spelling correction is that OCR system usually make mistakes in accent errors (dẫu -> dẩu) or white space errors (thành phố -> thànhphố) or both (thành phố - thànphố). Same problem for voice recognition appear usually. That's why people attempt to make computers able to automatically correct spelling errors.

Recently, there are various papers working with spell corrections. But there are few numbers for Vietnamese. The type of errors they are usually working with is real-word (da -> gia) or non-word (việt - > vieets) but not combine with white space errors. It results in the length of input sequences and output sequences are the same. This is convenient for calculate metrics like F1 or accuracy but

cannot address the errors like adding redundant words or remove random words in the sequences and the popular error in OCR is white space errors. In this project, we will implement and train transformer model that use both encoders and decoders to generate output sequence to deal with both word errors and white space errors.

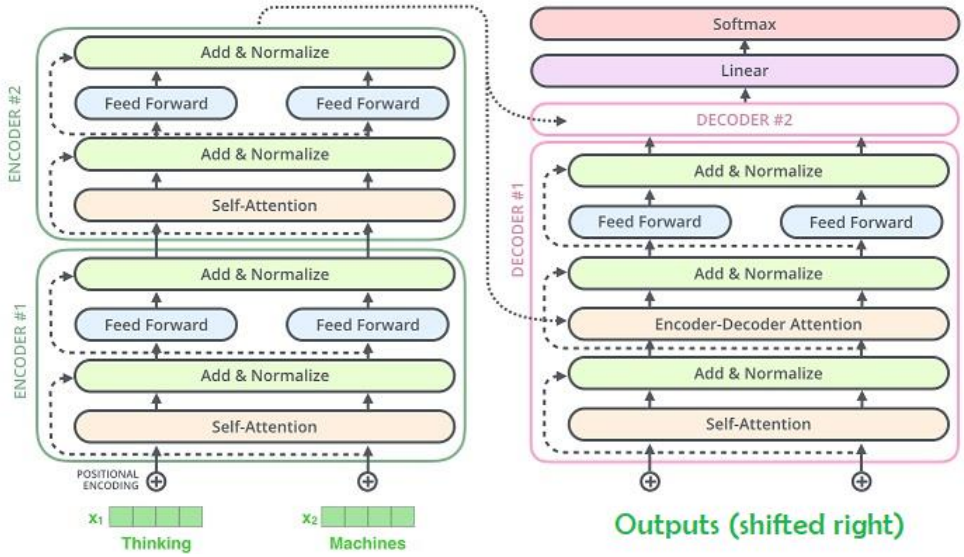
Further, special characteristic of spelling correction task is that there are not realistic datasets for training but is synthetically generated from a golden corpus or a random corpus. Papers usually not thoroughly describe their methods to generate data or public it. So, we implemented a confusion-set based and simple rule-based to generate training examples.

We summary our works as follows: (1) Confusion-set and rule-based synthetic training examples. (2) Implement and train our own Transformer with Tokenization Repair. (3) Evaluation of model on public datasets (VSEC) and our synthetic dataset. (4) A Website for spelling corrector demonstration of Speliing Correction API.

## 2. Related Work

### 2.1 Transformer

Transformer is proposed in paper Attention is all you need [1]. The novel architecture with its attention mechanism has proved its robustness in NLP especially Machine Translation. The idea is that instead of remembering the whole sentence and then generate word by word for the translate sequence, transformer use attention that allow it to focus on parts of input sequence while predicting output sequence autoregressive.



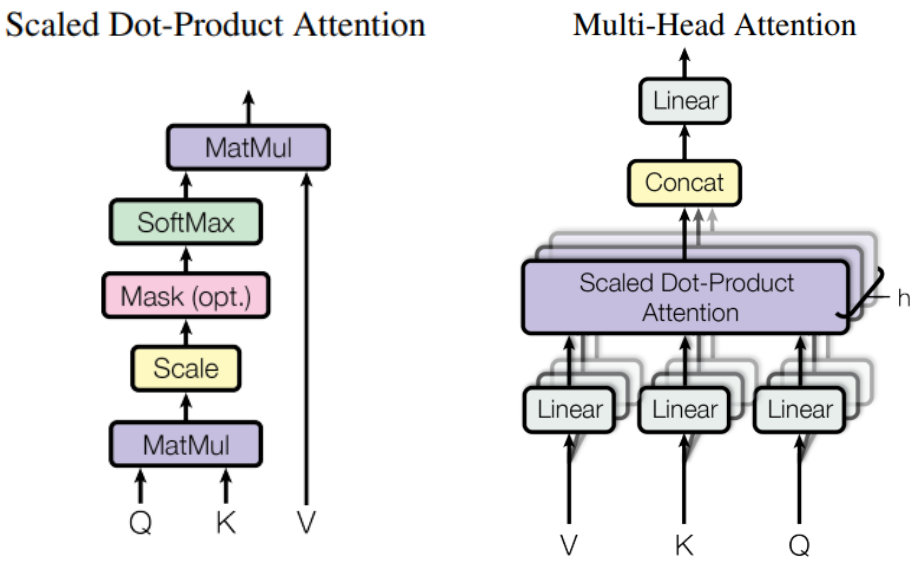

Transformer architecture contains Encoder and Decoder.

Encoder is a stack of **m** encoder layers. Each encoder layer includes 2 sub-layers. The first layer is Multi-Head Self-Attention and the second one is Fully Connected Feed-Forward. And there is residual connection per sub-layer which will follow by Layer Norm after it is added to output of sub-layer. Input feed to encoder is in parallel, go through Embedding layer to be converted into vectors and that is decorated by Positional Encoding to integrate position information.

Decoder is also a stack of n decoder layers. Architecture is same with Encoder except it add 1 more sub-layer at first position. It is called Masked Multi-Head Attention which makes use of attention mask to only allow unmask tokens to attend in attention. Because at $i$ time step of decoder, it should only know the words at position lower than $i$. This mechanism will mask the token over that position so that attention is applied only to the word that position is lower than $i$. Residual is also used. Input feed to decoder, however, isn't feed in parallel but autoregressively feed in at each time step which is the output of decoder at previous time step (**Start of Sentence** Token is be used for the first step). This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as RNN, … It is also gone through Positional Encoding and Embedding Layer.

Attention mechanism is the most important part of transformer. An attention function will be use, is described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. When a word does attention, it will use its Q matrix as query to search for content in other words in the sequence which is assumed to be store as K and V pair. All words in sentence have its own K(key) and V(value). This output is vector of weights (0.0 -> 1.0), allow model to know which words it should make attention on depend on the value. The Key/Value/Query concept is analogous to retrieval systems. For instance, when you search for videos on YouTube, the search engine will map your query (text in the search bar) against a set of keys (video title, description, etc.) associated with candidate videos in their database, then present you the best matched videos (values).

Original paper use Scaled Dot-Product Attention as function for attention mechanism.



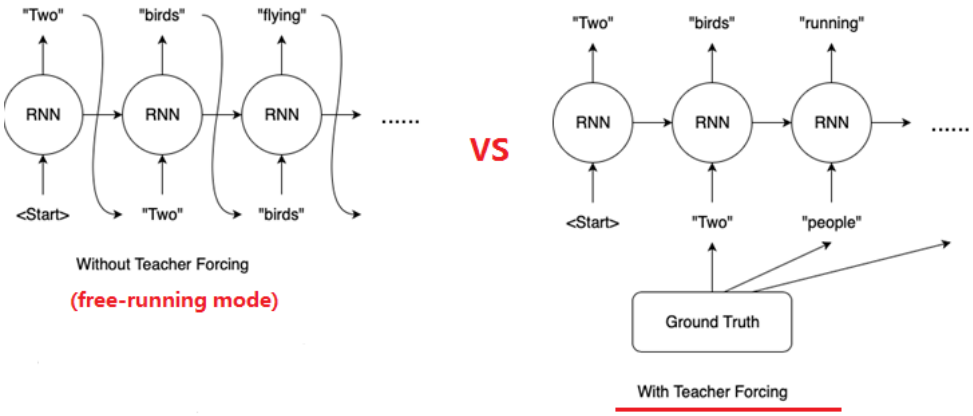

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Instead of performing a single attention function with d_model-dimensional keys, values, and queries. Transformer do Attention for h (num_heads) times using Queries, Keys and Values projected differently to be learnt. After that output of each head will be concatenated together and projected to d_model dimension.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

**Teacher Forcing** techniques will be used in training phase which is crutial point in training seq2seq model. The term "teacher forcing" can be motivated by comparing to a human student taking a multi-part exam where the answer to each part (for example a mathematical calculation) depends on the answer to the preceding part. In this analogy, rather than evaluating answer as a whole in the end, with the risk that the student fails every single part even though they only made a mistake in the first early one, a teacher records the score for each individual part and then tells the student the correct answer, to be used in the next part. This cased is analagous to seq2seq model especially machine translation task.

Formaly, Teacher Forcing is a strategy for training autoregressive model that uses ground truth as input, instead of model output from a prior time step as an input.




After all the answer of model has been recorded, model is learned by optimizing the cross-entropy between its decoder's output and the original text.

## 2.2 BART (Bidirectional Auto-Regressive Transformers)

BART is class of sequence-to-sequence pre-trained models. It is trained by large amounts of data and can easily be fine-tuned to downstream tasks like text-summarization, capitalization, puctualtion restoration, … The idea of BART comes when there is limitation of BERT (Bidirectional Encoder Representations from Transformers) in generative tasks.

Because of the similar in architecture, we will choose to fine-tune on $BARTPho_{word}$ (BART for Vietnamese public by Vinai [3]).

$BARTpho$ use transformer architecture and GeLU activation rather than ReLU and performing parameter initialization using Standard Probability Distribution from N(0, 0.02).

In the pre-training phase, they corrupt the input text with an arbitary noising function and model learning to construct the original text.

The below image is their public pre-trained with some properties.

**Pre-trained models**

| Model | #params | Arch. | Max length | Input text |
|---|---|---|---|---|
| `vinai/bartpho-syllable-base` | 132M | base | 1024 | Syllable level |
| `vinai/bartpho-syllable` | 396M | large | 1024 | Syllable level |
| `vinai/bartpho-word-base` | 150M | base | 1024 | Word level |
| `vinai/bartpho-word` | 420M | large | 1024 | Word level |

# 3. Methodology

## 3.1 Problem Formulation

Spelling Correction Problem can be formulated as follows:

Given corrupt text sequence: $X = (w_1, \dots, w_p)$ for word level, $X = (c_1, \dots, c_m)$ for char level.

The goal is to output: $Y = (y_1, \dots, y_n)$

Where X is original text with misspelled errors and Y is the golden text as label.

## 3.2 Confusion-Set and Rule-Based Error Generator

**Confusion Set:**

In our perspectives, Vietnamse words contain several syllables. For instance, the word (từ) "thành công" contains 2 syllables "thành" and "công". Inside individual syllable (âm tiết) there are 4 parts: start consonant (phụ âm đầu), vowel (nguyên âm), end consonant (phụ âm cuối) and its accent (dấu từ)

nguyễn -> { s_consonant: "t", "vowel": "uyê", e_consonant: "n", accent: "~" }

From now on, we will call "word" for "syllable".

With this structure of syllables, we try to employ a confusion set for generate real-word errors (errors that transform word as other word that have meaning for humans). By replace 4 parts of a word by candidates.

For instance: Candidates for S_Consonant, Vowel and E_Consonant respectively.

- q,ngh,v,c,nh,t,th,qu,ng,d,n,x,p,r,gi,l,h,ch,kh,tr,k,gh,g,m,b,đ,ph,s
- ộ,ặ,ằ,ụ,ầ,a,ũ,á,ể,uằ,iê,ấu,ượ,ùy,ạy,uỹ,ươ,ỗi,yệ,uỷu,uây,ươu,iệu,yếu,yểu,uyế, …
- ch,k,c,ng,t,nh,m,n,p

If new word formed are in a common vietnamese words usage [4], then we take that candidates to confusion-set entry. The one that is one-edit distance(word that is one edit operation from original word, edit operation includes: insertion, substitution and deletion) from the original word will be in the list of 0.7 probability of corrupt and 0.3 for the others (in the vowel candidates at second keys of dictionary)

Each entry in the confusionset look like:

```
{
  "cam": {
    "phu_am_dau": [ "lam", "nham", "sam", "pham","bam","tam"],
    "nguyen_am": [
      ["com","căm","cám","câm","cạm","cảm","càm","cum"],
      ["củm","cớm","cắm","cỡm","cẳm","cườm","cùm","cầm"]
    ],
    "phu_am_cuoi": ["cang","ca","canh","can"]
  }
}
```


**Rule-Based Error Generator:**

We corrupt sequence by choosing 30% words in the text.

We generate the following types of errors in the chosen words:

- Remove Diacritics
- Replace with Typo Letters (Telex, VNI)
- Random Editing
- Replace with common Homophone Letter
- Replace using Confusion Set

**Remove Diacritics (15%):** All accent of a word will be removed, integrate the problem of diacritics restoration. Ex: dấu -> dau, nguyễn -> nguyen

**Replace with Typo Letters(15%):** Vietnamese use two type of typing typos is TELEX, VNI. When fast typing, Vietnamese people usually make mistakes in this error. We will change all characters in a word to its corresponding typo with 70% of all the time for TELEX and remaining for VNI and sequentially apply 0-2 edit operations on the word with probabilities of 50%, 35%, 15% for 0, 1, 2 edit operations respectively.

**Random Editting(10%):** A word will be randomly apply 1-2 edit operation (insertion, substitution, deletion) with probs of 80% and 20% respectively.

**Replace with common Homophone Letter(20%)**: Vietnamese usually misunderstanding the use of some homophone letter pairs such as [“gi”, “d”], [“s”,”x”] or [”l”, “n”] for the case of voice recognition. We heuristically sample several pairs. 80% of all the time the homoleter will be replace by its candidates and 20% remaining we will remove that letter.

**Replace using Confusion Set(40%):** From the confusion set, we will select candidate list for each word by choosing 10% for S_Consonant, 30% for E_Consonant, 60% for Vowel.

After that randomly select candidate inside that list. Except for the case of Vowel, we need one more random operator to choose higher probability list of 70% and lower with 30%. See Confusion-Set.

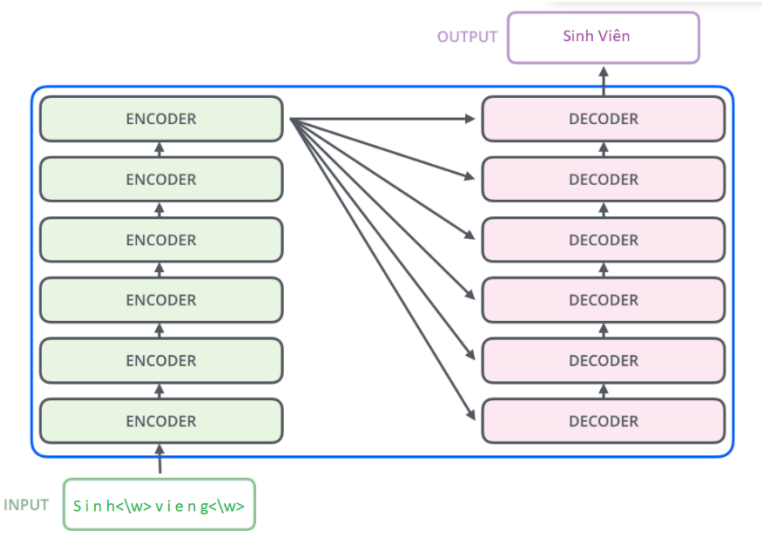
## 3.3 Architecture

The model **Transformer with Tokenization Repair** from thesis of Sebastian Walter [2] is model that uses original Transformer encoder-decoder architecture unchanged. But there is differences that its input is tokenized in character level and is train so that it is capable to generate sequence of BPE tokens as correct sentence. In the thesis, they use 12 encoders and 6 decoders and train with 100M sentences.

| | | | | | |
|---|---|---|---|---|---|
| TR & SEC | transformer with tokenization repair | 49.3M | Characters (99) | Sequence-level next token classification (10,000; BPE) | 12 encoder layers, 6 decoder layers |

**Overview of Transformer with Tokenization [2]**

However we do not possess large computational resource so we do not use the original architecture. Fortunately, vinai public their weights of $BARTpho_{word}$ model of which has similar architecture as Transformer with Tokenization Repair (Seq2Seq, Decoder, Open-Vocabulary Prediction) and there is the fact that large model needs large amounts of data to converge near local maximum. Therefore, we decide to fine-tune our data on $BARTpho_{word}base$.




$BARTpho_{word}base$ is transformer architecture contains 6 encoder layers and 6 decoder layers.

This model has 150M parameters, max input length of 1024 and hidden size of 768.

Teacher Forcing techinques is used for training, optimizing the cross-entropy loss:
With input $c = (c_1, \dots, c_m)$ golden target $y = (y_1, \dots, y_n)$ , at each time step $t$ input is fed in to model and the loss is

$$CE = -\sum_{t=1}^{n} \log\, p_\theta(y_t | y_{<t}, c)$$

With $t$ is time step, $p_\theta(y_t | y_{<t}, c)$ is the probability that the golden word $y_t$ is predicted knowing that previous golden words $y_{<t}$ is provided (teacher-forcing).

At inference, instead of feeding ground truth as in training, output of previous time step will be used as input for the next time step.

### 3.4 Tokenization

Tokenization is process of splitting sequences into tokens for easier descriptions of sequences.

The chosen model Transformer with Tokenization and Repair working with input tokenized at char-level and output sequence tokenized at word-level.

Since $BARTpho_{word}base$'s tokenizer only tokenize text to sequence of BPE tokens, we reuse the embedding layer of $BARTpho_{word}$ by tokenize the input sequence at char-level at follow. This is the customized version of $BARTpho_{word}base$'s tokenizer.

Sinh Viêng -> [ "S@@", "i@@", "n@@", "h", "V@@", "I@@", "ê@@", "n@@", "g" ]

For output sequence will be tokenized default tokenizer of BARTpho-word (**VNCoreNLP, BPE**)

Sinh Viên -> ["sinh", "viên"] or ["sinh", "viê@@", "n"]

## 4. Experimental Results

### 4.1 Datasets

Our training dataset is taken from binhvq/new-corpus [5]. The corpus is crawled from 14.896.998 news in the internet, which have 111.274.300 sentences ~ 18.6GB. Due to limitation in computational resources, we only take 12M sentences. After filtering sentence with number of words less than 10 and over 150 words, 512 num_characters, we obtain ~ 20M samples for training. We also split all sentences by regex pattern \w\S+\w | \w+ | [^\w\s]{1} (modified version compared to the one in thesis [2]) to separate all standalone punctuation. This will not split words like 20/3 for datetime or H.U.S.T for abreviations. We will merge back punctuation in the inference phase.

To evaluate the performance of the model, we use public available dataset is VSEC [7] and our own artificial dataset.

### 4.2 Evaluation Metrics

We use Precision, Recall as F1 score as main metrics for evaluation. F1 score for spelling correction is different from normal one so we use modified one. Formulation is as follows:

TP = Number of wrong words that are restored

FP = Number of word that are not corrupted but is modified, make it wrong

FN = Number of wrong words that are not restored

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1score = \frac{Precision*Recall}{Precision+Recall}$$

Since number of words at output and ground truth may different so we use Needleman-Wunsch algorithms [9] to align texts to reveal which words is corrected, modified or not. We will align 2 times. After alignment of predict_text-> true_text, we split two string by the index of space at true_text. For instance:

```
wrong_text = "Xã này códiện tí ch km² , sổ nawm 2004 là ngư ời ."
true_text = "Xã này có diện tích km² , dân số năm 2004 là người ."
predict_text = "Xã n ày có diệntích km² , thân sổ năm 2004 là người ."
```

Alignment of predict_text -> true_text:

```
Xã này có-diện tí ch km² ,---- sổ nawm 2004 là ngư ời .
Xã này có diện tí-ch km² , dân số n-ăm 2004 là ngư-ời .
```

After splitting,

```
['Xã', 'này', 'có@@', '@@diện', 'tí ch', 'km²', ',@@', '@@---', 'sổ', 'nawm', '2004', 'là', 'ngư ời']
['Xã', 'này', 'có', 'diện', 'tích', 'km²', ',', 'dân', 'số', 'năm', '2004', 'là', 'người']
```

One more time alignment wrong_text -> true_text

```
['Xã', 'n ày', 'có', 'diện@@', '@@tích', 'km²', ',', 'thân', 'sổ', 'năm', '2004', 'là', 'người']
['Xã', 'này', 'có', 'diện', 'tích', 'km²', ',', 'dân', 'số', 'năm', '2004', 'là', 'người']
```

Combine them into a list of tuple 3 elements.

```
[('Xã', 'Xã', 'Xã'),
 ('này', 'n ày', 'này'),
 ('có@@', 'có', 'có'),
 ('@@diện', 'diện@@', 'diện'),
 ('tí ch', '@@tích', 'tích'),
 ('km²', 'km²', 'km²'),
 (',@@', ',', ','),
 ('@@---', 'thân', 'dân'),
 ('sổ', 'sổ', 'số'),
 ('nawm', 'năm', 'năm'),
 ('2004', '2004', '2004'),
 ('là', 'là', 'là'),
 ('ngư ời', 'người', 'người')]
```

We can easily calculate F1 score from that list.

## 4.3 Training Details

We implemented Transformer with Tokenization Repair using huggingface library, using MBartForConditionalGeneration and use vinai/bartpho-word-base pre-trained to fine-tune on. Use the following components and hyperparameters to train model.

- **Optimizer**: AdamW with a weight decay of 0.01, base learning rate of 3e-5

- **Scheduler**: Linear Schedule with Warm Up [8] with warmup steps 5% first training examples.
- **Batch size**: We implement bucket sampling as described below. We set maximum of 1024 tokens per batch.
- **Epochs**: Iterating over the whole dataset. But due to the limitation of RAM we have, we devide the dataset into 5 epochs (5 parts) and load lazy each part individually. Without this important feature, we can't train the model since there aren't enough RAM to fit the whole dataset of 20M samples.

**Bucket sampling:** Instead of using a fixed batch size, we group samples with similar lengths (in tokens) into buckets until it reaches the maximum number of tokens. This is useful to speed up training because it reduces large amounts of padding tokens needed to batch sequences of different length into one Tensor.

## 4.4 Main Results

The following tables depict the results of evaluating model. Note that VSEC here is the model from paper VSEC: Transformer-based Model for Vietnamese Spelling Correction [6].

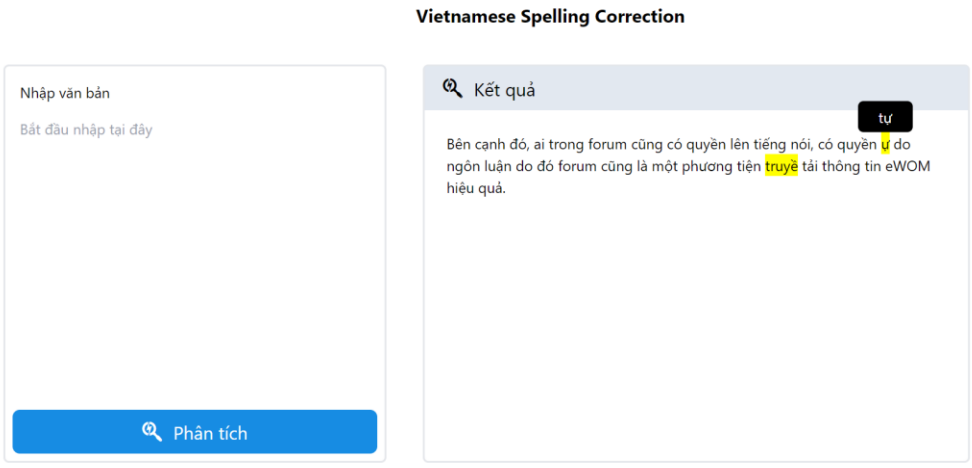
| Model | Dataset | |
|---|---|---|
| | Artificial | VSEC |
| Transformer with Tokenization Repair 20M | 0.980 | 0.825 |
| VSEC 5M | X | 0.815 |

**Performance of different method in F1**

## 4.4 Website Interface

We've integrated Spelling Correction API to a website and here is it GUI:

The wrong text after analyzing will be yellow bold and if we move the cursor on it, the recommendation text will be displayed.



## 5. Conclusions

Seq2Seq models that make open-vocabulary prediction can perform well on Spell Correction task with high F1 score. Tranformer with Tokenization Repair prove its capability to correct both word errors as well as whitespace errors.

To conclude, here is our innovations in this project:

- ✓ Confuset-set and simple rule-based error generator for Vietnamese
- ✓ Make use of $BARTpho_{word}base$ and customize behavior of its default tokenizer in implementation of Transformer with Tokenization Repair
- ✓ Using Needleman-Wunsch algorithm as util to calculate the F1 score since there is difference between the number of words in output sentence and ground truth sentence

## References


[1] Attention is all you need

[2] Transformers and Graph Neutral Networks for Spell Checking of Sebastian Walter

[3] BARTpho: Pre-trained Sequence-to-Sequence Models for Vietnamese

[4] An insight to Vietnamese syllables usage on Luong Hieu Thi's Blog

[5] Binhvq News Corpus

[6] VSEC: Transformer-based Model for Vietnamese Spelling Correction

[7] VSEC: A Real-world Dataset for Vietnamese Spell Correction

[8] Huggingface Optimization

[9] Needleman Wunsch Algorithm of alevchuk on Github