Correcting Arabic Soft Spelling Mistakes Using Deep Machine Learning and Transformers

By

Mohammed Adil Fadhil Al-Qaraghuli

Supervisor

Dr. Gheith Ali Abandah, Prof.

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer and Networks Engineering

Faculty of Graduate Studies

The University of Jordan

August, 2021

COMMITTEE DECISION

This Thesis (Correcting Arabic Soft Spelling Mistakes Using Deep Machine Learning and Transformers) was Successfully Defended and Approved on

Examination Committee	**Signature**
Dr. Gheith Abandah (Supervisor) Prof. of Computer Engineering	----------
Dr. Ramzi Saifan (Member) Assoc. Prof. of Computer Engineering	----------
Dr. Fahed Jubair (Member) Assist. Prof. of Computer Engineering	----------
Dr. Ahmad AlJaafreh (Member) Assoc. Prof. of Computer Engineering (Tafila Technical University)	----------

ACKNOWLEDGEMENT

My thanks are due to Prof. Gheith Abandah for his guidance, patience, and encouragement. I am grateful for giving me the chance to learn machine learning with him.

LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BART	Bidirectional and Auto-Regressive Transformer
BERT	Bidirectional Encoder Representations from Transformers
BiGRU	Bidirectional Gated Recurrent Unit
BiLSTM	Bidirectional LSTM
BLEU	Bilingual Evaluation Understudy
CA	Classical Arabic
CER	Character Error Rate
LSTM	Long Short Term Memory
MSA	Modern Standard Arabic
NLP	Natural Language Processing
OCR	Optical Character Recognition
QALB	Qatar Arabic Language Bank
RNNs	Recurrent Neural Networks
SVM	Support Vector Machine
WER	Word Error Rate

Correcting Arabic Soft Spelling Mistakes Using Deep Machine Learning and Transformers

By

Mohammed Adil Fadhil Al-Qaraghuli

Supervisor

Dr. Gheith Ali Abandah, Prof.

ABSTRACT

Spelling mistakes constitute a common problem for the Arabic language. Several techniques have been used to solve this problem such as confusion matrices, language models, and neural networks. In recent years, a neural network type called the *transformer* has been introduced as a machine translation model. Since then, the transformer and its variants has become a very popular solution for most of the natural language processing (NLP) tasks.

In this thesis, we use the transformer to correct four types of Arabic soft spelling mistakes, namely, confusion among various shapes of *hamza*, shapes of *alef* at the end of the word, insertion and omission of *alef* after *waw aljamaea*, and confusion among *teh*, *teh marbuta*, and *heh* at the end of the word.

We used artificial errors for training and evaluation. These errors were generated using an approach called stochastic error injection. We used Wiki-40B and Tashkeela datasets for training and evaluation, and Test200 dataset to report our results. We produced two transformer models: a Wiki model and a Tashkeela model.

We used three configurations and five error injection rates. The best model is the one that is trained on Wiki-40B set at 90 % error injection rate using 4-layer configuration. The model can correct 97.37% of the artificial errors that are injected into the test set, and achieves a character error rate (CER) of 0.86% on a set that contains real soft spelling mistakes. This error rate is 32.8% lower than the best previous reported results using bidirectional long short-term memory (BiLSTM) networks.

Chapter 1

Introduction

Arabic is the fifth most spoken language in the world, with more than 400 million speakers (UNESCO, 2019). Similar to other languages, spelling mistakes are a common issue in Arabic. These mistakes happen due to many reasons that will be explained shortly. Therefore, we aim in this thesis to correct some of these mistakes that are called soft spelling mistakes, namely, confusion among various shapes of *hamza*, shapes of *alef* at the end of the word, insertion and omission of *alef* after *waw aljamaea*, and confusion among *teh*, *teh marbuta*, and *heh* at the end of the word.

1.1 Spelling Mistakes Types

Spelling mistakes can be categorized into four types: discourse structure and pragmatic-level errors, morpho-syntactic errors, lexical and semantic errors, and soft errors (Abandah *et al*., 2021).

1.1.1 Discourse Structure and Pragmatic-level Errors

Discourse structure errors occur when a single or multiple words with correct form and spelling is used in a sentence but is not compatible with the sentence meaning. Some words may have various meanings depending on the context of the sentence if the word meaning is misunderstood within a certain context, then the error is considered as a pragmatic. We show this error type with the following translation example:

The sentence "*The Biology test was piece of cake. I think I will pass*" has the Arabic translation: "كان اختبار علم الأحياء عبارة عن قطعة من الكعكة ، أعتقد أنني سأنجح". The

“piece of cake” phrase is literally translated to “قطعـة مـن الكيـك” the literal Arabic translation for this phrase is out of context for it has replaced the English phrase word by word to refer to an actual piece of cake, whereas the proper translation should have used the Arabic equivalent for the word easy. So, this is a pragmatic error. In the same example, the misuse of the diacritization for the word “علـم” by using *fatha* instead of *kasra* changes its meaning from science to flag.

1.1.2 Morpho-syntactic Errors

This type of errors occurs due to the confusion between the feminine and masculine forms, using the wrong tense for a verb, or wrong positioning of a word. An example for this type of errors is the sentence “اشـتريتُ خمـس كتـب”, the word “خمـس” should be written in the feminine form “خمسة”.

1.1.3 Lexical and Semantic Errors

This type of errors occurs due to typographical mistakes. For example, writing the word based on its phonetics rather than its typographic form. These mistakes generate words that either do not belong to the lexicon (lexical errors), or words that are not suitable for the sentence (semantic errors). For example, the sentence “ يعتبر الظفدع من البرمائيـات” has a lexical error in the word “ظفـدع” as it is not a real word, for the letter “ض” is phonetically confused with the letter “ظ” whereas the correct word should be “ضفدع”.

1.1.4 Soft Errors

This type of errors mainly occurs due to the confusion among various shapes of a certain letter. Letters such as *hamza* (ء), *alef* (ا), *teh* (ت), and *heh* (ه) have various

shapes depending on the position of the letter and certain rules. For example, in the word "تفـاؤل", the *hamza* is written on a *waw* "ؤ" when the word is a noun, but if it is verb, the *hamza* will be written alone "تفاءل".

Al Ameri (2015) carried out a study that shows which errors are the most frequent among a group of students in a teaching Institute. The number of students that participated in the study is 100 students (40 males, and 60 females). Table 1 shows the frequent errors that were occurred in the study.

Table 1. Soft Spelling Mistakes Based on Al-Ameri Study (Al-Ameri, 2015)

Index	Spelling error type	Percentage
1	Writing *hamza mutwsita* on an *alef*	73%
2	Writing *alef maqswra* instead of *alef mamdwda*	71%
3	Writing *alef mamdwda* instead of *alef maqswra*	70%
4	Omitting *alef* following a *waw* at the end of some verb forms	67%
5	Writing *teh* instead of *teh marbuta*	67%
6	Writing *hamza mutwsita* on *waw*	64%
7	Writing *hamza* at the end of the word on *alef*	51%
8	Writing *hamza* on the line at the end of the word	47%
9	Writing *hamza* at the end of the word on *yeh*	47%
10	Dropping *Lam* before the "solar letter"	38%
11	Writing *teh marbuta* instead of *teh*	37%
12	Writing *hamza mutwsita* on *yeh*	30%
13	Writing *hamza alqt* instead of *hamza alwsl*	28%
14	Inserting *alef* after *waw* at the end of a word	25%

1.2 Importance of Arabic Spelling Correction

In order to improve the performance of the systems that use Arabic language such as optical character recognition (OCR), and automatic speech recognition (ASR), a good spelling correct system is required. Even in communication platforms, such as Twitter, it would be very useful to provide the user with a correction mechanism that facilitates the communication process.

Arabic spelling correction techniques have been evolving through the years. Various methods have been used such as confusion matrices, language models, neural networks, and long short-term memory (LSTM) recurrent neural networks.

In this thesis, we aim to use a state of art neural network, called the *transformer*, to provide a system that corrects Arabic soft spelling mistakes and help Arabic learners and native Arabic speakers to produce Arabic text without soft spelling mistakes.

1.3 Thesis Objectives

The objectives of this thesis are:

1. Can we use transformers to correct Arabic soft spelling mistakes?
2. Is it better to use a pre-trained transformer model such as AraBERT model or build a new model form scratch?
3. Can we develop an efficient system using transformers to automatically correct Arabic soft spelling mistakes?

1.4 Thesis Contribution

In this thesis, we introduce a transformer model that corrects Arabic soft spelling mistakes, namely, confusion among various shapes of *hamza*, shapes of *alef* at the end of the word, insertion and omission of *alef* after *waw aljamaea*, and confusion among *teh*, *teh marbuta*, and *heh* at the end of the word.

We used an approach named stochastic error injection to generate artificial errors that were used to train our model. Stochastic error injection has been introduced in previous work that uses BiLSTM models to correct Arabic soft spelling mistakes (Abandah *et al*., 2021).

1.5 Thesis Methodology

- **Survey of related work**: review the related work that deals with spelling mistakes correction for Arabic and other languages.
- **Evaluation of related work**: show the results of the related work and outline the tools that been used.
- **Dataset gathering and preparation**: prepare the datasets namely Wiki-40b and Tashkeela to be usable for our model.
- **Applying transformers to correct Arabic Soft Spelling Mistakes**: prepare the transformer model and apply it to the datasets.
- **Fine-tuning and accuracy improvement**: test the model and fine-tuning it to achieve better accuracy.
- **Developing**: using our model to develop a graphical user interface that represents our work.
- **Documentation**: document our work and show the results.

1.6 Thesis Outline

The rest of this thesis is structured as follows: in Chapter 2, we review the previous works that deal with spelling correction for both Arabic and other languages. In Chapter 3, we review the deep neural networks that have been used in spelling correction. In Chapter 4, we describe datasets processing along with model building steps. In Chapter 5, we explain how we fine-tuned our model and show our results, and lastly, in Chapter 6, we provide the conclusions and ideas for future work.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, we will discuss previous works that deal with spelling correction for both Arabic and other languages. Most Arabic spelling correction works use non-machine learning techniques and few of them use machine learning, specifically BiLSTM. Most recent works in other languages started using transformers for spelling correction.

2.2 Arabic Spelling Mistakes Correction

We categorized these works based on whether they use machine learning or not. Thus, we have machine learning approaches and general approaches.

2.2.1 Machine Learning Approaches

Azmi *et al*. (2019) introduced a system that detects and corrects Arabic semantic errors. The system was trained by inserting one artificial error in the sentence then the support vector machine classifier (SVM) will detect if the word contains an error or not, if it has an error the N-gram language model will generate a candidate word to replace the word with error. The system achieved an F_1 score of 90.7%.

Alkhatib *et al*. (2020) introduced word-level model with BiLSTM and polynomial network classifier for Arabic spelling and grammatical mistakes detection and correction. They trained the model using artificial errors that were generated using linguistic knowledge algorithms. For testing, they used a set that contains 15 million

words with errors. These errors were manually inserted and corrected by experts. The authors report an F-measure of 93.89%.

Abandah *et al*. (2021) introduced two character-level BiLSTM models that correct Arabic soft spelling mistakes identical to the ones we target in this thesis. They trained the models on Tashkeela and ATB3 sets. These sets are not designed for spelling correction purposes, but they implemented two training approaches, transformed input and stochastic error injection that allowed them to produce artificial errors on the mentioned sets. Transformed input was designed to map a set of targeted letters into a one-letter form. Stochastic error injection was used to replace a letter from a certain set with another letter from the same set based on a parameter called error injection rate *p*. The two models were trained using both approaches and a third set called Test200 that contains real soft spelling mistakes was used for evaluation. The best model was the one that trained using the Tashkeela set with stochastic error injection approach and 40% error injection rate. This model obtained a character error rate (CER) of 1.28%.

2.2.2 General Approaches

Mubarak *et al*. (2014) introduced a system with character-level correction model and language model to correct Arabic spelling mistakes. The correction model works by searching the lexicon for similar words to those who contain errors. The language model is dedicated to the errors that are difficult for the correction model to detect or correct. The system achieved an F_1 score of 63.43% on QALB dataset.

AlShenaifi *et al*. (2015) introduced a system named Arib that combines several algorithms and tools to detect and correct Arabic spelling mistakes. The system uses MADAMIRA tool to correct the errors that are related to the letters (ى ,أ). A rule based

corrector is used to convert English punctuations marks to Arabic punctuations, insert space after (إلى, على) Prepositions and the letters (ة, ـة) if they merged with the next word, and removes the letter that repeated sequentially three times or more in the same word. The type of errors that can be corrected with more than one word is handled using Bayes probability algorithm that uses a large dictionary constructed from various sets. Arib also uses Ghaltawi tool that contains an external dictionary to correct errors. Levenshtein distance algorithm is also used in this system. The final output is obtained after the system runs punctuation recovery algorithm. Arib achieved an F_1 score of 57.8% on QALB dataset.

Noaman *et al*. (2016) introduced a system that uses both confusion matrix and noisy channel spelling correction model to detect and correct Arabic spelling mistakes. The system works by searching a dictionary with 35.5k words, if the word is not in the dictionary then it considers an error and the confusion matrix will generate a candidate word. The system achieved an accuracy of 89.69%.

Attia *et al*. (2016) introduced a system to detect and correct Arabic spelling mistakes. The detection phase uses either a list that contains the correct word or two character-level language models. One was trained on the correct words and the other on the words that contain errors. When an error detected, an error model works to produce a list of candidates and pass it to an N-gram language model that handles the correction. Their system achieved an accuracy of 93.64%.

Abdellah *et al*. (2020) introduced a system that corrects space omitting mistakes in Arabic text base on Levenshtein distance algorithm. The system works by inserting a space between two words that were merged together due to the omitting of space. If the words contain errors after inserting space, the system corrects them using a lexicon

with 30k words. They tested the system with a set contains 1k words with errors. The system was able to achieve a correction rate of 99.14%.

2.3 Spelling Correction in Other Languages

Zhang *et al*. (2020) introduced a new method called soft-masked BERT to detect and correct Chinese spelling mistakes. The detection stage works by implementing bidirectional gated recurrent unit network (BiGRU) to detect the errors in the input. The correction stage uses bidirectional encoder representations from transformers (BERT). BERT is a transformer model that consists of 12 bidirectional encoders. Their model was able to achieve an F_1 score of 73.5% for detection and 66.4% for correction.

Tran *et al*. (2021) introduced a transformer model to detect and correct Vietnamese spelling mistakes. They built their model with character-level transformer encoder with four self-attention layers, word-level transformer encoder with 12 self-attention layers, a detector and corrector classifier. The model achieved an F_1 score of 68.88% for detection, and a correction accuracy of 96.01%.

Kuznetsov *et al*. (2021) introduced a transformer model for spelling correction. The model was trained with a method that generates artificial spelling errors for any input language. The authors built this method by collecting data from search engines and applying a set of rules to generate errors. Their model was tested on four languages and has an accuracy of 83.33% for Arabic, 93.97% for Greek, 91.83% for Russian, and 94.48% for Setswana.

Zhao *et al*. (2021) introduced a transformer model for semantic errors correction in Mandarin automatic speech recognition system. They used bidirectional and auto-

regressive transformer (BART) model to correct the output of the ASR system and improve its performance. The ASR system that was integrated with their model achieved a CER of 6.08%.

In Table 2, we summarize the previous works based on the language that been targeted, the method that been used, the evaluation metric that been used, and the score that been reported.

Table 2. Summary of the Literature Review

Author(s)	Language	Method	Metric	Score
Mubarak *et al*. (2014)	Arabic	Correction and Language Model	F_1	63.43%
AlShenaifi *et al*. (2015)	Arabic	Multiple Algorithms and Tools	F_1	57.8%
Noaman *et al*. (2016)	Arabic	Confusion Matrix	Accuracy	89.69%
Attia *et al*. (2016)	Arabic	N-gram	Accuracy	93.64%
Azmi *et al*. (2019)	Arabic	SVM and N-gram	F_1	90.7%
Abdellah *et al*. (2020)	Arabic	Levenshtein Distance	Correction Rate	99.14%
Alkhatib *et al*. (2020)	Arabic	BiLSTM	F-Measure	93.89%
Abandah *et al*. (2021)	Arabic	BiLSTM	CER	1.28%
Zhang *et al*. (2020)	Chinese	BERT (transformer)	F_1	66.4%
Tran *et al*. (2021)	Vietnamese	ALBERT (transformer)	Accuracy	96.01%
Kuznetsov *et al*. (2021)	Multiple Languages	Transformer	Accuracy	Multiple Scores
Zhao *et al*. (2021)	Mandarin	BART (transformer)	CER	6.08%

2.4 Arabic Spelling Mistakes Datasets

The Arabic spelling correction field lacks a large annotated dataset that can be used in research. Therefore, most of the previous works used various methods to generate artificial errors for correction. However, there are some sets that contain real errors such as Qatar Arabic Language Bank (QALB) which contains various spelling errors collected form comments written by real users and student essays (Zaghouani *et al*., 2014).

Unfortunately, QALB is not suitable for our work for two reasons: first it contains dialectal words that were not converted to its corresponding modern standard Arabic (MSA) format. Second is the size of the set. It contains 21k sequences, such size is considered small and transformers require a big set in order to achieve good results (Xu *et al*., 2020).

Chapter 3

Deep Neural Networks in Spelling Correction Field

3.1 Introduction

In this chapter, we will discuss deep neural networks that are commonly used in spelling correction. We start with brief overview for RNNs, LSTM, and RNNs with attention then we will discuss transformer model, how self-attention works in transformer, and model architecture.

3.2 RNNs and LSTM

Recurrent neural networks (RNNs) are a type of neural networks that can handle sequential data quite well. Recurrent neural networks process sequences and generate hidden state for each input sequence as shown in Figure 1. This hidden state is used in the next step to produce the output based on the following equations:

$$h_t = g(W_{hx}\, x_t + W_{hh}\, h_{t-1} + b_h) \quad (1)$$

$$\hat{y}_t = g(W_{yh}\, h_t + b_y) \quad (2)$$

where h_t is the hidden state at time t, x_t is the input at time t, $\hat{y}_t$ is the output at time t, h_{t-1} is the hidden state at time $t-1$, b is bias parameter, W is the weight parameter, and g is the activation function (Rumelhart *et al*., 1986).

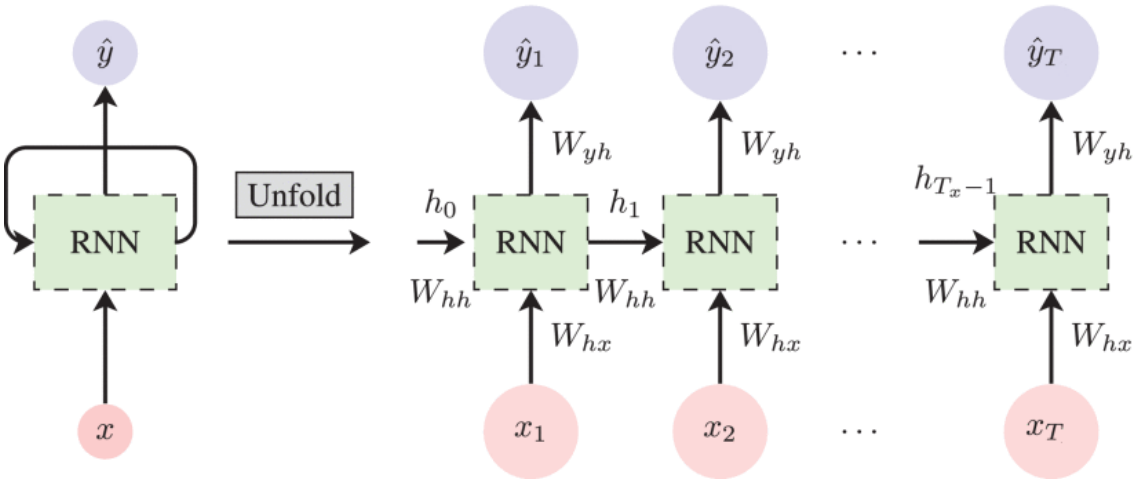

Figure 1. RNNs Structure (Madhfar *et al.*, 2020)

Some tasks like speech recognition, machine translation, and spelling correction require a model that can capture long dependences in the input sequence. Recurrent neural networks struggle with this kind of tasks due to the vanishing gradient problem. The solution for this problem is to use memory cells called long short-term memory (LSTM). Long short-term memory can preserve the hidden states for the previous steps; thus, the model will be able to process long sequences (Hochreiter and Schmidhuber, 1997).

Bidirectional LSTM (BiLSTM) is another version of LSTM that process the sequences in forward and backward manner as shown in Figure 2. This type of models can learn more information from the input sequences and that leads to more improvement in the performance (Schuster and Paliwal, 1997).

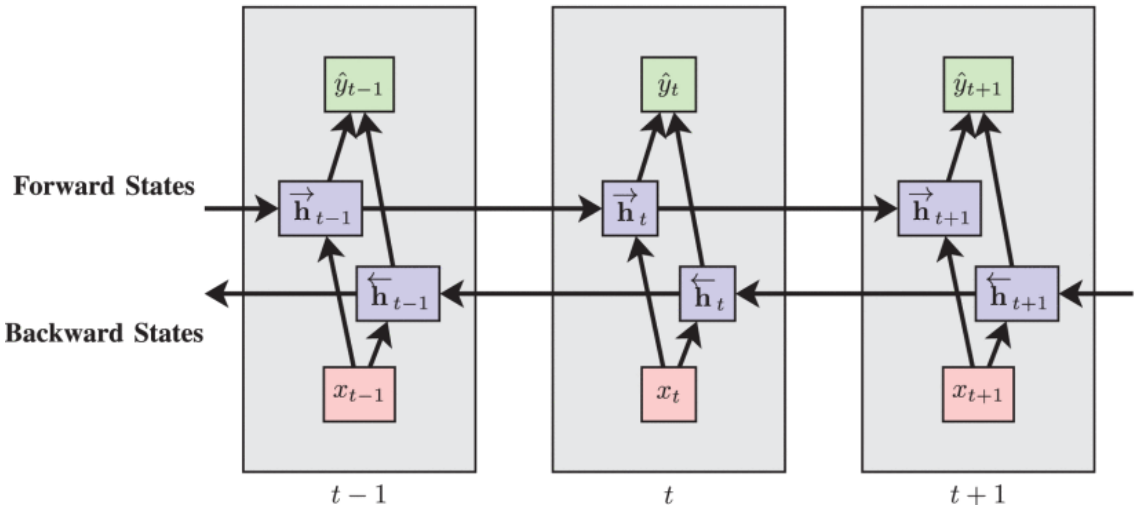

Figure 2. BiLSTM Structure (Madhfar *et al.*, 2020)

3.3 RNNs with Attention

Attention mechanism can be described as a method that helps the model to capture useful information by focusing on the relevant part in the sequence using a context vector as shown in Figure 3 (Madhfar *et al.*, 2020). Let's take the following sequence:

"*Paris is the capital and most populous city of France, with an estimated population of 2,175,601 residents as of 2018, in an area of more than 105 square kilometers. Since the 17th century, Paris has been one of Europe's major centers of finance, diplomacy, commerce, fashion, gastronomy, science and arts.*"

If the model task was question answering and it got the following question as an input:

What is France capital city?

To answer this question the model will process the sequence then outputs the word "Paris". But it is not necessary to go through the entire sequence, focusing on the words (Paris, capital, city, France) by assigning high scores to them compared to other words in the sequence can produce the correct answer.

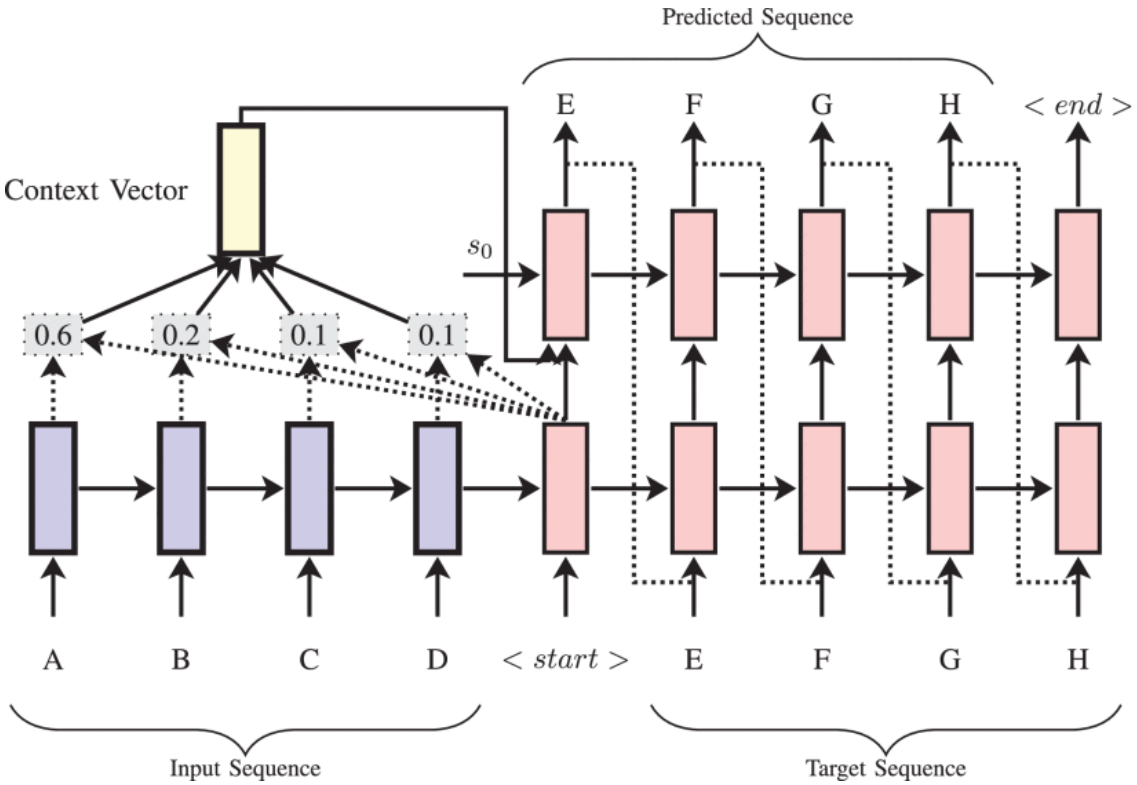

Figure 3. Encoder-Decoder RNNs with Attention (Madhfar *et al*., 2020)

3.4 Transformer

The success of attention mechanism in RNNs has led to the idea of creating a model known as transformer that depends entirely on attention and abandons the RNNs (Vaswani *et al*., 2017). One of the major advantages that transformer has is the ability to process the input through various paths.

Let's take the three words shown in Figure 4. These words enter the self-attention layer in three paths. These paths are dependent in the self-attention layer but not in the feed forward layer. Therefore, these paths can be processed through the feed forward layer in parallel not series like RNNs. This operation improves training time and helps the model to train faster (Alammar, 2018).

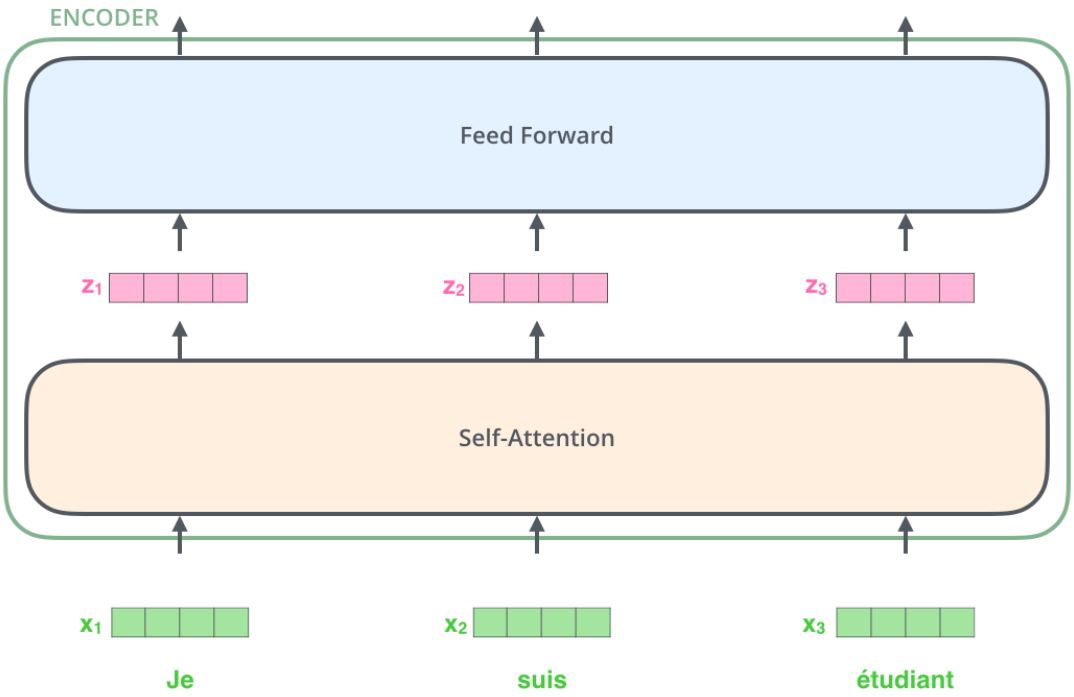

Figure 4. Input Paths in the Encoder (Alammar, 2018)

3.4.1 Self-Attention

Transformer model is built with new concept of attention called self-attention or scaled dot-product attention. Self-attention allows the model to search the input sentence to find representation for a certain word or to form relations between the word that is currently being processed and the other words in the input. We illustrate the concept of self-attention in Figure 5. One can observe that “it” in the left-side sentence is referring to the word “animal” due to the presence of the word “tired”. The right-side sentence contains the word “wide” instead of the word “tired”, thus, the best representation for “it” is the word “street”.

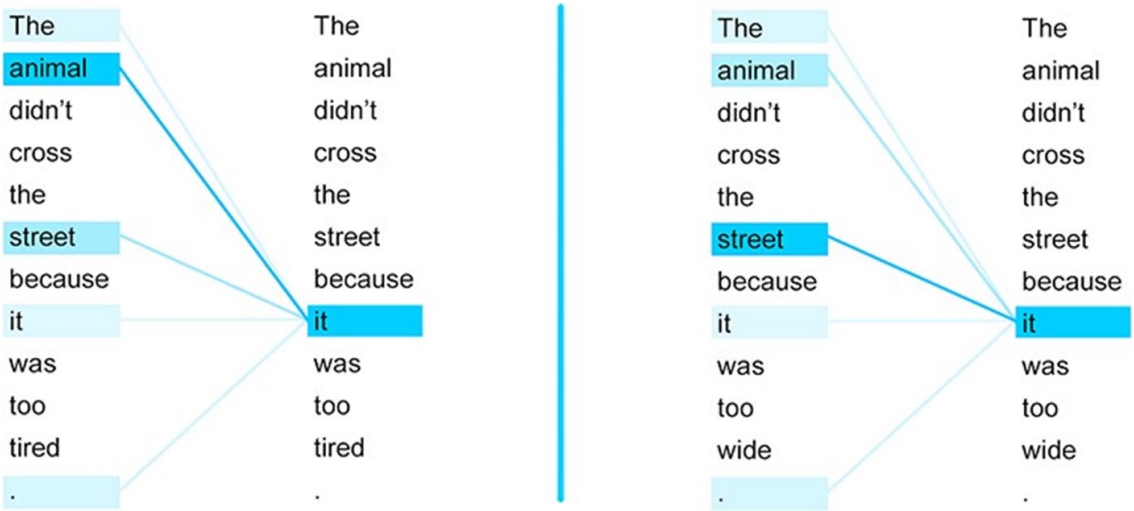

Figure 5. Example of Self-attention (van den Berg, 2020)

3.4.1.1 Calculating Self-Attention

Self-attention is calculated by the following equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q\,K^T}{\sqrt{d_k}}\right)V \tag{3}$$

where Q, K, V are the Query, Key, and Value matrices and d_k is the dimension of the key vector which is equal to 64 and it is calculated by dividing the model dimension (d_{model}) over the number of heads (h).

Transformer model has eight heads (h) or self-attention layers, combine they called multi-head attention. Using more than one head helps the model to represent different positions in the input sentence.

There are five steps to calculate self-attention as shown in Figure 6. Step one is to obtain the input sequence. Step two is to convert the input to embedding matrix X this step is only for first encoder; the rest of the encoders use the output of the previous encoder R.

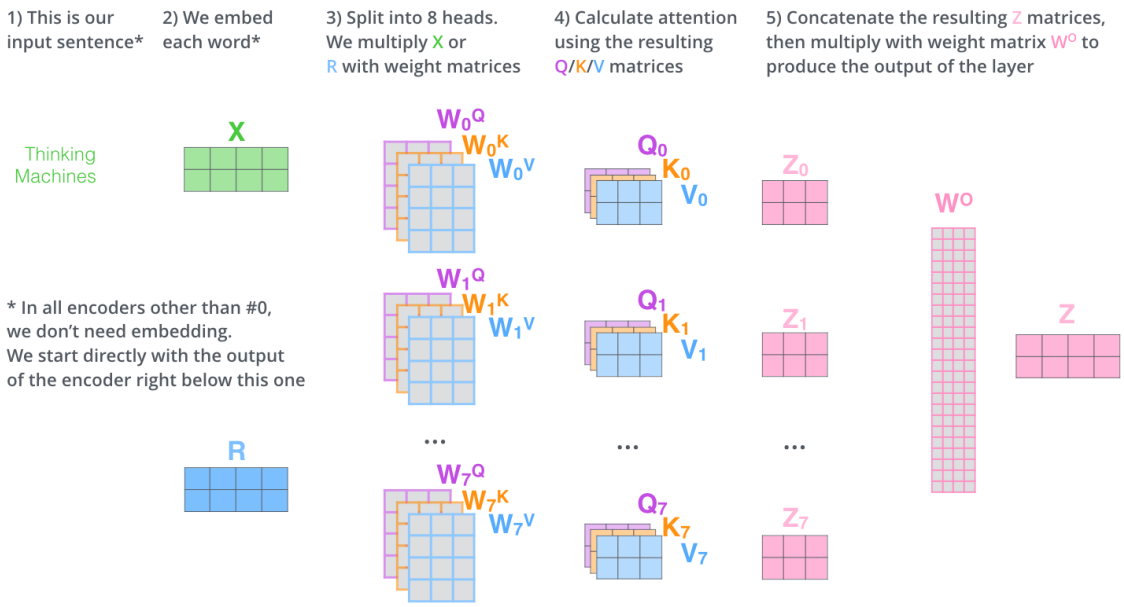

Figure 6. Self-attention Calculation Steps (Alammar, 2018)

Step three is to obtain Q, K, V matrices by multiplying X or R with the weights matrices that were obtained during the training as shown in Figure 7.

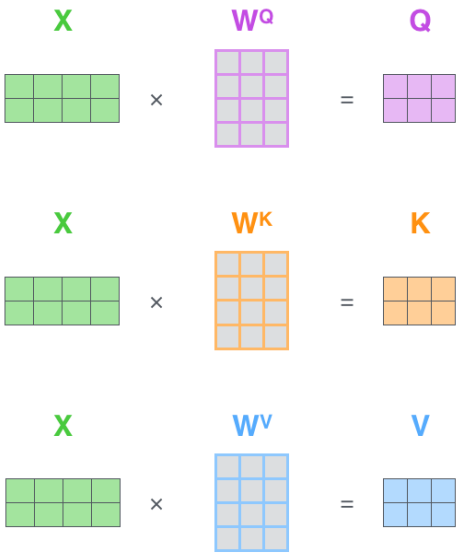

Figure 7. Query, Key, and Value Matrices Calculation (Alammar, 2018)

Step four is to calculate the output of each head (Z^0 ... Z^7) using Equation 3 as shown in Figure 8.

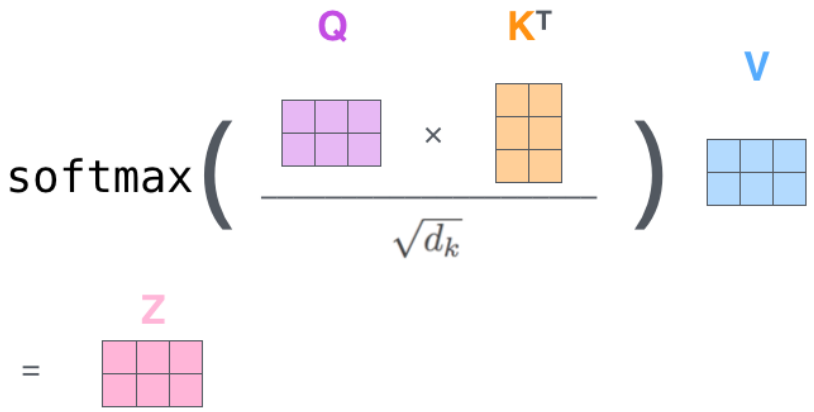

Figure 8. Attention Head Calculation Using Equation 3 (Alammar, 2018)

Lastly, the outputs of the heads are concatenate and multiply with W^0 matrix as shown in Figure 9. In result, the output of the encoder is obtained after the Z matrix go through the feed forward network (Alammar, 2018).

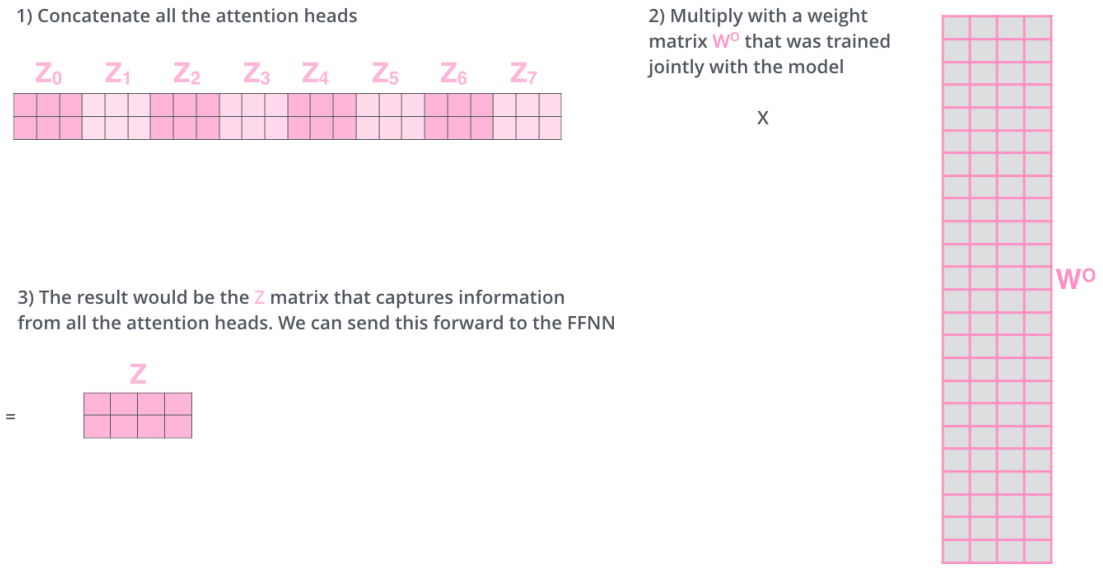

Figure 9. The Output of Self-attention Layers (Alammar, 2018)

3.4.2 Transformer Architecture

Transformer is built with encoder-decoder architecture, embedding and positional encoding to handle the input, and linear layer with *softmax* to handle the output. In Figure 10, we show an example of 2-layer transformer architecture that consists of two encoder-decoder stacks.

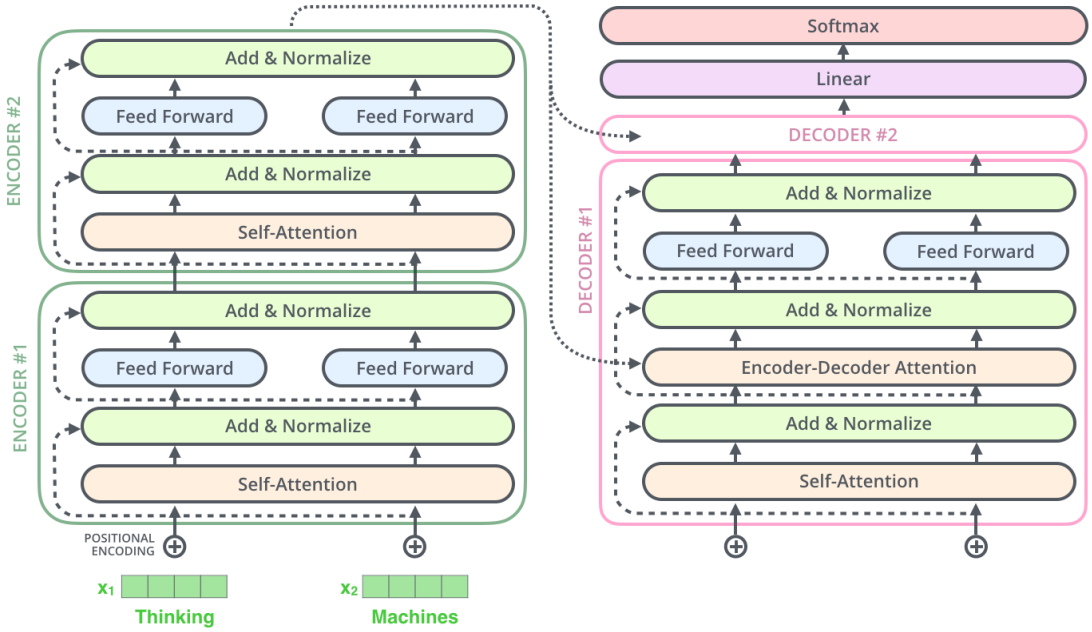

Figure 10. 2-Layer Transformer Architecture (Alammar, 2018)

3.4.2.1 Embedding

The embedding layer task is to convert the input into vectors of dimension d_{model} then stores these vectors in embedding matrix of dimension (vocabulary size, d_{model}). The embedding happens once for both encoder and decoder (Alammar, 2018).

3.4.2.2 Positional Encoding

Transformer model does not have recurrence. Therefore, positional encoding is the method that transformer is used to represent the order of the words in the input.

The model creates vectors of dimension d_{model} based on the following equations:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (4)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (5)$$

where pos is the position and i is the dimension. The even positions use the sine and the odd positions use cosine. These positions vectors are added to the embedding vectors to make the words in the embedding matrix represented by their meaning and how close they are to each other in the input (Vaswani *et al*., 2017).

3.4.2.3 Encoder

The encoder contains self-attention layer and feed forward layer as shown in Figure 11. These layers have residual connection and layer normalization to help the model keeps the information that was obtained from the input when the data travel to the upper encoders.

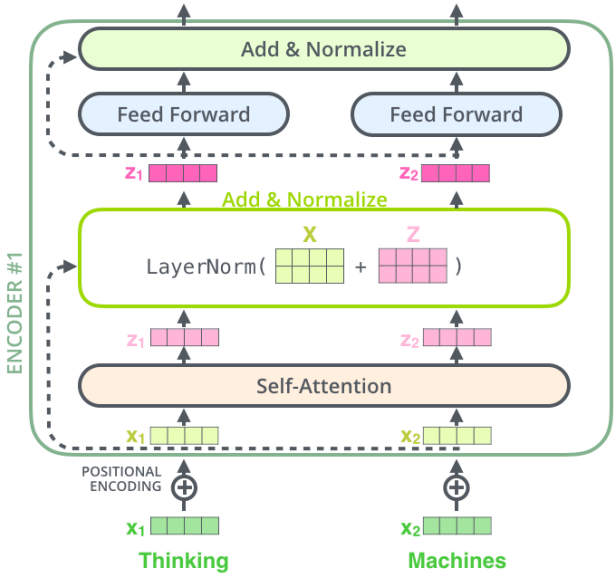

Figure 11. Encoder Structure (Alammar, 2018)

There are N encoders in the transformer and the default value of N is six. Self-attention layer depends on the previous encoder output to generate its key, value, and query (Alammar, 2018).

3.4.2.4 Decoder

Similar to the encoder part, there are N decoders in the transformer with $N = 6$, a self-attention layer and a feed forward layer with residual connection and layer normalization as well. However, there is an additional layer called encoder-decoder attention. This layer contains the key and value matrices from the encoder stack and generates the query matrix in the decoder. This layer helps the decoder to see the input from the encoder and form a better understanding with its own input.

Self-attention layer works diffrently in the decoder. It is only allowd to use the previous words from the decoder inputs to predict the current word (Alammar, 2018).

3.4.2.5 Linear Layer

The linear layer is the final layer in the transformer. It converts the output vector from the decoder into logits vector. The *softmax* converts logits into probabilities. The highest probability will be selected and its index will turn back into a word (Alammar, 2018).

Chapter 4

Datasets Preparation and Model Building

4.1 Introduction

In this chapter, we will describe the datasets that were used and how we prepared them for the model. Although, these sets are not related to spelling mistakes correction, the training approach that we used made it possible to correct spelling mistakes using these sets. We also will discuss how we built our transformer model and why we did not use a pre-trained model form Hugging Face library. Lastly, we will state the evaluation metrics that we used to evaluate our model.

4.2 Datasets

In this work, two sets were used, the Wiki-40B and Tashkeela.Wiki-40B is a set that contains cleaned articles from Wikipedia for over 40 languages. The Arabic version contains 220,885 articles for training, 12,271 articles for testing, and 12,198 articles for validation written in modern standard Arabic (MSA). The set was released as part of multilingual language modeling research that introduced a language model for over 40 languages (Guo *et al*., 2020). Table 3 shows a sample text from the Arabic version.

We used the training and validation subsets as are and for testing, we randomly selected 2000 sequences from the test subset. The reason for that is to reduce the testing time since it takes approximately 100 hours for the entire test subset and Kaggle only offers 30 hours per week to use its accelerators.

Table 3. Sample Text from Wiki-40B Arabic Version

Index	Text	Wikidata_id
1	_START_ARTICLE_ أوبال كونز _START_SECTION_ تاريخ _START_PARAGRAPH_ شخصي وُلِدت أوبال في عام 1894 أو 1896 في ميسوري، ووالدها إدوارد جبرسون. تَخرجت في مَدرسة دانا هول في عام 1923، تَزوجت مِن _NEWLINE_.في ويسلي، ماساتشوستس جورجوج فريدريك كونز(1856-1932). لكن تمَّ إلغاء الزواج في عام 1929. ظَل الزَّوجان عَلى عَلاقة جَيدة، وعَمِلت أوبال عَلى رِعاية جورج لِبقية حياته. وعِند وفاتِه تَرَك لَها وَصية كبيرة...	Q7095545
2	_START_ARTICLE_ ميكرونيسيا _START_PARAGRAPH_ ميكرونيسيا هي مجموعة جزر في النصف الجنوبي من المحيط الهادي. تقع الفلبين إلى غرب ميكرونيسيا، وإندونيسيا في الجنوب الغربي، وبابوا غينيا الجديدة وميلانيسيا إلى الجنوب، وبولنيسيا إلى المنطقة الجنوب الشرقي والشرق. _START_SECTION_ جغرافيا تقع ميكرونيسيا ضمن أرخبيل جزر الهند _START_PARAGRAPH_ الشرقية ،كما تقع إلى الشمال من أستراليا وتنقسم جزيرة غنيا الجديدة بين أندونسيا (ايريان الغربية)وأستراليا (بيوا) وتتبعها مجموعة من الجزر تقع في شرقها وتبلغ مساحة بيوا 234ألف كم، وقدر عدد سكانها في سنة 1408 هـ - 1988 م بحوالي 380 ألف نسمة...	Q3359409

The second set that we used is a cleaned version of the Tashkeela set that contains text written in classical Arabic (CA) and text from the Holy Quran as shown in Table 4. The set is split into 50k lines for training, 2.5k lines for validation, and 2.5k lines for testing. The set was released as part of Arabic text diacritization research (Fadel *et al*., 2019).

We used the test subset as is. The training and validation subsets were merged then redistributed and split into 80% training and 20% validation.

Table 4. Sample Text from Tashkeela

Index	Text
1	وَأَمَّا عَدَمُ تَعَلُّقِ حَقِّ الْغَيْرِ كَالرَّهْنِ وَالْإِجَارَةِ فَلَيْسَ بِشَرْطٍ فَلَوْ آجَرَ أَرْضًا عَامَيْنِ فَوَقَفَهَا قَبْلَ مُضِيِّهِمَا لَزِمَ الْوَقْفُ بِشَرْطِهِ وَلَا يَبْطُلُ عَقْدُ الْإِجَارَةِ فَإِذَا انْقَضَتْ الْمُدَّةُ رَجَعَتْ الْأَرْضُ إلَى مَا جَعَلَهُ لَهُ مِنْ الْجِهَاتِ وَكَذَا لَوْ رَهَنَ أَرْضَهُ ثُمَّ وَقَفَهَا قَبْلَ أَنْ يَفْتِكَهَا لَزِمَ الْوَقْفُ وَلَا تَخْرُجُ عَنْ الرَّهْنِ بِذَلِكَ ، وَلَوْ أَقَامَتْ سِنِينَ فِي يَدِ الْمُرْتَهِنِ ثُمَّ أَفْتَكَّهَا فَتَعُودُ إلَى الْجِهَةِ وَلَوْ مَاتَ قَبْلَ الِافْتِكَاكِ وَتَرَكَ قَدْرَ مَا تُفْتَكُّ بِهِ أُفْتُكَّتْ وَلَزِمَ الْوَقْفُ وَإِنْ لَمْ يَتْرُكْ وَفَاءً بِيعَتْ وَبَطَلَ الْوَقْفُ
2	حَدَّثَنَا عَفَّانُ حَدَّثَنَا شُعْبَةُ عَنْ سُلَيْمَانَ الْأَعْمَشِ عَنْ أَبِي وَائِلٍ عَنْ عَبْدِ اللَّهِ عَنْ النَّبِيِّ صَلَّى اللَّهُ عَلَيْهِ وَسَلَّمَ قَالَ لِكُلِّ غَادِرٍ لِوَاءٌ يَوْمَ الْقِيَامَةِ

Lastly, we used the Test200 set for evaluation. This set contains 200 sequences with real soft spelling mistakes and an average of 6.5 mistakes per sequence (Abandah *et al*., 2021). Table 5 shows the characteristics of Test200 set.

Table 5. Test200 Set Characteristics (Abandah *et al*., 2021)

Criterion	Count
Sequence count	200
Word count	2,443
Character count	24,002
Number of mistakes	1,306
Mistakes per sequence	6.5

4.2.1 Wiki-40B Set Preparation

The text in this set contains four special tokens. These tokens are start article, start section, start paragraph, and newline. Alongside the dot these tokens were replaced with the new line character (\n) then the text split at (\n). This process transformed the set from long articles based set to short and understandable sentences based set. The text also contains unnecessary characters such as punctuation marks, numbers, and English letters, all these characters were removed. Finally, all the lines longer than 300 were wrapped in order to get the maximum benefit from all the text.

4.2.2 Tashkeela Set Preparation

The first step in preparing this set is to remove the diacritics from the text alongside punctuation marks and numbers. The second step is to wrap the lines longer than 300.

Table 6 shows the characteristics for both sets in term of sequence count, word count, Arabic letter count, words per sequence, letters per word, and the portion of sequences that are shorter than or equal to the maximum sequence length.

Table 6. Wiki-40B and Tashkeela Sets Characteristics

Criterion	Wiki-40B Set	Tashkeela Set
Sequence count	4,015k	55k
Word count	73,525k	2,312k
Arabic letter count	354,047k	9,189k
Words per sequence	18.31	42.05
Letters per word	4.82	3.97
Sequences ≤ 300 chars	95.78%	78.06%

Table 7 shows the occurrences of the target characters. Character (ا) is the most occurring character with 14.27% and 11.37% for Wiki-40B and Tashkeela sets, respectively. Character (ء) is the least occurring character for Wiki-40B with 0.04%, and characters (ـوا) are the least occurring characters for Tashkeela with 0.05%.

Table 7. Target Characters Occurrences in Wiki-40B and Tashkeela Sets

Character(s)	Wiki-40B Set	Tashkeela Set
ء	0.04%	0.09%
أ	1.94%	3.27%
ئ	0.40%	0.25%
إ	0.75%	1.26%
ؤ	0.09%	0.08%
ا	14.27%	11.37%
آ	0.10%	0.10%
ـة	3.41%	1.65%
ى	0.81%	0.91%
ـو	0.46%	0.92%
ـت	1.16%	0.55%
ـوا	0.07%	0.05%
ـه	0.70%	3.49%
اء	0.27%	0.25%
Total	24.47%	24.24%

4.3 Training Approach

In this work, we base the training approach on stochastic error injection (Abandah *et al*., 2021), an approach that built entirely on the concept of artificial errors. The errors are injected onto the set randomly using a parameter called *error injection*

rate p. The authors in (Abandah *et al*., 2021) reported best model results using an error injection rate of 40%. Thus, we used it as a baseline when evaluating the error injection rate. We also adopted the one-to-one mapping approach in (Abandah *et al*., 2021), where certain two-letter combinations and the letters that appear at the end of the word are represented by one special token each, as shown in Table 8.

Additionally, our target characters are split into four sets based on which characters are often confused in place of others. The first set contains the various shapes of *hamza* (ا, أ, إ, آ, ؤ, ئ, ء, اء), the second set contains the two shapes of *alef* at the end of the word (ا, ى), the third set contains the two shapes of *waw* at the end of the word (ـو, ـوا), and the fourth set is for the *teh*, *teh marbuta*, and *heh* at the end of the word (ـت, ـة, ـه). Each character in these sets is randomly replaced with another character from its own set in the stochastic error injection approach.

Table 8. Target Characters and their Corresponding Tokens

Character(s)	Corresponding Token
ـو	O
ـوا	A
ـه	H
ـت	T
اء	J

4.4 Model Building

In NLP domain, models usually are word-level or character-level. To show which type of models is suitable for our work, we will take this simple example: in the word "ابـن", the letter (ا) may be randomly replaced with any letter in its represented set. Thus, the word "ابن" may become one of these words (أبن, إبن, آبن, ئبن, ءبن, ؤبن, اءبن) some of these words are not even a real words and a word-level model can't recognizes it since this type of model is trained on a huge amount of Arabic text from various

sources and any word that does not recognized will be mapped to an *unknown token* (UNK). On the other hand, a character-level model can handle this type of variations quite well due to the fact that these changes are only applied in one character which is (ا) and the model already knows all the other characters that character (ا) may change to it. Therefore, the nature of our training approach obligates us to use a character-level model.

There are two options to use transformers either by using a pre-trained model or by building a model from scratch.

4.4.1 Using Pre-Trained Model

Hugging Face transformers is a library that provides a collection of state of the art transformers models such as AraBERT, an Arabic version of BERT. It achieved splendid results in various tasks like sentiment analysis, named entity recognition, and question answering (Antoun *et al*., 2020a). ARAGPT2, an Arabic version of GPT2, the model was able to generate a readable Arabic text similar to the human text (Antoun *et al*., 2020b). These models are word-level and they are not suitable for our work and to the best of our knowledge there is no pre-trained character-level transformer for Arabic language.

4.4.2 Building the Model from Scratch

We built our character-level model based on the original transformer paper (Vaswani *et al*., 2017). The model has encoder-decoder stacks with number of layers $N = 6$, self-attention layers or heads $h = 8$, point wise feed forward network with dimension $d_{ff} = 2048$, dropout rate $p_{drop} = 0.1$, and model dimension $d_{model} = 512$.

We used sparse categorical crossentropy as a loss function and *adam* optimizer with custom learning rate scheduler based on the following equation:

$$\text{Learning rate} = d_{model}^{-0.5} \times \min(\text{step\_num}^{-0.5}, \text{step\_num} \times \text{warmup\_steps}^{-1.5}) \quad (6)$$

where the $\text{warmup\_steps} = 4000$.

4.5 Model Training

In order to train the model, we need to prepare the input pipeline and make the data suitable for the model. The data consist of input sequences which contain text with soft spelling errors, and target sequences which contain the correct text. The first step in preparing the data is to insert a start token (S) and end token (E) for both input and target sequences, then insert a pad token (P) for the sequences that are shorter than the maximum sequence length which is equal to 300. We chose this value to avoid crashing the model when it operates on GPU environment. The second step is to convert the input and target sequences into Ids, where each Id represents a letter in the sequences. The last step is to prepare the encoder input, the decoder input, and the labels. The encoder input is the input sequences, the decoder input is the target sequences without the end token, and the labels are the target sequences without the start token. For training, we used TPU accelerator on Kaggle to reduce the training time. Training with a GPU is slower (22 vs. 0.8 hrs. per epoch for the model that was trained on Wiki-40B set). Table 9 shows the specifications of kaggle platform that was used for training.

Table 9. Kaggle Platform Specifications

Aspect	Specification
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz, 2 cores, 56 MB cache
TPU	v3-8
GPU	Nvidia P100 @ 1.32GHz, 16 GB
Memory	16 GB
Libraries	Python 3.7.9, TensorFlow 2.4.1

In result, we produced two models, the first one called Wiki model that was trained on Wiki-40B set for 20 epochs. Figures 12 and 13 show the model performance in term of loss and accuracy on the training and validation sets.

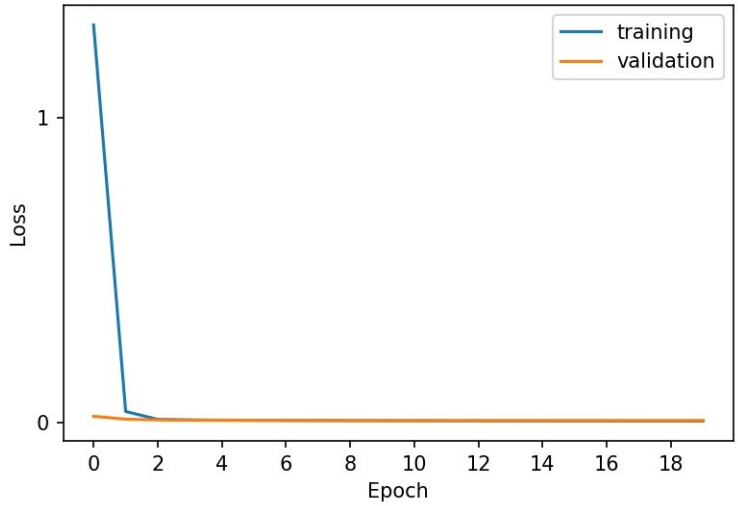

Figure 12. Wiki Model Loss on Training and Validation Sets

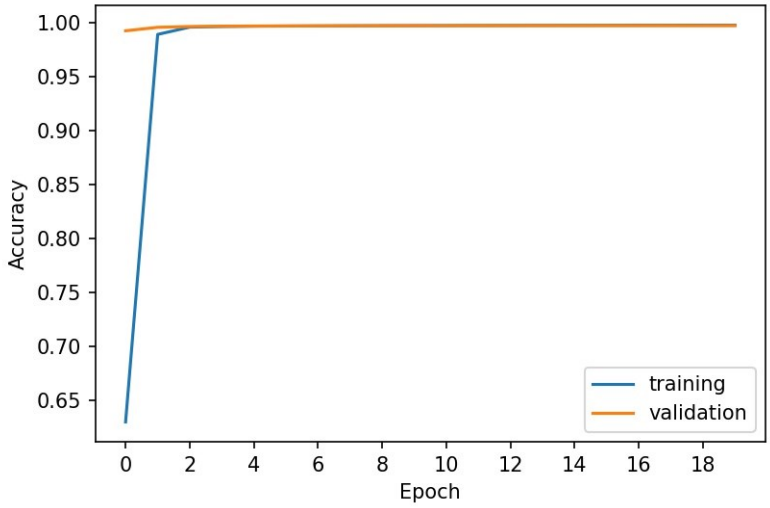

Figure 13. Wiki Model Accuracy on Training and Validation Sets

The second model called Tashkeela model that was trained on Tashkeela set for 88 epochs. Figures 14 and 15 show the model performance in term of loss and accuracy on the training and validation sets. We can observe that after epoch 40 the model is

overfitting the training data due to the small size of the Tashkeela set and the large size of the model. Yet, the model is able to stabilize after epoch 70.

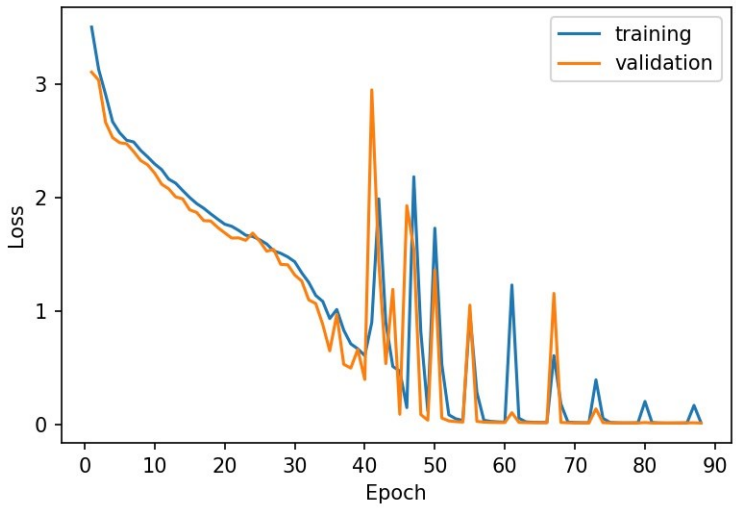

Figure 14. Tashkeela Model Loss on Training and Validation Sets

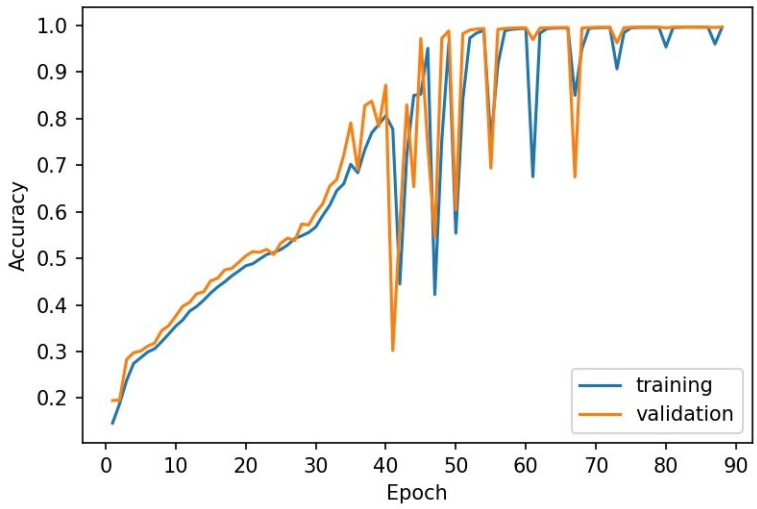

Figure 15. Tashkeela Model Accuracy on Training and Validation Sets

4.6 Evaluation Metrics

We evaluate our models with the following metrics: accuracy, BLEU score, CER, WER, and FP/Changes. Accuracy is a general metric that measures the overall

performance of a model. CER, WER, and BLEU are more specific metrics that measure the performance of the model for specific tasks such as speech recognition, machine translation and spelling correction.

4.6.1 Accuracy

Spelling correction can be considered as a classification task. In this kind of tasks, four types of predictions can be produced by a model. These types are true positive (TP), true negative (TN), false positive (FP), and false negative (FN). When a letter x is correctly predicted as letter x this is a TP prediction, when a letter is not x and correctly predicted as not x this is a TN prediction, when a letter is not x is incorrectly predicted as x this is a FP prediction, when a letter x is incorrectly predicted as not x this is a FN prediction.

The accuracy is calculated using Equation 7 and the higher value, the better the performance of the model.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{7}$$

4.6.2 BLEU Score

Bilingual Evaluation Understudy (BLEU) is a metric often used in machine translation tasks to measure how close the translated text that was generated by a model to the reference translation. In this work, spelling correction can also be considered a translation task, when a model translates an error text into a correct text.

Post (2018) introduced a unified version of BLEU called SACREBLEU to facilitate comparison between results from different papers. Thus, we used

SACREBLEU in our work. Like accuracy, the higher BLEU score, the better the performance of the model.

4.6.3 CER

Character Error Rate (CER) is the ratio of the incorrect characters in the text. It can be calculated using the following equation:

$$CER = \frac{S+D+I}{N} \tag{8}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, and N is the number of characters in the target sequences.

The performance of the model is better when CER value is low.

4.6.4 WER

Word Error Rate (WER) similar to CER but it works on word-level. It calculated using the following equation:

$$WER = \frac{S+D+I}{N} \tag{9}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, and N is the number of words in the target sequences.

4.6.5 FP/Changes

Abandah *et al* (2021) introduced this metric in order to obtain the error rate in the predicted text. It is calculated by dividing the false positive cases in the predicted text over the number of changes (artificial errors) in the input text.

Chapter 5

Experiments and Results

5.1 Introduction

In this chapter, we will discuss the experiments we conducted in order to achieve the optimal results. We used accuracy and BLEU score to show the models' ability to produce a well-structured and understandable text. As for the ability to correct, we used the remaining metrics CER, WER, and FP/Changes. We also used confusion matrix to show the FP, FN, TP, and TN for each of our targeted characters in the test set. We compared our results with the ones reported in (Abandah *et al*., 2021). Lastly, we represent our work using graphical user interface.

5.2 Data Scarcity

In general, the model task is to correct a character with error and to output the correct character without any modification. Wiki model does that with no issues but Tashkeela model does not. The reason for that is the size of the set, when the Tashkeela set is considered a small set. To illustrate this problem, Table 10 contains a simple sentence with no errors and we modified it by inserting seven errors. Tashkeela model corrected five errors out of seven but it also modified four characters that were supposed to stay as they are. Wiki model corrected five errors out of seven with no modification.

The solution for this problem is to use transfer learning. We trained Tashkeela model to output the same input that was provided to it. In this case we used the target sequences that contain the correct text. Later on, we transfer the model weights into a

new one and train it to correct the text. Using transfer learning did not change the five corrected characters by Tashkeela model but it helped reducing the modified characters from four to one.

Table 10. Example on Models Performance

Source	Text
Original Text	بدأت الفرق الإنجليزية تحضيراتها لبطولة الدوري بعد عودة اللاعبين الذين شاركوا مع منتخباتهم في بطولة الأمم الأوربية
Text with Errors	بدأت الفرق الانجليزيه تحضيراتها لبطوله الدوري بعد عودة اللاعبين الذين شاركو مع منتخباتهم في بطولة الامم الاوربيه
Tashkeela Model Prediction	بدأة الفرق الانجليزية تحضيراتها لبطوله الدوري بعد عوده اللأعبين الذين شاركوا مع منتخباتهم في بطوله الأمم الأوربية
Tashkeela Model with Transfer Learning Prediction	بدأت الفرق الانجليزية تحضيراتها لبطوله الدوري بعد عودة اللاعبين الذين شاركوا مع منتخباتهم في بطوله الأمم الأوربية
Wiki Model Prediction	بدأت الفرق الإنجليزية تحضيراتها لبطولة الدوري بعد عودة اللاعبين الذين شاركوا مع منتخباتهم في بطولة الامم الاوربية

5.3 Model Size

The two models were able to correct soft spelling mistakes. The CER on Test200 set for Wiki model was 3.21%, and for Tashkeela model was 2.18%. These results are not better than the 1.28% that was reported in (Abandah *et al.*, 2021). Therefore, fine-tuning is required to achieve better results.

We began by reducing the model size from six layers to four and two layers. Table 11 shows the rest of the model parameters.

Table 11. Model Configurations with Various Numbers of Layers

Parameter	Conf. 1	Conf. 2	Conf. 3 (base)
N	2	4	6
d_{model}	64	128	512
d_{ff}	128	512	2,048
h	8	8	8
p_{drop}	0.1	0.1	0.1
Batch Size	128	128	128

In Figure 16, we observe that Tashkeela model performance using four-layer configuration is nearly identical to six-layer configuration with an accuracy of 99.65% for the four layers and 99.62% for the six layers. With similar behavior Wiki model performance using four-layer configuration is nearly identical to six-layer configuration with an accuracy of 99.75% for four layers and 99.77% for six layers.

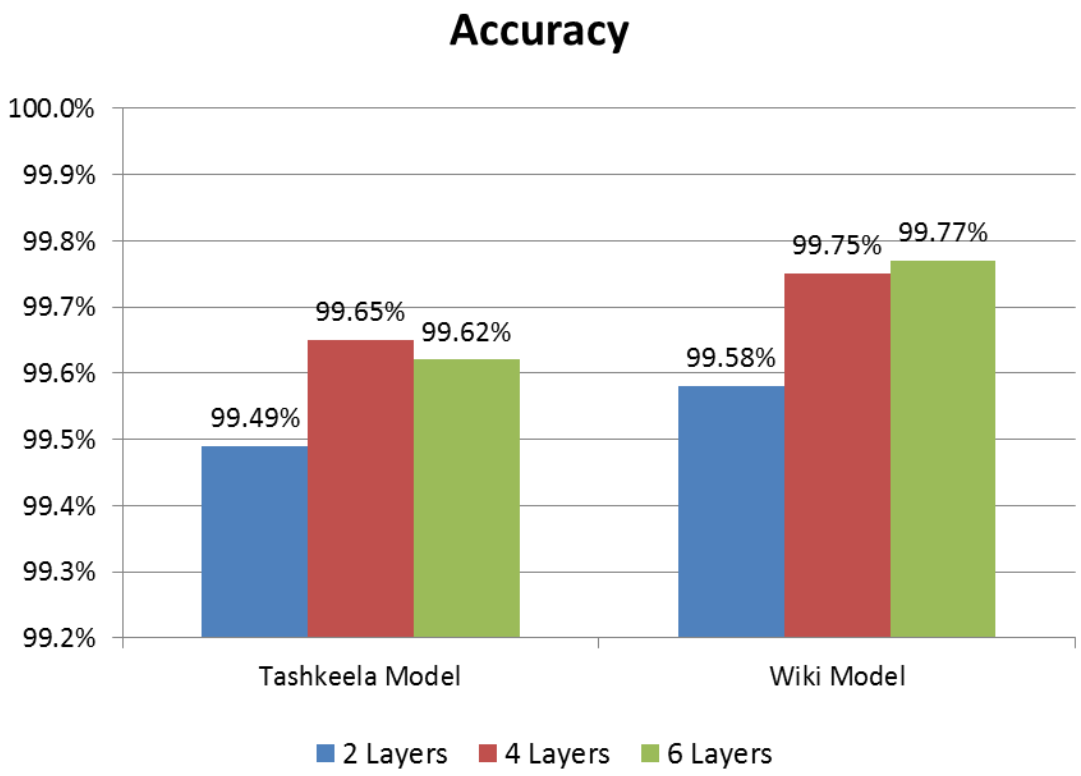

Figure 16. Tashkeela and Wiki Models Accuracy Results with Various Numbers of Layers

In Figure 17, the Tashkeela model with four layers performs better in term of BLEU with a score of 95.79% for four layers and 95.35% for six layers. Wiki model scores 96.58% for six layers and 96.33% for four layers. Both models achieved high results in accuracy and BLEU, which indicates that the models can produce a well-structured text like the one provided in the target sequences.

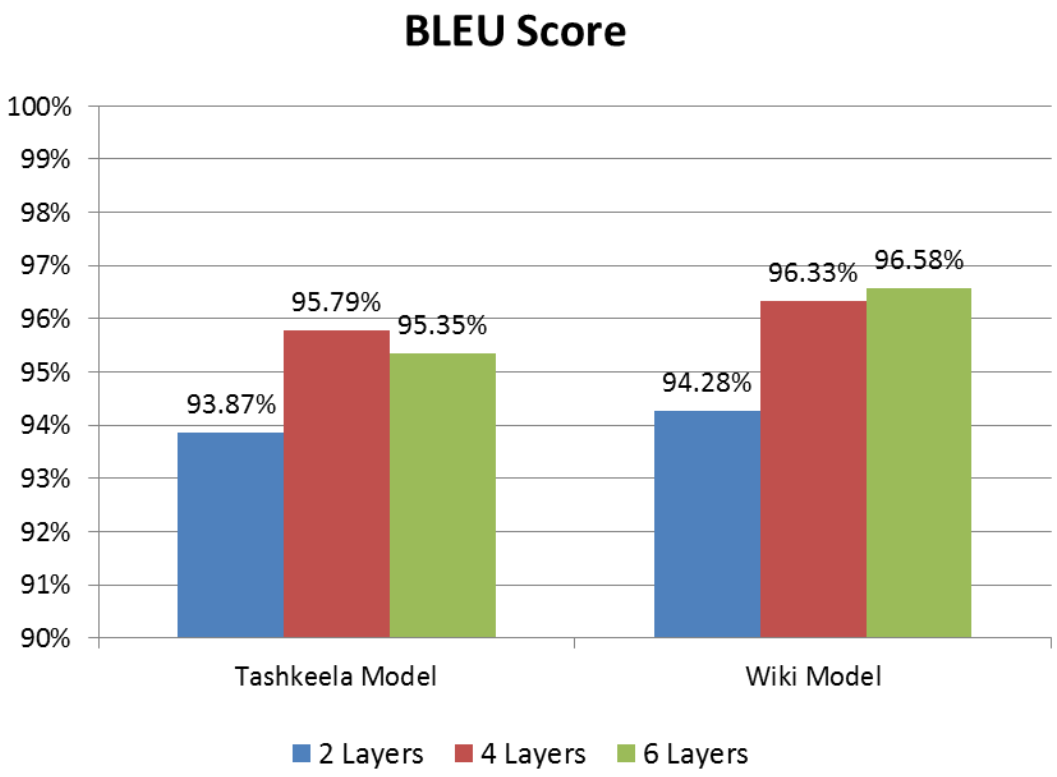

Figure 17. Tashkeela and Wiki Models BLEU Results with Various Numbers of Layers

The two models behave differently with the various numbers of layers due to the different size of each set. Thus, we cannot decide which size is best based on these results only. Therefore, we used CER on Test200 to determine the best model size. In Figure 18, we observe that the four-layer configuration achieves the lowest CER for both models with a score of 2.05%, and 3.11% for Tashkeela model and Wiki model, respectively. Therefore, we adopted four-layer model size.

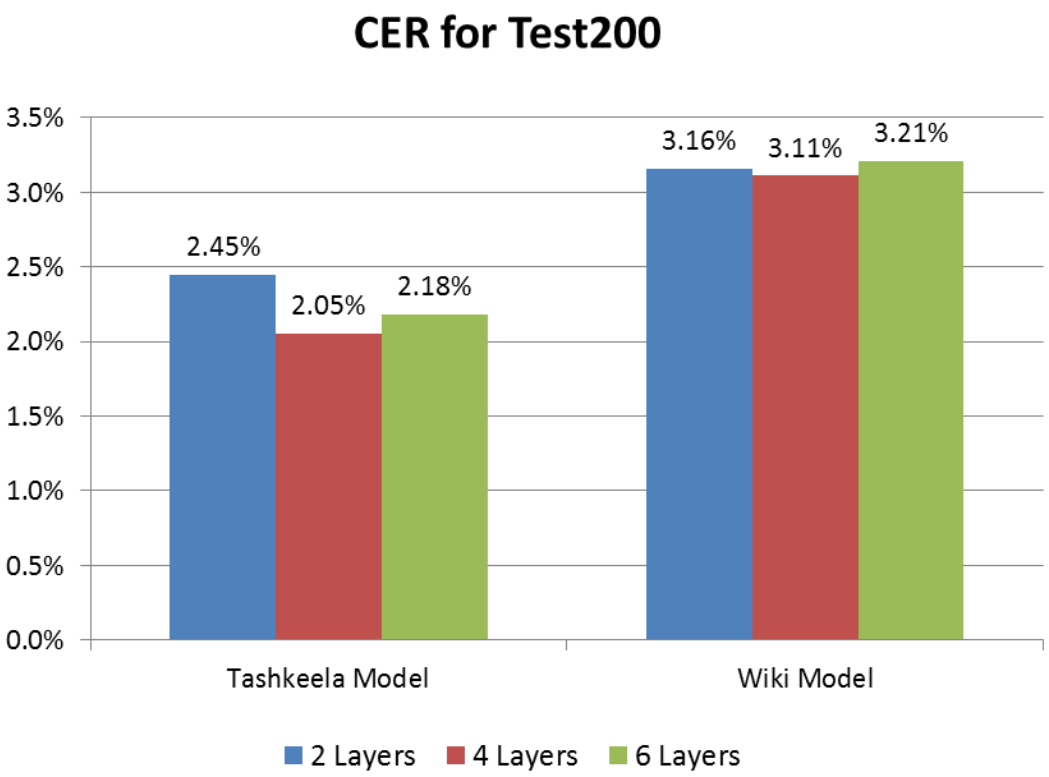

Figure 18. Tashkeela and Wiki Models CER Results on Test200 Set with Various Numbers of Layers

5.4 Error Injection Rate

The results on Test200 that we obtained after reducing the model size were also not better than 1.28% that was reported in (Abandah *et al.*, 2021). So we start changing the 40% error injection rate. We decreased the error injection rate to 30%, the results on Test200 weren't better than the ones of 40% error injection rate, so we went for increasing instead. We increased the rate to 60% – 80% – 90%. The results on Test200 started to improve with the increasing of the error injection rate. The models achieved the best results on Test200 at 90% error injection rate. In Figures 19 and 20, we show the accuracy and BLEU score for both models with various error injection rates. The models at 30% error injection rate achieved best results. Both accuracy and BLEU score decrease with an increase in the error injection rate.

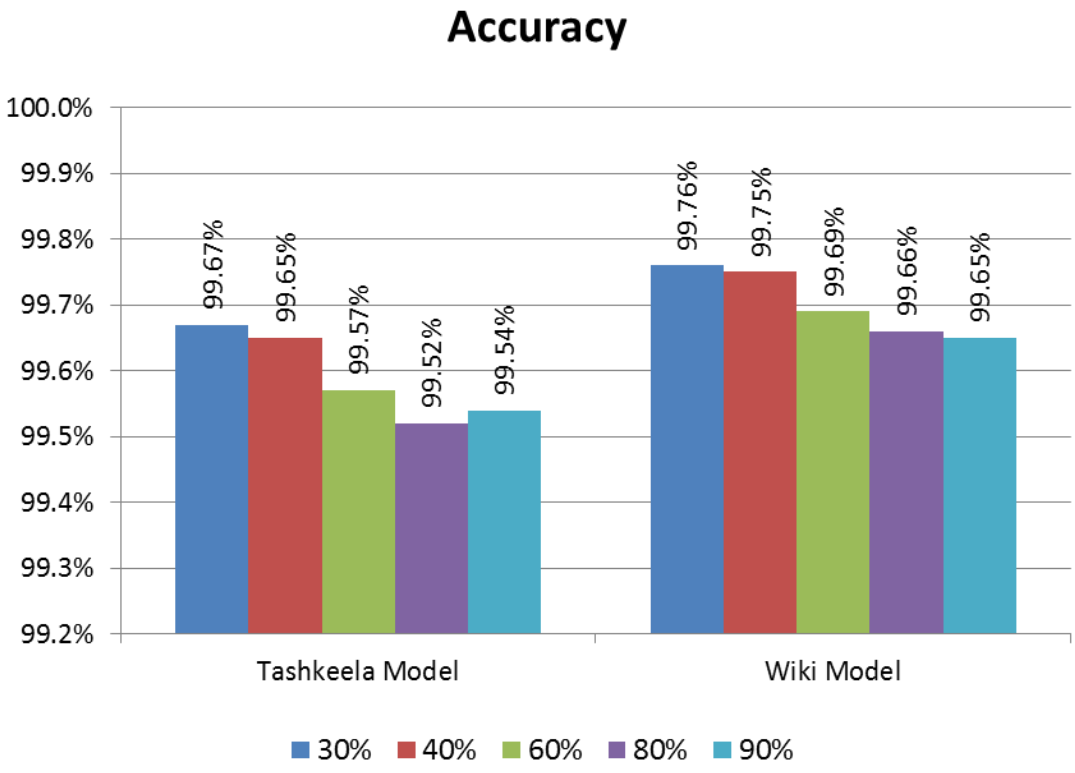

Figure 19. Tashkeela and Wiki Models Accuracy Results with Five Error Rates

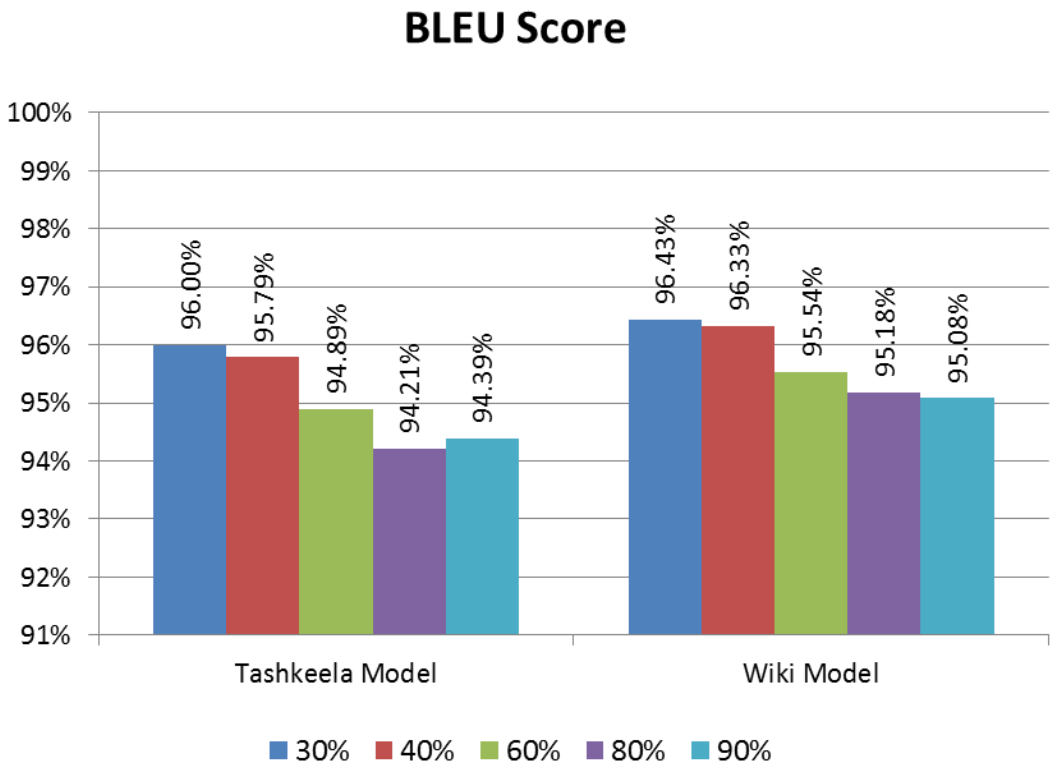

Figure 20. Tashkeela and Wiki Models BLEU Results with Five Error Rates

This behavior extends to the CER and WER metrics as shown in Figures 21 and 22. It also like the behavior of the BiLSTM models that reported in (Abandah *et al.*, 2021). The reason behind this behavior is the fact that these metrics are calculated for the entire set of characters. Therefore, we used FP/Changes metric to show that increasing the error injection rate helps the model to correct better. The CER results on Test200 also support this assumption since the models achieved the best results at 90% error injection rate.

In Figure 23, we show the results of FP/Changes metric to assert the observation that increasing the error injection rate helps the model to correct better. One can observe that both models trained with 90% error injection rate have the lowest values. While the ones trained with 30% error injection rate have the highest values even though they have fewer errors.

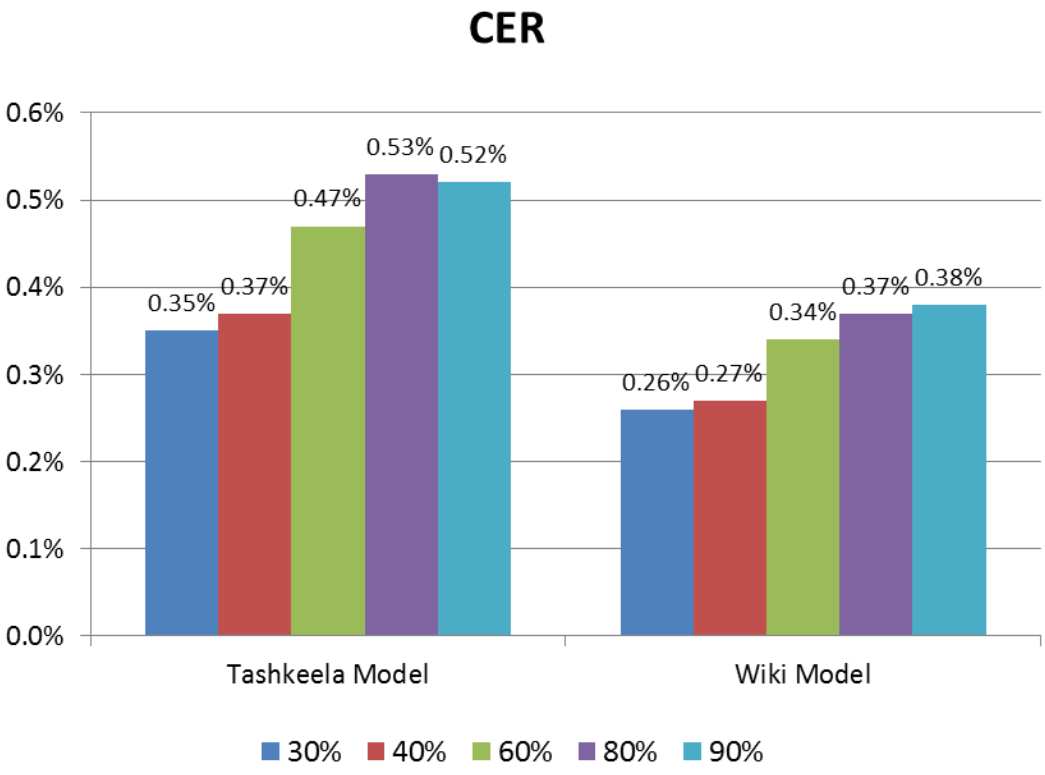

Figure 21. Tashkeela and Wiki Models CER Results with Five Error Rates

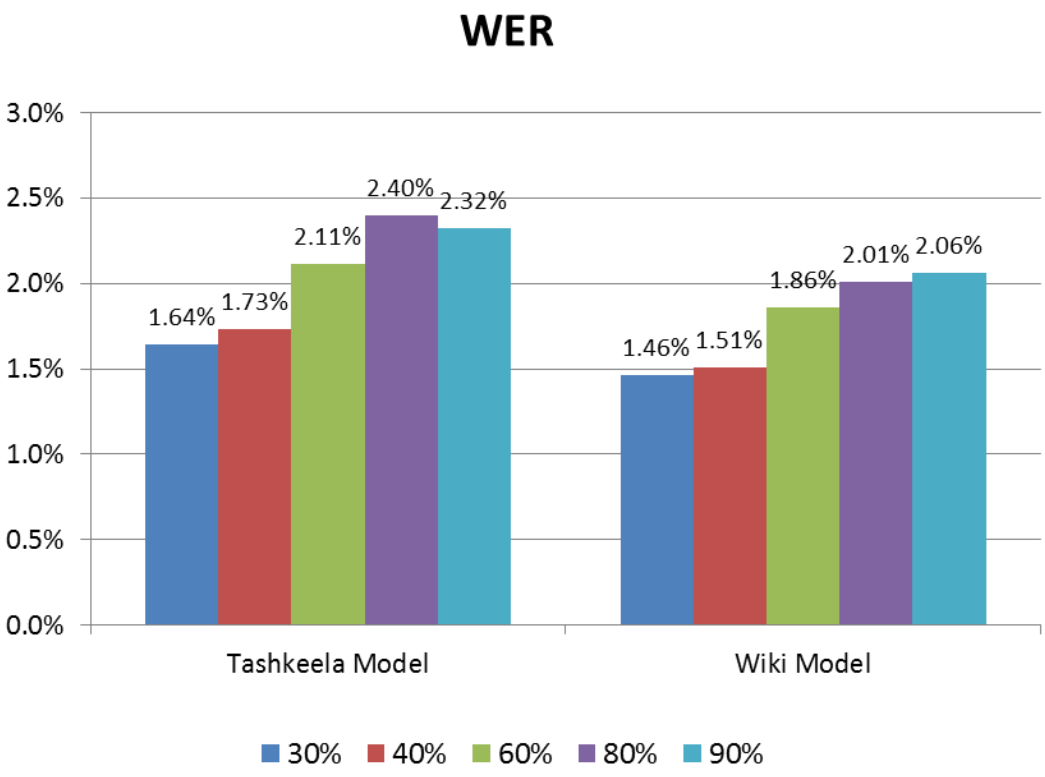

Figure 22. Tashkeela and Wiki Models WER Results with Five Error Rates

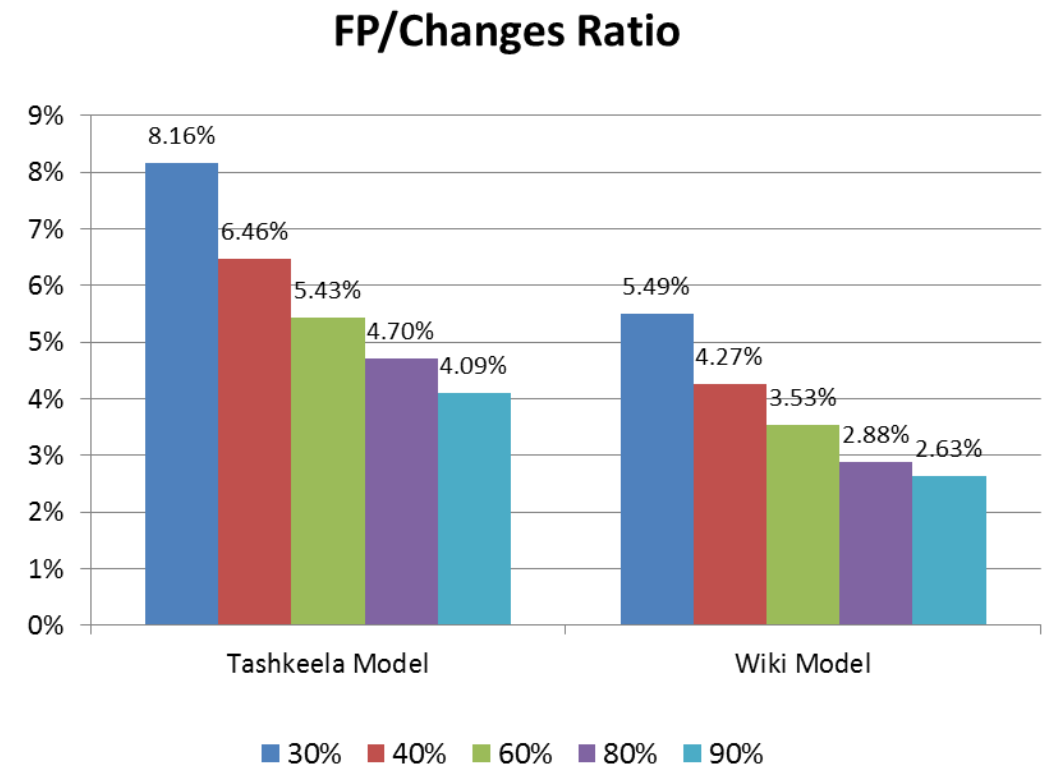

Figure 23. Tashkeela and Wiki Models FP/Changes Results with Five Error Rates

The results on Test200 in Figure 24 show that the Wiki model shows lowest CER of 0.86% with 90% error injection rate. This is a reduction of 72.35% compared to the 3.11% that is obtained with 40% error injection rate. The Tashkeela model does not show this large improvement with the error injection rate increase. The score is 1.88% with 90% error injection rate, it is only 8.29% reduction compared to the 2.05% CER with 40% error injection rate. These results demonstrate the importance of the dataset size that is used in transformer model. Small sets like Tashkeela are likely to work better with LSTM models rather than transformer model.

Lastly, we tested whether increasing the error injection rate to 100% will improve the results of the Wiki model. The CER was 0.89% which is not better than the one obtained using 90% error injection rate.

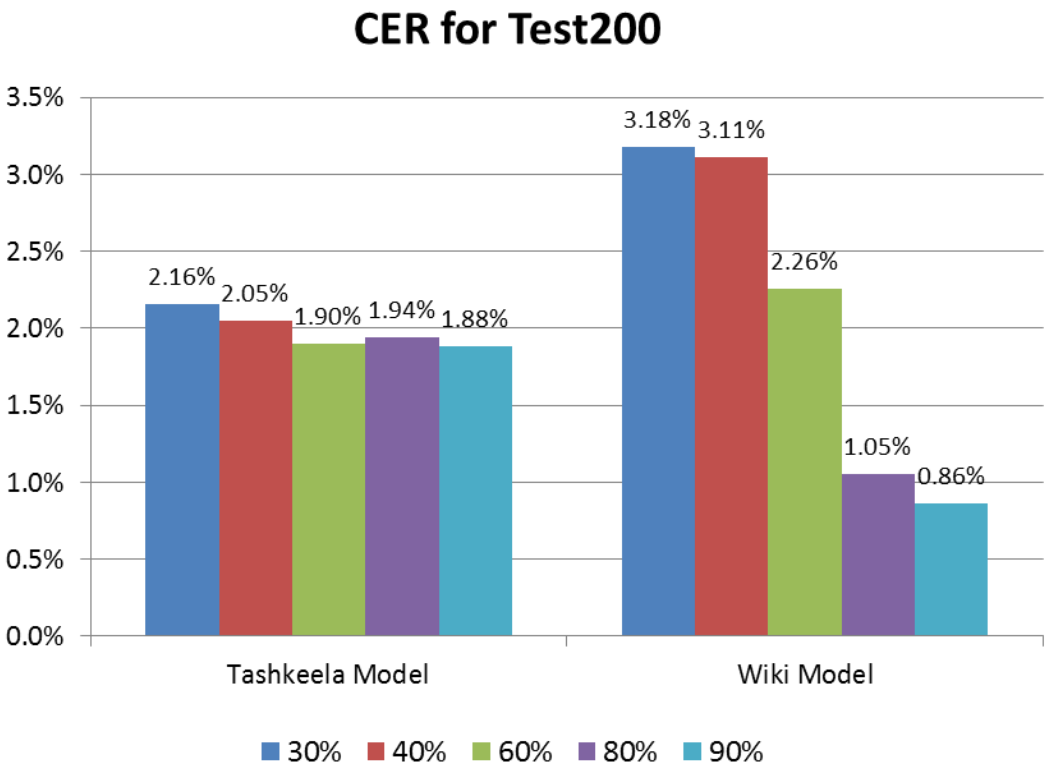

Figure 24. Tashkeela and Wiki Models CER Results on Test200 Set with Five Error Rates

5.5 Confusion Matrix

In Figure 25, we show the confusion matrix of Wiki model that was trained with 90% error injection rate. The matrix shows that most confusion occurs with the letters (ا), (أ), and (إ). We can observe that letter (ا) got confused with all letters that belong to its set except the letter (ء).

Actual Letter \ Predicted Letter	ـوا	ـه	اء	ـو	ـت	ء	آ	أ	ؤ	إ	ئ	ا	ة	ى
ـوا	105	0	0	7	0	0	0	0	0	0	0	0	0	0
ـه	0	1131	0	0	19	0	0	0	0	0	0	0	74	0
اء	0	0	484	0	0	0	0	0	0	0	0	2	0	0
ـو	3	0	0	853	0	0	0	0	0	0	0	0	0	0
ـت	0	13	0	0	2313	0	0	0	0	0	0	0	23	0
ء	0	0	0	0	0	60	0	0	0	0	0	0	0	0
آ	0	0	0	0	0	0	157	11	0	1	0	6	0	0
أ	0	0	0	0	0	0	15	3552	1	60	5	58	0	0
ؤ	0	0	0	0	0	0	1	2	164	0	3	3	0	0
إ	0	0	0	0	0	0	1	44	0	1306	0	76	0	0
ئ	0	0	1	0	0	1	0	1	7	0	745	2	0	0
ا	0	0	4	0	0	0	8	219	4	88	7	26430	0	10
ة	0	29	0	0	19	0	0	0	0	0	0	0	6462	0
ى	0	0	0	0	0	0	0	0	0	0	0	17	0	1515

Figure 25. Confusion Matrix of Wiki Model Trained with 90% Error Injection Rate

5.6 Model Timing

In Figures 26 and 27, we show the training time and the number of epochs for Wiki and Tashkeela models. The longest time was 24.9 hours and 28.2 minutes for Wiki model and Tashkeela model, respectively, using 4-layer configuration with 40% error rate. The largest number of epochs was 89 and 145 epochs for Wiki model using 4-layer configuration with 40% error rate and Tashkeela model using 2-layer configuration with 40% error rate, respectively.

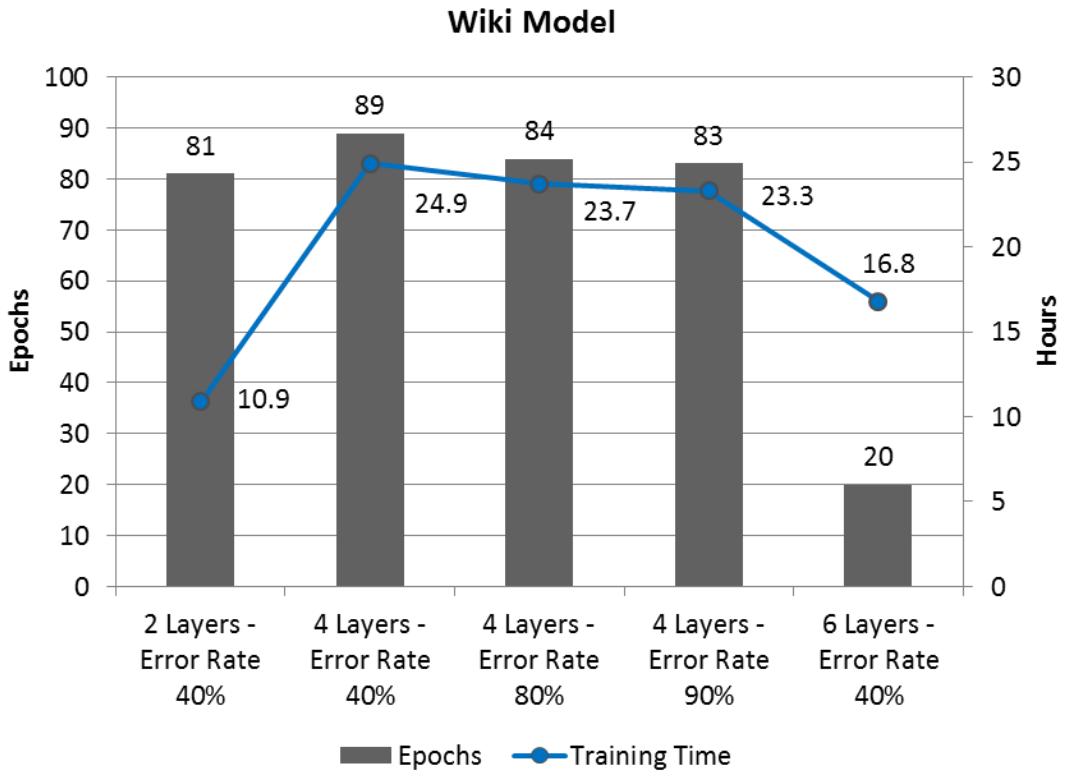

Figure 26. Wiki Model Timing

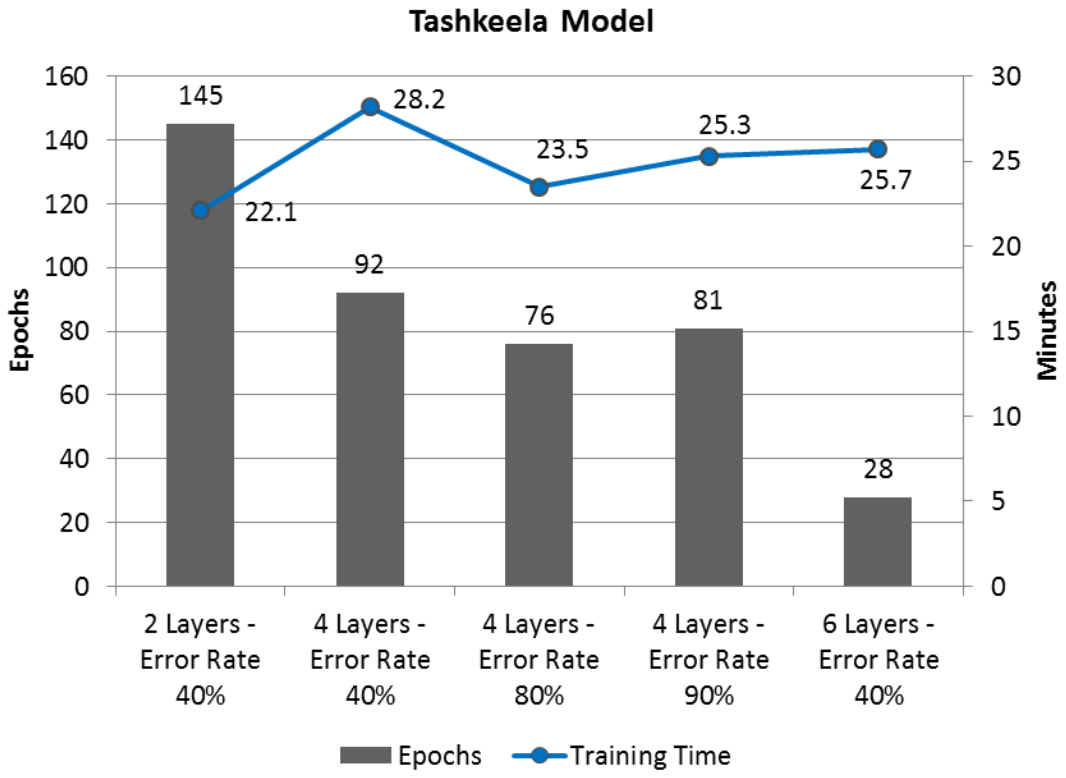

Figure 27. Tashkeela Model Timing

5.7 Comparison

We compared our results on Test200 with the ones reported in (Abandah *et al*., 2021) using the Tashkeela and ATB3 datasets and BiLSTM network as shown in Figure 28. Our Wiki transformer model achieves a CER of 0.86%, which is lower than the 1.28% obtained by Tashkeela BiLSTM model. On the other hand, our Tashkeela transformer model does not outperform Tashkeela and ATB3 BiLSTM models.

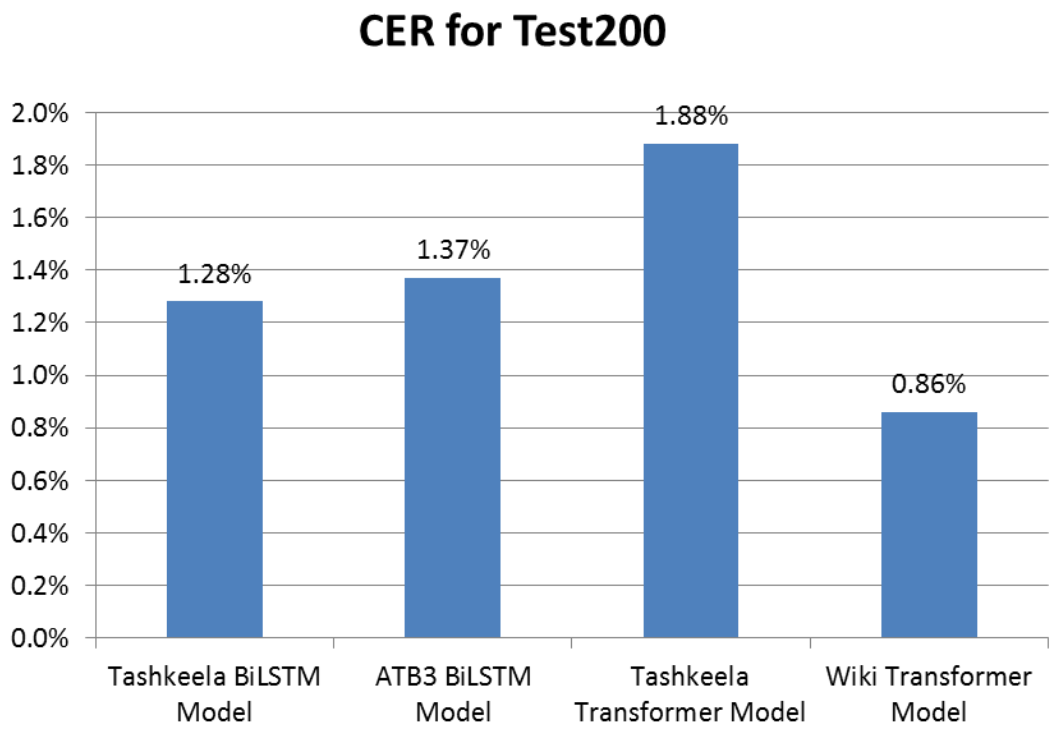

Figure 28. Our Tashkeela and Wiki Transformer Models CER Results on Test200 Compared to Tashkeela and ATB3 BiLSTM Models in (Abandah *et al*., 2021)

5.8 Discussion

Given that the work in (Abandah *et al*., 2021) uses a maximum sequence length of 400, while in this work we use a sequence length of 300, we conducted a final experiment to test the effects of sequence length on the obtained results. We re-trained the Tashkeela transformer model with maximum sequence length of 200 and 400, and

we used 90% and 40% error injection rates, since these are the best rates for transformer and BiLSTM, respectively. The results on the Test200 are shown in Figure 29.

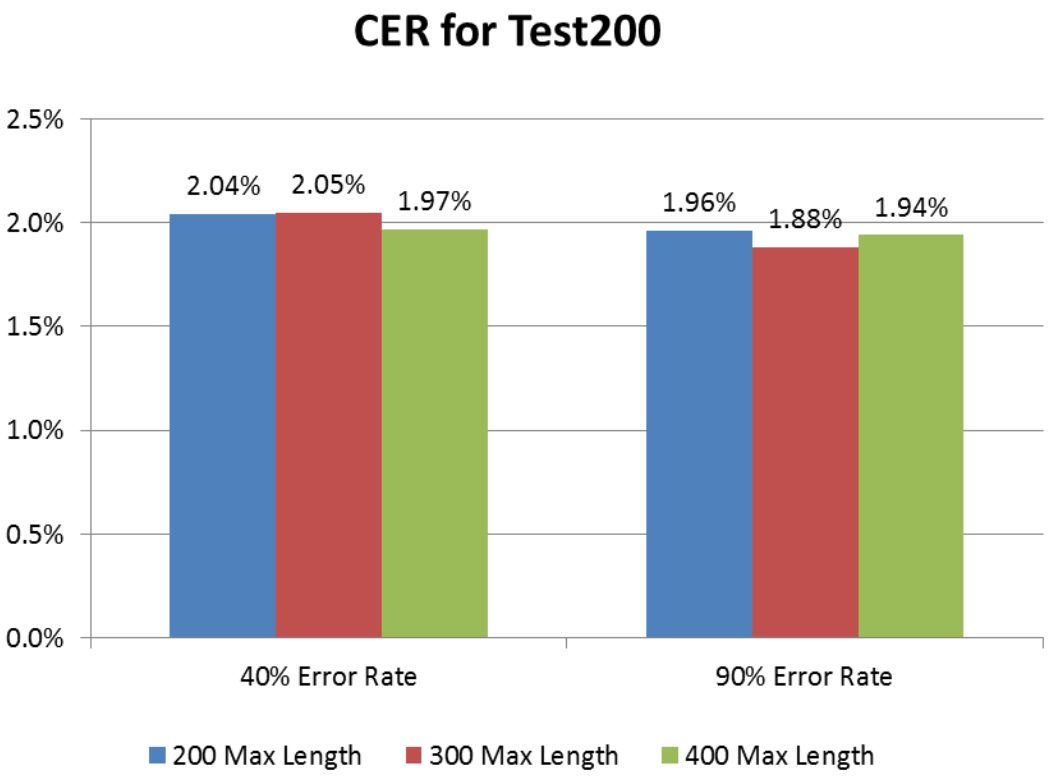

Figure 29. Tashkeela Model CER Results Using Various Maximum Sequence Lengths

At 40% error injection rate, a maximum sequence length of 400 has a better result than a maximum sequence length of 300. Yet, this result is still not better than 1.28% of the BiLSTM Tashkeela. Using a 90% error injection rate lowered CER values for all lengths. Moreover, a maximum sequence length of 300 seems to be suitable more than 400 and 200 for our Tashkeela transformer model at 90% error injection rate. From these results, we can say that the maximum sequence length is not the reason behind the performance of our Tashkeela transformer model. The main reason is the set size that is not suitable for a transformer model that was built from scratch.

As for Wiki model performance, we can say that using transformer model, which accepts high rates of error injection and utilizes self-attention mechanism, is the reason behind the good performance of the model.

5.9 Work Representation

We designed a graphical user interface to represent our work using Tkinter library. It takes an input sentence that contains only Arabic letters with max length of 300 as shown in Figure 30. After pressing the correct button, our Wiki model will be initialized and correct the sentence that was given as an input as shown in Figure 31.

Figure 30. Input Sentence in the Graphical User Interface

Figure 31. Model Prediction after Pressing Correct Button

Chapter 6

Conclusions and Future Work

In recent years, transformer models have proven to be powerful neural networks that can handle many NLP tasks quite well. Spelling correction is one of these tasks where transformers started to replace LSTM models and language models in the correction process.

In this thesis, we implemented a machine learning approach based on the original transformer to correct Arabic soft spelling mistakes in modern standard Arabic and classical Arabic. We handled four types of soft spelling mistakes, namely, confusion among *hamza* shapes, both shapes of *alef* at the end of the word, insertion and omission of *alef* after *waw aljamaea*, and confusion among *teh*, *teh marbuta*, and *heh* at the end of the word.

The previous work lacks a large annotated dataset designed for Arabic spelling correction systems. To overcome this problem, we used artificial errors that were generated using a stochastic error injection approach that was proposed in (Abandah *et al*., 2021).

We used Wiki-40B and Tashkeela sets for training and evaluation, and Test200 set to report our results. We produced two transformer models: a Wiki model and a Tashkeela model.

We compared our results on Test200 set with the ones that were obtained using BiLSTM models in (Abandah *et al*., 2021). Our Wiki model achieves a CER reduction of 32.8% compared with the best BiLSTM model that has been reported.

Additionally, the Wiki model can correct 97.37% of the artificial errors that were injected into the test set.

For future work, we suggest looking at more advanced transformer models such as BART (Lewis *et al*., 2019) and T5 (Raffel *et al*., 2019). These models contain encoder-decoder architecture and may be useful in spelling correction. We also suggest the proposed model to correct other types of Arabic spelling mistakes such as semantic errors.

REFERENCES

Abandah, G., Suyyagh, A., and Khader, M. (2021), Correcting Arabic Soft Spelling Mistakes Using BiLSTM-Based Machine Learning. **arXiv preprint** arXiv:2108.01141.

Abdellah, Y., Lhoussain, A. S., Hicham, G., and Mohamed, N. (2020), Spelling Correction for the Arabic Language Space Deletion Errors. **Procedia Computer Science**, 177, 568-574.

Al-Ameri, AMH. (2015), Common Spelling Mistakes Among Students of Teacher Education Institutes. **The Islamic College University Journal**, 1(33), 445–474.

Alammar, J. (2018), The Illustrated Transformer. **Github**. http://jalammar.github.io/illustrated-transformer/

Alkhatib, M., Monem, A. A., and Shaalan, K. (2020), Deep Learning for Arabic Error Detection and Correction. **ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)**, 19(5), 1-13.

AlShenaifi, N., AlNefie, R., Al-Yahya, M., and Al-Khalifa, H. (2015, July), ARIB@ QALB-2015 Shared Task: A Hybrid Cascade Model for Arabic Spelling Error Detection and Correction. **In Proceedings of the Second Workshop on Arabic Natural Language Processing,** pp. 127-132.

Antoun, W., Baly, F., and Hajj, H. (2020, a), Arabert: Transformer-Based Model for Arabic Language Understanding. **arXiv preprint** arXiv:2003.00104.

Antoun, W., Baly, F., and Hajj, H. (2020, b), AraGPT2: Pre-Trained Transformer for Arabic Language Generation. **arXiv preprint** arXiv:2012.15520.

Attia, M., Pecina, P., Samih, Y., Shaalan, K., and Van Genabith, J. (2016), Arabic Spelling Error Detection and Correction. **Natural Language Engineering**, 22(5), 751-773.

Azmi, A. M., Almutery, M. N., and Aboalsamh, H. A. (2019), Real-Word Errors in Arabic Texts: A Better Algorithm for Detection and Correction. **IEEE/ACM Transactions on Audio, Speech, and Language Processing**, 27(8), 1308-1320.

Fadel A, Tuffaha I, Al-Ayyoub M, et al. (2019), Arabic Text Diacritization Using Deep Neural Networks. **In: 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)**, IEEE, pp. 1–7.

Guo, M., Dai, Z., Vrandečić, D., and Al-Rfou, R. (2020, May), Wiki-40B: Multilingual Language Model Dataset. **In Proceedings of the 12th Language Resources and Evaluation Conference**, pp. 2440-2452.

Hochreiter, S., and Schmidhuber, J. (1997), Long Short-term Memory. **Neural computation**, 9(8), 1735-1780.

Kuznetsov, A., and Urdiales, H. (2021), Spelling Correction with Denoising Transformer. **arXiv preprint** arXiv:2105.05977.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., and Zettlemoyer, L. (2019), Bart: Denoising Sequence-To-Sequence Pre-Training for Natural Language Generation, Translation, and Comprehension. **arXiv preprint** arXiv:1910.13461.

Madhfar, M. A. H., and Qamar, A. M. (2020), Effective Deep Learning Models for Automatic Diacritization of Arabic Text. **IEEE Access**, 9, 273-288.

Mubarak, H., and Darwish, K. (2014, October), Automatic Correction of Arabic Text: A Cascaded Approach. **In Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP)**, pp. 132-136.

Noaman, H. M., Sarhan, S. S., and Rashwan, M. (2016), Automatic Arabic Spelling Errors Detection and Correction Based on Confusion Matrix-Noisy Channel Hybrid System. **Egypt Comput Sci J**, 40(2).

Post, M. (2018), A Call for Clarity in Reporting BLEU Scores. **arXiv preprint** arXiv:1804.08771.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., and Liu, P. J. (2019), Exploring the Limits of Transfer Learning with a Unified Text-to-text Transformer. **arXiv preprint** arXiv:1910.10683.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986), Learning Representations by Back-propagating Errors. **Nature**, 323(6088), 533-536.

Schuster, M., and Paliwal, K. K. (1997), Bidirectional Recurrent Neural Networks. **IEEE transactions on Signal Processing**, 45(11), 2673-2681.

Tran, H., Dinh, C. V., Phan, L., and Nguyen, S. T. (2021), Hierarchical Transformer Encoders for Vietnamese Spelling Correction. **arXiv preprint** arXiv:2105.13578.

UNESCO. (2019), World Arabic Language Day. **UNESCO**. https://en.unesco.org/commemorations/worldarabiclanguageday

van den Berg , T. (2020), Parametric Neural Networks. **CQF Institute**. https://www.cqfinstitute.org/sites/default/files/param-nn-1.3f.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., and Polosukhin, I. (2017), Attention Is All You Need. **In Advances in neural information processing systems**, pp. 5998-6008.

Xu, P., Yang, W., Zi, W., Tang, K., Huang, C., Cheung, J. C. K., and Cao, Y. (2020), Optimizing Deeper Transformers on Small Datasets: An Application on Text-to-sql Semantic Parsing. **arXiv preprint** arXiv:2012.15355.

Zaghouani, W., Mohit, B., Habash, N., Obeid, O., Tomeh, N., Rozovskaya, A., and Oflazer, K. (2014), Large Scale Arabic Error Annotation: Guidelines and Framework. **Carnegie Mellon University**.

Zhang, S., Huang, H., Liu, J., and Li, H. (2020), Spelling Error Correction with Soft-masked BERT. **arXiv preprint** arXiv:2005.07421.

Zhao, Y., Yang, X., Wang, J., Gao, Y., Yan, C., and Zhou, Y. (2021), BART Based Semantic Correction for Mandarin Automatic Speech Recognition System. **arXiv preprint** arXiv:2104.05507.

تصحيح الأخطاء الإملائية الشائعة في اللغة العربية باستخدام التعلم الآلي العميق والمحولات

إعداد

محمد عادل فاضل القره غولي

المشرف

الأستاذ الدكتور غيث علي عبندة

ملخص

تعتبر الأخطاء الإملائية من المشاكل الشائعة في اللغة العربية. هناك عدة تقنيات تم استخدامها لحل هذه المشكلة مثل مصفوفات الخطأ ونماذج اللغة والشبكات العصبونية. في السنوات الأخيرة تم ابتكار شبكة عصبونية تدعى بالمحولة لكي تقوم بعملية الترجمة الآلية. ومنذ ابتكارها أصبحت هذه المحولة ومشتقاتها حل شائع لأغلب التطبيقات في مجال معالجة اللغات الطبيعية.

نهدف في هذه الرسالة إلى استخدام المحولة لتصحيح أربعة أنواع من الأخطاء الإملائية الشائعة وهي: الخلط بين أنواع الهمزة والاختلاف بشكل الألف في نهاية الكلمة وحذف وإضافة الألف بعد واو الجماعة والخلط بين التاء والتاء المربوطة والهاء في نهاية الكلمة.

لقد استخدمنا الأخطاء المصطنعة للتدريب والتقييم. هذه الأخطاء تم إنشاؤها بواسطة نهج يسمى إضافة الخطأ بطريقة عشوائية. ولقد استخدمنا قاعدة بيانات Wiki-40B وقاعدة بيانات Tashkeela للتدريب والتقييم، و مجموعة Test200 لتقييم النتائج النهائية. لقد أنتجنا نموذجين من المحولات: نموذج Wiki ونموذج Tashkeela.

لقد استخدمنا ثلاثة تصاميم وخمس معدلات لإضافة الخطأ. أفضل نموذج قمنا بتدريبه كان الذي يستخدم قاعدة بيانات Wiki-40B وبمعدل إضافة خطأ 90% وتصميم الأربعة طبقات. استطاع هذا النموذج تصحيح 97.37% من الأخطاء المصطنعة التي أضيفت إلى مجموعة التقييم، إضافة إلى ذلك تمكن النموذج من الحصول على نسبة خطأ على مستوى الحرف بمقدار 0.86% باستخدام مجموعة تقييم تحتوي على أخطاء إملائية شائعة حقيقية. نسبة الخطأ هذه أقل بمقدار 32.8 عن أفضل النتائج السابقة التي سجلت باستخدام شبكات ذاكرة ثنائية الاتجاه طويلة المدى (BiLSTM).