Correcting Arabic Soft Spelling Mistakes using BiLSTM-based Machine Learning

Gheith Abandah, Ashraf Suyyagh, Mohammed Z. Khedher
Department of Computer
Engineering
University of Jordan
Amman, Jordan 11942

***Abstract*—Soft spelling mistakes are a class of mistakes that is widespread among native Arabic speakers and foreign learners alike. Some of these mistakes are typographical in nature. They occur due to orthographic variations of some Arabic letters and the complex rules that dictate their correct usage. Many people forgo these rules, and given the identical phonetic sounds, they often confuse such letters. In this paper, we investigate how to use machine learning to correct such mistakes given that there are no sufficient datasets to train the correction models. Soft errors detection and correction is an active field in Arabic natural language processing. We generate training datasets using proposed transformed input approach and stochastic error injection approach. These approaches are applied to two acclaimed datasets that represent Classical Arabic and Modern Standard Arabic. We treat the problem as character-level, one-to-one sequence transcription problem. This one-to-one transcription of mistakes that include omissions and deletions is possible with adopted simple transformations. This approach permits using bidirectional long short-term memory (BiLSTM) models that are more effective to train compared to other alternatives such as encoder-decoder models. Based on investigating multiple alternatives, we recommend a configuration that has two BiLSTM layers, and is trained using the stochastic error injection approach with error injection rate of 40%. The best model corrects 96.4% of the injected errors and achieves a low character error rate of 1.28% on a real test set of soft spelling mistakes**

Keywords*—Arabic text; natural language processing; spelling mistakes; recurrent neural networks; bidirectional long short-term memory

I. Introduction

Arabic is one of the world's five major languages with over 290 million native speakers and a total of 422 million world speakers [1], [2]. Arabic is the fourth most common language on the Internet [3], [4] and the fastest growing language online [3]. A recent British Council report [5] ranks Arabic as the fourth needed language for the future based on economic and market factors, diplomatic and security priorities, mobility, tourism, and public interest.

The Arabic language has 28 letters which we show alongside their Unicode encoding in Table I. Of the 44 letters and diacritics listed, eight are letter variants: 0621–0626, 0629, and 0649.[1]

Western linguists distinguish between two forms of standard Arabic: Classical Arabic (CA) and Modern Standard Arabic (MSA). CA is the language of the Quran, ancient religious and liturgical texts, and old Arabic literature. MSA is the modern form that is syntactically, morphologically, and phonologically based on CA. MSA is the primary form of Arabic language used in education, business, media, news, and drafting laws and regulations. The contrast between MSA and CA "is mostly reflected in topic, vocabulary, and style rather than grammatical structure" [7].

Arabic is a diglossic language and Arabic linguists and speakers refer to both CA and MSA as *al-fusha* الفصحى to differentiate between the standard forms and the colloquial variants spoken throughout the Arab world. These dialects are not standardized and vary significantly socially and geographically. Furthermore, they are widely popular in use on social and messaging applications as well on the Internet [8].

The U.S. state department categorizes Arabic among the exceptionally difficult languages to learn for native English speakers [9]. Consequently, it is vital to develop and modernize automated tools that aid native speakers and learners alike in communicating in Arabic using correct grammar and spelling, especially in online communication. To this end, our previous work [10], [11], [12] investigated machine learning and hybrid approaches for Arabic text diacritization with recurrent neural networks (RNN). This novel work investigates Arabic text correction using RNN in contrast to traditional techniques.

Research efforts have normally focused on a subset of the typical spelling and writing mistakes encountered in Arabic texts; usually by addressing one or two at a time. We target in this work the most common type of spelling mistakes, which is the soft spelling mistakes.

A. Soft Spelling Mistakes

Soft spelling mistakes are a special type of lexical and semantic errors that are due to the orthographic variations of some Arabic letters. For example, at the beginning of a word, the Arabic letter *alef* - comes in bare *alef* and and *hamza* forms: *alef with madda above*, *alef with hamza above* and *alef with hamza below*. The former bare *alef* shape is called همزة وصل (*hamzaẗ waṣl*) while the *alef* with *hamza* above or below is called همزة قطع (*hamzaẗ qaṭc*). Replacing one form of *alef* with another is a major case of soft-spelling mistakes [13].

Despite having standard rules for the shape and placement of the *hamza* inside the word, or as known in Arabic الهمزة المتوسطة (*ạl-hamzaẗ ạl-mutawsiṭaẗ*); these rules are quite

[1]The transliteration used in this paper is Romanization using Glosbe transliteration tools [6].

Table I. Arabic Letters and Diacritics in the Unicode Arabic Code Block

x	U+062x	Name	U+063x	Name	U+064x	Name	U+065x	Name
0			ذ	*thal*			ِ	*kasra*
1	ء	*hamza*	ر	*reh*	ف	*feh*	ّ	*shadda*
2	آ	*alef/madda above*	ز	*zain*	ق	*qaf*	ْ	*sukun*
3	أ	*alef/hamza above*	س	*seen*	ك	*kaf*		
4	ؤ	*waw/hamza above*	ش	*sheen*	ل	*lam*		
5	إ	*alef/hamza below*	ص	*sad*	م	*meem*		
6	ئ	*yeh/hamza above*	ض	*dad*	ن	*noon*		
7	ا	*alef*	ط	*tah*	ه	*heh*		
8	ب	*beh*	ظ	*zah*	و	*waw*		
9	ة	*teh marbuta*	ع	*ain*	ى	*alef maksura*		
A	ت	*teh*	غ	*ghain*	ي	*yeh*		
B	ث	*theh*			ً	*fathatan*		
C	ج	*jeem*			ٌ	*dammatan*		
D	ح	*hah*			ٍ	*kasratan*		
E	خ	*khah*			َ	*fatha*		
F	د	*dal*			ُ	*famma*		

complex for natives and learners alike. We can write the middle *hamza* above an *alef* as in the noun 'head' رأس (*rẚs*), or above the letter *waw* as in the noun 'vision' رؤية (*rűyaẗ*), or on a mark نبرة (*nabirẗ*) as in the noun 'well' بئر (*bŷr*), or standalone *hamza* form as in the noun 'reading' قراءة (*qiraˀaẗ*). We can determine the correct spelling by examining the diacritics of the *hamza* itself and those of the preceding letter. Some exceptions do apply.

Similar Arabic spelling rules dictate how we write the *hamza* at the end of a word, which is called الهمزة المتطرفة (*al-hamzaẗ al-mutaṭarifaẗ*). We can write the *hamza* above an *alef* as in the noun 'refuge' ملجأ (*maljẚ*). We can place it above a *waw* as in the verb 'to dare' يجرؤ (*yajrű*) or over the *alef maksura* as in the noun 'ports' الموانئ (*al-mawanŷ*). Finally, it can rest alone as in the noun 'warmth' دفء (*difˀ*). Clearly, the shape that the *hamza* takes based on its possible placement within a word could be quite confusing.

There are other notorious examples in the soft misspellings category that are not necessarily related to the *hamza* . These include inserting or omitting the *alef* following a *waw* letter at the end of a word. A common example for the insertion error is adding an *alef* at the end of present tense verbs that end with a *waw*; such as writing the verb 'to kneel' as يجثوا (*yajtẖwa*) instead of يجثو (*yajtẖw*). We frequently encounter the omission error while spelling conjugated imperative plural verbs. For example, in the imperative sentence 'Don't look' addressing a group of people, the verb can by incorrectly spelled as لا تنظرو (*la tanẓurw*) instead of لا تنظروا (*la tanẓurwa*).

Furthermore, common soft misspellings include confusing the *teh marbuta* at the end of the word for a *teh*. One example is writing the noun 'watches' as ساعاة (*saˁaẗ*) instead of ساعات (*saˁat*). The *teh marbuta* is also often confused for the letter *heh* as in the word for library مكتبه/مكتبة (maktabaẗ/maktabah).

Finally, due to the similar shapes between the letters for *alef maksura/yeh* written at the end of Arabic words, we can see the letters mistakenly interchanged as in the adjective 'interactive' تفاعلى/تفاعلي (*tafaˁuly /tafaˁuly*). The correct shape of the *alef* at the end of a word is dictated by grammatical rules based on the Arabic word root system. Many people forgo the rules, and given the identical phonetic sounds, the *alef maksura* and the *alef* are often written one for the other. To illustrate, the correct spelling for the singular masculine past tense of the verb 'to cry' is بكى (*baky*) where it is often misspelled as بكا (*baka*). In contrast, the singular masculine past tense of the verb 'to forgive' is عفا (*ˁafa*) and it is misspelled as عفى (*ˁafy*).

Al-Ameri [14] analyzed the frequency of Arabic spelling mistakes in a sample of Teacher Education Institutes attendees. Table II summarizes the most common soft-spelling errors encountered in his study. We notice that the majority of reported errors relate to typographical errors due to phonetic mistranscription (*i.e.*, mistaking a diacritic for a letter or vice versa, phonetically close letters); errors due to *hamza*; and finally confusing the end *alef* or *teh* for other orthographic forms or letters.

It is worth noting that the lack of a common and sufficiently large enough benchmark dataset for Arabic spelling errors has hindered the continuous progress in the field of Arabic spelling errors detection and correction. This holds for both classical and modern standard Arabic sets. Many of the existing sets suffer from the lack of proper and comprehensive annotation, variety, and consistency. For example, within the same set, one can find texts that are fully diacritized while others have minimal or lack diacritization altogether. The lack of annotation makes the tasks of mapping the dataset texts to grammatically sound and misspellings-free texts difficult. Many times, researchers spend cumbersome manual effort in preparing sets for study. Other times, it is easier to introduce artificial mistakes from known correct texts. While this could work most of the time; the injection of artificial errors might not necessarily correspond to the real errors made by language speakers and learners alike.

TABLE II. THE FREQUENCY OF SOFT SPELLING ERRORS ANALYZED IN AL-AMERI STUDY [14]

No.	Spelling Error Type	%	Wrong Spelling	Correct Spelling
1	Writing *middle hamza* on an *alef*	73%	قرأة (*qirạ̉ẗ*)	قراءة (*qirạʾẗ*)
2	Writing *alef maksura* instead of *alef*	71%	عفى (*ᶜafy̱*)	عفا (*ᶜafạ*)
3	Writing *alef* instead of *alef maksura*	70%	بكا (*bakạ*)	بكى (*baky̱*)
4	Omitting *alef* following a *waw* at the verb end	67%	لا تنظرو (*lạ tanẓurw*)	لا تنظروا (*lạ tanẓurwạ*)
5	Writing *teh* instead of *teh marbuta*	67%	كثرت (*katẖrat*)	كثرة (*katẖarẗ*)
7	Writing *middle hamza* on *waw*	64%	المروؤة (*ạl-murww̉aẗ*)	المروءة (*ạl-murwʾaẗ*)
8	Writing *hamza* at the end of the word on *alef*	51%	أشيأ (*ạ̉sẖyạ̉*)	أشياء (*ạ̉sẖyạʾ*)
9	Writing a standalone *hamza* at the word end	47%	خطء (*kẖaṭʾ*)	خطأ (*kẖaṭạ̉*)
10	Writing *hamza* at the end of the word on *yeh*	47%	شئ (*sẖaỷ*)	شيء (*sẖayʾ*)
11	Dropping *lam* before a "solar letter"	38%	اسماء (*ạsmạʾ*)	السّماء (*ạl-sãmạʾ*)
12	Writing *teh marbuta* instead of *teh*	37%	ساعاة (*sạᶜạẗ*)	ساعات (*sạᶜạt*)
13	Writing *middle hamza* on *yeh*	30%	يتفائل (*yatafạʾal*)	يتفاءل (*yatafạỷl*)
14	Writing *hamzaẗ qaṭᶜ* instead of *hamzaẗ waṣl*	28%	إبن (*ạbn*)	ابن (*ạbn*)
15	Inserting *alef* after *waw* at the end of a word	25%	يجثوا (*yajtẖwạ*)	يجثو (*yajtẖw*)
16	Confusing *teh marbuta* and *heh* at the word end	*	مكتبه(*makatabh*)	مكتبة (*maktabaẗ*)

* Common mistake yet not reported in this study

B. Approach and Contribution

In this paper, we propose using a tuned bidirectional long short-term memory (BiLSTM) recurrent neural network to detect and correct spelling mistakes written in either classical or modern standard Arabic. We target a subset of the most commonly encountered spelling mistakes in the Arabic language [14]. Mainly, errors in the soft misspelling category pertaining to *ạl-hamzạt* (الهمزات), the different shapes of *alef*, and the common errors in shaping *teh* at the end of the word. We propose tackling the problem at the character-level and we propose letter conversion scheme that allows one-to-one training of the input sequences against the target sequences.

We propose and evaluate two approaches to train models to correct these mistakes. In the *transformed input* approach, the network is trained to predict correct spelling from transformed unified input. Whereas the *stochastic error injection* trains the network to correct randomly injected spelling mistakes. We recommend best configuration and approach based on evaluation on two training datasets and a sample of real mistakes.

We organize the rest of the paper as follows: In Section II, we survey the state-of-the-art techniques in detecting and correcting spelling mistakes in European, Indo-Iranian and the Arabic languages. In Section III, we review the basic concepts of recurrent neural networks, long short-term memory, and the sequence transcription problem. Section IV details the experimental setup used in this work, the datasets, the used training approaches, and the performance evaluation metrics. Section V presents and discusses the results of our experiments. We provide a summary and conclude the paper in Section VI.

II. RELATED WORK

The natural language processing community has an ongoing interest in spell checking and correction. Traditionally, post-OCR spell check and correction has been a driving force behind research and application. Yet, in the past decade, the proliferation of social media and the high reliance on instant messaging demand more efficient and accurate spell checkers and on-the-fly accurate correction. The accuracy level of spelling and grammatical mistakes correction varies in maturity between different languages.

It is worth noting that research experiments have been usually evaluated based on artificially-created or proprietary corpora and less so on a corpus of authentic misspellings [15]. Moreover, generic spelling checkers (GSCs), such as the ones packaged in popular text editors like Microsoft Word, are designed for native writers and as such fail to detect and correct mistakes commonly introduced by second language learners [16]. In an ever-interconnected world where hundreds of millions of people are bilingual and multilingual, designing accurate spell-checkers is more challenging. In this section, we review some of the most recent works in spelling correction for three world language groups.

A. Spelling Correction for European Languages

Whereas spelling correction for the English language is well-established compared to other languages, most of this research was evaluated on proprietary corpora of native English texts or well-formed texts with artificially injected errors. Yet, spelling correction of non-native speakers is far more challenging as they feature multi-character edits compared to a single-character edit produced by native speakers [17]. To this end, the authors in [15] developed a minimally supervised model based on contextual and non-contextual features. These features include orthographic similarity, phonetic similarity, word frequency, n-grams and word embeddings among others. Notably, they present and use a corpus of real-world learner essays from the TOEFL exam, and further test their model on out-of-domain medical notes. They report an accuracy level of 88.12% on the TOEFL set, and 87.63% on the medical set. The authors in [18] propose a nested RNN model for English word spelling error correction. They generate pseudo data based on phonetic similarity to train the network. Their proposed system has a precision of 71.77%, a recall rate of 61.26%, and an $F_{0.05}$ score of 69.39%.

D'hondt *et al.* [19] employ many-to-many character sequence learning network using LSTM for French text correction. Their model stacks two LSTM layers: an encoder layer that reads the sequence of characters, and a decoder layer that generates the output. They further use a drop-out layer to enhance performance. They train and evaluate their system on a dataset of OCRed French medical notes using two models that introduce noise and confusion into the text. They report an accuracy rate of 73% for the former and 71% for the latter model. The same authors extended their work by using BiLSTM [20], and used various corpora based on structured English, structured French, and free-text French with artificially corrupted strings. They report accuracy rates no less than 85% for the structured English and French, and 60% for the free-text French. They show that for an original text with a character error rate (CER) of 34.3%, the BiLSTM system reduces the CER to 7.1%.

B. Spelling Correction for Indo-Iranian and Asian Languages

Dastgheib *et al.* [21] introduced Perspell; a semantic-based spelling correction system for the Persian language which is based on an n-gram model. Perspell handles both real-word and non-word errors. The authors' experiments show that for non-word errors, the precision, recall, and F_1 score are 87.7%, 88.9%, and 88.3%, respectively; while for real-word errors, the authors report a precision of 92.4%, a recall rate of 93%, and an F_1 score of 92.6%.

More recently, Yazdani *et al.* [22] use dictionary-based methods to detect word misspells. They rely on a generic Persian dictionary and a specialized medical dictionary as their system is oriented towards health care applications, specifically ultrasound reports. They employ an n-gram model to dictate suggestions based on orthographic and edit distances. They test their system on actual ultrasound free-text reports and achieve a detection performance of up to 90.29% and a correction accuracy of 88.56%.

Salavati *et al.* [23] introduce Rênûs, a spell checker for the Sorani dialect of the Kurdish language. The error detection phase in Rênûs is based on an n-gram frequency model, and the error correction phase is based on the edit distance, as a measure of similarity and frequency. The authors carried out experiments to investigate error correction once with the use of lexicon and another without. They report a correction accuracy of 96.4% for the former and 87% for the latter case.

The work in [24] introduces the SCMIL system which stands for sequence-to-sequence text correction model for Indic languages. SCMIL uses an attention model with a bidirectional RNN encoder and attention decoder. The decoder is trained end-to-end and it has a character-based representation on both encoder and decoder sides. They have synthesized a dataset from the Hindi and Telugu languages with data lists comprised of a maximum of five words. They subsequently introduced errors which include insertion, deletion, substitution, and word fusion. The authors show that SCMIL has an accuracy rate of 85.4% for the Hindi language and 89.3% for the Telugu language.

Zhang *et al.* [25] proposed a system for Chinese spelling error detection which consists of a network for error detection and a network for error correction based on BERT. The two networks are connected to each other through a technique they called soft-masking. For a training set of five million examples, the authors' error detection system achieved an accuracy of 80.8%, a precision of 65.5%, a recall of 64% and an F_1 score of 64.8%. The error correction system; however, achieved an accuracy of 77.6%, a precision of 55.8%, a recall of 54.5% and an F_1 score of 55.2%.

C. Spelling Correction for the Arabic Language

Most recent works for Arabic spelling detection and correction still use traditional techniques in the field of natural language processing (NLP). For example, the authors in [26] introduced a spell checker which targets both lexical and semantic spelling mistakes. He uses a sequential combination of approaches including lexicon-based, rule-based, and statistical-based methods. He achieves an F_1 score of 67%.

Al-Shneifi *et al.* [27] developed a cascade system called Arib that detects and corrects a range of spelling errors. Errors that are discovered by Arib include: edit, add, split, merge, punctuation, phonological, and other observed common mistakes. They employed two core models: a probabilistic model based on Bayes probability theory and a Levenshtein distance-based model. They further add three extra models; two of which are based on 3rd party error detection tools: MADAMIRA and Ghaltawi, and the third additional module is a rule-based correcter derived from analyzing samples of the QALB database. Overall, Arib has an F_1 score of 57.8%, and precision and recall rates of 66.6% and 51.1%, respectively.

Mubarak and Darwish [28] also used a cascaded approach for word-level errors, followed by punctuation correction. For word-level correction, the authors used a statistical character-level transformation model and a language model to handle letter insertions, deletions, and substitutions and word merges. The author subsequently use a case-specific system aided by a language model to handle specific error types such as dialectal word substitutions and word splits. For punctuation recovery, the authors employ a simple statistical word-based system and a conditional random fields sequence labeler (CRF). For different experiments, the authors were able to achieve a precision rate up to 71.7%, a recall rate up to 60.32%, and an F-measure up to 63.43%.

Bouamor *et al.* [29] introduced another hybrid system that is based on a morphology-based corrector; rule-based linguistic techniques, language modeling, statistical machine translation (SMT), as well as an error-tolerant finite-state automata method. They target common error types which include split, delete, edit, merge, move and add errors based on the 2014 QALB set. They report an F_1 score of 68.4%.

Noaman *et al.* [30] developed a hybrid system based on the concept of confusion matrix and the noisy channel spelling correction model. They automatically detect and correct Arabic spelling errors of the edit and split types based on the QALB dataset. They report a word error correction rate up to 89.7%.

Zahui *et al.* [31] introduced Al- Mossahih tool that detects and corrects one-letter typographical and phonetic transcription errors. Their detection phase is dictionary-based where a collection of around two million words are sorted in alphabetical order. The correction module encompasses four techniques: one is based on a correspondence table between pairs of commonly confused characters, the second is permutation-based where all possible words from the word letters are generated, the third is a neighborhood module which considers letters whose keys are nearby on the keyboard, and finally a language model that deals with word locations within a sentence. Al-Mossasih tool has a word error detection rate of 74.75% and a correction rate of 80.2%.

Semantic errors have been addressed by few recent works. A major approach is based on confusion matrices, which despite being powerful, they limit the number of errors that can be detected and corrected. Al-Jefri and Mahmoud [32] compiled a corpus of 7.4 million words from the set of words most confused by non-native Arabic speakers and from the set of mis-recognized words by Arabic OCR systems. The authors compiled these words into 28 confusion sets with assigned probabilities. Errors detection only targets words listed in the confusion matrices and error correction is based on picking the word with highest probability using the computed n-gram model. They report an average accuracy of 95.4%.

For non-confusion set-based approaches, Zribi and Ahmed [33] detected semantic errors through the use of four combined statistical and linguistic methods. They have introduced semantic errors on a set of sentences collected from economic articles from the Egyptian Al-Ahram newspaper. The semantic errors were all a single edit away from the correct word. The reported detection performance was 90% and 83% for precision and recall, respectively.

Rokaya [34] introduced a small variation into the previous method by using the power link method instead of traditional frequency to detect and correct semantic errors coupled with confusion sets as a hybrid approach. They only report results for the detection stage where their system achieves 94.35% and 85.57% for precision and recall, respectively.

More recently, Watson *et al.* [35] utilized sequence-to-sequence models and character and word embeddings for Arabic Text Normalization. Azmi *et al.* [36] combine the language model with machine learning in the detection stage. For the correction step, they only use a language model. They have used word n-grams as features which are subsequently fed into a support vector machine classifier (SVM) to detect and mark words with semantic errors. For the correction step, they generate candidate words which are one-edit distance away from the erroneous word. The candidates are ranked and sorted based on the n-gram language model and then they select the best suggestion accordingly. Their system has an F_1 score of 90.7%, and an precision and recall rates of 83.5% and 99.2%, respectively.

Alkhatib *et al.* [37] recently used an LSTM model to detect and correct spelling and grammatical mistakes at the *word-level*. Their model uses word-embeddings and a polynomial classifier. They report an F_1 score of 93.89%, and for morpho-syntactic mistakes pertaining to word form, noun number, verb form, and verb tense, they report a precision of 95.6% and a recall rate of 94.88%. Solyman *et al.* [38] employed CNNs for the automatic correction of Arabic grammar. After fine-tuning their different developed and tested models, the authors achieved a precision of 80.23%, a recall rate of 63.59%, and an F_1 score of 70.91. Kuznetsov and Urdiales [39] proposed a method of performing spelling correction on short input strings, such as search queries or individual words using denoising auto-encoder transformer model to recover the original query. They used datasets for four langauges and achieved an accuracy of 83.33% (Arabic), 91.83% (Russian), 93.97% (Greek), and 94.48% (Setswana).

III. Machine Learning and Sequence Transcription

A general definition of sequence transcription is the process of transforming an input sequence into a corresponding output sequence. Within the context of machine learning spelling detection and correction, the input sequence is the set of letters forming the text that may have spelling errors. The corresponding output sequence is the text on which the machine learning algorithm attempted corrections. Sequence transcription is quite common in similar problems in the fields of language translation, voice recognition and diacritizing Arabic texts [12]. In all these applications, we need to infer relationships and provide outputs depending on past input data. Therefore, the need to preserve correlations between data points in the sequence is necessary.

Recurrent neural networks (RNN) provide the capability to learn from data sequences and consequently infer relevant output data. In this section, and for the sake of completeness, we briefly review RNN and a special RNN network cell called long short-term memory (LSTM) that we adopted in this work. We will further provide details of how we handle and process the input sequence prior to its use in the LSTM network.

A. Basic Recurrent Neural Networks

In general, RNN maintain hidden states that are functions of previous input. This enables such networks to infer outputs from past sequences making them quite suitable for applications where we require sequence transcription. A standard RNN cell can be described by two equations that relate the input sequence $x_t \in (x_1, x_2, \ldots x_T)$ to the hidden states $h_t \in (h_1, h_2, \ldots h_T)$ and output sequence $y_t \in (y_1, y_2, \ldots y_T)$. Equations 1 describe the standard RNN model:

$$h_t = \sigma(W_h x_t + U_h h_{t-1} + b_h) \tag{1a}$$

$$y_t = W_y h_t + b_y \tag{1b}$$

where *W*, *U*, and *b* denote the weight and bias matrices. We can clearly observe that the current hidden state h_t depends on both the current input x_t and the previous hidden state h_{t-1}. The model uses a sigmoid function σ to limit the output within a prefixed range. Hyberbolic tangents *tanh*, for example, limit the output to between -1 and 1, whilst logistic sigmoid limits the output to between 0 and 1. Both functions are often used in RNN among others. In this paper, we strictly use the σ symbol to denote a logistic sigmoid whilst we express the hyperbolic tangent as *tanh*. As stated earlier, the hidden state equips the network model with the ability to learn from input sequences, remember past data and infer outputs.

However, if the input sequence is long, and the output data to be inferred depends on a much older (past) input data sequence (*i.e.*, the gap or the distance in sequence between the inferred output and relevant input data is large), RNN in its standard form might not be able to deliver an accurate desired output sequence. To this end, researchers developed the refined RNN cell which they called long short-term memory

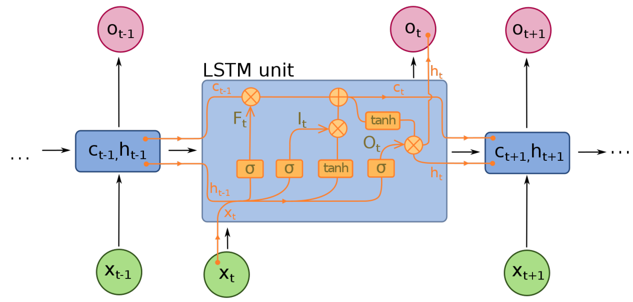

Figure 1. Basic Structure of an LSTM Cell [41]

(LSTM) [40]. LSTM can handle much longer sequences as well as avoid an inherent problem in standard RNN: the vanishing gradient problem. The vanishing gradient problem halts the update of network weights when the gradient is too small; thus stopping the learning stage quite early resulting in a poor network model.

B. Long Short-Term Memory Cells

The internal architecture of an LSTM cell vastly improves on the basic RNN cell by incorporating a *set* of gates which govern the operation of each individual cell. These gates equip the cell with the capacity to work with short or long-term contexts. LSTM cells are relatively insensitive to long gaps or large distance between the output to be inferred and relevant old data in the sequence. Figure 1 illustrates the internal architecture of an LSTM cell. An LSTM cell has an *input* gate **I**, a *forget* gate **F**, an *output* gate **O**, and a *cell activation* unit **C**.

We provide the governing equations of these gates in Equations 2. For each gate type, we use the small letter notation to denote the output of the gate at time *t*:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1} + b_i) \tag{2a}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1} + b_f) \tag{2b}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \mathit{tanh}(W_c x_t + U_c h_{t-1} + b_c) \tag{2c}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t + b_o) \tag{2d}$$

$$h_t = o_t \circ \mathit{tanh}(c_t) \tag{2e}$$

where W, U, V, and b denote the weight and bias matrices and the initial values for $c_0 = h_0 = 0$. The $\circ$ operator denotes the Hadamard (element-wise) product. Similar to the basic RNN cell, an update to a short-term state h_t or long-term state c_t depends on the current input x_t, and previous short-term and long-term memory states h_{t-1} and c_{t-1}.

LSTM is likely to quickly over-fit the training data; thus rendering them less powerful in predicting correct outputs. Dropout is a computationally cheap way that reduces over-fitting. It further improves generalization error and model performance in all deep neural networks. Dropout simply implies probabilistically excluding nodes from activation and weight updates while training a network.

The bidirectional LSTM is a version of the LSTM architecture that exploits future contexts as well as past contexts. That is, in sequence transcription problems, it could be beneficial to have access to subsequent (future) sequences to infer correct current outputs. In BiLSTM, we couple the conventional unidirectional LSTM network that works with past sequences with another unidirectional network that works with future sequences. We train the former in a forwards fashion whilst the latter in a backwards fashion. The final output is a concatenation of both the forward and backward LSTM networks.

TABLE III. LETTERS CONVERTED TO INTERMEDIATE CODES

Letter(s)	**Intermediate Code**
اء	J
ـوا	A
ـو	O
ـت	T
ـه	H

C. Sequence Processing and Data Encoding

We can classify RNN based on the relationship between the input and output sequence lengths into four types:

- *One-to-one* networks where the length of the input sequence matches the length of the output sequence.
- *Many-to-one* networks where input sequences are transcribed into one final output, for example classification problems.
- *One-to-many* networks where one input vector is used to produce an output sequence.
- *General many-to-many* networks where the number of items in the input sequence differs than that in the output sequence. We use encoder-decoder architectures to tackle these transcription problems by using an intermediary fixed-length vector. The encoder maps a variable-length source sequence to the intermediary fixed-length vector, and the decoder maps the vector representation to a variable- length target sequence.

We can model the Arabic spelling correction problem as a one-to-one problem. Intuitively, this should be straightforward as most soft Arabic spelling errors result from confusing shapes of الهمزات (al-hamzạt) or the similar sounding characters at the end of the word. Simple one-character replacement fixes the misspelled word. However, some of these soft mistakes fall under the addition/omission type. For example, the spelling error in cases 4 and 15 in Table II add or omit the *alef* at the end of the word after a *waw*. Similarly, case one corrects writing *middle hamza* on an *alef* (one letter) by writing it properly following an *alef* (two letters). These particular cases result in differing-length sequences. Ordinarily, the encoder/decoder RNN architecture handles this well, yet with extra cost and overhead.

To mitigate and simplify our approach, we propose a simple yet effective technique that maintains one-to-one sequencing by processing the input sequence stream prior to applying it to the neural network, and then post-processing the output sequence to restore readable Arabic form. We convert some letters and two-letter combinations to intermediate arbitrary codes of English letters, as specified in Table III. The conversion involves letters that are prone to the spelling mistakes under study: the combination *alef-hamza* (اء) and the letters at word ends (ـوا, ـو, ـت and ـه). We emphasize that this conversion is positional in nature; that is; we convert the *waw-alef* combination (ـوا) to an 'A' only when it appears at word endings where the associated error frequently occurs. Should this combination appear in the middle, no conversion is performed. However, we convert the letter combination *alef-hamza* to 'J' wherever it occurs in the word. This combination is susceptible to soft mistakes both in the middle and at the end of the word.

In our machine training approach, we use this converted sequence as the target sequence and a copy of it as the input sequence after injecting some artificial spelling mistakes (refer to Section IV-B). These sequences are stored using the Unicode UTF-8 encoding. However, when presented to the neural network, they are put in 3D $(B \times T \times C)$ tensors (dense matrices). Each tensor holds a batch of B sequences of a maximum length $T = 400$ characters. Note that we wrap sequences longer than 400 characters to improve training performance. The third dimension encodes each of the C distinct characters using one-hot encoding.

IV. Experimental Setup

In this work, we use an experimental setup similar to the one used in our past research [42]. We list the specifications of the experimental platform in Table IV.

A. Machine Learning Model Configurations

We develop our machine learning models using Python deep learning libraries ensuring we use the latest versions of the algorithms. Specifically, we use Keras with TensorFlow at the backend. Given that we tackle the problem of Arabic spelling correction at the character level, we adopt BiLSTM RNN. These networks can handle longer sequences which can be beneficial for correcting misspelled words based on past and future context. We develop and compare three models. The baseline model has only two BiLSTM layers. We add an input masking layer before the two hidden BiLSTM layers, and connect their output to a fully-connected (dense) output layer, as shown in Fig. 2.

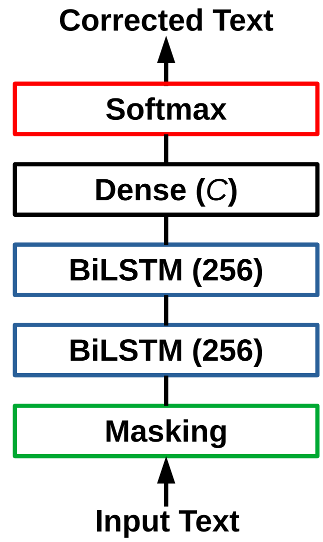

Figure 2. The Base 2-Layer Network Model

The second model maintains the same settings of the previous one but further employs the *dropout* method to reduce or avoid the over-fitting problem. The third and final model has four BiLSTM hidden layers instead of two and also uses *dropout*. We use the same configuration settings for the three models. Each bidirectional layer has 256 cells. We use the *softmax* as the activation function of the output layer and the *RMSprop* optimizer in training. We use categorical cross entropy as the loss function, and a batch size of 64 sequences with a sequence length of 400 while wrapping longer sequences similar to our work in [12]. To combine the forward and backward layers in our BiLSTM layers, we use *concatenation*. For the models that use dropout, we use the base settings of dropout = 0.1 and recurrent_dropout = 0.3. We set the training time to a maximum of 50 epochs with early stopping and 5-epoch patience. We show the skeleton code of the 2-Layer model with dropout in Listing 1.

B. Data Sets

We use two widely used datasets for Arabic NLP [43], [44], [45], [46], [47], [48], [49] to train, validate, and test our proposed BiLSTM models. The first is a processed subset of the Tashkeela corpus as extracted in [50]. The second is from the Linguistic Data Consortium's (LDC) Arabic Treebank (LDC2010T08), specifically speaking: the Arabic Treebank Part 3 (ATB3) v3.2 [51]. We will now-forth refer to this dataset as ATB3 in this paper. The major difference between the Tashkeela and ATB3 datasets is the form of the Arabic text they consist of. Tashkeela mainly contains 55K sequences from classical Arabic texts. On the other hand, ATB3 contains samples of modern standard Arabic of 599 distinct news-wire stories from the Lebanese publication An-Nahar. We show in Table V the characteristics of the two datasets in terms of sequence count, word count, character count, average words per sequence, average letters per word, and fraction of sequences shorter than or equal to 400 characters.

In Table VI, we present the Arabic letters and their variations that account for the majority of the soft spelling errors that we presented in detail in Section I and cited in examples in Table II. We break them down in order of their absolute frequency within the dataset texts relative to the character count. We also break them down in terms of their relative appearance to each other within the same texts. This table demonstrates that about one fifth of the characters are involved in the common soft spelling mistakes under study and that the relative frequencies of these letters are highly skewed ranging from 0.13% to 58.23%.

We split the ATB3 dataset as proposed by Zitouni *et*

TABLE IV. EXPERIMENTAL PLATFORM SPECIFICATIONS

Aspect	Specification
CPU	Intel Core i7-9700KF @ 3.6 GHz, 8 cores, 12 MB cache
GPU	Nvidia GeForce RTX 2080 @ 2.1 GHz, 2944 CUDA cores, 8 GB memory
Memory	32 GB DDR4-SDRAM @ 2666 MHz
OS	Ubuntu 20.04 LTS, 64-bit
Libraries	Python 3.8.2, TensorFlow 2.2.0, Keras 2.3.0-tf

Listing 1: A 2-Layer BiLSTM Model with Dropout

```
model = Sequential()
model.add(Masking(mask_value = 0,input_shape=(seq_len, num_inp_tokens)))
model.add(Bidirectional(LSTM(256, return_sequences=True,dropout=0.1, recurrent_dropout=0.3),
    merge_mode='concat'))
model.add(Bidirectional(LSTM(256, return_sequences=True,dropout=0.1, recurrent_dropout=0.3),
    merge_mode='concat'))
model.add(TimeDistributed(Dense(num_tar_tokens, activation='softmax')))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])
```

al. [52] such that we use the first 509 news-wire stories, in *chronological* order, to train the model and use the remaining 90 stories to validate and test the model. This accounts for 22,170 sequences for training and 3,857 sequences for validation. Similarly, we split the Tashkeela dataset into 50K lines for training, 5,000 lines for testing. In our previous work [12], we analyzed the best maximum sequence length to use based on the same datasets. We found that a maximum sequence of 400 characters provides the best speed versus accuracy trade off. We consequently adopt this sequence length in this work as well.

In addition to the above datasets, we test the proposed solutions using samples of real soft spelling mistakes (Test200). These samples were collected in a previous work [8] and are summarized in Table VII. They have a challenging collection of soft spelling mistakes with an average of 6.5 mistakes per sequence.

C. Training Approaches

We have experimented with the following two approaches to train BiLSTM networks to correct the soft spelling mistakes.

1) Transformed input: This approach trains the BiLSTM network to predict the correct form of the letters under study given unified transformed input. Once the sequences are converted to the intermediary form, as described in Section III-C, we transform the letters that are often confused with each other into one final form. We show the used mappings of the letters affected by this transformation in Table VIII. All *hamza* forms are transformed to plain *hamza* (ء), the *heh* and *teh* forms at word ends are transformed to *teh marbuta* (ة), *waw-alef* at word ends are transformed to *waw* (و), and *alef* at word ends are transformed to *alef maksura* (ى).
2) Stochastic error injection: This approach trains the network to correct artificial errors randomly injected in the input sequences. We inject errors in the input sequence by replacing the target letters pertaining to the soft Arabic spelling mistakes. With an *error injection rate* p, a letter belonging to the four groups shown in Table VIII is randomly replaced by one of the letters in its group. For example, we replace *alef maksura* with an *alef* and vice versa (cases 2 and 3 in Table II). We have investigated using multiple error injection rates p as described in Section V. For example, 10% of the letters under investigation are replaced with $p = 10\%$.

TABLE V. TASHKEELA AND ATB3 DATASETS CHARACTERISTICS

Criterion	Tashkeela	ATB3
Sequence count	55K	26K
Word count	2,312K	305K
Character count	12,464K	1,660K
Words per sequence	42.1	11.3
Letters per word	4.0	4.6
Sequences $\leq$ 400 chars.	84.1%	99.9%

D. Evaluation Metrics

We measure and evaluate the performance of BiLSTM networks using their computational time and multiple performance metrics, namely: *accuracy*, *precision*, *recall*, F_1 *score*, character error rate (*CER*), and word error rate (*WER*). These metrics are quite common [36], [50], [53], [54], [37] in evaluating the performance of solutions pertaining to error detection and correction, voice recognition, and similar sequence-based problems.

The first four measures are readily computed and understood through means of a simple confusion matrix that we will explain within the context of our work. At the character level, any character in any sequence can either be correctly spelled or not. When comparing the predicted outcome of our model with the actual (target) sequence, we can thus have four cases that we show in Fig. 3. For a certain character c, the counts of these four cases are:

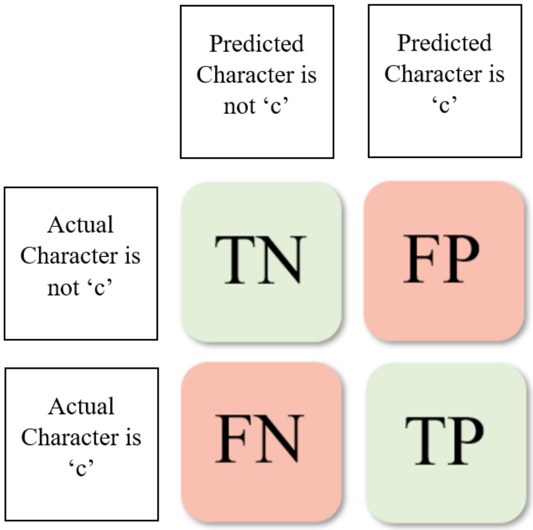

Figure 3. Confusion Matrix of Correct and Erroneous Characters between Predicted and Actual Sequences

- True positive (*TP*): The number of times character c is correctly predicted as c.
- True negative (*TN*): The number of times characters other than c are predicted correctly.
- False positive (*FP*): The number of times characters other than c are incorrectly predicted as c.
- False negative (*FN*): The number of times character c is incorrectly predicted as another character.

It is evident that we need our model to maximize the correct cases *TP* and *TN* (cases shown in green). We desire our model also to keep the incorrectly predicted characters *FP* and *FN* to a minimum (cases shown in red). This readily translates to the definition of the *accuracy* metric that we present in Equation 3:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

The *precision* metric is the ratio of correct predictions of character c to total number of characters predicted as c and is given by Equation 4:

$$Precision\ (P) = \frac{TP}{TP + FP} \tag{4}$$

The *recall* metric is the ratio of correct predictions of character c to the total count of actual c characters. Equation 5 mathematically defines the recall as:

$$Recall\ (R) = \frac{TP}{TP + FN} \tag{5}$$

The F_1 score combines the *precision* and *recall* metrics into one score by applying the weighted harmonic mean on both giving equal weights to each. We get the F_1 score using Equation 6:

$$F_1 = 2 \times \frac{P \times R}{P + R} \tag{6}$$

We additionally use a new metric (*FP/Changes*) to assess the prediction error rate with respect to the number of replacements in the input sequence. We divide the number of false positive cases of a letter over the number of times this letter was changed in the input sequence. The accuracy is reported in this work for the entire set of characters. Whereas, precision, recall, F_1, and *FP/Changes* scores are reported as weighted averages of the target letters shown in Table VI.

We also evaluate our models using the character error rate (CER) and word error rate (WER). CER is the percentage of letters that are misspelled whereas WER is the percentage of misspelled words. A word is considered misspelled if it has at least one incorrectly spelled letter.

Table VI. The Subset of Target Arabic Letters (and their Variations) under Study and their Absolute and Relative Frequencies within the Datasets

	Frequency		Relative Frequency	
Letter(s)	**Tashkeela**	**ATB3**	**Tashkeela**	**ATB3**
ا	8.38%	11.55%	46.96%	58.23%
ـه	2.55%	0.58%	14.31%	2.93%
أ	2.41%	1.70%	13.52%	8.57%
ة	1.22%	2.49%	6.81%	12.55%
إ	0.93%	0.79%	5.21%	3.97%
ـو	0.68%	0.24%	3.79%	1.21%
ى	0.67%	0.72%	3.77%	3.65%
ـت	0.40%	0.81%	2.24%	4.11%
اء	0.18%	0.24%	1.03%	1.24%
ئ	0.18%	0.40%	1.03%	2.04%
آ	0.07%	0.08%	0.40%	0.41%
ء	0.07%	0.03%	0.37%	0.13%
ؤ	0.06%	0.14%	0.37%	0.69%
ـوا	0.04%	0.06%	0.34%	0.28%
Subtotal	17.85%	19.83%	100.00%	100.00%
Other chars.	82.15%	80.17%	0.00%	0.00%
Total	100.00%	100.00%	100.00%	100.00%

Table VII. Test200 Test Set of Real Soft Spelling Mistakes

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

Table VIII. Letters Changed in the “Transformed Input” Approach

Intermediate Forms	Mapped To
[ء, آ, أ, ؤ, إ, ئ, ا, and J (اء)]	ء
[ة, T (ـت), and H (ـه)]	ة
[O (ـو) and A (ـوا)]	O (ـو)
[ا and ى]	ى

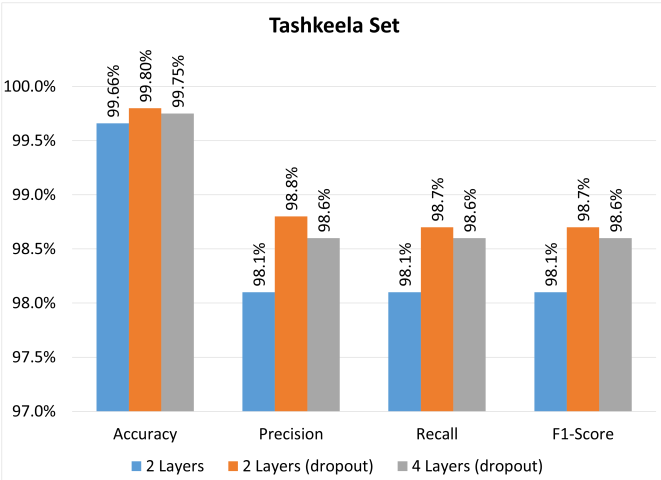

Figure 4. Performance of Three BiLSTM Networks using the Transformed Input Training on the "Tashkeela" Set.

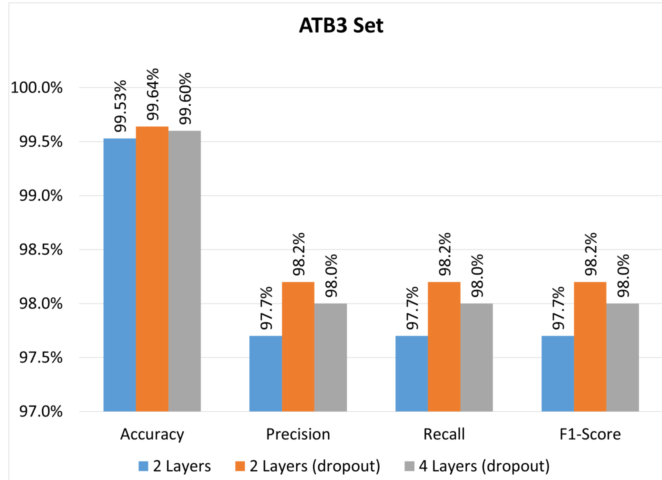

Figure 5. Performance of Three BiLSTM Networks using the Transformed Input Training on the "ATB3" set.

V. Results and Discussion

This section presents the results of evaluating alternative network topologies and the two training approaches proposed above. This section also presents and discusses the detailed performance evaluation results.

A. Network Topology

We started by evaluating three network topologies: the base network of two BiLSTM layers, another that adds the dropout technique to the base network, and a third that extends the second network to having four BiLSTM layers instead of two. These three alternatives are inspire by experience from previous work, *e.g.*, [42]. In this initial evaluation, we use the *transformed input* training approach. For each of these three networks, we use the Tashkeela and ATB3 sets to separately train two separate instances of the BiLSTM network; one trained for classical Arabic, and another for MSA Arabic. We repeat this procedure for the three networks using the same training sequences and compare their performance in terms of *accuracy*, *precision*, *recall*, and F_1 score on the test set.

In Fig. 4, we show these metrics for the networks trained with the Tashkeela set. Despite the results being marginally close, we concur that networks with the *dropout* technique employed perform generally better than the one without. We observe that there is no large difference between the performance of the 2-layer and 4-layer networks with dropout, with only a slight edge towards the 2-layer network. Fig. 5 similarly illustrates the same metrics but for the case when we evaluate the BiLSTM networks using the ATB3 set. We notice that our observations and conclusions for the Tashkeela set carry over to the ATB3 set. However, the accuracy with the ATB3 set is generally lower than the accuracy on the larger Tashkeela set.

We notice also that the accuracy is higher than the other three metrics because it is calculated for the entire character set while the other metrics are weighted averages of the target letters only. For the remaining experiments, and given the slight performance edge and lesser complexity of the 2-layer BiLSTM network with dropout used, we conduct and evaluate the remaining experiments and report their results based on this model only.

B. Training Approach

In this section, we present the performance of the adopted 2-layer network on the two proposed training approaches: *transformed input* and *stochastic error injection*. The error injection rate p correlates with the network's ability to correct errors. Therefore, we experimented with multiple rates, *e.g.*, 2.5%, 10%, and 40%, alongside the transformed input approach.

For this evaluation, we show the results also in terms of *accuracy*, *precision*, *recall*, and F_1 score. Recall that these metrics are found not only from the letters that we actually changed, but also include the whole letters in the subset under study. Therefore, we expect that the inclusion of all the letters that could possibly be changed and those actually changed dilutes the results. For example, for the case when the error injection rate is 2.5%, we already know that a high percentage of the remaining 97.5% letters are correct and match the target output.

Despite the expected result dilution, we have to present these results for in most cases the analysis and the interest is in the overall output character sequence and that it should be error free. Given these disproportional ratios between unchanged letters and error-injected letters, we expect better performance with lower error injection rates. We indeed observe these results in Figures 6 and 7. The best performance appears here for the network with the smallest error injection rate of $p = 2.5\%$.

It is difficult to make solid conclusions about the two training approaches from the previous data alone. So we compare the two training approaches using the *FP/Changes* ratio. This allows us to observe more insights that we could not deduce from the diluted results in Fig. 6 and 7. We present

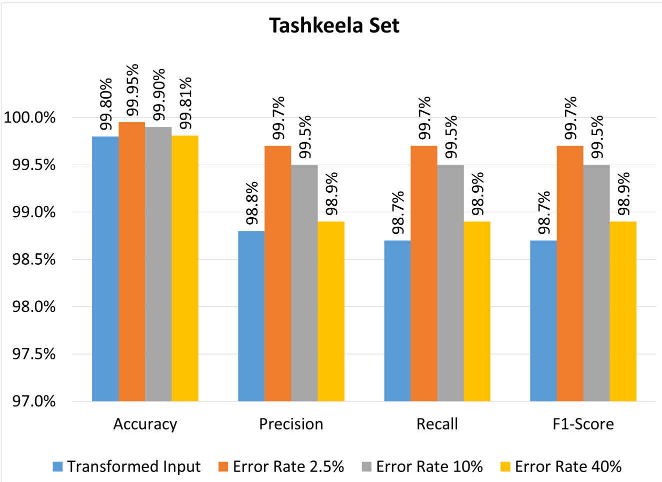

Figure 6. Performance of the Transformed Input and Stochastic Error Injection Training Approaches on the "Tashkeela" Set using the 2-Layer with Dropout Network. Three Error Injection Rates are Investigated.

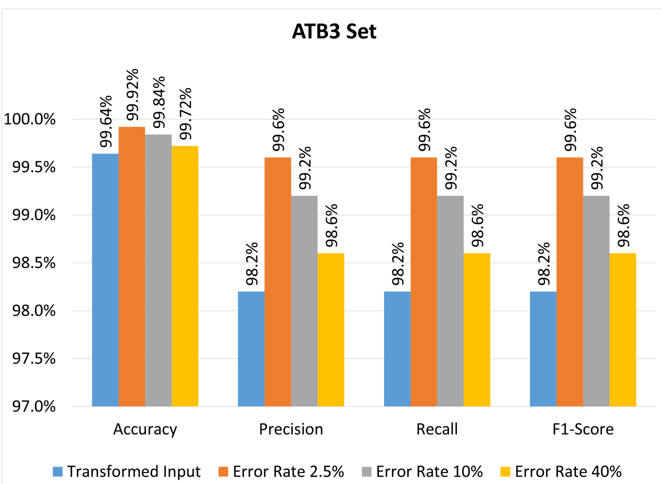

Figure 7. Performance of the Transformed Input and Stochastic Error Injection Training Approaches on the "ATB3" Set using the 2-Layer with Dropout Network. Three Error Injection Rates are Investigated.

this ratio in Fig. 8 for the Tashkeela and ATB3 sets. In this analysis, we easily see that increasing the error injection rate helps train the network to better detect and correct errors. For example for Tashkeela, we observe a decrease of the percentage of false positives from 14.4% to 3.6% as we increase the error injection rate from 2.5% to 40%. Despite the more errors introduced, the network was able to handle them well. In the case of transformed input, we even observe better performance. This is because transforming the most confused characters into one form is another way of introducing different errors into the network training.

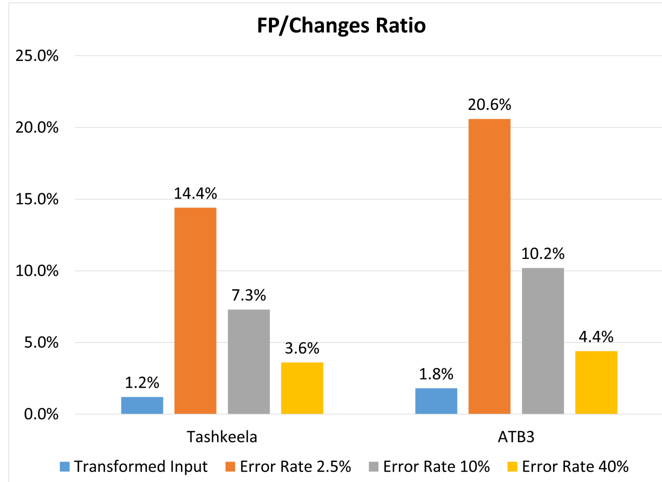

Figure 8. The *FP/Changes* Ratio of the Two Training Approaches on the Tashkeela and ATB3 Sets.

In Fig. 9 and 10, we show the character and word error rates for the proposed training approaches. These two metrics are global metrics similar to the accuracy. Therefore, they are generally low because most characters in the input are correct and only need to be passed to the model output as is. We present these metric as any application will handle entire sequences with possible spelling mistakes and attempt to provide an error-free version. Essentially, CER and WER are related so we expect a similar pattern for both. We expect this pattern to resemble that for the analysis presented in above, specifically, $1-$ *accuracy*. This is due to the ratio of already correct and unchanged characters that are in the input, predicted, and target sequences. Despite this, we observe that for both experiments on the Tashkeela and ATB3 datasets that in the worst case the CER did not exceed 0.36% while the WER did not exceed 1.88%. Referring to the analysis of the datasets that we show in Table V, we note that the average number of letters per word is 4.0 and 4.6 for the Tashkeela and ATB3 sets, respectively. The results we show for WER in Fig. 10 are approximately five times than those for CER in Fig. 9. This is in line with the letters per word statistic for these datasets under consideration.

C. Test200 Results

The above results are not conclusive about which training approach is best. Therefore, we used the BiLSTM models trained on the two training approaches to correct the mistakes in the Test200 set. We use the networks which we trained using the Tashkeela and ATB3 sets here. We report the CER of the predicted Test200 sequences on the eight training configurations shown in Fig. 11. Except for the 2.5% error injection rate, the two training sets provide results within 0.2% for each other. Yet, we note that despite that the reported CER for this external set is quite low, it is an order of magnitude higher than that reported for the test sequences of the Tashkeela and ATB3 datasets. This is because the BiLSTM models were trained using artificially injected errors that might not necessarily always correspond to errors committed by real-users. This

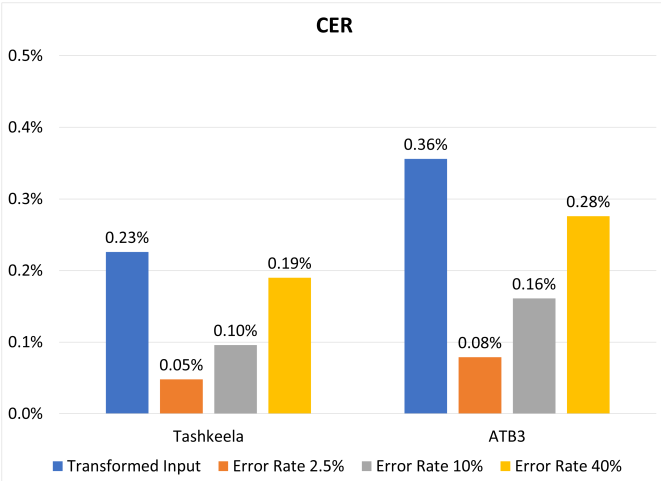

Figure 9. Character Error Rate of the Two Training Approaches on the Tashkeela and ATB3 Sets.

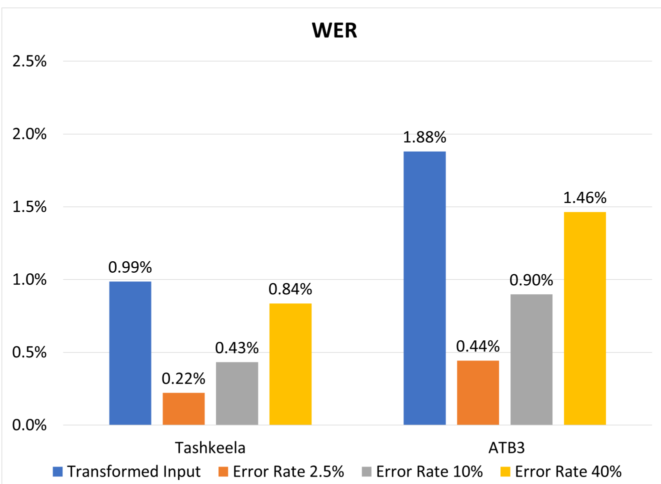

Figure 10. Word Error Rate of the Two Training Approaches on the Tashkeela and ATB3 Sets.

stresses the need for a large Arabic corpora collected from real users, annotated, and corrected by linguistic experts to enrich the Arabic NLP research domain.

Fig. 11 shows also that training on the larger Tashkeela set gives better results and the transformed input and error injection with 40% rate are better than lower rates. The best results with only 1.28% CER is for the case when training using Tashkeela set and 40% error injection rate. To test whether increasing the error injection rate beyond 40% would further decrease CER, we experimented with higher rates (50%, 70%, and 100%). Fig. 12 shows that the model with $p = 40\%$ performs best and yielded the least character error rate. Therefore, we recommend using error injection training approach with $p = 40\%$. We further analyze the results of this approach in the following subsection.

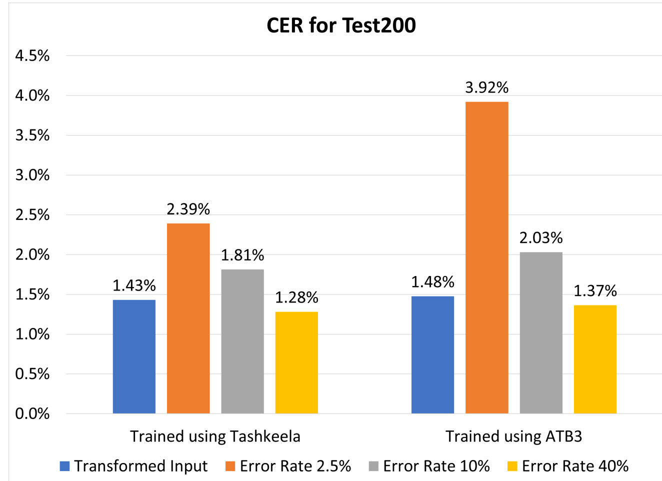

Figure 11. CER of the Two Training Approaches on the Test200 Set for Models Trained on the Tashkeela and ATB3 Sets.

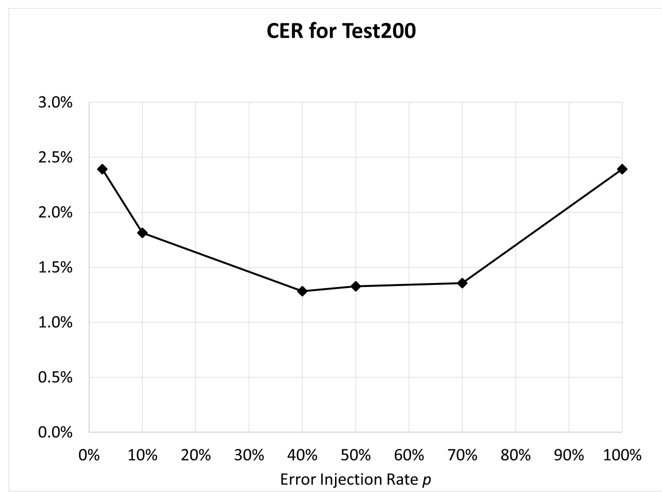

Figure 12. CER of Test200 Set for Models Trained with the Stochastic Error Injection Approach and the Tashkeela Set as a Function of the Error Injection Rate p.

D. Confusion Matrix Results

Fig. 13 shows the confusion matrix for the predicted against actual of the Tashkeela test set of the letters under investigation. We present here this matrix for the best trained model (error injection with $p = 40\%$) and Tashkeela set because it is the largest set we use. This matrix allows us to analyze the letters that our machine learning model mostly confuses with each other. While the majority of the letters are correctly classified and corrected (diagonal), most confusion occurs within the three terminal letters: *heh* (ـه), *teh* (ـت), and *teh marbuta* (ة), and *alef with hamza above* (أ) and *alef with hamza below* (إ).

Actual Letter \ Predicted Letter	ـوا	ـه	ـاء	ـو	ـت	ء	آ	أ	ؤ	إ	ئ	ا	ة	ى
ـوا	264	0	0	3	0	0	0	0	0	0	0	0	0	0
ـه	0	14, 699	0	0	66	0	0	0	0	0	0	0	102	0
ـاء	0	0	1, 083	0	0	0	0	3	0	0	2	7	0	0
ـو	9	0	0	3, 888	0	0	0	0	0	0	0	0	0	0
ـت	0	123	0	0	2, 242	0	0	0	0	0	0	0	30	0
ء	0	0	0	0	0	381	0	0	0	0	0	0	0	0
آ	0	0	0	0	0	0	359	25	0	2	0	2	0	0
أ	0	0	3	0	0	1	20	13, 660	19	79	6	31	0	0
ؤ	0	0	0	0	0	0	0	15	315	0	17	4	0	0
إ	0	0	0	0	0	0	6	75	0	5, 358	1	3	0	0
ئ	0	0	2	0	0	2	0	4	5	0	957	4	0	0
ا	0	0	2	0	0	1	6	63	1	3	3	48, 505	0	40
ة	0	151	0	0	46	0	0	0	0	0	0	0	6, 881	0
ى	0	0	0	0	0	0	0	0	0	0	0	107	0	3, 735

Figure 13. Confusion Matrix for the 40% Error Injection Training Approach on Tashkeela Test Set.

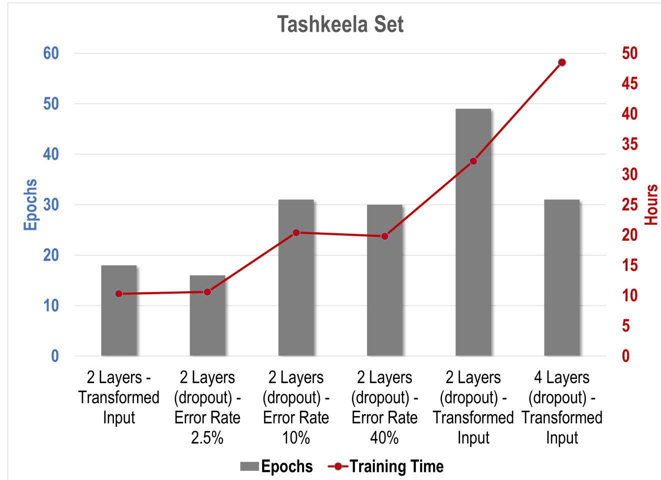

Figure 14. BiLSTM Timing Metrics under different Configurations for the Tashkeela Set.

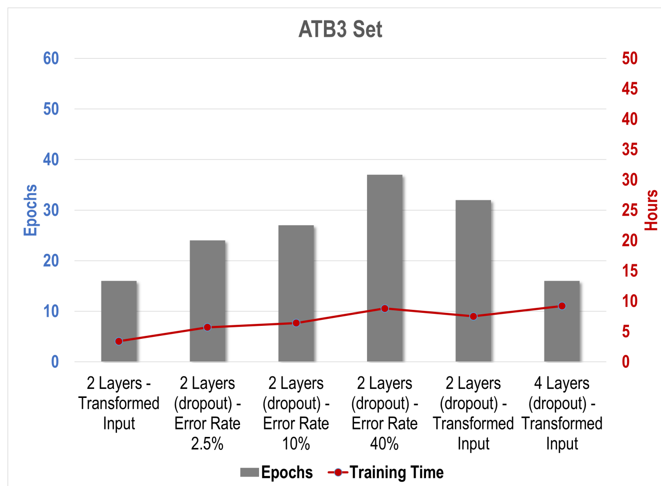

Figure 15. BiLSTM Timing Metrics under different Configurations for the ATB3 Set.

We list Examples 5, 12, and 16 of these common mistakes in Table II. For example, out of $(151 + 46 + 6,881 = 7,078)$ *teh marbuta* letters, 151 and 46 are wrongly predicted as *heh* and *teh*, respectively.

E. Model Timing

Finally, we report the timing metrics (training time and number of training epochs) of six selected configurations in Fig. 14 and 15 for Tashkeela and ATB3, respectively. In all cases, the training time for the Tashkeela set is between two to five times more than that for the ATB3. It took the longest to train the 4-layer network for both sets; 48.5 hours and 9.2 hours for Tashkeela and ATB3 using the transformed input approach, respectively. When training the network using variable error injection rate, we observe that the training time *generally* increases as this rate increases.

In these experiments, we set the maximum number of training epochs to 100. Yet, all models converged before half the set number of epochs. In contrast to the number of hours parameter, we observe that in some cases it took from $\frac{2}{3}$ to twice the number of epochs to train the same model for the two datasets.

VI. Conclusion

In this work, we addressed the problem of correcting common Arabic soft spelling errors. We developed variant configurations of bidirectional LSTM networks with either two or four hidden layers, while using or forgoing the dropout technique. We use sequences of Arabic texts that are either written in classical Arabic (Tashkeela set), or MSA Arabic (ATB3 set) to train and validate our models. The 2-layer network with dropout had the highest F_1 score of 98.7% on Tashkeela test set using the transformed input training approach.

We also experimented with a second training approach where we introduce stochastic errors and train the BiLSTM network to correct them. We deliberately varied the percentage of injected errors in the input sequence to assess the BiLSTM network performance and sensitivity given how well it can learn from lower or higher injected error rates. Networks trained with higher error rates (from 2.5%, 10%, through 40%) have worse accuracy, precision, recall, F_1 score, CER, and WER. However, they are more capable of correcting errors as reflected by their *FP/Changes* ratio. Noting that injecting more errors in the input stream beyond 40% does not improve the network performance in correcting soft spelling mistakes.

The transformed input training approach has better *FP/Changes* ratio than that of the stochastic error injection approach. However, the latter approach is better in correcting the soft spelling mistakes in the Test200 test set with error injection rate of $p = 40\%$. The best result on this test set is an CER of 1.28%. This lowest error rate is for the network trained on Tashkeela set and $p = 40\%$.

For future work, we consider expanding beyond the class of soft Arabic spelling errors. Further, a much larger annotated and pre-processed dataset will open doors to improving accuracy. We could possibly handle Arabic spelling correction and letter diacritization in one problem space.

References

[1] UNESCO, "World Arabic language day," Dec 2019. [Online]. Available: https://en.unesco.org/commemorations/worldArabiclanguageday

[2] Ethnologue, "Arabic language statistics," 2020. [Online]. Available: https://www.ethnologue.com/language/ara

[3] K. Darwish and W. Magdy, "Arabic information retrieval," *Foundations and Trends in Information Retrieval*, vol. 7, no. 4, pp. 239–342, 2014.

[4] J. Clement, "Internet: Most common languages online 2019," Jul 2019. [Online]. Available: https://www.statista.com/statistics/262946/share-of-the-most-common-languages-on-the-internet/

[5] T. Tinsley and K. Board, "Languages for the future," British Council, Report ISBN 978-0-86355-883-2, 2017.

[6] Glosbe, "Transliteration and Romanization utilities," 2020. [Online]. Available: https://glosbe.com/transliteration/Arabic-Latin

[7] K. C. Ryding, "Learning Arabic as a foreign language," 2020. [Online]. Available: https://Arabic.georgetown.edu/afl/

[8] G. Abandah, M. Khedher, W. Anati, A. Zghoul, S. Ababneh, and M. S. Hattab, "The Arabic language status in the Jordanian social networking and mobile phone communications," in *7th Int'l Conf. on Information Technology (ICIT 2015)*, 2015, pp. 449–456.

[9] "The school of language studies - language categories," 2017. [Online]. Available: https://2009-2017.state.gov/m/fsi/sls/orgoverview/languages/index.htm

[10] G. A. Abandah, A. Graves, B. Al-Shagoor, A. Arabiyat, F. Jamour, and M. Al-Taee, "Automatic diacritization of Arabic text using recurrent neural networks," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 18, no. 2, pp. 183–197, 2015.

[11] S. Alqudah, G. Abandah, and A. Arabiyat, "Investigating hybrid approaches for Arabic text diacritization with recurrent neural networks," in *2017 IEEE Jordan Conf. on Applied Electrical Engineering and Computing Technologies (AEECT)*, Oct 2017, pp. 1–6.

[12] G. Abandah and A. Abdel-Karim, "Accurate and fast recurrent neural network solution for the automatic diacritization of Arabic text," *Jordanian Journal of Computers and Information Technology*, vol. 6, no. 2, pp. 103–121, 2020.

[13] M. Attia, P. Pecina, Y. Samih, K. Shaalan, and J. Van Genabith, "Arabic spelling error detection and correction," *Natural Language Engineering*, vol. 22, no. 5, pp. 751–773, Sep. 2016.

[14] A.-M. H. Al-Ameri, "Common spelling mistakes among students of teacher education institutes," *The Islamic College University Journal*, vol. 1, no. 33, pp. 445–474, 2015.

[15] M. Flor, M. Fried, and A. Rozovskaya, "A benchmark corpus of English misspellings and a minimally-supervised model for spelling correction," in *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 76–86.

[16] M. BIazquez and C. Fan, "The efficacy of spell check packages specifically designed for second language learners of Spanish," *Pertanika Journal of Social Sciences & Humanities*, vol. 27, no. 2, 2019.

[17] M. Flor, Y. Futagi, M. Lopez, and M. Mulholland, "Patterns of misspellings in L2 and L1 English: a view from the ETS spelling corpus," *Bergen Language and Linguistics Studies*, vol. 6, 2015.

[18] H. Li, Y. Wang, X. Liu, Z. Sheng, and S. Wei, "Spelling error correction using a nested RNN model and pseudo training data," *arXiv:1811.00238 [cs]*, Nov. 2018.

[19] E. D'hondt, C. Grouin, and B. Grau, "Low-resource OCR error detection and correction in French clinical texts," in *Proceedings of the Seventh International Workshop on Health Text Mining and Information Analysis*. Auxtin, TX: Association for Computational Linguistics, 2016, pp. 61–68.

[20] E. D'hondt, C. Grouin, and B. Grau, "Generating a training corpus for OCR post-correction using encoder-decoder model," in *Proceedings of the Eighth International Joint Conf. on Natural Language Processing*, 2017, pp. 1006–1014.

[21] M. B. Dastgheib, S. M. Fakhrahmad, and M. Z. Jahromi, "Perspell: A New Persian Semantic-based Spelling Correction System," *Digital Scholarship in the Humanities*, vol. 32, no. 3, pp. 543–553, 03 2016.

[22] A. Yazdani, M. Ghazisaeedi, N. Ahmadinejad, M. Giti, H. Amjadi, and A. Nahvijou, "Automated misspelling detection and correction in Persian clinical text," *Journal of Digital Imaging*, pp. 1–8, 2019.

[23] S. Salavati and S. Ahmadi, "Building a lemmatizer and a spell-checker for Sorani Kurdish," *CoRR*, vol. abs/1809.10763, 2018. [Online]. Available: http://arxiv.org/abs/1809.10763

[24] P. Etoori, M. Chinnakotla, and R. Mamidi, "Automatic spelling correction for resource-scarce languages using deep learning," in *Proceedings of ACL 2018, Student Research Workshop*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 146–152.

[25] S. Zhang, H. Huang, J. Liu, and H. Li, "Spelling error correction with soft-masked bert," *arXiv preprint arXiv:2005.07421*, 2020.

[26] M. Mars, "Toward a robust spell checker for Arabic text," in *International Conf. on Computational Science and Its Applications*. Springer, 2016, pp. 312–322.

[27] N. AlShenaifi, R. AlNefie, M. Al-Yahya, and H. Al-Khalifa, "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*, 2015, pp. 127–132.

[28] H. Mubarak and K. Darwish, "Automatic correction of arabic text: A cascaded approach," in *Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP)*, 2014, pp. 132–136.

[29] H. Bouamor, H. Sajjad, N. Durrani, and K. Oflazer, "Qcmuq@ qalb-2015 shared task: Combining character level mt and error-tolerant finite-state recognition for Arabic spelling correction," in *Proceedings of the Second Workshop on Arabic Natural Language Processing*, 2015, pp. 144–149.

[30] H. M. Noaman, S. S. Sarhan, and M. Rashwan, "Automatic Arabic spelling errors detection and correction based on confusion matrix-noisy channel hybrid system," *Egypt Comput Sci J*, vol. 40, no. 2, pp. 54–64, 2016.

[31] A. Zahui, W. Elhor, and M. A. Cheragui, "EL-Mossahih V1.0: A hybrid approach for detection and correction of typographical and phonetic transcription errors in Arabic texts," in *2017 8th International Conf. on Information Technology (ICIT)*. IEEE, 2017, pp. 774–779.

[32] M. M. Al-Jefri and S. A. Mahmoud, "Context-sensitive Arabic spell checker using context words and n-gram language models," in *2013 Taibah University International Conf. on Advances in Information Technology for the Holy Quran and Its Sciences*. IEEE, 2013, pp. 258–263.

[33] C. Zribi and M. Ahmed, "Detection of semantic errors in Arabic texts," *Artificial Intelligence*, vol. 195, pp. 249–264, 2013.

[34] M. Rokaya, "Arabic semantic spell checking based on power links," *International Information Institute (Tokyo). Information*, vol. 18, no. 11, p. 4749, 2015.

[35] D. Watson, N. Zalmout, and N. Habash, "Utilizing character and word embeddings for text normalization with sequence-to-sequence models," *arXiv preprint arXiv:1809.01534*, 2018.

[36] A. M. Azmi, M. N. Almutery, and H. A. Aboalsamh, "Real-word errors in Arabic texts: A better algorithm for detection and correction," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 8, pp. 1308–1320, 2019.

[37] M. Alkhatib, A. A. Monem, and K. Shaalan, "Deep learning for Arabic error detection and correction," *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, vol. 19, no. 5, pp. 1–13, 2020.

[38] A. Solyman, W. Zhenyu, T. Qian, A. A. M. Elhag, M. Toseef, and Z. Aleibeid, "Synthetic data with neural machine translation for automatic correction in arabic grammar," *Egyptian Informatics Journal*, 2020.

[39] A. Kuznetsov and H. Urdiales, "Spelling correction with denoising transformer," *arXiv preprint arXiv:2105.05977*, 2021.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] Wikimedia_Commons, "Long short-term memory," 2007. [Online]. Available: https://en.wikipedia.org/wiki/File:Long_Short-Term_Memory.svg

[42] G. A. Abandah, M. Z. Khedher, M. R. Abdel-Majeed, H. M. Mansour, S. F. Hulliel, and L. M. Bisharat, "Classifying and diacritizing Arabic poems using deep recurrent neural networks," *Journal of King Saud University - Computer and Information Sciences*, 2020.

[43] I. H. Ali, Z. Mnasri, and Z. Lachiri, "Dnn-based grapheme-to-phoneme conversion for arabic text-to-speech synthesis," *International Journal of Speech Technology*, vol. 23, no. 3, pp. 569–584, 2020.

[44] N. Habash, A. Shahrour, and M. Al-Khalil, "Exploiting arabic diacritization for high quality automatic annotation," in *Proceedings of the Tenth International Conf. on Language Resources and Evaluation (LREC'16)*, 2016, pp. 4298–4304.

[45] I. Bounhas, N. Soudani, and Y. Slimani, "Building a morpho-semantic knowledge graph for arabic information retrieval," *Information Processing & Management*, vol. 57, no. 6, p. 102124, 2020.

[46] A. Alosaimy and E. Atwell, "Diacritization of a highly cited text: a classical arabic book as a case," in *2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR)*. IEEE, 2018, pp. 72–77.

[47] A. Abdelli, F. Guerrouf, O. Tibermacine, and B. Abdelli, "Sentiment analysis of arabic algerian dialect using a supervised method," in *International Conf. on Intelligent Systems and Advanced Computing Sciences (ISACS)*. IEEE, 2019, pp. 1–6.

[48] R. Moumen, R. Chiheb, R. Faizi, and A. El Afia, "Arabic diacritization with gated recurrent unit," in *Proceedings of the International Conf. on Learning and Optimization Algorithms: Theory and Applications*, 2018, pp. 1–4.

[49] M. A. H. Madhfar and A. M. Qamar, "Effective deep learning models for automatic diacritization of arabic text," *IEEE Access*, vol. 9, pp. 273–288, 2020.

[50] A. Fadel, I. Tuffaha, M. Al-Ayyoub *et al.*, "Arabic text diacritization using deep neural networks," in *2019 2nd International Conf. on Computer Applications & Information Security (ICCAIS)*. IEEE, 2019, pp. 1–7.

[51] M. Maamouri, A. Bies, T. Buckwalter, and W. Mekki, "The Penn Arabic treebank: Building a large-scale annotated Arabic corpus," in *NEMLAR Conf. on Arabic Language Resources and Tools*, vol. 27. Cairo, 2004, pp. 466–467.

[52] I. Zitouni, J. Sorensen, and R. Sarikaya, "Maximum entropy based restoration of Arabic diacritics," in *Proceedings of the 21st International Conf. on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 2006, pp. 577–584.

[53] M. M. Alamri and W. J. Teahan, "Automatic correction of Arabic dyslexic text," *Computers*, vol. 8, no. 1, p. 19, 2019.

[54] Y. J. Choe, J. Ham, K. Park, and Y. Yoon, "A neural grammatical error correction system built on better pre-training and sequential transfer learning," *arXiv preprint arXiv:1907.01256*, 2019.