Hindawi
Advances in Human-Computer Interaction
Volume 2022, Article ID 3613710, 13 pages
https://doi.org/10.1155/2022/3613710

Review Article

A Comparative Study of Some Automatic Arabic Text Diacritization Systems

Ali Mijlad and Yacine El Younoussi

SIGL Laboratory, ENSATe, Abdelmalek Essaadi University, Tetouan, Morocco

Correspondence should be addressed to Ali Mijlad; a.mijlad@uae.ac.ma

Received 22 January 2022; Accepted 15 July 2022; Published 13 August 2022

Academic Editor: Armando Bennet Barreto

Copyright © 2022 Ali Mijlad and Yacine El Younoussi. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Arabic diacritization is the task of restoring diacritics or vowels for Arabic texts considering that they are mostly written without them. This task, when automated, shows better results for some natural language processing tasks; hence, it is necessary for the field of Arabic language processing. In this paper, we are going to present a comparative study of some automatic diacritization systems. One uses a variant of the hidden Markov model. The other one is a pipeline, which includes a Long Short-Term Memory deep learning model, a rule-based correction component, and a statistical-based component. Additionally, we are proposing some modifications to those systems. We have trained and tested those systems in the same benchmark dataset based on the same evaluation metrics proposed in previous work. The best system results are 9.42% and 22.82% for the diacritic error rate DER and the word error rate WER, respectively.

1. Introduction

Arabic is the language of over 422 million natives in the Arab world. It is present in the religious life of over a billion Muslims. It is also present, with its modern standard and dialectical forms, in the daily life of native speakers. The Arabic texts follow the Abjad writing system. Every letter is considered a consonant, while diacritics or vowels are mostly omitted and are left to the readers to deduce based on their knowledge of the language and the words' context.

The vowel marks or diacritics are written either above or below the letters. It is worth mentioning that typing speed of the Arabic script may decrease to half if we include diacritics in the written text. They could also improve the reading comprehension of people suffering from dyslexia. This is due to the cognitively demanding characteristics of inferring the diacritics, and as shown by Al-Wabil et al. [1] Arabic dyslexics lack skills in working memory and phonological skills. Moreover, using intensively the nonvowelized form in Arabic web content becomes an obstacle for dyslexic and visually impaired readers.

Another problem cited by Al-Wabil et al. [1] is that diacritics can help infer the right words, but, at the same time, they add visual complexity to the text. This adds more effort for dyslexics since reading requires visual discrimination and memory skills. A proposed solution to address this problem is to offer the text in its undiacritized form in addition to diacritization options by levels (partial or full).

Furthermore, not only diacritics are important to children and Arabic novice learners, but they can also offer great help in some natural language processing NLP tasks. Those tasks can be text-to-speech, machine translation, automatic speech recognition, Part-Of-Speech (POS) tagging, etc. For example, the diacritized form can narrow down the result list of an information retrieval system. In addition, it can lead to better text classification, which could be utile for sentiment analysis systems. Ergo, the automatic Arabic diacritization task, is important in the Arabic NLP field, and in the real life of Arabic beginner learners and people with Specific Learning Difficulties (SpLD).

In this paper, we are going to present a comparison of some diacritization systems. The first one uses the hidden Markov model (HMM) combined with smoothing techniques. The second system is a pipeline of multiple components, starting with a deep learning model, followed by a

rule-based model and a statistical-based model. The third system is quite similar to the second one, but we modified the input layer by using char embedding to replace the one-hot encodings. The study was done based on the same benchmark dataset and using the same evaluation metrics proposed in previous work [2]. The best results are done by the HMM-based system using Laplace smoothing technique. When we include the count of nonvocalized letters in the original text, the system does 9.42% and 22.82% for the DER and WER, respectively. Without the count of nonvocalized letters in the original text, the system does 10.60% and 21.89% for DER and WER, respectively.

For this comparison, this paper will have first a section that represents the language background. After that, we are going to look at some works that tackled the same problem. Then, we are going to describe the main systems we compared in this study. Finally, we are going to present the experimental settings and results of the comparison we have made.

2. Arabic Language Background

In this section, we are going to look into the Arabic language background, specifically the diacritization area.

Arabic diacritic marks can be grouped as shown in Table 1. First short diacritics or *Harakat: Fathah, Kasrah,* and *Dammah*. Second, *tanwīn* or nunation diacritics are visually double short diacritics and are present at the end of some words; when read, it gives the sound of the short vowel followed by an/*n*/ sound. The third contains *shaddah*, which geminates the letters, and it can be present with the short diacritics and the nunations. The last group comprises the *sukūn*, which is written above a consonant to mark the absence of vowels and indicate the closure of a finished syllable.

Diacritics have two purposeful classes. The lexical or morphological ones (core-diacritics) help distinguish the lexical characteristics of the word, while the inflectional class helps with the syntactic characteristics of the same lexeme within a sentence. They are also called case-ending vowels, and they are harder to infer than the former class [3].

3. Related Works

In this section, we are going to summarize some previous works related to automatic diacritization.

Elshafei et al.'s system [4] firstly gives a dictionary and words with their frequencies. Then, it gives bigram and trigram distributions. Then, a hidden Markov model (HMM)—with nonvocalized sequences as observed states and diacritized forms as hidden ones—determines the best vocalization. The system was trained based on a large diacritized corpus related to different fields. The vocabulary size is 18,623 diacritized words (179,910 characters). The list of the corresponding undiacritized words is of size 15,006 (about 80,864 letters). The test set was about 50 sentences (about 995 words made up of 7657 characters) chosen randomly from the Quran text. The diacritic error rate (DER) is about 4.1%.

Ya'kov Gal's work [5] is used for Arabic and Hebrew vowels' restoration. The unigrams and bigrams are extracted at first. Then, an HMM model with the Viterbi decoding algorithm helps find the best output. As an Arabic dataset, the work used a publicly accessible version of the Qur'an corpus (https://www.sacred-texts.com/) that contains 90,000 words. The system achieved 14% and 19% word error rate (WER) for the Arabic and the Hebrew test data, respectively.

Zayyan et al.'s system [6] used multilexical layers to infer diacritics. The first layer is word-based, and it uses n-grams models, taking into account left and right contexts. The second layer is letter-based, and it uses n-grams also taking into account left and right contexts. The dataset used is made up of LDC Arabic Treebank (LDC Arabic Tree Bank Part 3: http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2005T20) (about 340K words), and the dialectal corpus of CallHome Arabic (https://catalog.ldc.upenn.edu/LDC97T19) (120 transcripted telephone conversations). The solution achieved 16.8% and 11.7% for WER and DER, respectively.

Shaalan et al. [7] combined lexicon retrieval methods, a bigram model, and a support vector machine SVM, which determines the Part-of-Speech POS tags. The latter—POS tags—are used to select the best bigrams. The system also used an external analyzer to tackle the inflectional characteristic of Arabic texts. The training and evaluation dataset used for this system is the LDC Arabic Treebank (Diacritized news Part 2 v2.0: catalogue number LDC2004T02 and 1-58563-282- 1). It includes 144,199 tokens. The system achieved 11.79% as WER and 3.24% as DER.

To restore diacritics, Shahrour et al. [8] used MADAMIRA (https://camel.abudhabi.nyu.edu/madamira/) analyzer, J48 decision tree classifiers, and syntax rules. As a dataset, they used the Penn Arabic Treebank (PATB parts 1, 2, and 3). The achieved WER is 9.4%.

Ahmed Said et al.'s work [9] is in the form of sequence. Firstly, it starts with the autocorrection of common Arabic mistakes and the tokenization. Second, the out-of-vocabulary OOV generation comes accompanied by the extraction of morphological features using rules and a statistical-based analyzer. Thirdly, an HMM for POS tagging returns the most likely sequence and helps with the case endings' restoration. Lastly, a statistical-based model handles the OOV. The system was trained and tested using the standard LDC Arabic Treebank corpus (part 3 version 1.0, #LDC2004T11). The training set contains around 288K words; the blind set contains 52K words. The system has achieved a WER of 11.4% and a DER of 3.6%.

Bebah et al. [10] used Alkhalil Morpho Sys (https://sourceforge.net/projects/alkhalil/) to extract morphological information. Then, an HMM chooses the most likely vowelization based on the frequency distribution of words in the corpus. The HMM uses undiacritized sequences as observations, while it either sees the possible vocalization forms or the possible diacritics as hidden states. The corpus used in the training (respectively, testing) phase is derived mainly from NEMLAR (http://catalog.elra.info/en-us/repository/browse/ELRA-W0042/), Tashkeela (https://sourceforge.net/projects/tashkeela/), and RDI (http://www.rdi-eg.com/RDI/TrainingData/) corpora. The training corpus consists of 2,463,351 vowelized words. The

Table 1: Arabic diacritics with their International Phonetic Alphabet representation (IPA).

Diacritic mark	Name	Type	Transl.	IPA	Word position
ـَ	Fathah	Short vowels	a	/*a*/	Any
ـُ	Dammah	Short vowels	u	/*u*/	Any
ـِ	Kasrah	Short vowels	i	/*i*/	Any
ـً	Tanwin Fath	Tanwin	F	/An/	End
ـٌ	Tanwin Damm	Tanwin	N	/Un/	End
ـٍ	Tanwin Kasr	Tanwin	K	/In/	End
ـّ	Shaddah	Shaddah	~	:	Any
ـْ	Sukun	Sukun	o	∅	Any

proposed vocalizer has a WER of 21.11%, and a DER of 7.37%.

Samah Alansary's system [11] is based on rules where the Arabic diacritization processing is done in seven steps grouped into three modules: morphological processing, syntactic processing, and morph-phonological processing. The data set was selected from the International Corpus of Arabic (ICA) (http://www.bibalex.org/ica/en/about.aspx). The vocabulary size of the training set is about 300,000 Arabic words, and the testing vocabulary size is 100,000. Moreover, they used a testing set derived from LDC Arabic Treebank (nearly 52,000 words). The system achieved a WER of 15.7%.

Mohsen Rashwan et al.'s work [12] utilizes a context memory applied to each tokenized word. Then, two deep networks simultaneously use them (the word and its context). One extracts the features, and the other deep network finds POS tags. The results of the two networks are fed then to a deep net for classification. The used dataset is composed of LDC's Arabic Treebank (ATB) part 3 corpus (catalog identifier = LDC2005T20) and some customized datasets. Based on the ATB dataset, the case-ending accuracy is shown to be 88.4%, while the morphological accuracy is about 97%.

Yonatan Belinkov et al.'s system [13] uses a bidirectional Long-Short Term Memory (BLSTM) network to infer the diacritics. It is composed of a char embedding layer, three BLSTM layers, and an output layer that generates probability distributions over labels. The data set was from the Arabic Treebank. The train, dev, and test sets contain, respectively, 470K, 81K, and 80K words. The system's DER is about 4.85%.

Gheith Abandah et al.'s system [14] also used a deep BLSTM for diacritics' restoration. The dataset was composed of ten books from the Tashkeela corpus, the Holy Quran, and the LDC's Arabic Treebank part 3, v3.2 (#LDC2010T08). The vocabulary size of the LDC ATB3 corpus is about 305,000 words. The average vocabulary size of all the corpora (including LDC ATB3) is about 402,000 words. Testing on the Tashkeela corpus gave a DER of 2.09%, and a WER of 5.82%. Testing on the ATB3 gave a DER of 2.72% and a WER of 9.07%.

Aya Metwally et al. [15] have proposed a system that contains three phases. The first one is for restoring morphological diacritics and POS tags, using an HMM with Laplace smoothing technique. In the second step, the system infers the same things, yet for the unseen words (Out of vocabulary OOV). The last phase infers the syntactic vowels using morphological features, POS tags, and a Conditional Random Fields (CRFs) classifier. For the experiment, they used the LDC Arabic Treebank part 3 dataset. It consists of 600 articles (about 340,000 words). The WER was reported to be 13.7%.

Chennoufi et al.'s system [16] uses firstly Alkhalil Morpho Sys to extract morphological features and all different vowelization for each word. Then, the syntactic rules are used to throw invalid sequences. Thirdly, the HMM is used to choose the best one. Finally, a char-based HMM is used to process unseen words. The dataset consists of the Tashkeela corpus (63 million diacritized words), NEMLAR text (500,000 diacritized words), and the RDI corpus (about 8.5 million diacritized words). The achieved WER and DER are 6.22% and 1.98%, respectively.

Amany Fashwan et al.'s system [17] has two levels. The first one is for morphological processing through unimorphological processing, morphological rules, statistical processing, and processing of unseen words. The second level is for syntactic rules, which help restore case endings. The testing was done based on the LDC's Arabic Tree-bank part 3 (about 52,000 words). The WER is about 14.78%, while the DER is about 4.11%.

The work of Kareem Darwish et al. [18] does the task in two phases. One is for internal vowels' inferring using bigrams, unigrams' stems, stem patterns' templates, and sequence labeling of stems. The second phase deals with the case endings via the SVM ranking model and heuristics. The training corpora (acquired from a commercial vendor) contain more than 9.7 million words. The testing data—containing about 18,300 words—was made up of 70 WikiNews articles. The work achieved 12.76% WER and 3.54% DER.

Saba' Alqudah et al. [19] followed also the hybrid approach by fusing the MADAMIRA analyzer and a deep bidirectional LSTM network. The outputs of MADAMIRA with a high confidence parameter are input to the network. The experimental data set is the LDC Arabic Treebank part 3 v3.2 (catalog id = LDC2010T08). It comprises 305,000 words. The system achieved 2.39% and 8.40% as DER and WER, respectively.

Badr Alkhamissi et al. [3] proposed a system that has two models, each with two levels. The first has a word-level encoder. The second is a character encoder. In addition, a cross-level attention unit is used to improve the character's representation, by utilizing embeddings of characters to access every word in the sentence. The second model has a

forward LSTM that inputs char embeddings and one-hot encoding of the previous model. In addition, it has a final classifier utilizing a $Softmax$. The dataset was generated from the Tashkeela corpus. It was split into the train (2,449K tokens), validation (119K tokens), and holdout (125K tokens) sets. The system achieved 5.34% WER and 1.83% DER.

Ismail Hadjir et al. [20] presented a system based on a modification of the HMM using the Viterbi decoding algorithm. The training set is made up of 26 books from the Tashkeela dataset, while the testing set is made up of three other books from the same corpus. The system achieved a precision of up to 80% at the word level.

4. Presentation of the Three Systems

In this section, we are going to present the three compared systems based on which we did the comparison.

4.1. Arabic Diacritizer That Uses a Multilevel Statistical Model. Mohamed Hadj Ameur et al. [21] proposed—as shown in Figure 1—a model consisting of multiple layers, which are statistically based ones. The first is a bigram-word-based model using a modified version of HMM. The second phase is dedicated to unsolved cases. It is based on a 4-gram character-based model, which uses also a modified version of HMM.

The first phase follows the following steps: Firstly, a dictionary is built; it associates each undiacritized word with its possible vocalizations. The second step generates a lattice, for the undiacritized sequence $W_1, W_2, \ldots, W_n$, where each word W_i is associated with its possible diacritized forms from the dictionary. Then, for each possible diacritization, a probability is calculated using the bigram model assumption expressed in the following equation:

$$P\left(\frac{W_n}{(W_1 \ldots W_{n-1})}\right) = P\left(\frac{W_n}{W_{n-1}}\right). \quad (1)$$

Finally, among all the possible diacritizations, the one with the highest probability is generated as formulated in the following equation:

$$d_1, d_2, \ldots, d_n = \arg\max\left(\prod_{k=1}^{n} P\left(\frac{d_k}{d_{k-1}}\right)\right). \quad (2)$$

The Viterbi algorithm uses a variation of HMM as input to generate the best vocalization.

The HMM is defined by the following:

(i) A set of states representing the diacritized words $d_1, d_2, \ldots, d_n$

(ii) A set of observations representing the undiacritized words $W_1, W_2, \ldots, W_n$

(iii) The matrix of transitions: contains transition probabilities

It is to note that the HMM variation proposed in this model is that *the transition probabilities are considered, while the emission probabilities are neglected*. The Viterbi algorithm uses a recursive relation to pick the best diacritized sequence from the hidden Markov model.

The probability, for each transition indexed (i, j), in the level i, is computed based on its precedent ones in level $i-1$, as shown in the following equation:

$$P(i,\ j)\ =\ \max_{k=1, v_{i-1}}\left(P\left(\frac{i, j}{i-1, k}\right) \times P(i-1, k)\right), \quad (3)$$

where v_{i-1} is the number of all diacritization possibilities for the $(i-1)^{th}$ word. In the forward propagation of this algorithm, all the probabilities are calculated while recording the best transition probabilities on this path. Then, the back tracing helps in returning the optimal path.

To solve the problem of unseen bigrams, the smoothing technique (Laplace smoothing or Absolute Discounting) is used.

The second phase is a 4-gram letter-based model used for nonvocalized words from the first phase. It uses a letter-based HMM, which consists of a set of the following:

(i) States representing the diacritized letters $q_1, q_2, \ldots, q_n$

(ii) Observation states: represented by the non-diacritized letters $l_1, l_2, \ldots, l_n$

(iii) The matrix of transition probabilities $P(q_i/q_{i-1}, q_{i-2}, q_{i-3})$

(iv) The matrix of emissions for emission probabilities $P(l_i/q_i)$

This letter-based model is used similarly to the word-based model with the smoothing techniques. Besides, the best path is selected based on the Viterbi algorithm.

It is also worth mentioning that the source code (https://github.com/Ycfx/Arabic-Diacritizer), related to this work, seems to be incomplete, so we restored some missing methods—algorithms 1–4— which were used mainly in the letter-based component. Besides, we corrected some lines of code in other methods.

4.2. Multicomponent System for Automatic Arabic Diacritization. The system proposed by Hamza Abbad et al. [22] is a pipeline of three main components. At the outset, a preprocessing phase is needed. Its main rule is to simplify the presentation of the data. First, it keeps the Arabic letters and the spaces and replaces numbers with 0 s. The other chars are inferred after diacritics restoration. Second, each sentence is presented as an input and its corresponding outputs. The outputs are a one-hot encoded vector for the presence of *Shadda* diacritic on a letter, and a 2D array of one-hot encoded vectors representing the primary diacritics for each character in the sentence. Along with that, the input is mapped to the 38 numeric labels of the kept characters. They are then one-hot-encoded as a two-dimensional array with shape *(length of sentence, number of labels)*. After that, the input array is extended to be a 3D tensor with shape *(length of sentence, number of time steps, number of labels)*.

The first component of the pipeline is a recurrent neural network RNN-based model composed of two stacked

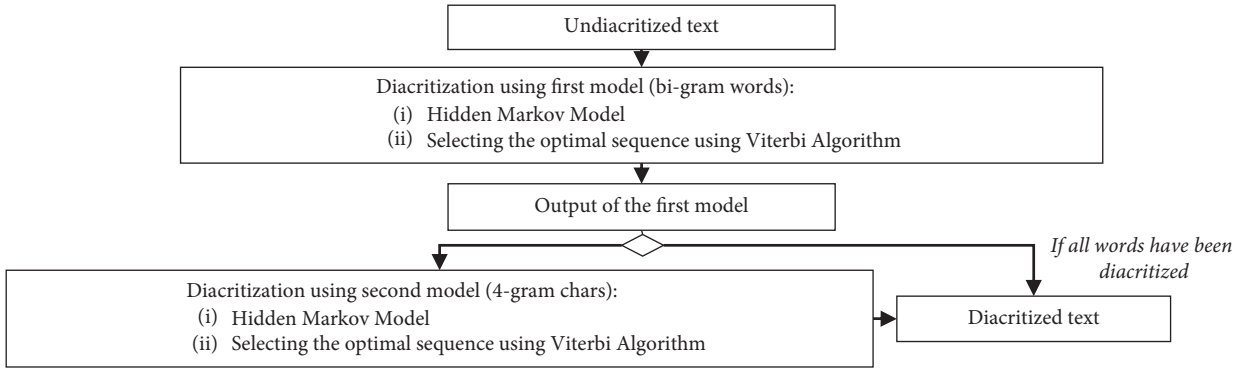

Figure 1: Vocalization system using hidden Markov model (HMM).

BLSTM layers of 64 cells in each direction. Then, there come two parallel dense layers of sizes 8 and 64. The dense layers and their previous layers use the *tanh* activation function. The first parallel layer, which helps predict the presence of *Shadda*, is connected to a single perceptron with a *sigmoid* activation function. The second parallel layer, which helps in the prediction of primary diacritics, is connected to seven perceptrons followed by a $Softmax$ function. Figure 2 shows the architecture of the deep learning component.

The second phase is for rule-based corrections. It is connected to the input and output of the previous component, to do the *Shadda* and the primary diacritics' corrections based on Arabic rules.

The third component is for statistical-based corrections. Outputs and inputs of the previous correction component are merged and transformed. Then the resultant sentence is split into words. Each word is checked, and it goes through four sublevels. The first one is dedicated to word trigram corrections; it generates trigrams from the nonvocalized sentence and selects the most frequent vocalization for the second word of the trigram. The second sublevel is for word bigram corrections; it is similar to the first one, and it chooses the most frequent vocalization of the second word in the bigram. The third sublevel calculates the Levenshtein distance of the nonvocalized word, which has known diacritized forms. This distance is mainly the minimum edit distance between the nonvocalized word and its diacritized forms. The last sublevel is applied when the predicted word is never seen. It calculates the Levenshtein distance between the corresponding pattern of that predicted word and the saved vocalized forms of the same pattern.

4.3. The Modified Proposed Version. In this part, we are going to present the modifications we tried to make to the pipeline proposed by Abbad et al. [22]. In fact, instead of using the char one-hot encoding, we used char embedding to see its impact on the system.

4.3.1. How the Char Embedding Matrix Is Created. The Arabic chars were embedded using a simple neural network from previous work available on Github (https://github.com/sonlamho/Char2Vec). The network inputs a one-hot encoding representation corresponding to a character and outputs its context vector, which is the distribution of the neighboring chars.

Suppose that $c[i]$ is a character from the corpus, x is its one-hot encoding representation having v as a dimension, $2 \times k$ is the number of characters encompassing $c[i]$, and y is the context vector of dimension $2 \times k \times v$.

The network learns the matrices of weights U and W, which have, respectively, the shapes (v, d) and $(d, 2 \times k \times v)$, and they verify equation (4). Finally, $x \cdot U$ becomes the embedding vector for the char c[i], and it has the shape d.

$$y \sim Sigmoid\,(x \cdot U \cdot W). \tag{4}$$

4.3.2. Including Char Embeddings in the Old System. The proposed modification uses the same architecture of the deep learning component proposed in Abbad et al.'s work [22] as shown in Figure 2. Nevertheless, the input has quietly changed, so that its tensor has three dimensions. The first one is for the sentence dimension, the second is for the time steps, and the third dimension is for char embedding representations instead of one-hot encodings. Figure 3 shows the proposed input modification.

Additionally, as shown in Figure 4, using char embedding means modifications in both the input layer of the deep learning component and the input of the rule-based components.

The rule-based part uses the char indexes, which should be extracted from the input sequence of the char embeddings. Therefore, we created—in algorithm 5— a method dedicated to that purpose. The algorithm takes *Seq* a 2D array representing the sequence of char embeddings and *emb_Mat* a 2D array representing all the learned char embeddings. The method outputs the sequence of char indexes corresponding to char embeddings in the sequence *Seq*.

5. Experience and Results

In this section, we are going to describe the dataset and evaluation metrics used in this study. Besides, we are going to look at the results based on those metrics and for the same dataset.

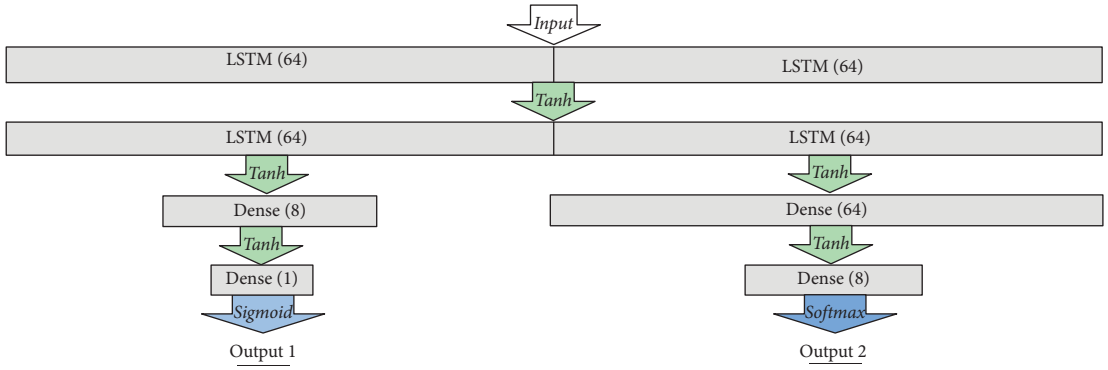

Figure 2: The architecture of the deep learning component.

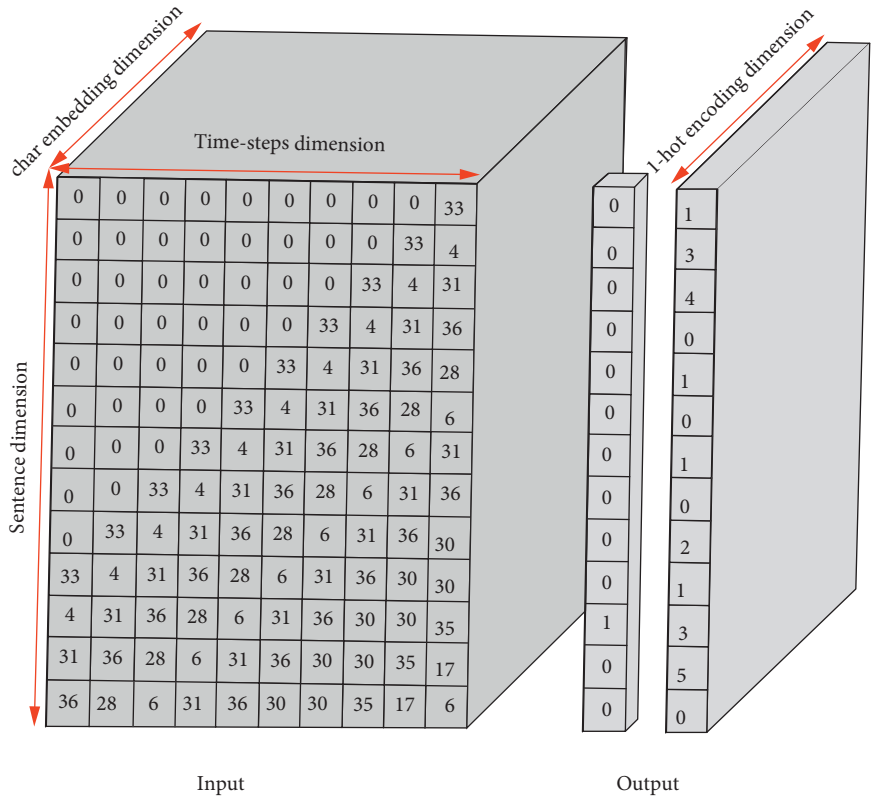

Figure 3: Input and output tensors in the proposed modification.

5.1. Dataset. To make a good comparison with other works, we used the free available benchmark dataset proposed by Fadel et al. [2]. The corpus was chosen—in that work (https://github.com/AliOsm/arabic-text-diacritization)—randomly from classical Arabic books and Holy Quran, which belong to the Tashkeela corpus (https://sourceforge.net/projects/tashkeela/) distribution. To make use of it properly, it was cleaned and preprocessed. The resulting output was then split into training, validation, and holdout sets with a percentage of 90%, 5%, and 5%, respectively.

From the dataset provided, we generated more statistics about the train, validation, and test sets. Table 2 shows some statistics about different chars and tokens, while Table 3 shows the percentage of Out-Of-Vocabulary words in the validation and holdout sets compared to the training set.

5.2. Evaluation Metrics. The metrics used to evaluate this comparison are the same used in Fadel et al.'s work [2]:

(i) Diacritic Error Rate (DER): the percentage of diacritics that are attributed wrongly

(ii) Word Error Rate (WER): the ratio of words having at least one wrong diacritic attribution

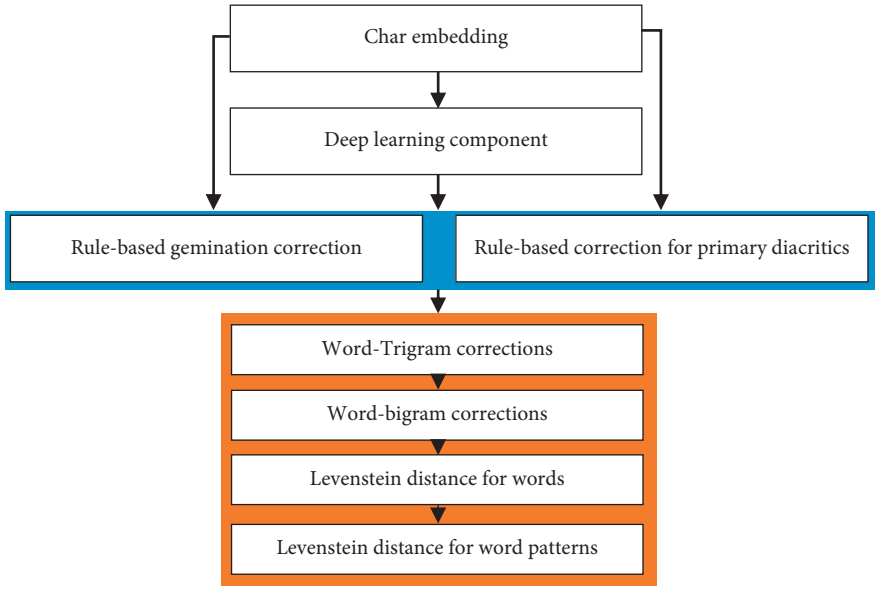

Figure 4: The architecture of the system.

(i) **Input:** list of diacritized sentences *s*,
(ii) **Output:** distribution of diacritized letters *lDist*
(iii) SET words EQUAL TO EMPTY LIST
(iv) SET letters EQUAL TO EMPTY LIST
(v) SET lDist EQUAL TO EMPTY DICTIONARY
(vi) tool: = MyToolKit()//this is a toolkit class from the same work, it contains a group of methods
(vii) FOR EACH sent IN s
(viii) sentenceWords: = tool.words(sent)//splitting according to Arabic regular expression
(ix) words.extend(sentenceWords)
(x) FOR *w* IN words
(xi) wordLetters: = tool.LettersDiac(w)//diacritized letters of the word w
(xii) letters.extend(wordLetters)
(xiii) lDist: = freqDist(letters)
(xiv) RETURN lDist

Algorithm 1: create_Distribution_Of_Diacritized_Letters.

(i) **Input:** list of all letters combined with all diacritics *l*,
(ii) **Output:** list of tuples d
(iii) //each tuple contains the letter and its' diacritizations separated with spaces
(iv) SET dict EQUAL TO EMPTY DICTIONARY
(v) SET *d* EQUAL TO EMPTY LIST
(vi) tool: = MyToolKit()
(vii) FOR EACH *e* IN *l*
(viii) dict[tool.deleteDiacritics(e)].append(e)//The value here is a list of diacritization of the letter e
(ix) FOR key IN dict
(x) d.append((key, ' '.joined(dict[key])))
(xi) RETURN d

Algorithm 2: create_Dictionary_Of_Letters.

```
(i) Input: list of diacritized sentences sents,
(ii) The size of the n-grams n
(iii) Output: dictionary d distribution of letter n-grams
(iv) SET grams EQUAL TO EMPTY LIST
(v) SET words EQUAL TO EMPTY LIST
(vi) SET ngrDist EQUAL TO EMPTY DICTIONARY
(vii) tool: = MyToolKit()
(viii) FOR EACH s IN sents
(ix)     words.extend(tool.words(s))//tool.words(): splits s to words
(x) FOR EACH w IN words:
(xi)     grams: = grams + n-grams(tool.LettersDiac(w),n)//n-grams() is a method that extract n-grams
(xii) ngrDist: = freqDist(grams)
(xiii) RETURN ngrDist
```

Algorithm 3: create_Distribution_Of _Letter_N-grams.

```
(i) //Extract the 3-gram letters that have a successor to form the 4-grams.
(ii) //This method can be used when calculating the Absolute Discount Smoothing
(iii) Input: list of diacritized 4-grams grams,//each 4-gram is a tuple
(iv) Output: a dictionary d representing the distribution of 3-gram letters that have a successor
(v) SET unzip EQUAL TO EMPTY LIST
(vi) SET n_minus_one_grams EQUAL TO EMPTY LIST
(vii) SET d EQUAL TO EMPTY DICTIONARY
(viii) unzip: = LIST(zip(*grams))//Unzip the grams into 4 tuples
(ix) unzip.pop()//delete the last tuple
(x) n_minus_one_grams: = zip(unzip[0], unzip [1], unzip [2])
(xi) d: = freqDist(list(n_minus_one_grams))
(xii) RETURN d
```

Algorithm 4: create_Distribution_Of_Letter_3-grams_Having_A_Successor.

```
(i) Input: 2D tensor representing the sequence of embeddings seq,
(ii) //shape of seq is (time_steps, embedding_dim)
(iii) 2D tensor for the learned embedding matrix emb_Mat,
(iv) //shape of Emb_Mat is (number of chars, embedding_dim)
(v) Output: a tensor t//of shape (time_steps) where each row contains the index of the char
(vi) seq_shape: = shape(seq)
(vii) b_shape: = shape(emb_Mat)
(viii) //tile seq along new dimension
(ix) seq_tiled: = tile(seq, [1,b_shape[0]])
     //reshape
(x) seq_tiled: = reshape(seq_tiled, [seq_shape[0], b_shape[0],seq_shape [1]])
(xi) //Elementwise comparison
(xii) eq: = equal(emb_Mat, seq_tiled)
(xiii) //Reduce the last dimension
(xiv) red: = reduce(eq, -1)
(xv) //element where condition eq is True
(xvi) z: = where(red)
(xvii) t: = z[:,1]
(xviii) RETURN t
```

Algorithm 5: convert_A_Sequence_Of_Embeddings_to_A_Sequence_Of_Indexes

Table 2: Some statistics about the corpus splits used in this study.

	Train set	Validation set	Test set
Characters	16082164	784570	820022
Tokens	2460405	120075	125220
Numbers	33260	1648	1637
Digits	75963	3794	3774
Arabic words	2103156	102479	107291
Arabic letters	8356030	407434	426469
Diacritics	7290312	355666	371726
Undiacritized forms	105720	19515	20520
Diacritized forms	163237	26129	27298

Table 3: Percentage of out of vocabulary (OOV) in validation and test sets compared to the training set.

	Validation set	Test set
Diacritized OOV Arabic words (%)	15.52	16.70
Undiacritized OOV Arabic words (%)	11.84	13.19

Numbers and punctuations are not taken into account when calculating those metrics defined in that work [2], unlike other previous comparative works.

When calculating DER and WER, we can either include the case-ending or not. Furthermore, we can include or not the 'no diacritic' class, which gives the choice of whether to count or not the letters with no diacritics from the original text.

6. Results

In this part, we are going to present the results of the tested models.

Before that, it is fair to mention that, for the deep learning models, we did not use *Adadelta* optimizer as in the original work [22]. Instead, we used the Adam optimization method, because it is computationally efficient, is less memory-consuming, and converges faster. Besides, the training dataset used for this model is smaller than the one used in the original work [22]. In addition, we tried to replace the LSTM layers in the systems described above by using the GRU layers to see if we will get better results.

The systems that are based on HMM achieved the errors shown in Table 4. When observing some errors given by the HMM-based systems, we found that one of the common errors done by those systems is that the *normalizeAlif* method substitutes the repetitions of the forms of the Arabic *Alef* letter "أ" , آ" ", إ" ", ٱ," or "ا," with the "ا." This caused problems for the n-gram distributions and the dictionaries. Consequently, this had an impact on the test results. For example, when the system tries to vowelize the word "فلأن"/*fli'ana*/, it replaces "أ" with "ا". This is diacritized as "فُلَانٌ"/*fulanun*/(meaning: John Doe, or so-and-so), or diacritized sometimes as "فلَان"/*falana*/(meaning: it became soft or flexible). However, the diacritized form we want is "فلِأَنَّ"/*fali'anna*/ (which means: and because). To solve this issue, we altered the method to substitute the repetitions of each Arabic *Alef* letter form with its corresponding one. The systems showed some improvements as shown in Table5.

As shown in Table 6, the HMM-based systems have common mistakes that are related to the case-ending. It has also errors that are related to the letter-based component where sometimes the nunations can be put in the middle of a word. Also, sometimes this component—letter-based HMM—inserts nunation at the end of verbs. Generally, the letter-base component has generated some errors concerning the internal diacritics. Sometimes, it affects negatively some well-diacritized—by the previous word-based HMM—words.

In Table 7, we have some common mistakes done by the systems that use the deep learning models (either with LSTM or with GRU). As shown, the systems have different kinds of errors. Those mistakes are related to either the case endings or the internal diacritics, or both. As far as we know, those models have some common mistakes with the HMM-based ones. Besides, some of the mistakes of the HMM-based systems are not present here, such as the nunation problem in the middle of the word or the beginning.

Table 5 shows the comparison results of all the systems, and the best results are in italic.

For the WER, when we exclude case endings, all the systems get a much less WER, which means that a good percentage of words have case ending errors. Besides, by the use of the 'no diacritic' class and looking at the DER, we can observe from the table that the original test set has at least 1% of chars that are not diacritized.

We can observe also from the results that the letter-based HMM makes a good amelioration for the error rate, especially the DER. Furthermore, for this dataset, we can see that the HMM-based model by Mohamed Ameur et al. [21] has better results compared to the other systems that use the deep learning component. Nevertheless, this cannot justify some errors that are related to the letter-based-HMM component.

Moreover, for the system that uses one-hot encoding and the other one using char embedding, they both have similar error rates. This can be explained by the fact that the embedding is more useful for word-level because, for

Table 4: Error rates of the HMM-based systems

Systems	Diacritic error rate: DER (%)				Word error rate: WER (%)			
	Including 'no diacritic' class		Excluding 'no diacritic' class		Including 'no diacritic' class		Excluding 'no diacritic' class	
	With case ending	Without case ending	With case ending	Without case ending	With case ending	Without case ending	With case ending	Without case ending
Mohamed Hadj Ameur et al.**'s work** [21] **(word-level HMM using Absolut discount smoothing technique)**	10.58	8.13	12.36	9.45	23.84	12.22	22.88	11.80
Mohamed Hadj Ameur et al.'s work [21] (word-level and letter-level HMMs using Absolute discount smoothing technique)	**9.74**	**7.10**	**11.00**	**7.84**	**23.90**	**12.19**	**22.87**	**11.75**
Mohamed Hadj Ameur et al.**'s work** [21] **(word-level HMM using Laplace smoothing technique)**	10.49	8.20	12.22	9.52	23.10	12.06	22.16	11.67
Mohamed Hadj Ameur et al.'s work [21] (word-level and letter-level HMMs using Laplace smoothing technique)	**9.65**	**7.19**	**10.82**	**7.86**	**23.16**	**12.02**	**22.14**	**11.60**

Table 5: Comparison of word error rate (WER) and diacritic error rate (DER).

Systems	Diacritic error rate: DER (%)				Word error rate: WER (%)			
	Including 'no diacritic' class		Excluding 'no diacritic' class		Including 'no diacritic' class		Excluding 'no diacritic' class	
	With case ending	Without case ending	With case ending	Without case ending	With case ending	Without case ending	With case ending	Without case ending
Mohamed Hadj Ameur et al.'s work [21] (word-level HMM using Absolute discount smoothing technique)	10.29	7.78	12.09	9.12	23.37	11.44	22.49	11.12
Mohamed Hadj Ameur et al.'s work [21] (word-level & letter-level HMMs using Absolute discount smoothing technique)	*9.43*	*6.73*	*10.69*	*7.46*	*23.43*	*11.41*	*22.47*	*11.07*
Mohamed Hadj Ameur et al.'s work [21] (word-level HMM using Laplace smoothing technique)	10.27	7.94	12.02	9.28	22.76	11.53	21.90	11.24
Mohamed Hadj Ameur et al.'s work [21] (word-level & letter-level HMMs using Laplace smoothing technique)	*9.42*	*6.90*	*10.60*	*7.59*	*22.82*	*11.49*	*21.89*	*11.17*
Hamza Abbad et al.'s work [22] (using two LSTM layers)	*11.52*	*10.70*	*13.02*	*11.91*	*29.62*	*22.58*	*27.61*	*21.02*
Hamza Abbad et al.'s work [22] (replacing the LSTM layers with two GRU layers)	12.53	11.41	14.21	12.71	31.43	23.50	29.53	22.07
Hamza Abbad et al.'s work [22] (replacing only the second LSTM layer with a GRU layer)	12.10	11.09	13.69	12.34	30.66	22.98	28.81	21.60
Hamza Abbad et al.'s work [22] (replacing only the first LSTM layer with a GRU layer)	11.81	10.94	13.37	12.18	30.14	22.79	28.16	21.26

Table 5: Continued.

Systems	Diacritic error rate: DER (%)				Word error rate: WER (%)			
	Including 'no diacritic' class		Excluding 'no diacritic' class		Including 'no diacritic' class		Excluding 'no diacritic' class	
	With case ending	Without case ending	With case ending	Without case ending	With case ending	Without case ending	With case ending	Without case ending
Our proposed modified version of Hamza Abbad et al.'s work (using LSTM layers)	*12.11*	*10.97*	*13.73*	*12.20*	*31.46*	*23.15*	*29.45*	*21.59*
Our proposed modified version of Hamza Abbad et al.'s work (using GRU layers instead of LSTM layers)	13.21	11.62	15.03	12.97	33.57	24.21	31.53	22.65
Our proposed modified version of Hamza Abbad et al.'s work (using an LSTM layer followed by a GRU layer)	12.63	11.31	14.36	12.61	32.55	23.81	30.56	22.27
Our proposed modified version of Hamza Abbad et al.'s work (using a GRU layer followed by an LSTM layer)	12.50	11.21	14.21	12.51	32.18	23.61	30.20	22.08

Table 6: Examples of some common mistakes done by the HMM-based systems

Common mistakes done by the HMM-based systems		
The Predicted	The correct	Description
مْ أَرَهُ ضَحِكَ ضِحُكَ اَلِ أَكْثَرَ مِنْهُ	مْ أَرَهُ ضَحِكَ ضِحِكًا اَل أَكْثَرَ مِنْهُ	(i) For "ضِحُكَ اَ" /Duhaka/ it is an error done by the letter-based component. It should be "ضَحِكًا" /dahikan/ instead. (ii) For "أَلْكْثَرُ" / Aktharu/ it is a case-ending error. It should be "أَكْثَرَ" /Akthara/ instead.
مَاذَا تَصُنِعَانِ	مَاذَا تَصْنَعَانِ	(i) For "تَصُنِعَانِ"/Tasunin'anin/ it is a case-ending error: since it is a verb it should not have a nunation at the end. Also, it has a morphological error: nunation cannot be in the middle. This error is generated by the letter-based HMM. The correct form is "تَصْنَعَانِ" /tasna'ani/.
فَلَا تُخْلَفُ فِيهِ	فَلَا تَخَلَّفَ فِيهِ	(i) For "فِيهِ"/Feyeh/ it has a case-ending and an internal error. The correct form is "فِيهِ" /Fihi/. (ii) For "تُخْلَفُ" /Tukhlafu/ all the structure is wrong. The correct form is "تَخَلَّفَ" /Takhallufa/. Here the letter-based component in the system, when it diacritized the unseen word "تخلف" it modified some letters of the words surrounding the unseen word. The word "فيه" /Fih/ is correctly diacritized in other places where there is no OOV.
قْبَلَوِنْ	قْبَلُهُنْ	For "قْبَلَوِنْ" /Nin-qbalahu/: the nunation cannot be in the middle nor the beginning of a word. The internal diacritization is all-false. The correct form is "قْبَلُهُنْ" /naqbaluhu/.
قَوْلٌ مَرْغُوبٍ عَنْهُ	قَوْلٌ مَرْغُوبٌ عَنْهُ	"مَرْغُوبٍ" /marghubin/ here has a case-ending error. It should be "مَرْغُوبٌ" /marghubun/
يَا ابْنُ أَخِي	يَا ابْنَ أَخِي	"ابْنُ" /Bnu/ here has a case-ending error. It should be "ابْنَ" /Bna/. It is a context-related error.

large word vocabulary, one-hot-encoding words will cause very sparse vectors.

For the models that use the deep learning component, we can see that the models that have two LSTM layers give better results compared with the case when we replace either the first or the second or both LSTM layers with the GRU. This confirms the fact that LSTM is more accurate on large sequences.

All the systems have some pros and cons related to the results. As an example, the HMM-based models have good error rates, but they output some bad mistakes especially when they use the letter-based model. Another example is that both kinds of systems—the HMM-based ones and the deep-net-based ones—have mistakes related to the case-ending diacritics or the internal diacritics. Eventually, the studied systems can be ameliorated by training on a larger

Table 7: Common mistakes done by the deep-net-based systems.

Common mistakes done by the deep-network-based systems		
The predicted	The correct	Description
وَلَدُ الوَاحِدلا	وَلَدُ الْوَاحِدُلا	(i) For "وَاحِدلا"/Alwahidi/here it is a case-ending error. The right form is "ـوَاحِدُلا"/Alwahidu/.
أَنَّهُ تَمْلِكْ	أَنَّهُ تَمَلَّكَ	(i) Here the form "تَمْلِكْ"/Tamlik/is wrong because the internal diacritics and the case ending are wrong. The right answer would be "تَمَلَّكَ"/Tamallaka/
ضَحِكَ ضَحِكَا	ضَحِكَ ضَحِكًا أَكْثَرَ	(i) For the word "ضَحِكَا"/Dahhikan/it has a case-ending error because a nunation is required in the *kaf* letter "ـك". It has also an internal error where the germination should not appear. The correct form is "ضَحِكًا"/dahikan/. (ii) For the word "أَكْثَرُ"/Akthara/, it is a case-ending error. In this context, the correct form should be "أَكْثَرَ".
تَحْرِجْتَ	تَحَرَّجْتَ	(i) For the word "تَحْرِجْتْ"/Tahrijut/the whole form is incorrect. The correct form is "تَحَرَّجْتَ"/Taharrajta/.
مَاذَا تَصْنَعَانِ	مَاذَا تَصْنَعَانِ	(i) "تَصْنَعَانِ"/Tassini'ani/here it has an error related to the internal diacritics. The correct form is "تَصْنَعَانِ"/Tasna'ani/.
أَعْلَمُ أَنَّهُمْ	أَعْلَمُ أَنَّهُمْ	(i) The word "أَنَّهُمْ"/Ann-hum/has a wrong internal diacritic over the letter "ـن". The germination should be accompanied by the short vowel *Fat-ha*. The correct form is "أَنَّهُمْ"/Annahum/.

dataset and by solving the common problems by adding rules.

7. Conclusion

To sum up, we have seen in this work a comparative study of some Arabic vowelization systems. The study we have done de facto has the purpose of assuring the reproducibility of those previous works, making some alterations to them, and summarizing the impacts. Additionally, we assured a fair comparison of those systems by training and testing them based on the same benchmark dataset. As result, we noticed that the system, which uses the hidden Markov model combined with the smoothing techniques, gives the best results. Additionally, for the second and third systems that have a deep learning component, when their deep network uses two LSTM layers, they perform better than the case when we used different layer architectures. This can be explained by the fact that, theoretically, LSTM can remember longer sequences than GRU. Furthermore, we can use a much larger dataset to see how the deep learning model would perform. This could give a conclusive decision about whether to use char embedding in this deep learning model, or the one-hot encoding system is already a sufficient solution.

Data Availability

The dataset used in this comparison study is publicly available in the GitHub repository: https://github.com/AliOsm/arabic-text-diacritization, from a previous work done by Fadel et al. [2]. For further information about the trained models in this work, you can contact the corresponding author. via a.mijlad@uae.ac.ma

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] A. Al-Wabil, P. Zaphiris, and S. Wilson, "Web design for dyslexics: accessibility of Arabic content," *Computers Helping People with Special Needs*, vol. 4061, pp. 817–822, 2006.

[2] A. Fadel, I. Tuffaha, B. Al-Jawarneh, and M. Al-Ayyoub, "Arabic text diacritization using deep neural networks," in *Proccedings of the 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pp. 1–7, Riyadh, Saudi Arabia, 2019.

[3] B. AlKhamissi, M. N. ElNokrashy, and M. Gabr, "Deep diacritization: efficient hierarchical recurrence for improved Arabic diacritization," 2020, https://arxiv.org/abs/2011.00538.

[4] M. Elshafei, H. Al-Muhtaseb, and M. M. Alghamdi, "Statistical methods for automatic diacritization of Arabic text," in *Proceedings 18th National Computer Conference*, Riyadh, Saudi Arabia, 2006.

[5] Y. Gal, "An HMM approach to vowel restoration in Arabic and Hebrew," in *Proceedings of the ACL-02 Workshop on Computational Approaches to Semitic Languages*, pp. 1–7, Philadelphia, PA, USA, 2002.

[6] A. A. Zayyan, M. Elmahdy, H. Binti Husni, and J. M. Al Ja'am, "Automatic diacritics restoration for dialectal Arabic text," *International Journal of Computing and Information Sciences*, vol. 12, no. 2, pp. 159–165, 2016.

[7] K. Shaalan, H. M. Abo Bakr, and I. Ziedan, "A hybrid approach for building Arabic diacritizer," in *Proceedings of the EACL 2009 Workshop on Computational Approaches to Semitic Languages - Semitic '09*, Athens, Greece, 31 March, 2009.

[8] A. Shahrour, S. Khalifa, and N. Habash, "Improving Arabic diacritization through syntactic analysis," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1309–1315, Lisbon, Portugal, 2015.

[9] A. Said, M. El-sharqwi, A. Chalabi, and E. Kamal, "A hybrid approach for Arabic diacritization," in *Proceedings of the Natural Language Processing and Information Systems: 18th International Conference on Applications of Natural Language to Information Systems*, pp. 53–66, Valencia, Spain, 2013.

[10] M. Bebah, C. Amine, M. Azzeddine, and L. Abdelhak, "Hybrid approaches for automatic vowelization of Arabic texts," *International Journal on Natural Language Computing*, vol. 3, no. 4, pp. 53–71, 2014.

[11] S. Alansary, "Alserag: an automatic diacritization system for Arabic," in *Intelligent Natural Language Processing: Trends and Applications (Studies in Computational Intelligence, 740)*, K. Shaalan, A. E. Hassanien, and F. Tolba, Eds., 1st edition, pp. 523–543, 2018.

[12] M. Rashwan, A. Al Sallab, H. Raafat, and A. Rafea, "Automatic Arabic diacritics restoration based on deep nets," in *Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing*, pp. 523–543, Doha, Qatar, 2014.

[13] Y. Belinkov and J. Glass, "Arabic diacritization with recurrent neural networks," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 2281–2285, Lisbon, Portugal, 2015.

[14] 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*, vol. 18, no. 2, pp. 183–197, 2015.

[15] A. S. Metwally and M. A. Rashwan, "A multi-layered approach for Arabic text diacritization," in *Proceedings of the 2016 IEEE International Conference on Cloud Computing and Big Data Analysis*, pp. 389–393, Chengdu, China, 2016.

[16] A. Chennoufi and A. Mazroui, "Morphological, syntactic and diacritics rules for automatic diacritization of Arabic sentences," *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 2, pp. 156–163, 2017.

[17] A. Fashwan and S. Alansary, "SHAKKIL: an automatic diacritization system for modern standard Arabic texts," in *Proceedings of the Third Arabic Natural Language Processing Workshop*, pp. 84–93, Valencia, Spain, 2017.

[18] K. Darwish, H. Mubarak, and A. Abdelali, "Arabic diacritization: stats, rules, and hacks," in *Proceedings of the Third Arabic Natural Language Processing Workshop*, pp. 9–17, Valencia, Spain, 2017.

[19] S. Alqudah, G. Abandah, and A. Arabiyat, "Investigating hybrid approaches for Arabic text diacritization with recurrent neural networks," in *Proccedings of the 2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies*, pp. 1–6, Aqaba, Jordan, 2017.

[20] I. Hadjir, M. Abbache, and F. Z. Belkredim, "An approach for Arabic diacritization," *Natural Language Processing and Information Systems*, vol. 11608, pp. 337–344, 2019.

[21] M. H. Ameur, Y. Moulahoum, and A. Guessoum, "Restoration of Arabic diacritics using a multilevel statistical model," in *Proccedings of the 5th International Conference on Computer Computer Science and its Applications*, pp. 181–192, Kyiv, Ukarine, 2015.

[22] H. Abbad and S. Xiong, "Multi-components system for automatic Arabic diacritization," *Lecture Notes in Computer Science*, vol. 12035, pp. 341–355, 2020.