JOURNAL OF QURANIC SCIENCES AND RESEARCH
e-ISSN: 2773-5532

JQSR

Vol. 5 No. 1 (2024) 12-29
https://publisher.uthm.edu.my/ojs/index.php/jqsr

Auto-Diacritization and Stylistic Realization of Arabic Text using Deep Neural Networks

Adel Sabour[1], Abdeltawab Hendawi[2], and Mohamed Ali[1]

[1] *Computer Science and Systems, University of Washington, Tacoma, USA*
[2] *Computer Science and Statistics, University of Rhode Island, Rhode Island, USA*

*Corresponding Author: adelsabour@gmail.com
DOI: https://doi.org/10.30880/jqsr.2024.05.01.002

Article Info

Received: 25th May 2024
Accepted: 12th June 2024
Available online: 30th June 2024

Keywords

Arabic diacritization, quranic linguistic styles, NLP, LSTM, GRU, transformer-based models, arabic dialect, textual data processing, Quran

Abstract

This paper presents QRDiaRec, an advanced diacritization system for Arabic Quranic texts. In Arabic, linguistic style refers to the variations in diacritic markings used to convey different pronunciations, dialects, meanings, and contextual understandings. QRDiaRec addresses the challenge of interpreting Arabic diacritics across multiple linguistic styles, which is crucial for accurate language processing. Unlike traditional systems that generate only one correct form of diacritics, QRDiaRec can recognize and produce multiple valid diacritic forms. This capability is due to its training on a dataset that encompasses seven Quranic linguistic styles. The Qur'an is an ideal case study because it is one text with multiple linguistic patterns, allowing us to recognize different forms of correct diacritization. QRDiaRec employs bidirectional LSTM, GRU, and transformer-based models to convert non-diacritic texts into annotated formats, achieving up to 94.2% accuracy. The system enhances Arabic language processing, impacting NLP, machine translation, and Arabic linguistics.

1. Introduction

In Arabic, diacritics are essential for determining pronunciation and meaning Shaalan *at al.* (2019). Without understanding diacritical letters, it is challenging to grasp word meanings, sentences, and context. We aim to develop an auto-diacritical solution for Arabic text called Automatic Diacritization and Dialect System for Arabic Quran Text Recognition (*QRDiaRec*). This solution serves as a fundamental component in enhancing the comprehension of the Arabic language, specifically in understanding diacritical letters and recognizing various Arabic dialects. Our QRDiaRec system is one of many development works related to the Qur'an featured on our website (Sabour and Ali, 2023). The website was constructed through the comprehensive efforts of Siddiqui et al. (2022). There are various Arabic dialects (Benaissa , M, 2021), including those with linguistic errors and limited fluency in Arabic, as well as correct dialects with well-known origins Farghaly and Shaalan (2009). These correct dialects are known as linguistic styles. The Arabic linguistic style encompasses various aspects of grammar, rhetoric, and pronunciation. In this context, the Qur'an is an Arabic diacritic-annotated text with 20 authentic linguistic styles Najeeb *at al.* (2015) referred to as "Riwayah" or "Qira'ah". There are 20 linguistic styles, but only 7 of them have been digitized. We use the dataset from Sabour *at al.* (2024), which includes seven linguistic styles.

Initially, Arabic writing consisted of 19 letters without dots or diacritics Sabour *at al.* (2023). The Arabic word can be written with the same letters more than once, and each word can have a different meaning or even antonym. What distinguishes between the different meanings of words that share the same letters are the diacritics (See figure 1). Diacritics are responsible for conveying subtle differences in meaning along with

grammatical rules Najeeb *et al.* (2015). Without understanding diacritics, resolving ambiguity, resolving ambiguity in Arabic text becomes extremely challenging, leading to misunderstandings and misinterpretations. For example, the word كتب written at the second level can have multiple meanings. With diacritic marks, كَتَبَ means 'He wrote', كُتِبَ means 'A text has been written', and كُتُبٌ means 'books'.
In Figure 1, it is evident that the Arabic text was misunderstood when processed by Google Translator and Bing Translator, highlighting the potential consequences.

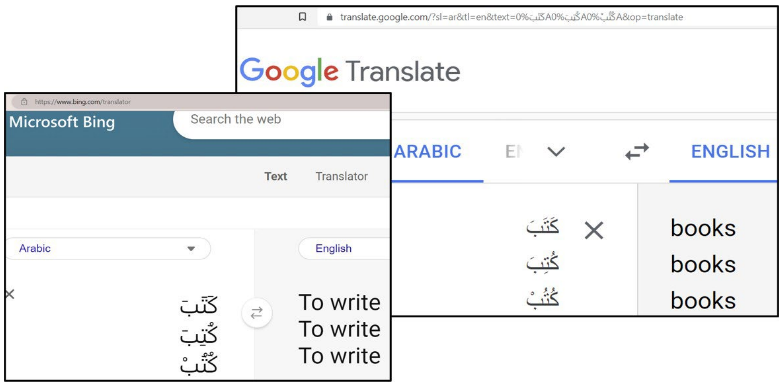


Fig. 1 *Impact of diacritics on arabic text understanding: comparison between Bing translator and Google translator*

As a case study, the text of the Qur'an is chosen due to its representation of the highest level of the Arabic language Sabour *at al.* (2024) and its inclusion of various linguistic styles. The linguistic style encompasses any linguistically correct dialect, can be traced back to its origin, is associated with known proficient speakers, and can be represented in written form Sabour *at al.* (2024). The term "linguistic style" in Islamic studies is referred to as "*Riwayah*" or "*Qira'ah*". In the provided table 1, the Arabic word هيت is examined across different linguistic styles. Illustrates the variation in the representation and pronunciation of the word across different Arabic linguistic styles. The first column lists the names of the linguistic styles. The second column shows how the word is written in each style. The third column details the pronunciation in each respective style. Although the basic shape of the word remains consistent, variations in dots and diacritics alter its pronunciation. This can either provide additional semantic information or reflect a regional dialect's pronunciation. The system's input consists of words without diacritics. The expected output is the same words, but with diacritics added. The system learns to add diacritics based on the patterns it finds in the linguistic styles. This learning process enables the system to generate words with diacritics from input words without diacritics. The table illustrates that while some styles share the same form, others display notable differences.

Table 1 *Example of the orthographic and phonetic diversity for one word across different Arabic linguistic styles*

Linguistic Styles	Word	Pronunciation
Nafie, AboJaafr, Ibn-Thakwan	هِيْتَ	hey-ta
Ibn-Kathir	هَيْتُ	hay-tu
Hisham	هِئْتَ	h'-eta
The other styles	هَيْتَ	hay-ta

QRDiaRec encounters each sentence in various forms, representing different linguistic styles. This exposure allows the system to grasp and apply a wider range of possible diacritics to Arabic sentences. A key difference from prior systems is their single "correct" output for each sentence. QRDiaRec acknowledges the existence of multiple valid diacritic forms due to the diverse linguistic styles it has learned. As a result, a single input sentence can have several correct outputs with diacritics. The automatic diacritization is challenging due to the presence of multiple valid diacritization forms for the same word. This paper investigates this challenge by employing three neural network models: a bidirectional LSTM, a bidirectional GRU, and a transformer-based model Gupta and Agrawal (2022). We compare the performance of these models to identify the most effective approach for Arabic text diacritization through the different linguistic styles.

This paper is organized as follows. We first review existing research in automatic diacritization (Section 2). Then, Section 3 explores the integration of linguistic styles and the diacritization dataset. Section 4 details the methods used for automated Arabic diacritization. In Section 5, we evaluate the performance of the diacritization algorithms. Section 6 compares various performance metrics and analyzes accuracy trends. Section 7 provides a discussion of the findings and outlines future research directions. Finally, Section 8 provides the conclusion.

2. Related Work

Starting with an in-depth survey of Arabic Natural Language Processing (ANLP) by Guellil et al. (2021), the authors delve into the research landscape in this field. They highlight the complexity of the Arabic language and the challenges it presents for NLP tasks. The authors' analysis of 90 research papers reveals a predominant focus on Arabic dialects, particularly Modern Standard Arabic (MSA) and Dialectal Arabic (DA).
One notable finding of the study is the need for further research on Classical Arabic (CA), which has received comparatively less attention. Building upon this insight, our research endeavors to make contributions to the advancement of CA. Recognizing the authors' emphasis on the importance of resource development in ANLP, our work aims to address this need. We accomplish this by building various systems to support Arabic NLP. Their results emphasize the significance of resource construction in various tasks, including semantic analysis, speech recognition, and text processing. In alignment with their findings, our research is dedicated to advancing these aspects within the field of Arabic NLP.

The research by Abandah and Abdel-Karim (2020) focuses on developing a fast and accurate machine-learning solution for the automatic diacritization of Arabic text. They employ long short-term memory (LSTM) recurrent neural networks to predict diacritics in Arabic text. The research recommends a solution using four bidirectional LSTM layers, achieving diacritization error rates. Specifically, they report a diacritization error rate of 2.46% on the LDC ATB3 dataset and 1.97% on the larger Tashkeela dataset, showcasing a 47% improvement over the best-published previous result.

The research conducted by Fadel et al. (2019) addresses the task of diacritization in Arabic text and emphasizes the scarcity of open-source resources for this problem. They utilize a dataset comprising 55K lines, consisting of approximately 2.3M words, obtained from the Tashkeela Corpus and a simplified version of the Holy Quran. The experimental results demonstrate the superiority of the neural approach, specifically the Shakkala system, over other methods in terms of diacritic error rate (DER) and word error rate (WER). The neural Shakkala system achieves a DER of 2.88%, surpassing the best DER obtained by non-neural approaches, which stands at 13.78%.

The research conducted by Belinkov and Glass (2015) focused on diacritization of Arabic text using long short-term memory (LSTM) layers. They compared their models to simple feed-forward networks and found that LSTM models outperformed them, particularly when using bidirectional LSTM (B-LSTM) and deeper models. Their best model achieved a diacritic error rate (DER) of 5.39% on all diacritics and 8.74% on case endings on a separate test set. Importantly, their results surpassed previous models that relied on segmenters and part-of-speech taggers, demonstrating the effectiveness of their model in diacritizing Arabic text without relying on additional resources. It is worth noting that their study primarily focused on diacritization performance and the comparison to existing methods. The research did not explicitly report accuracy as a performance metric. However, the DER values provided can be considered analogous to accuracy, as they represent the percentage of correctly predicted diacritics.

Elshafei et al. (2006) addressed the diacritization of Arabic text using statistical methods based on language modeling. The researchers employed a hidden Markov Model framework, treating the un-diacritized word sequence as an observation sequence and the diacritized word expressions as hidden states. The Viterbi Algorithm was used to obtain the optimal sequence of diacritized words. The study achieved a 4.1% letter error rate using the basic hidden Markov Model approach. Incorporating a preprocessing stage and utilizing trigrams for selected short and frequent words reduced the letter error rate to around 2.5%. The algorithm assumed that all words in the un-diacritized sequence existed in the provided vocabulary list, emphasizing the need for statistical methods to generate fully diacritized words based on letter sequences. Additionally, the algorithm's success rate for restoring diacritical marks based on letter statistics was less than 72%.

Comparative analysis with existing systems is challenging for the QRDiaRec system. This is due to the unique focus of our model on the linguistic styles. Traditional metrics like Diacritization Error Rate (DER) and Word Error Rate (WER) are not applicable. These metrics assume a single correct form for diacritics, which does not hold in our study. Our model acknowledges these variations and can produce multiple correct diacritic outputs for the same sentence. Therefore, applying DER and WER, which penalize these valid alternatives, wouldn't be an accurate measure of QRDiaRec's effectiveness. We have evaluated QRDiaRec's performance using standard accuracy measures, which provide a clearer picture in this context. Despite the challenges of multiple valid forms, our models achieved accuracy ranging from 91.5% to 92.2%. Our approach employs character-level sequence-to-sequence models. These models include bidirectional LSTM, bidirectional GRU, and transformer-based neural networks. Our research goes beyond just diacritization by delving into the different Quranic linguistic styles, an aspect not addressed in prior studies.

3. Integrating Linguistic Styles and Diacritization Dataset

This section explores data organization, focusing on two key aspects: modeling linguistic styles in the database and the training and evaluation dataset for automated Arabic diacritization. The database, created in our previous research Sabour *at al.* (2024), organizes Quran Arabic linguistic styles at multiple levels, including: chapters, verses, words, letters, and diacritics. The dataset is derived from this database, prepared for various learning models, and divided into training and evaluation subsets to improve diacritization accuracy.

3.1 Modeling Linguistic Styles in the Database

We present a glimpse into part of our database that illustrates the relationship and organization of textual data of the Arabic Qur'an. The database includes information on *linguistic styles* and the Qur'an books known as *Mushafs*. These *Mushafs* are divided into *chapters, verses, words,* and more. The tables within the database provide both statistical data and basic information, enabling various text-level analyses.

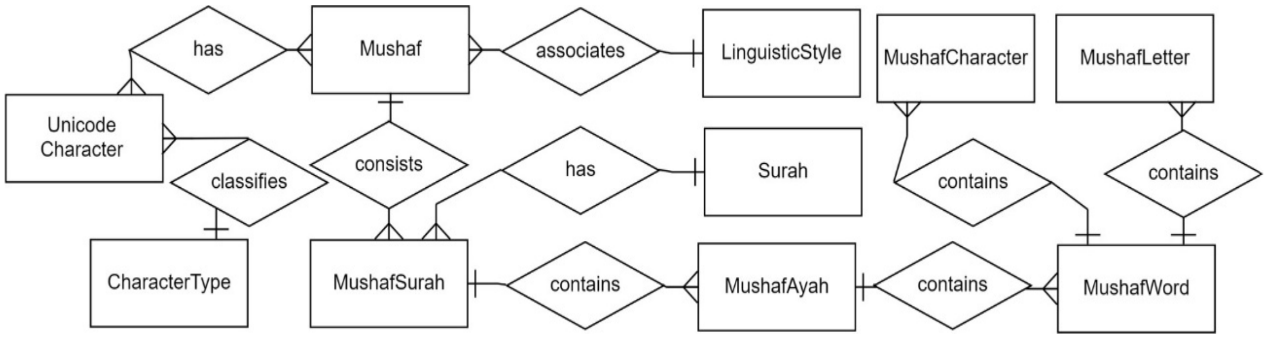


Fig. 2 *Database schema for modeling linguistic styles in the Qur'an*

Figure 2 displays the tables utilized in this section, encompassing the following elements:

- *Linguistic styles* are associated with multiple *Mushafs*. Each *Mushaf* represents a complete book of the Qur'an presented in a specific *linguistic style*.
- Each *Mushaf* consists of a collection of chapters, referred to as *Surahs*. A complete Quran comprises 114 chapters. The *MushafSurah* table records data for the processed *Surahs* in each *Mushaf*.
- Each *Surah* is further divided into multiple verses, known as *Ayahs*.
- Each *Ayah* is subdivided into *words*, and each *word* is composed of *characters*.
- The *Characters* table contains information about each character used in the script, along with its position.
- The classification of characters includes *Letters*, *Diacritics*, and *OtherCharacters*, representing special char- acters. The Characters include the corresponding Unicode representation used in each *Mushaf*, which may vary for the same *character* from *Mushaf* to another.

This structure enables analysis of the Arabic Qur'an's textual content, facilitating the creation of a dataset for training and evaluation capable of supporting the Automated Arabic Diacritization System

3.2 Training and Evaluation Dataset for Automated Arabic Diacritization: Multistyle Dialects Approach

The training and evaluation dataset we used for the diacritization task contains seven linguistic styles. Table 2 provides an overview of the dataset used in the Arabic auto-diacritization system, including statistics on the number of linguistic styles, tokens/words, characters without diacritics, diacritics, and characters in diacritics.

Table 3, represents statistical information about the training and evaluation dataset. Each style represents the same text but in different Arabic dialects specifically different linguistic styles. Each record in the dataset contains a verse without diacritics and the same verse with diacritical letters. A verse may contain more than one sentence. This pairing facilitates the training and evaluation of diacritization models.

Table 2 *Dataset statistics*

Linguistic Style Variations	7

Tokens/Words Count	542,020
Undiacritized Character Count	2,283,177
Diacritic Mark Count	2,203,148
Total Character Count	4,486,325

We divided the Quranic data into sections for training, testing, and evaluation purposes. Furthermore, we ensured that these same sections were distributed across various linguistic styles. This ensures that the model is thoroughly assessed on unseen data during evaluation, providing a realistic gauge of its generalization capabilities. Our division of the Quranic data into sections per a linguistic style then taking the same sections for each linguistic style serves as a safeguard against over-fitting enhances generalization, and ensures a comprehensive evaluation of the model's performance.

Table 3 *Dataset statistics: multistyle dialects for automated arabic diacritization*

StyleID	Style	Count	Percentage	Training	Validation	Testing
1	Shueba	6,236	14.32	4,488	1,248	500
2	Hafs	6,236	14.32	4,488	1,248	500
3	Qonbl	6,220	14.28	4,476	1,244	500
4	Al-Bazi	6,220	14.28	4,476	1,244	500
5	Warsh	6,214	14.27	4,471	1,243	500
6	AlDawri	6,217	14.27	4,473	1,244	500
7	Qalun	6,214	14.27	4,471	1,243	500
Totals	7	43,557	100	31,343	8,714	3,500

Table 3 represents the division of the Arabic multi-style dialects across our datasets. The column *Style:* The name of the Arabic dialect or linguistic style represented by the text in each book. The column *Count:* The total number of sentences in the dataset for each book or dialect. The column *Percentage:* The percentage of sentences represented by each book or dialect in the entire dataset. The percentages are calculated based on the total number of sentences (43,557 sentences). The column *Training:* The number of sentences from each book or dialect reserved for training purposes. The column *Evaluation:* The number of sentences from each book used for evaluating the diacritization model. The column *Testing:* The number of sentences from each book or dialect used for testing the diacritization model. These sentences are employed to assess the model's performance. The column *Totals:* Represents the aggregate statistics for the entire dataset. The total number of sentences in the dataset is 43,557 for non-diacritic sentences and the same number for the diacritic sentences. The total count of sentences used for training is 31,343, while the total count of sentences used for validation is 8,714, and for testing is 3,500. These values demonstrate how the dataset is split into training, validation, and testing subsets for the diacritization task. This diversity in the dataset allows the models to handle the different dialectal styles, enhancing the overall performance and usability of the automated Arabic diacritization system.

4. Automated Arabic Diacritization Module

Automatic diacritization opens the door to a world of possibilities in Arabic language processing. We are interested in diacritics as a means to comprehend the Arabic language. Removing diacritic letters from diacritic-annotated text is a straightforward task, where all Unicode diacritics are deleted from the text. However, the real challenge lies in automatically adding diacritics to text when they are absent. This process significantly enhances language understanding and clarification of meaning. Diacritics provide phonetic and grammatical information that is essential for disambiguating the text. By introducing diacritics automatically, we improve the processing of Arabic NLP, which yields benefits across various fields. This advancement positively impacts sentiment analysis, machine translation, information retrieval, text-to-speech (TTS) systems, consistency, and standardization in texts, and others.

This module is essential for automating the conversion of non-diacritic Arabic text into diacritic-annotated text, enabling interpretation and understanding of Arabic, enhancing linguistic analysis, and facilitating various language processing applications. The character-level sequence-to-sequence model is chosen because it operates directly on the character level, allowing it to capture fine-grained linguistic features and patterns (Zhang et al., 2016). It can learn the complex relationships between the input non-diacritic Arabic text and the corresponding diacritic-annotated text, which is the objective of the diacritization system. The model has the ability to generate sequences of variable length (Ruzsics and Samardzic, 2017), making it suitable for handling the varying lengths of Arabic words and sentences. By utilizing bidirectional LSTM or GRU layers, the model can effectively capture both past and future context (Yang et al., 2020), enabling it to make informed decisions about appropriate diacritics based on the surrounding characters.

LSTM and GRU are different types of recurrent neural network (RNN) architectures used for sequence modeling (Ruzsics and Samardzic, 2017). Both bidirectional LSTM and bidirectional GRU incorporate information from both past and future contexts by processing the input sequence in both forward and backward directions (Zhang et al., 2016). The main difference lies in their internal mechanisms. LSTM utilizes memory cells, input gates, forget gates, and output gates to control the flow of information, whereas GRU uses update gates and reset gates to regulate the flow of information (Dey and Salem, 2017). LSTM has a more complex structure than GRU, with more parameters, which allows it to capture longer-term dependencies in the input sequence (Yang et al., 2020). GRU has a simpler structure with fewer parameters, making it computationally more efficient and easier to train (Zhang et al., 2016).

Algorithm 1: Training a Character-Level Language Model

```
Input:
        Training dataset $D_{train}$ with input sequences $X$ and target sequences $y$
        Testing dataset $D_{test}$ with input sequences $X_{test}$ and target sequences $y_{test}$
        Maximum sequence length $L_{max}$
        Embedding dimension $E_{dim}$
        Number of training epochs $N_{epochs}$
        Batch size $B_{size}$
Output:
        Trained character-level language model $M$
  1:  function PreprocessData
  2:      Convert sentences in $D_{train}$ and $D_{test}$ into character lists
  3:      Convert character lists into integer representations
  4:      Pad the sequences to ensure the uniform length
  5:  function InitializeModel
  6:      // The implementation of this function varies depending on the model used.
  7:  function CompileAndTrainModel
  8:      Compile the model $M$ using the sparse categorical cross-entropy loss function and the Adam optimizer
  9:      Train the model $M$ on the preprocessed training data,
              iterating over $N_{epochs}$ epochs with a batch size of $B_{size}$
 10:      Evaluate the model's performance on the validation data
 11:      Save the trained model $M$ for future use

 12:  PreprocessData
 13:  InitializeModel
 14:  CompileAndTrainModel
```

The transformer-based model is employed for diacritizing Arabic text, offering a broader range of alternative approaches. This model leverages self-attention mechanisms to model long-range dependencies and capture contextual relationships within the text (Ding et al., 2022). Inspired by the influential "Attention Is All You Need" architecture (Vaswani, et al., 2017), our model excels at handling large-scale datasets and effectively captures global dependencies across the entire sequence (Vaswani et al., 2017). The character-level sequence-to-sequence model is effective in capturing sequential dependencies, while the transformer-based model excels at modeling long-range dependencies and capturing contextual relationships.

Algorithm 1 is a general algorithm for the three models (GRU, LSTM, and Transformer). It consists of three functions:

- **PreprocessData:** Preprocesses the training and testing data uniformly across all models.
- **InitializeModel:** Initializes the model, with implementation varying for each model due to their unique architecture and initialization process.
- **CompileAndTrainModel:** Compiles and trains the model, following the same process for all three models.

Our models, implemented using the Keras framework, are powerful deep-learning algorithms for capturing and predicting patterns in sequential data (Moolayil, 2019). The algorithmic steps for training the models are summarized in Algorithm 1. To prepare the training and testing data during PreprocessData function, the sentences are initially converted into character lists. This conversion allows the models to operate at the character level, capturing the fine-grained details of the text.

By representing the text as characters, the models can learn the relationships between different characters and their corresponding diacritics. The character lists are further transformed into integer representations because neural networks typically work with numerical data. Assigning unique integers to each character enables the models to process and learn from the input data effectively. In order to ensure uniformity in the input data, padding is applied (Dwarampudi and Reddy, 2019). This involves adding zeros padding tokens to the sequences so that they all have the same length. Creating equal-length input sequences is crucial as neural networks typically require fixed-size inputs (Dwarampudi and Reddy, 2019). By padding the sequences, we ensure that the models can handle inputs of varying lengths without encountering issues.

During model compilation inside the CompileAndTrainModel function, the sparse categorical cross-entropy loss function is used (Cosentino et al., 2019). This loss function is suitable for multi-class classification tasks like diacritization, where each character can have multiple possible diacritics. The Adam optimizer is chosen for training the models, as it efficiently updates the model's parameters based on the gradients computed during the training process (Zhang, 2018).

Regarding the InitializeModel function, we can combine the idea of implementing LSTM and GRU into Algorithm 2. The model architecture for both the LSTM and GRU models is defined using the Sequential API, a straightforward way to build neural networks in a sequential manner (Manaswi, 2018). The architecture consists of multiple layers, including an embedding layer, which maps each character to a continuous vector representation. This allows the models to capture semantic relationships between characters based on their contextual usage. Next, a bidirectional layer (LSTM or GRU) is employed. The bidirectional nature of these layers enables the models to consider both past and future contexts when making predictions. This is beneficial for diacritization as it allows the models to take into account the surrounding characters when predicting the diacritics for a specific character. A dense layer with softmax activation (Luus et al., 2015) is utilized to produce the final output probabilities for each character's diacritics. This layer interprets the outputs as a probability distribution over diacritic options for each character.

Algorithm 3 provides a high-level overview of the InitializeModel function for the Transformer model. The Transformer model is constructed with an embedding layer followed by multiple TransformerEncoder layers (Wang et al., 2019). The embedding layer maps the integer representations to dense vectors of size E_{dim}, allowing the model to learn meaningful representations of the characters.

The TransformerEncoder layers leverage self-attention mechanisms to capture the dependencies between characters and generate contextualized representations (Ding et al., 2022). The output of the last TransformerEncoder layer is fed into a dense layer with softmax activation to obtain the predicted diacritics. We chose specific hyperparameters with the transformer:

Algorithm 2: Initialize the GRU and LSTM Models

```
Input:
      Number of LSTM/GRU units $U_{LSTM/GRU}$
1:  function InitializeModel
2:      Create an embedding layer with input dimension $L_{max}$ and output dimension $E_{dim}$
3:      Create a bidirectional LSTM/GRU layer with $U_{LSTM/GRU}$ units
4:      Create a dense layer with softmax activation
```

- d_model=32: It determines the size of the model's internal representation. A smaller value makes the model more efficient for processing Arabic text.
- num_heads=4: It represents the number of attention spots in the model's brain. More heads allow the model to capture different patterns in the text, essential for understanding diacritic placement.
- dff=64: It controls the capacity for processing and transformations in the model. A moderate value balances efficiency and learning capability.
- rate=0.1: It indicates the dropout rate, which helps in generalization and prevents the model from overfitting to specific examples.

These hyperparameters were chosen to make the model computationally efficient and effective in handling the complexities of Arabic text with diacritics.

References to algorithms 2 and 3 denote the instantiation of the "InitializeModel" function specific to GRU, LSTM, and Transformer models. These algorithms constitute essential components within the broader framework of the comprehensive algorithm 1. By incorporating these algorithms, the implementation process for the Automated Arabic Diacritization Module is accomplished, encompassing all requisite steps.

Algorithm 3: Initialize the Transformer Model

```
Input:
    Number of Transformer heads $N_{heads}$ Feed-forward dimension $F_{dim}$
    Dropout rate $D_{rate}$
1: function InitializeModel
2:         Create an embedding layer with input dimension $V_{size}$ and output dimension $E_{dim}$
3:         Create $N_{heads}$ TransformerEncoder layers with $E_{dim}$, $N_{heads}$, and $F_{dim}$
4:         Create a dense layer with softmax activation
```

These models have demonstrated high accuracy rates, as indicated by achieving accuracies of 91.5% with bidirectional LSTM, 92.1% with bidirectional GRU, and 94.2% with the transformer-based model, making them effective solutions for diacritizing Arabic letters.

5. Performance Evaluation of Diacritization Algorithms

In this section, we present the performance evaluation of diacritization algorithms, specifically on the LSTM, GRU, and Transformer models. These models are powerful deep-learning algorithms for capturing and predicting patterns in sequential data (Zhang et al., 2016).

5.1 Experimental Results of the LSTM Algorithm

This section presents an analysis of the LSTM algorithm's performance for diacritization. It is structured into three segments, each focusing on different aspects of the algorithm's performance evaluation. By delving into these analyses, we gain valuable insights into the LSTM algorithm's proficiency in accurately predicting and capturing patterns while minimizing errors.

5.1.1 Performance Analysis of LSTM Algorithm: Metrics and Results

The LSTM algorithm achieved a training accuracy of 94.3%, a validation accuracy of 93.5%, and a testing accuracy of 92.1%. The training loss, which measures the error during training, is 0.179, and the validation loss is 0.214.

Table 4 provides an evaluation of the performance of the LSTM model for diacritization. These results indicate that the LSTM algorithm performed well in capturing the patterns and relationships between characters and their corresponding diacritics, leading to accurate predictions.

Table 4 *Performance metrics of LSTM algorithm in diacritizing arabic text*

LSTM Metric	Accuracy	Precision	Recall	F1 Score	Training Loss	Training Accuracy	Val. Loss	Val. Accuracy
Value	92.14	90.55	92.14	91.29	0.179	94.3	0.214	93.5

5.1.2 Assessment of LSTM Model Generalization: Training vs. Validation Accuracies

Figure 3, illustrates the training and validation accuracies achieved during the evaluation of the LSTM model. The accuracies are computed over 75 epochs, with the following summarization:

- The training accuracy gradually improves from 91.3% to 94.3%, indicating the model's learning and performance enhancement on the training data.
- The validation accuracy displays a similar increasing trend, rising from 91.8% to 93.5%. This suggests the model's ability to generalize well to unseen data and make accurate predictions.
- The proximity of the training and validation accuracies suggests a balanced model that avoids overfitting or underfitting (Jabbar and Khan, 2015). It demonstrates the model's capability to generalize without merely memorizing the training examples.
- The validation accuracy consistently tracks the increasing trend of the training accuracy, indicating the model's improvement on both training and unseen data.
- The final accuracies of 94.3% for training and 93.5% for validation highlight the model's reasonably high accuracy in accomplishing the given task.

  These findings strongly support the hypothesis that the LSTM model effectively learns and generalizes for the specified task.

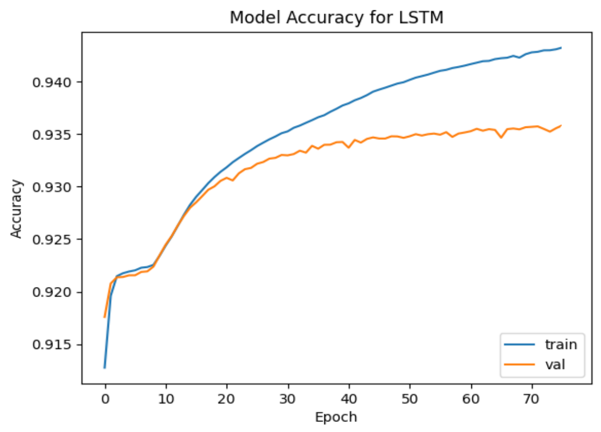


Fig. 3 *Accuracy comparison: training and validation results for the LSTM model*

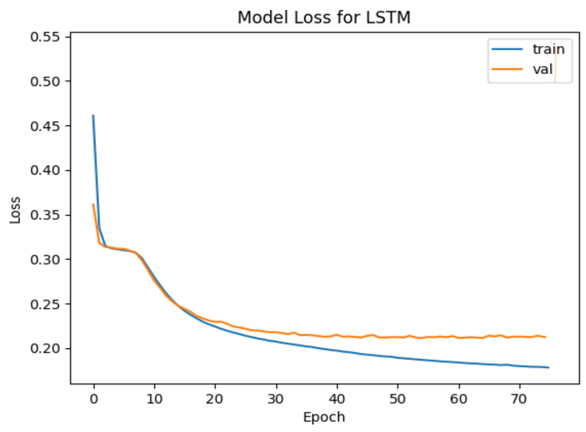


Fig. 4 *Loss comparison: training and validation results for the LSTM model*

5.1.3 Analyzing Loss in LSTM Model: Training vs. Validation Results

Figure 4, displays the training and validation losses obtained during the evaluation of the LSTM model. It is important to note that these loss values are not expressed as percentages but rather represent a measure of the model's performance and error reduction. These losses were computed over multiple epochs and can be summarized as follows:

- The training loss gradually decreases from 0.461 to 0.179. This decreasing trend indicates that the model improves its performance and reduces errors as it learns from the training data.
- Similarly, the validation loss also exhibits a decreasing trend, ranging from 0.361 to 0.214. This suggests that the model effectively minimizes errors and generalizes well to unseen data, indicating its ability to make accurate predictions on new instances.
- The close proximity of the training and validation losses indicates that the model achieves a balance between capturing patterns in the training data and generalizing to new instances. This suggests that the model neither overfits the training data nor under-fits the underlying patterns, resulting in reliable predictions.
- The consistent decrease in both training and validation losses indicates the model's ability to generalize learned patterns to unseen data.
- The final losses of 0.179 for training and 0.214 for validation demonstrate the model's effectiveness in achieving low loss levels, reflecting its capability to capture underlying patterns in the given task.

These findings provide evidence supporting the LSTM model's performance in minimizing loss and capturing patterns in both training and validation data.

5.2 Experimental Results of the GRU Algorithm

This section presents an analysis of the GRU algorithm's performance for diacritization. It is structured into three subsections, each focusing on different aspects of the algorithm's performance evaluation. These subsections employ various metrics to assess the algorithm's effectiveness in capturing patterns, its ability to generalize to unseen data, and the analysis of loss values during training and validation.

5.2.1 Performance Analysis of GRU Algorithm: Metrics and Results

The GRU algorithm achieved an accuracy of 94.2% a precision of 93.26%, a recall of 94.20%, and an F1 score of 93.46%. These metrics measure the algorithm's accuracy in predicting the diacritics for Arabic text in the testing dataset. Additionally, the training loss of the GRU model is 0.150, indicating the error during the training process, while the training accuracy is 95.2%, showing the percentage of correctly predicted diacritics on the training data. For the validation phase, the GRU algorithm achieved a validation loss of 0.171, which measures the error on unseen data, and a validation accuracy of 94.7%, indicating the accuracy of diacritic predictions on the validation data. Table 5 provides an evaluation of the performance of the GRU model for diacritization. The GRU algorithm performed well in capturing the patterns and relationships between characters and their corresponding diacritics, leading to accurate diacritization of Arabic text.

Table 5 *Performance metrics of GRU algorithm in diacritizing arabic text*

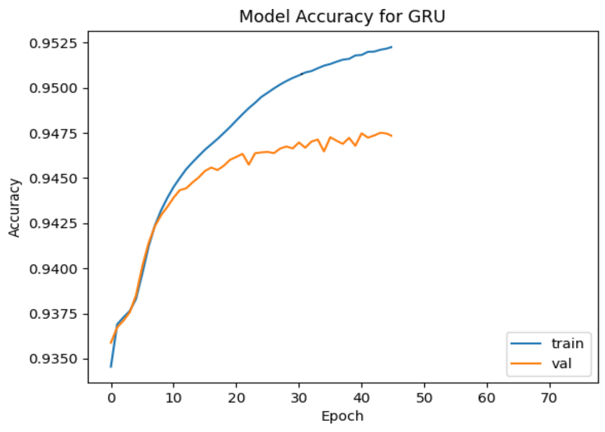
GRU Metric	Accuracy	Precision	Recall	F1 Score	Training Loss	Training Accuracy	Val. Loss	Val. Accuracy
Value	94.20	93.26	94.20	93.46	0.150	95.20	0.171	94.7

5.2.2 Assessment of GRU Model Generalization: Training vs. Validation Accuracies

Figure 5, illustrates the training and validation accuracies achieved during the evaluation of the GRU model. After epoch 47, the GRU model shows a significant decrease in accuracy. We applied an early stop until epoch 47 to overcome the overfitting (Rice et al., 2020).

- The training accuracy gradually improves from 93.5% to 95.2%, indicating the model's learning and performance enhancement on the training data.
- The validation accuracy also shows an increasing trend, rising from 93.6% to 94.7%. This suggests the model's ability to generalize well to unseen data and make accurate predictions.
- The proximity of the training and validation accuracies indicates a balanced model that avoids overfitting or underfitting. It demonstrates the model's capability to generalize without merely memorizing the training examples.
- The validation accuracy consistently tracks the increasing trend of the training accuracy, indicating the model's improvement on both training and unseen data.
- The final accuracies of 95.2% for training and 94.7% for validation highlight the model's high accuracy in accomplishing the given task.

These findings provide evidence that the GRU model effectively learns and generalizes for the specified task, similar to the LSTM model.



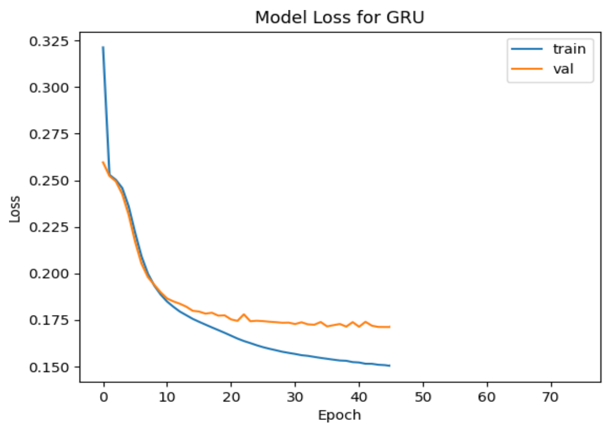


Fig. 5 *Accuracy comparison: training and validation results for the GRU model*

Fig. 6 *Loss comparison: training and validation results for the GRU model*

5.2.3 Analyzing Loss in GRU Model: Training vs. Validation Results

Figure 6, presents the training and validation losses obtained during the evaluation of the GRU model. These losses are computed over 75 epochs, but we applied an early stop at epoch 47 and we can be summarized as follows:

- The training loss gradually decreases from 0.321 to 0.150, indicating improved performance and error reduction in the training data.
- The validation loss also exhibits a decreasing trend, ranging from 0.260 to 0.171, suggesting effective error minimization and generalization to unseen data.
- The close proximity of the training and validation losses suggests a balanced model that avoids overfitting or underfitting, effectively capturing patterns in the training data while generalizing to new instances.
- The consistent decrease in both training and validation losses indicates the model's ability to generalize learned patterns to unseen data.

- The final losses of 0.150 for training and 0.171 for validation demonstrate the model's effectiveness in achieving low loss levels, reflecting its capability to capture underlying patterns in the given task.

These findings provide evidence supporting the GRU model's performance in minimizing loss and capturing patterns in both training and validation data.

5.3 Experimental Results of the Transformer Algorithm

The Transformer model is a state-of-the-art architecture that has revolutionized NLP tasks, including language modeling (Irie et al., 2019). This section presents the outcomes of the experiments conducted using the Transformer model and examines its effectiveness in capturing patterns and making accurate predictions.

5.3.1 Performance Analysis of the Transformer Algorithm: Metrics and Results

Table 6 *Performance metrics of the transformer algorithm in diacritizing arabic text*

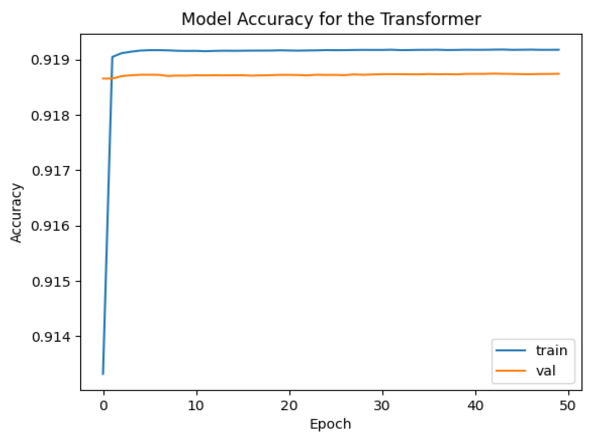
Transformer Metric	Accuracy	Precision	Recall	F1 Score	Training Loss	Training Accuracy	Val. Loss	Val. Accuracy
Value	92.10	89.17	92.10	89.84	0.447	91.9	0.448	91.9

The experimental results of the Transformer algorithm for diacritizing Arabic text are summarized in Table 6. The Transformer algorithm achieved an accuracy of 92.10% a precision of 89.17%, a recall value of 92.10%, and an F1 score, which is 89.84% during the testing phase. In terms of training performance, the algorithm achieved a training loss of 0.447. The training accuracy reached 91.9%. During the validation phase, the algorithm achieved a validation loss of 0.448, and the validation accuracy is 91.9%.

5.3.2 Assessment of Transformer Model Generalization: Training vs. Validation Accuracies

Figure 7, visualizes the training and validation accuracies achieved during the evaluation of the Transformer model. These accuracies are computed over multiple 75 epochs and can be summarized as follows:

- The training accuracy gradually improves from 91.5% to 91.9%, indicating the model's learning and performance enhancement on the training data.
- The validation accuracy shows a relatively stable trend, 91.9%. This suggests that the model maintains a consistent level of accuracy on unseen data.
- The proximity of the training and validation accuracies suggests a balanced model that avoids overfitting or underfitting. It indicates that the model generalizes well to unseen data without relying heavily on the training examples.
- The validation accuracy remains relatively stable throughout the epochs, indicating that the model's performance on unseen data remains consistent.
- The final accuracies of 91.9% for training and validation highlight the model's reasonably high accuracy in accomplishing the given task.
-



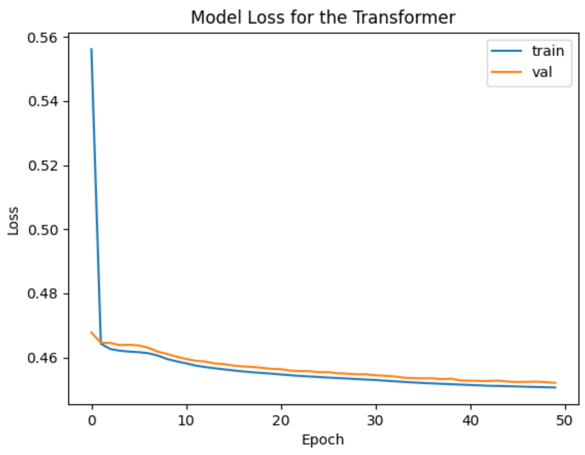


Fig. 7: *Accuracy comparison: training and validation results for the transformer model*

Fig. 8 *Loss comparison: training and validation results for the transformer model*

5.3.3 Analyzing Loss in Transformer Model: Training vs. Validation Results

Figure 8, showcases the training and validation losses obtained during the evaluation of the Transformer model. These losses are computed over 75 epochs and can be summarized as follows:

- The training loss gradually decreases from 0.506 to 0.447, indicating improved performance and error reduction in the training data.
- The validation loss also exhibits a decreasing trend, ranging from 0.465 to 0.448, suggesting effective error minimization and generalization to unseen data.
- The close proximity of the training and validation losses suggests a balanced model that avoids overfitting or underfitting, effectively capturing patterns in the training data while generalizing to new instances.
- The consistent decrease in both training and validation losses indicates the model's ability to generalize learned patterns to unseen data.
- The final losses of 0.447 for training and 0.448 for validation demonstrate the model's effectiveness in achieving low loss levels, reflecting its capability to capture underlying patterns in the given task.

6. Comparing Performance Metrics and Accuracy Trends

The main focus of this section is to compare the performance metrics, accuracy trends, and loss analysis of the Transformer, GRU, and LSTM models. Through the analysis, we aim to provide an understanding of the performance of these models and their implications for Arabic diacritization tasks.

6.1 Evaluating Performance Metrics

The comparison of performance metrics among the Transformer, GRU, and LSTM models is visualized in Figure 9. The GRU model achieved the highest accuracy of 94.2%, followed by the LSTM and the transformer model with an accuracy of 92.1%.

The GRU model achieved the highest precision score of 93.26%, indicating that it has a strong ability to classify the diacritics in the Arabic text. The Transformer model achieved a precision score of 89.17%, while the LSTM model achieved a precision score of 90.55%. The GRU model achieved a recall score of 94.2%, indicating its capability to effectively capture the true positive instances. The LSTM and Transformer models also demonstrated a high recall score of 92.1%, closely following the performance of the GRU model. The F1 score combines precision and recall into a single metric, providing an overall assessment of a model's performance. The GRU model achieved the highest F1 score of 93.46%, which indicates a good balance between precision and recall. The LSTM model closely followed with an F1 score of 91.3%, while the Transformer model obtained a slightly lower F1 score of 89.8%. Overall, the comparison of precision, recall, and F1 score suggests that the GRU model exhibits the highest performance among the three models, closely followed by the LSTM model. The Transformer model, while achieving slightly lower scores, still demonstrates competitive performance.

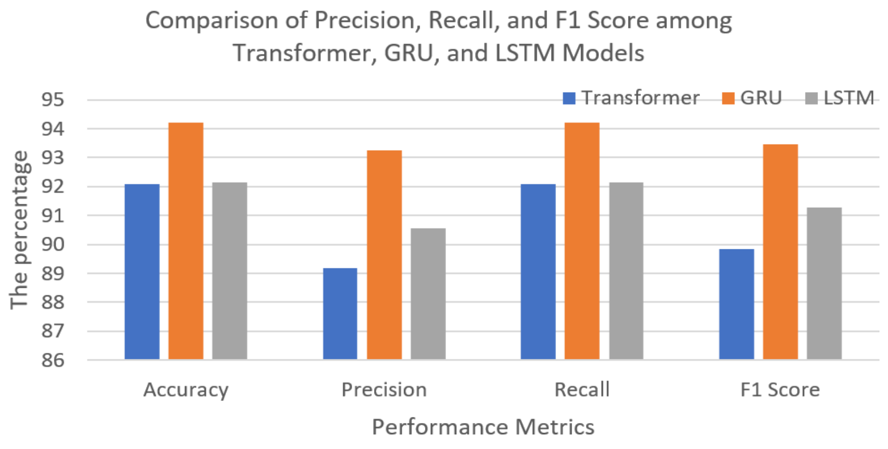


Fig. 9 *Comparison of precision, recall, and F1 score among transformer, GRU, and LSTM models*

6.2 Analysis of Training Accuracy Trends

Figure 10 compares the Training Accuracy of the Transformer, GRU, and LSTM models over 75 epochs. The key observations:

- The GRU is stopped early during epoch 47. The reason is that this is the epoch in which the GRU achieves the highest accuracy. Based on that, we assumed the stability of the GRU at this epoch. The GRU model

consistently shows the highest Training Accuracy throughout 47 epochs, with values ranging from 93.5% to 95.2%. It demonstrates strong learning capabilities and performance on the training data.
- The LSTM model follows closely behind the GRU model, with Training Accuracy ranging from 91.3% to 94.3%. It also exhibits consistent and steady improvement over the epochs.
- The Transformer model has the lowest Training Accuracy among the three models, ranging from 91.5% to 91.9%. Although it starts with slightly lower accuracy, it shows improvement and reaches a similar level as the LSTM model towards the end of the epochs.

The GRU model consistently achieves the highest Training Accuracy, indicating its superior performance in learning and capturing patterns in the training data. The LSTM model closely follows behind, demonstrating its effectiveness in learning and generalizing. The Transformer model shows improvement over the epochs but initially starts with a lower accuracy compared to the other two models.

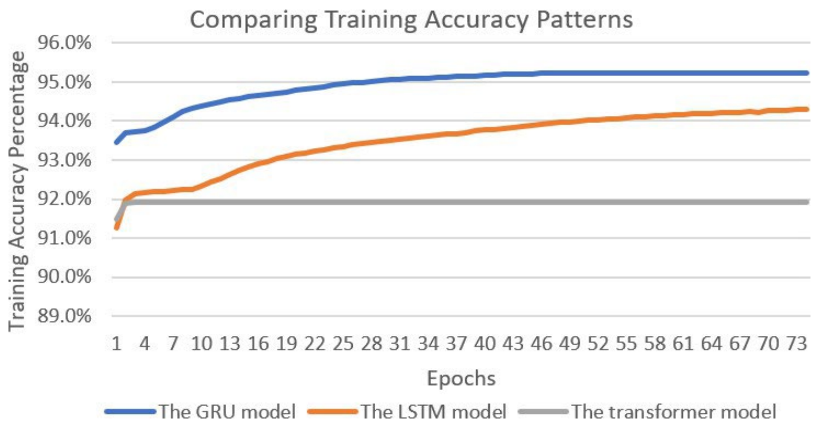


Fig. 10 *Comparing training accuracy patterns: transformer, GRU, and LSTM models*

6.3 Analysis of Evaluation Accuracy Trends

Figure 11 compares the Evaluation Accuracy of the models. The key observations are as follows:
- The GRU model consistently achieves the highest Evaluation Accuracy among the three models. It shows a gradual increase in accuracy from 93.6% to 94.7% over the 47 epochs. This indicates the GRU model's ability to generalize well to unseen data and make accurate predictions.
- The LSTM model closely follows the GRU model in terms of Evaluation Accuracy. It starts with an accuracy of 91.8% and shows consistent improvement throughout the epochs, reaching a final accuracy of 93.5%. This demonstrates the LSTM model's effectiveness in capturing patterns and generalizing for new instances.
- The Transformer model starts with a slightly lower Evaluation Accuracy compared to the other two models, at 91.9%.

The GRU model consistently outperforms the other models in terms of Evaluation Accuracy, indicating its superior performance in generalization.

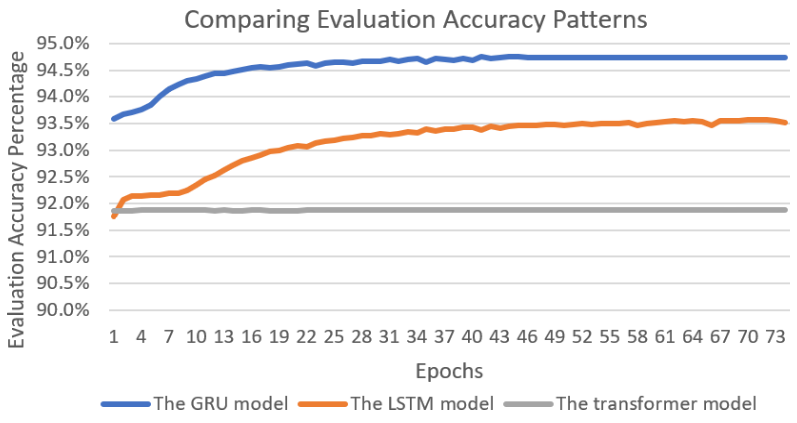


Fig. 11 *Comparing evaluation accuracy patterns: transformer, GRU, and LSTM models*

6.4 Analysis of Training Loss

We analyzed the training loss of the GRU, Transformer, and LSTM models over 75 epochs. The training loss reflects the discrepancy between the predicted diacritics and the actual diacritics in the training data. Lower training loss values indicate a better fit of the models to the data.

Figure 12 illustrates the training loss patterns of the three models:

- The GRU model exhibits the lowest training loss values, starting at 0.321 and gradually decreasing to 0.150. This suggests that the GRU model effectively captures the diacritical patterns in the Arabic text and converges to a better fit with the training data.
- The LSTM model follows a similar trend, starting with a training loss of 0.461 and reaching 0.179 by the end of the epochs. It demonstrates consistent improvement and convergence, albeit with slightly higher loss values compared to the GRU model.
- The Transformer model shows the highest training loss among the three models, starting at 0.506 and decreasing to 0.447. However, it still exhibits a notable improvement over the epochs and converges to a comparable level with the LSTM model.

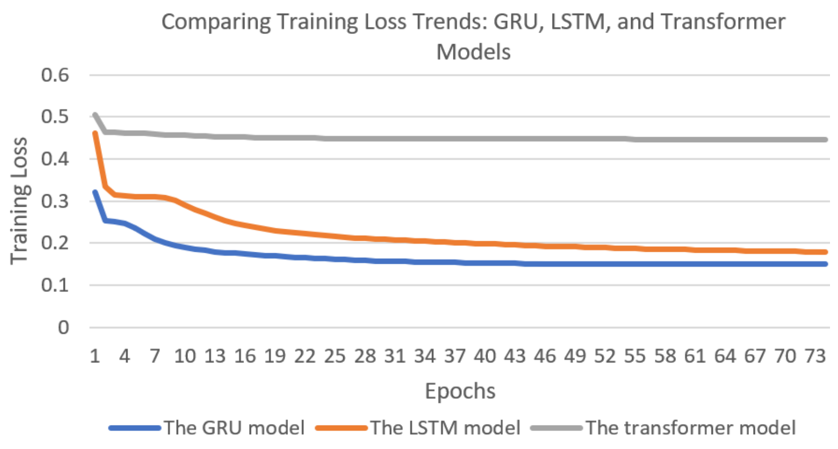


Fig. 12 *Comparing training loss patterns: transformer, GRU, and LSTM models*

The three models demonstrate a decreasing training loss over the course of training, indicating their ability to learn and capture the diacritical patterns in the training data. The GRU model achieves the lowest training loss, followed by the LSTM model, while the Transformer model lags slightly behind.

6.5 Analysis of Evaluation Loss

We conducted an analysis of the evaluation loss for the GRU, Transformer, and LSTM models over 75 epochs. The evaluation loss represents the discrepancy between the predicted diacritics and the actual diacritics in the evaluation data. Lower evaluation loss values indicate better generalization and performance on unseen data.

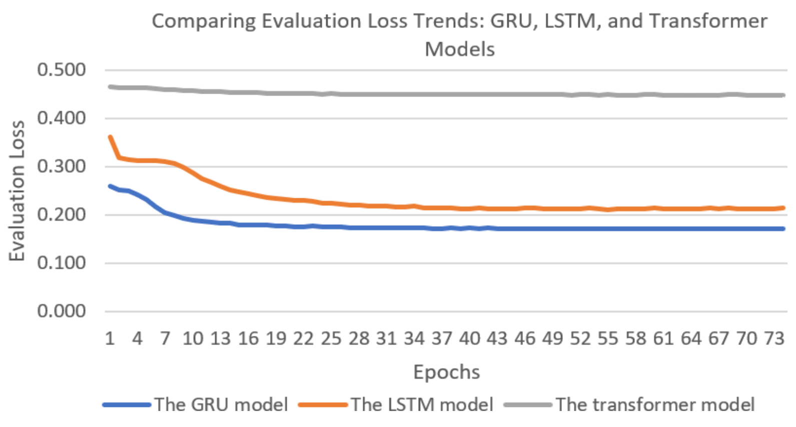


Fig. 12 *Comparing evaluation loss patterns: transformer, GRU, and LSTM models*

Figure 12 displays the evaluation loss patterns of the three models:

- The GRU model exhibits the lowest evaluation loss values, starting at 0.260 and gradually decreasing to 0.171. This suggests that the GRU model effectively generalizes to unseen data and performs well in diacritizing Arabic text.
- The LSTM model follows a similar decreasing trend, starting with an evaluation loss of 0.361 and reaching 0.214 by the end of the epochs. It demonstrates consistent improvement and showcases its ability to generalize to new instances.

The Transformer model starts with a higher evaluation loss compared to the other two models, at 0.448. The analysis of evaluation loss indicates that the GRU model consistently outperforms the other models, exhibiting the lowest evaluation loss values.

7. Discussion and Future Work

This section discusses the key findings of our research on automatic diacritization for Arabic text. We employed three neural network models and evaluated their performance. Additionally, we highlight the contributions of this study and explore potential areas for future work.

7.1 Results

We evaluated three neural network models for Arabic text diacritization: Transformer, GRU, and LSTM. Here's a breakdown of their performance:

- **Accuracy and Generalizability:**
    - The GRU model emerged as the champion, achieving the highest overall accuracy 94.2% and F1 score 93.46%.
    - This indicates its strong ability to both learn and perform well on unseen data.
    - LSTM followed closely with an accuracy of 92.1% and F1 score of 91.3%.
    - The Transformer model also showed competitive performance, reaching an accuracy of 92.1% but with a slightly lower F1 score 89.8%.
- **Training Efficiency:**
    - The GRU model impressed with the lowest training loss and the fastest convergence. This suggests it excels at capturing patterns within the training data.
    - All models exhibited a decrease in both training and evaluation loss as training progressed, indicating successful learning and improvement.

Based on our findings, the GRU model stands out as the most effective approach for Arabic text diacritization in this study. It achieved superior performance in accuracy, generalization, and training efficiency compared to the other models.

7.2 Future Work

Our study lays the groundwork for further exploration in several areas:

- **Use of Standard Metrics:** Research in auto diacritization typically uses metrics like Diacritization Error Rate (DER) and Word Error Rate (WER). These metrics assume a single correct form for each text instance. However, our research acknowledges multiple correct forms for diacritizing Arabic text due to various linguistic styles. Consequently, DER and WER may not fully capture the effectiveness of our models. Although we used standard metrics like accuracy, precision, and F1 score, future research should consider developing new metrics. These new metrics would better accommodate the variability inherent in linguistic styles.
- **Fine-tuning and Customization:** Implement techniques for fine-tuning and customizing the diacritization models to adapt to specific linguistic styles.
- **Linguistic Style Detection:** Develop a linguistic style detection module within the system to automatically identify and categorize the linguistic style of Arabic text.
- **Integration with Other NLP Tasks:** Integrate the diacritization system with other NLP tasks, such as sentiment analysis, named entity recognition, and part-of-speech tagging, to support more comprehensive Arabic language processing.
- **Cross-Linguistic Studies:** Conduct cross-linguistic studies to compare the diacritization and stylistic realization challenges in Arabic with other languages, potentially leading to valuable insights and improvements.

By addressing this future work, we can further refine diacritization systems for Arabic text. This refinement will enhance the effectiveness of these systems, particularly in handling diverse linguistic styles.

8. Conclusion

This study advances the understanding of Arabic through the QRDiaRec system. This system is focused on the auto-diacritization and stylistic realization of Arabic texts. Unlike traditional systems, QRDiaRec can produce multiple correct diacritic outputs for the same sentence. This ability significantly enhances comprehension of diacritical letters and diverse linguistic styles. The incorporation of deep learning models, including bidirectional LSTM, GRU, and transformer-based models, has enabled the QRDiaRec system to achieve a 94.2% accuracy in diacritization. This high accuracy underlines the system's effectiveness in Arabic text analysis and interpretation. The key contributions of this study are the development of the QRDiaRec system and the application of sophisticated deep learning models for diacritization. These advancements are poised to significantly impact the field of Arabic language processing, opening avenues for further research and practical applications.

Acknowledgement

The authors would like to thank Computer Science and Systems, University of Washington, for their invaluable assistance and support in this study.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Adel Sabour;* ***data collection:*** *Adel Sabour;* ***analysis and interpretation of results:*** *Adel Sabour, Abdeltawab Hendawi , and Mohamed Ali;* ***draft manuscript preparation:*** *Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. All authors reviewed the results and approved the final version of the manuscript.*

References

Adel Sabour and Mohamed Ali. Quran research, 2023. URL https://quranresearch.org/. Accessed: December 12, 2023.

Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. A Tapestry of Tongues: A Novel Provenance Approach for Arabic Linguistic Styles and Lineage Tracing. JQSR(Journal of Quranic Sciences and Research), 1, 2024.

Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. Diacritic-aware alignment and classification in arabic speech: A fusion of fuztpi and ml models. JISTech (Journal of Islamic Science and Technology), 8(2):169–191, 2023.

Ali Fadel, Ibraheem Tuffaha, Mahmoud Al-Ayyoub, et al. Arabic text diacritization using deep neural networks. In 2019 2nd international conference on computer applications & information security (ICCAIS), pages 1–7. IEEE, 2019.

Ali Farghaly and Khaled Shaalan. Arabic natural language processing: Challenges and solutions. ACM Transactions on Asian Language Information Processing (TALIP), 8(4):1–22, 2009.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

Benaissa , M. . . (2021). دراسة وتحليل دلالي في ألفاظ الأضداد في اللغة العربية / A semantic study and analysis of antonyms in Arabic Language. مجلة الدراسات اللغوية والأدبية (Journal of Linguistic and Literary Studies), 12(2), 98–111. Retrieved from https://journals.iium.edu.my/arabiclang/index.php/jlls/article/view/898

Francois PS Luus, Brian P Salmon, Frans Van den Bergh, and Bodhaswar Tikanath Jugpershad Maharaj. Multiview deep learning for land-use classification. IEEE Geoscience and Remote Sensing Letters, 12(12):2448–2452, 2015.

Gheith Abandah and Asma Abdel-Karim. Accurate and fast recurrent neural network solution for the automatic diacritization of arabic text. Jordanian Journal of Computers and Information Technology, 6(2), 2020.

H Jabbar and Rafiqul Zaman Khan. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). Computer Science, Communication and Instrumentation Devices, 70 (10.3850):978–981, 2015.

Haijun Zhang, Jingxuan Li, Yuzhu Ji, and Heng Yue. Understanding subtitles by character-level sequence-to-sequence learning. IEEE Transactions on Industrial Informatics, 13(2):616–624, 2016.

Imane Guellil, Houda Saˆadane, Faical Azouaou, Billel Gueni, and Damien Nouvel. Arabic natural language processing: An overview. Journal of King Saud University-Computer and Information Sciences, 33(5):497–507, 2021.

Jojo Moolayil and Jojo Moolayil. An introduction to deep learning and keras. Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python, pages 1–16, 2019.

Justin Cosentino, Federico Zaiter, Dan Pei, and Jun Zhu. The search for sparse, robust neural networks. arXiv preprint arXiv:1912.02386, 2019.

Kazuki Irie, Albert Zeyer, Ralf Schlu¨ter, and Hermann Ney. Language modeling with deep transformers. arXiv preprint arXiv:1905.04226,2019.

Khaled Shaalan, Sanjeera Siddiqui, Manar Alkhatib, and Azza Abdel Monem. Challenges in arabic natural language processing. In Computational linguistics, speech and image processing for arabic language, pages 59–83. World Scientific, 2019.

Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In International Conference on Machine Learning, pages 8093–8104. PMLR, 2020.

Mahidhar Dwarampudi and NV Reddy. Effects of padding on lstms and cnns. arXiv preprint arXiv:1903.07288, 2019.

Manish Gupta and Puneet Agrawal. Compression of deep learning models for text: A survey. ACM Transactions on Knowledge Discovery from Data (TKDD), 16(4):1–55, 2022.

Moath Najeeb, Abdelkarim Abdelkader, Musab Al-Zghoul, and Abdelrahman Osman. A lexicon for hadith science based on a corpus. International Journal of Computer Science and Information Technologies, 6(2):1336–1340, 2015.

Moustafa Elshafei, Husni Al-Muhtaseb, and Mansour Alghamdi. Statistical methods for automatic diacritization of arabic text. In The Saudi 18th National Computer Conference. Riyadh, volume 18, pages 301–306, 2006.

Navin Kumar Manaswi and Navin Kumar Manaswi. Understanding and working with keras. Deep learning with applications using Python: Chatbots and face, object, and speech recognition with TensorFlow and Keras, pages 31–43, 2018.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. arXiv preprint arXiv:1906.01787, 2019.

Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In 2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS), pages 1597–1600. IEEE, 2017.

Shudong Yang, Xueying Yu, and Ying Zhou. Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example. In 2020 International workshop on electronic communication and artificial intelligence (IWECAI), pages 98–101. IEEE, 2020.

Tatyana Ruzsics and Tanja Samardzic. Neural sequence-to-sequence learning of internal word structure. In Proceedings of the 21st conference on computational natural language learning (CoNLL 2017), pages 184–194, 2017.

Umar Siddiqui, Habiba Youssef, Adel Sabour, and Mohamed Ali. Scalability, availability, reproducibility and extensibility in islamic database systems. IJASAT (International Journal on Islamic Applications in Computer Science and Technology), 10(1):12–21, 2022.

Yifei Ding, Minping Jia, Qiuhua Miao, and Yudong Cao. A novel time–frequency transformer based on self–attention mechanism and its application in fault diagnosis of rolling bearings. Mechanical Systems and Signal Processing, 168:108616, 2022.

Yonatan Belinkov and James Glass. Arabic diacritization with recurrent neural networks. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 2281–2285, 2015.

Zijun Zhang. Improved adam optimizer for deep neural networks. In 2018 IEEE/ACM 26th international symposium on quality of service (IWQoS), pages 1–2. Ieee, 2018.