**American University of Beirut**

**School of Engineering and Architecture**

**Department of Electrical and Computer Engineering**

# An Enhancement of Arabic Diacritization Models

By:

**Mohamad Ayman Charaf** **Tamara Sadek**

**Mohamad Khaled Charaf** **Nadim Sukkar**

**Aya El Baba** **Hassan Baydoun**

**Dana Kossaybati**

December 2023

## Table of Contents

## I- Introduction

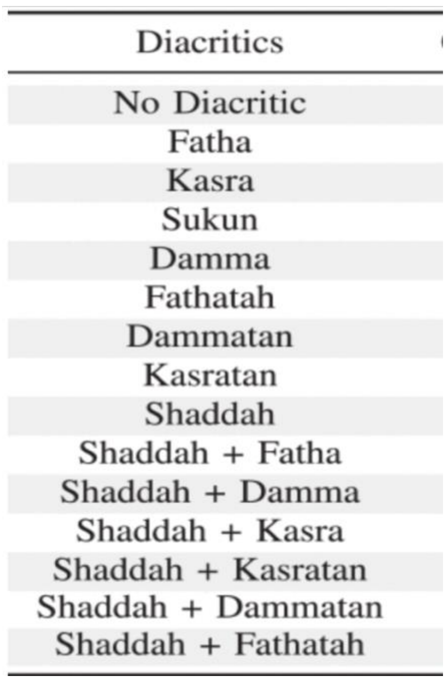
The Arabic language stands as a testament to the rich cultural heritage of the Arab world, serving as a linguistic tapestry interwoven with history, literature, and identity. Its significance transcends mere communication; Arabic is a source of pride, a medium for expression, and a vessel for preserving the narratives of a diverse and storied civilization. As technology advances, the imperative to maintain the integrity, beauty, and prevalence of Arabic becomes increasingly vital. In this context, our Natural Language Processing (NLP) project addresses a specific aspect crucial to the linguistic precision of Arabic—diacritization.

Diacritization, the process of adding diacritical marks to Arabic text, plays a pivotal role in ensuring the accurate representation of pronunciation, meaning, and syntactic nuances. These diacritical marks, commonly known as "Tashkeel" or "Harakat," include symbols like vowels, consonant points, and other linguistic cues that are indispensable for a comprehensive understanding of written Arabic. The absence of these diacritics can introduce ambiguity, leading to multiple interpretations and potential miscommunication.

In our pursuit to enhance the diacritization process, we explored a range of neural network models, namely Fully Connected Neural Networks (FCNNs) and Recurrent Neural Networks (RNNs). Leveraging the power of NLP and deep learning, our goal was to develop a model capable of accurately predicting diacritics in Arabic text. By doing so, we aim not only to streamline and automate the diacritization process but also to contribute to the preservation of the linguistic nuances that make Arabic a truly unique and vibrant language.
This report provides a comprehensive overview of our methodologies, experimental setups, and the outcomes derived from the application of different models. As we delve into the intricacies of diacritization, our endeavor is anchored in the belief that advancing technology can be a powerful ally in the mission to safeguard the essence of Arabic—a language that encapsulates the soul of a culture and the spirit of a people.

## II-Problem Fomulation

In the realm of diacritization, we confront the challenge of transforming a character sequence, denoted as $x=(x1,\ldots,xn)$, into a corresponding diacritic sequence $y=(y1,\ldots,yn)$. Every yi should signify the diacritic for its corresponding character xi, chosen from a set of 15 possible values, as illustrated in Figure 1.



| Diacritics |
|---|
| No Diacritic |
| Fatha |
| Kasra |
| Sukun |
| Damma |
| Fathatah |
| Dammatan |
| Kasratan |
| Shaddah |
| Shaddah + Fatha |
| Shaddah + Damma |
| Shaddah + Kasra |
| Shaddah + Kasratan |
| Shaddah + Dammatan |
| Shaddah + Fathatah |

*Figure 1: The 15 possible values our model should predict for each letter*

From a probabilistic standpoint, our objective is to identify a target sequence that maximizes the conditional probability of y given a source sentence x. In essence, the diacritization process involves predicting the most likely diacritic sequence for a given input character sequence, thereby enhancing the clarity and precision of the Arabic text. The 15 possible diacritic values encapsulate the variability inherent in Arabic script, making our diacritization model a pivotal component in deciphering and accurately representing the nuances of the language.

## III- Dataset

Ensuring the accuracy of training data is paramount for the robustness of any deep learning model. Our endeavor to obtain diacritized Arabic words proved to be a meticulous task, given that many texts lack comprehensive diacritization, potentially leading the machine model to learn inaccuracies. Notably, one of the 15 possible diacritization values is the absence of diacritization on specific letters exemplified by the word ",الذي" where diacritization is typically omitted for letters with "mad."

Our primary dataset originates from huggingface.co, comprising approximately 18000 lines of training data. For validation and testing, we utilized 2,500 lines each. The content is predominantly from classical Arabic sources. Notably, classical Arabic texts often lack full diacritization, posing a challenge that our model addresses.

Supplementing our dataset, we generated additional data from modern classical Arabic sources, including the children story "أبي صير وأبي قير" by Kamal Keilany and excerpts from Al Jazeera text. To enhance data quality, we implemented algorithms for diacritic removal, facilitating the separation of features from the output. Additionally, we developed an algorithm to eliminate English symbols, which can be extended to remove extraneous characters, including numbers and various symbols. The processing of Arabic sentences involved segmentation using known Arabic delimiters such as "،.!؟" To transform the diacritized text into a dataset of diacritized and undiacritized sentences.

It is crucial to note that our dataset's unique composition, blending classical and modern Arabic, contributes to the diversity and richness of our training data. As we delve into prior work, the dataset's distinctive characteristics set it apart, potentially impacting the model's performance positively.

## IV- Methodology

### A. Overview of Experimental Design

We opted to delve into the exploration of various distinct models, coupled with an extensive grid search to fine-tune their various parameters. The purpose of the grid search was to systematically adjust hyperparameters, seeking the most optimal configurations that enhance each model's performance. This approach allows us to rigorously assess the models' sensitivity to different parameter settings and strategically pinpoint the combinations that yield superior diacritization results.

## 1. Feed-Forward Neural Network (FFNN) Approach

In the paper by Fadel et al, the initial approach was to employ a Feed-Forward Neural Network (FFNN) that tackles diacritizing each character independently. This approach was realized through three distinct models, whose parameters we experimented with.

### 1.1 Basic Model

The first model, referred to as the Basic Model, presented in the paper operates with a 100-dimensional vector as input, capturing features for individual characters in a sentence. The vector is made up of 50 non-diacritic characters that precede the target character and the subsequent 50 characters, with padding when necessary. The model classifies inputs into one of the 15 predefined classes using a Softmax output unit (Fig 1), with the highest probability determining the correct output class.

The Basic Model was made up of 17 hidden layers of varying sizes, utilizing the Rectified Linear Unit (ReLU) activation function. To test different parameters, we changed the number of hidden layers and number of neurons in hidden layers to test different combinations. We tried three different architectures to be used for grid search: in the first we removed two hidden layers, in the second we added two hidden layers and for the third the number of neurons in the hidden layers was increased.

Table 1: Basic model with number of layers decreased

| Layer Name | Neurons | Activation Function |
|---|---|---|
| Hidden 1 | 200 | ReLU |
| Hidden 2 | 500 | ReLU |
| Hidden 3 | 500 | ReLU |
| Hidden 4 | 450 | ReLU |
| Hidden 5 | 400 | ReLU |
| Hidden 6 | 400 | ReLU |
| Hidden 7 | 350 | ReLU |
| Hidden 8 | 300 | ReLU |
| Hidden 9 | 300 | ReLU |
| Hidden 10 | 250 | ReLU |
| Hidden 11 | 200 | ReLU |
| Hidden 12 | 150 | ReLU |
| Hidden 13 | 100 | ReLU |
| Hidden 14 | 50 | ReLU |
| Hidden 15 | 25 | ReLU |
| Output | 250 | ReLU |

Table 2: Basic model with number of layers increased

| Layer Name | Neurons | Activation Function |
|---|---|---|
| Hidden 1 | 200 | ReLU |
| Hidden 2 | 500 | ReLU |
| Hidden 3 | 500 | ReLU |

| | | |
|---|---|---|
| Hidden 4 | 450 | ReLU |
| Hidden 5 | 400 | ReLU |
| Hidden 6 | 400 | ReLU |
| Hidden 7 | 350 | ReLU |
| Hidden 8 | 300 | ReLU |
| Hidden 9 | 300 | ReLU |
| Hidden 10 | 250 | ReLU |
| Hidden 11 | 250 | ReLU |
| Hidden 12 | 200 | ReLU |
| Hidden 13 | 200 | ReLU |
| Hidden 14 | 150 | ReLU |
| Hidden 15 | 150 | ReLU |
| Hidden 16 | 100 | ReLU |
| Hidden 17 | 100 | ReLU |
| Hidden 18 | 50 | ReLU |
| Hidden 19 | 25 | ReLU |
| Output | 15 | Softmax |

### 1.2 100-Hot Model

Each unique character in the dataset was assigned a number. The second model transforms each integer from the 100-integer inputs into its 1-hot representation as a 75-dimensional vector. These vectors are then concatenated, forming a 7,500-dimensional vector. The model architecture consists of five hidden layers incorporating dropout for regularization. The original model was trained using 50 epochs employing the Adam optimization algorithm and categorical cross-entropy as the loss function. In our training, we changed the architecture of the network twice, where we increased and then decreased the number of hidden layers to perform a grid search on these parameters.

Table 3: 100-Hot model with number of layers increased

| Layer Name | Neurons | Activation Function |
|---|---|---|
| One Hot | - | - |
| Flatten | - | - |
| Dropout (2.5%) | - | - |
| Hidden 1 | 250 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 2 | 200 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 3 | 150 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 4 | 125 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 5 | 100 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 6 | 75 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 7 | 50 | ReLU |
| Dropout (2.5%) | - | - |
| Output | 15 | Softmax |

Table 4: 100-Hot model with number of layers decreased

| Layer Name | Neurons | Activation Function |
|---|---|---|
| One Hot | - | - |
| Flatten | - | - |
| Dropout (2.5%) | - | - |
| Hidden 1 | 250 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 2 | 200 | ReLU |

|  |  |  |
|---|---|---|
| Dropout (2.5%) | - | - |
| Hidden 3 | 100 | ReLU |
| Dropout (2.5%) | - | - |
| Hidden 4 | 50 | ReLU |
| Dropout (2.5%) | - | - |
| Output | 15 | Softmax |

## 1.3 Embeddings Model

The last model replaces the 100-hot layer with an embeddings layer, which maps each unique character to a vector of values that are learned during training. This model consists of five hidden layers underwent training with the same settings as the 100-Hot Model. In our training, we also changed the architecture of the network twice, where we increased and then decreased the number of hidden layers to perform a grid search on these parameters like the procedure followed for the 100-Hot Model.

Table 5: Embeddings model with more layers

| Layer Name | Neurons | Activation Function |
|---|---|---|
| Embedding | - | - |
| Flatten | - | - |
| Dropout (10%) | - | - |
| Hidden 1 | 250 | ReLU |
| Hidden 2 | 200 | ReLU |
| Hidden 3 | 150 | ReLU |
| Hidden 4 | 125 | ReLU |
| Hidden 5 | 100 | ReLU |

| | | |
|---|---|---|
| Hidden 6 | 75 | ReLU |
| Hidden 7 | 50 | ReLU |
| Output | 15 | Softmax |

Table 6: Embeddings model with less layers

| Layer Name | Neurons | Activation Function |
|---|---|---|
| Embedding | - | - |
| Flatten | - | - |
| Dropout (10%) | - | - |
| Hidden 1 | 250 | ReLU |
| Hidden 2 | 200 | ReLU |
| Hidden 3 | 100 | ReLU |
| Hidden 4 | 50 | ReLU |
| Output | 15 | Softmax |

## 2. Recurrent Neural Networks (RNN) Approach

### 2.1 Basic Model

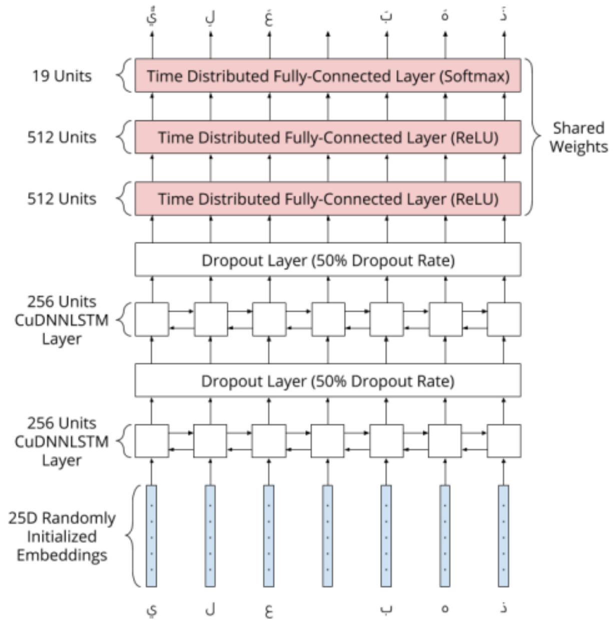

19 Units
Time Distributed Fully-Connected Layer (Softmax)
512 Units
Time Distributed Fully-Connected Layer (ReLU)
512 Units
Time Distributed Fully-Connected Layer (ReLU)
Shared Weights
Dropout Layer (50% Dropout Rate)
256 Units CuDNNLSTM Layer
Dropout Layer (50% Dropout Rate)
256 Units CuDNNLSTM Layer
25D Randomly Initialized Embeddings

*Figure 2: The Basic RNN Model Structure*

The model has embeddings which are 25D and randomly initialized. Following, the model features 2, 256 units, CuDNNLSTM layers. CuDNN is a GPU-accelerated library for deep neural networks, and CuDNNLSTM is specifically designed to accelerate the training and inference of LSTM (Long Short-Term Memory) networks on compatible NVIDIA GPUs. LSTM layer is used to capture the dependencies between the words. After that, the model has 3 Fully-Connected layers, 2 of which use 512 units and ReLU activation function, and the other uses 19 units and Softmax activation function.
Additionally, the model uses Adam optimization algorithm, and categorical cross-entropy loss function, with 50 epochs of training.

## 2.2 CRF Model

In this model, the CRF (Conditional Random Field) classifier is utilized instead of the Softmax layer for predicting the network output. CRF demonstrates enhanced capability compared to Softmax, particularly in handling sequence dependencies within the output layer, a characteristic present in diacritization problems. It's important to note that CRF is widely recognized as a best practice in addressing sequence labeling problems.

## B. Evaluation Metrics

In order to assess the accuracy of our models, we used two variations of the following metrics.

*Diacritization Error Rate (DER):* This metric, derived from all characters and their accurate diacritics, quantifies the percentage of characters that underwent incorrect diacritization. The computation involves extracting diacritics from both the original and predicted files, applying the following formula:

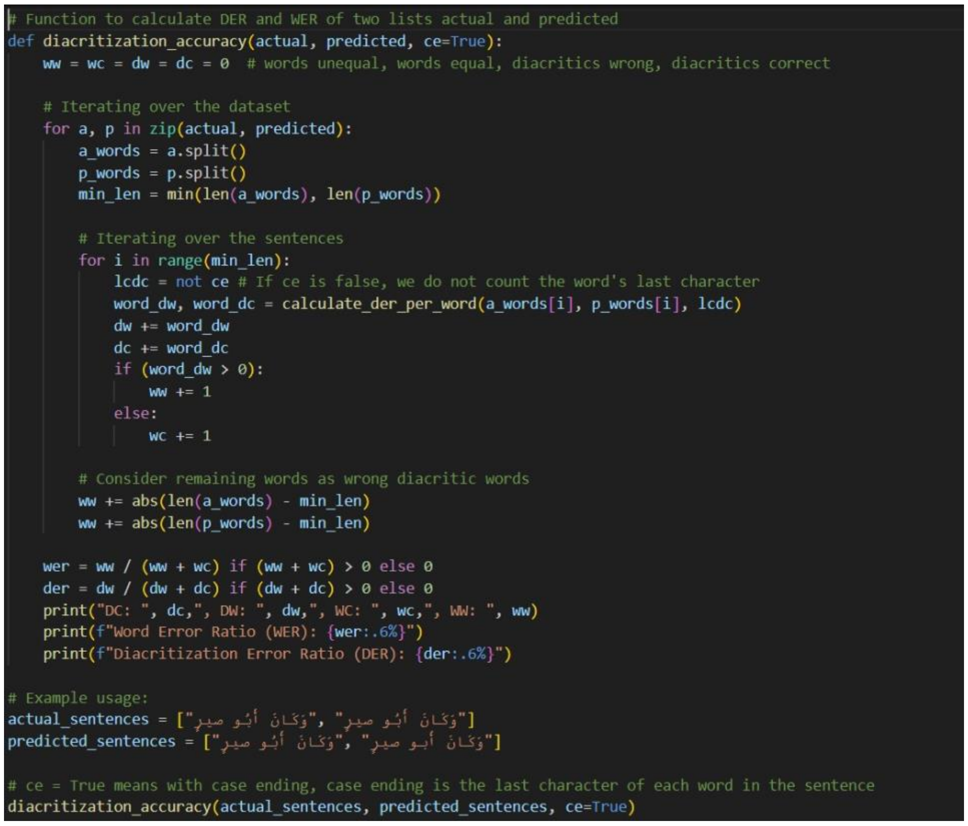
$$\mathrm{DER} = \frac{D_W}{D_W + D_C} \times 100$$

*Word Error Rate (WER):* Calculating the percentage of words containing at least one diacritization error, this metric compares all words between the original and predicted files to determine the proportion of unequal words:

$$\mathrm{WER} = \frac{W_W}{W_W + W_C} \times 100$$

The variation in our approach involves the consideration of predicting the last ending character, Case-Ending (CE). In the calculation without CE, each word's last character is excluded from the error computation, as it often adheres to grammatical rules. This provides insight into the core word's performance. Conversely, including CE in the calculation provides an overall assessment.

Based on this, we implemented a Python function that takes two string arrays, "predicted" and "actual," and a boolean "CE" (Case-Ending), capable of counting the mentioned variables (Dc: Diacritization correct, Dw: Diacritization wrong, Wc: Word diacritization correct, Ww: Word diacritization wrong), and thus calculating the accuracy metrics DER and CER.
This involved numerous loops and counters that iterate over each sentence to identify diacritization in place and determine correctness.



```python
# Function to calculate DER and WER of two lists actual and predicted
def diacritization_accuracy(actual, predicted, ce=True):
    ww = wc = dw = dc = 0  # words unequal, words equal, diacritics wrong, diacritics correct

    # Iterating over the dataset
    for a, p in zip(actual, predicted):
        a_words = a.split()
        p_words = p.split()
        min_len = min(len(a_words), len(p_words))

        # Iterating over the sentences
        for i in range(min_len):
            lcdc = not ce # If ce is false, we do not count the word's last character
            word_dw, word_dc = calculate_der_per_word(a_words[i], p_words[i], lcdc)
            dw += word_dw
            dc += word_dc
            if (word_dw > 0):
                ww += 1
            else:
                wc += 1

        # Consider remaining words as wrong diacritic words
        ww += abs(len(a_words) - min_len)
        ww += abs(len(p_words) - min_len)

    wer = ww / (ww + wc) if (ww + wc) > 0 else 0
    der = dw / (dw + dc) if (dw + dc) > 0 else 0
    print("DC: ", dc,", DW: ", dw,", WC: ", wc,", WW: ", ww)
    print(f"Word Error Ratio (WER): {wer:.6%}")
    print(f"Diacritization Error Ratio (DER): {der:.6%}")

# Example usage:
actual_sentences = ["وَكَانَ أَبُو صيرٍ", "وَكَانَ أَبُو صيرٍ"]
predicted_sentences = ["وَكَانَ أَبُو صيرٍ", "وَكَانَ أبو صيرٍ"]

# ce = True means with case ending, case ending is the last character of each word in the sentence
diacritization_accuracy(actual_sentences, predicted_sentences, ce=True)
```

*Figure 3: Snippet from the Diacritization Accuracy generator code*

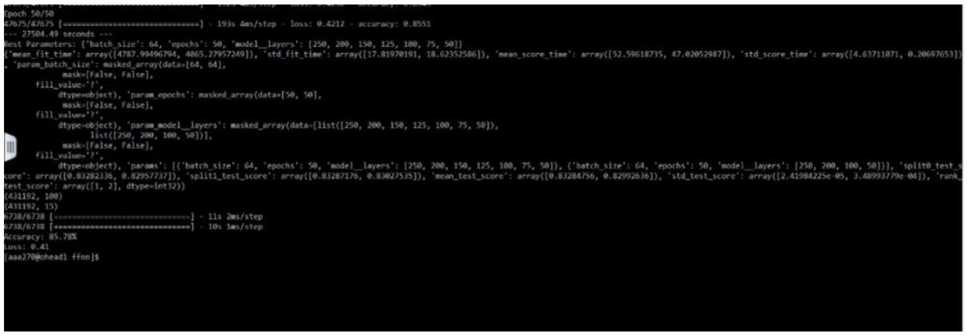
## V- Results and Analysis

The 100-Hot FFNN model was trained on an Nvidia V100 GPU. The grid search and fitting of the model took a total of 8 hours.

We used the grid search to compare two variations of this model. The first one is represented in Table 3, and the other is represented in Table 4.

Both models were trained for 50 epochs with a batch size of 64. The grid search revealed that the model represented by Table 3 yielded better results. It yielded a validation accuracy of 83.28%, while the other one yielded 82.95%.

Then, we refit the model with the best parameters, and ran it on the test set. It yielded a test accuracy of 85.78%



```
Epoch 50/50
47675/47675 [==============================] - 193s 4ms/step - loss: 0.4212 - accuracy: 0.8551
--- 27504.49 seconds ---
Best Parameters: {'batch_size': 64, 'epochs': 50, 'model__layers': [250, 200, 150, 125, 100, 75, 50]}
{'mean_fit_time': array([4787.99496794, 4065.27957249]), 'std_fit_time': array([17.81970191, 18.62352586]), 'mean_score_time': array([52.59618735, 47.02052987]), 'std_score_time': array([4.63711071, 0.20697653])
, 'param_batch_size': masked_array(data=[64, 64],
             mask=[False, False],
       fill_value='?',
            dtype=object), 'param_epochs': masked_array(data=[50, 50],
             mask=[False, False],
       fill_value='?',
            dtype=object), 'param_model__layers': masked_array(data=[list([250, 200, 150, 125, 100, 75, 50]),
                   list([250, 200, 100, 50])],
             mask=[False, False],
       fill_value='?',
            dtype=object), 'params': [{'batch_size': 64, 'epochs': 50, 'model__layers': [250, 200, 150, 125, 100, 75, 50]}, {'batch_size': 64, 'epochs': 50, 'model__layers': [250, 200, 100, 50]}], 'split0_test_s
core': array([0.83282336, 0.82957737]), 'split1_test_score': array([0.83287176, 0.83027535]), 'mean_test_score': array([0.83284756, 0.82992636]), 'std_test_score': array([2.41984225e-05, 3.48993779e-04]), 'rank_
test_score': array([1, 2], dtype=int32)}
(431192, 100)
(431192, 15)
6738/6738 [==============================] - 11s 2ms/step
6738/6738 [==============================] - 10s 1ms/step
Accuracy: 85.78%
Loss: 0.41
[aaa270@ohead1 ffnn]$
```

*Figure 4: Snippet from Joblib File of the 100-Hot-Encoding Model*

The embeddings model showed a training accuracy as high as 92.5% during the training process. It didn't complete training due to computational constraints (exceeding the memory limit).

Nevertheless, due to computational limitations, the other models were only partially trained.

## VI- Conclusion

Embarking on the realm of Natural Language Processing (NLP), our exploration into Arabic diacritization has shown the potential for technology to safeguard linguistic heritage. As we conclude our journey through diacritization complexities, it's evident that our pursuit holds promise in computational linguistics and preserving the Arabic language's sanctity.

The diverse array of models examined—ranging from Fully Connected Neural Networks (FCNNs) to RNNs (Recurrent Neural Networks)—underscored the challenge's complexity. Yet, it is precisely this complexity that reinforces the importance of our mission. By enhancing diacritization accuracy and efficiency, we contribute to fortifying the linguistic foundations of Arabic culture.

In the broader context, our work transcends algorithmic experimentation; it resonates with a commitment to cultural preservation. As technology evolves, so must our language approaches. By harnessing NLP, we position ourselves at the nexus of tradition and innovation, ensuring that the Arabic language remains vibrant.

While our journey through Arabic diacritization has yielded valuable insights, the incomplete nature of our model evaluations beckons further inquiry. The road ahead invites continued exploration of advanced models, linguistic nuances, and real-world applications. As we navigate this frontier, we remain steadfast in our dedication to upholding the integrity, beauty, and prevalence of the Arabic language—a language that continues to be the heartbeat of a civilization and a bridge to its rich cultural legacy.

## VII- References:

AliOsm. "shakkelha." GitHub, https://github.com/AliOsm/shakkelha/tree/master/models