Contents

[General Introduction 3](#_Toc202876553)

[Chapter 1 : Preliminaries 4](#_Toc202876554)

[1.1 Deep Learning Models 4](#_Toc202876555)

[1.1.1 Recurrent Neural Network : 4](#_Toc202876556)

[1.1.2 Long Short Term Memory 5](#_Toc202876557)

[6](#_Toc202876558)

[1.1.3 Transformers 7](#_Toc202876559)

[1.2 Deep Learning for Natural Language Processing (NLP) 9](#_Toc202876560)

[1.2.1 Introduction to NLP 9](#_Toc202876561)

[1.2.2 Arabic NLP 10](#_Toc202876562)

[1.2.3 The Task of Diacritization 10](#_Toc202876563)

[Chapter 2: State of the Art 11](#_Toc202876564)

[2.1 Introduction 11](#_Toc202876565)

[2.2 Early Computational Models: Rule Based and Statistical Approaches 11](#_Toc202876566)

[2.3 Bridging the Gap : Hybrid Approaches 12](#_Toc202876567)

[2.4 Deep Learning Models 14](#_Toc202876568)

[2.5 Conclusion 15](#_Toc202876569)

[Chapter 3: Resources and Methodology 16](#_Toc202876570)

[Dataset Description 16](#_Toc202876571)

[Preprocessing and Splitting 17](#_Toc202876572)

[Evaluation Metrics 19](#_Toc202876573)

[Chapter 4 : Offline Diacritization 19](#_Toc202876574)

[Bilstm Architectures 19](#_Toc202876575)

[Transformer Architectures 19](#_Toc202876576)

[Experiments and Analysis 19](#_Toc202876577)

[Chapter 5 : Online Diacritization 20](#_Toc202876578)

[Real-Time Inference Challenges 20](#_Toc202876579)

[Proposed Approach 20](#_Toc202876580)

[Implementation and Design 20](#_Toc202876581)

[Evaluation 20](#_Toc202876582)

[General Conclusion 21](#_Toc202876583)

[Bibliography 21](#_Toc202876584)

Table of Figures

[Figure 1.1 Recurrent Vs Feedforward networks 4](file:///C:\Users\Lenovo\Downloads\FYP%20report.docx#_Toc202956497)

[Figure 1.2 LSTM model 5](file:///C:\Users\Lenovo\Downloads\FYP%20report.docx#_Toc202956498)

[Figure 1.3 BiLSTM 6](file:///C:\Users\Lenovo\Downloads\FYP%20report.docx#_Toc202956499)

[Figure 1.4 The Transformer - model architecture 7](file:///C:\Users\Lenovo\Downloads\FYP%20report.docx#_Toc202956500)

[Figure 1.5 Arabic Morphology 10](file:///C:\Users\Lenovo\Downloads\FYP%20report.docx#_Toc202956501)

[Figure 1.6 The diacritizations of علم and their meanings 10](#_Toc202956502)

# General Introduction

# Chapter 1 : Preliminaries

## Deep Learning Models

Deep learning is a subset of machine learning that uses artificial neural networks to model complex patterns in data. A deep learning model is a compilation of nodes that connect and layer in neural networks, much like the human brain. These networks pass information through each layer, sending and receiving data to identity patterns. Unlike traditional algorithms that rely on explicit feature engineering, deep learning models automatically learn hierarchical representations from raw input, making them highly effective in the areas of speech recognition, computer vision, and natural language processing. Deep learning models use different types of neural networks to achieve specific solutions. In this section, we will focus on Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformers since these architectures have had a major influence on the processing of sequential data, mostly text.

### 1.1.1 Recurrent Neural Network :

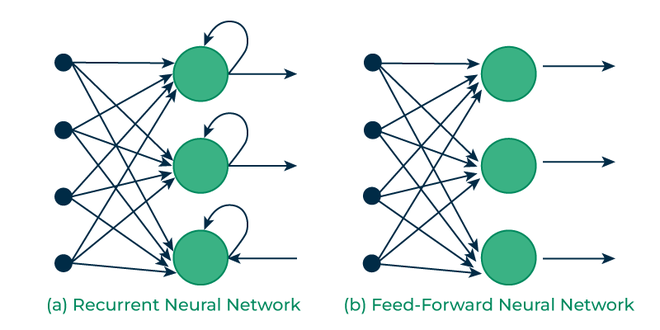
RNN is a particular form of neural network created to handle sequential data, which makes it particularly well-suited for applications like language processing where the order of input is important. RNNs are unlike typical feedforward networks in that they can "remember" past time step information in the sequence since they maintain a hidden state that captures information about previous inputs as shown in the figure below. An RNN is a system that consists of many interconnected components mimicking how humans perform sequential data conversions, such as translating text from one language to another. Therefore RNNs are particularly powerful for usage like speech recognition, text synthesis, and language modeling. [1]

Figure 1.1 Recurrent Vs Feedforward networks

RNNs' main advantage is their capacity to handle sequences of different lengths, however, they struggle to capture long-range relationships in sequences because of the vanishing gradient problem. It can be challenging for the network to learn dependencies from remote inputs when the gradient of the loss function diminishes to almost zero as it travels back through numerous layers in lengthy sequences.

### 1.1.2 Long Short Term Memory

In order to effectively address the vanishing gradient issue, LSTM Units were created. . They are a special kind of RNN capable of learning long-term dependencies. LSTMs do that by using structures known as gated cells to hold additional data outside of the conventional neural network flow [2]. These gates are as follows:

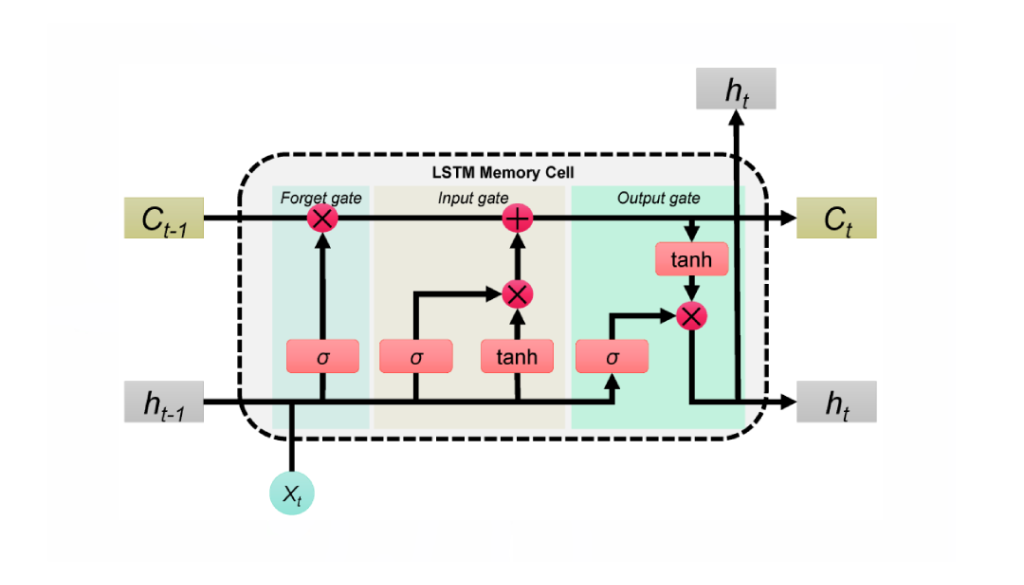


Figure 1.2 LSTM model

* **Input Gate:**The input gate controls how much of the new information from the current input should be added to the cell state. Here the network can use relevant new information while ignoring unnecessary data. The input gate decides this through a sigmoid activation function, which outputs values between 0 and 1, indicating the extent to which new data should be added.
* **Forget Gate:** The forget gate determines what information should be discarded from the cell state. It looks at the previous hidden state and the current input to decide which parts of the memory cell should be "forgotten." Also, the forget gate uses a sigmoid activation function to output values between 0 and 1, where 0 means "forget everything" and 1 means "keep everything."
* **Output Gate:** The output gate controls what information should be passed to the next time step and be included in the output of the LSTM unit. After updating the cell state, the output gate applies a tanh function to the cell state and multiplies it by the output of the forget gate to regulate how much information from the memory should be exposed.

This gating mechanism allows LSTMs to maintain long-term dependencies and alleviate the vanishing gradient problem, making them effective for tasks where the context from earlier in the sequence is important.

#### 

#### 1.1.2.1 BiLSTM

While standard LSTMs process sequences in a single direction, typically from past to future, this unidirectional approach can be limiting when the context from both past and future elements in the sequence is important for understanding. BiLSTM networks [3] address this limitation by incorporating two LSTM layers: one that processes the sequence from left to right (forward direction) and another that processes it from right to left (backward direction). The outputs from both directions are then combined, typically by concatenation, to form the final output. This bidirectional setup allows BiLSTM models to capture richer contextual information, making them particularly effective for tasks such as named entity recognition, part-of-speech tagging, sentiment analysis, and machine translation. By leveraging information from both past and future time steps simultaneously, BiLSTMs provide a more complete understanding of the sequence compared to unidirectional models.

### 

Figure 1.3 BiLSTM

### 

### 

### 

### 

### 

### 1.1.3 Transformers

The Transformer model, introduced in the paper “Attention is All You Need,” [4] has revolutionized natural language processing due to its efficiency and ability to handle long-range dependencies in sequences. Unlike RNNs and LSTMs, transformers process all elements of a sequence simultaneously rather than sequentially, which allows them to capture relationships between words in parallel.

There are various types of Transformer architectures that modify this structure to suit different tasks and improve performance. For instance, **Encoder-only Transformers**, such as **BERT** [5], focus solely on the encoder portion and are highly effective for tasks like text classification, question answering, and diacritization. These models rely on the encoder’s ability to capture context from both directions in the input sequence. Other variations, such as **Decoder-only Transformers** like **GPT** [6], focus on the decoder part and are typically used for text generation tasks, where the model generates a sequence one token at a time.

Here’s a breakdown to highlight how the Transformers Layer interaction in order to transform the input data:

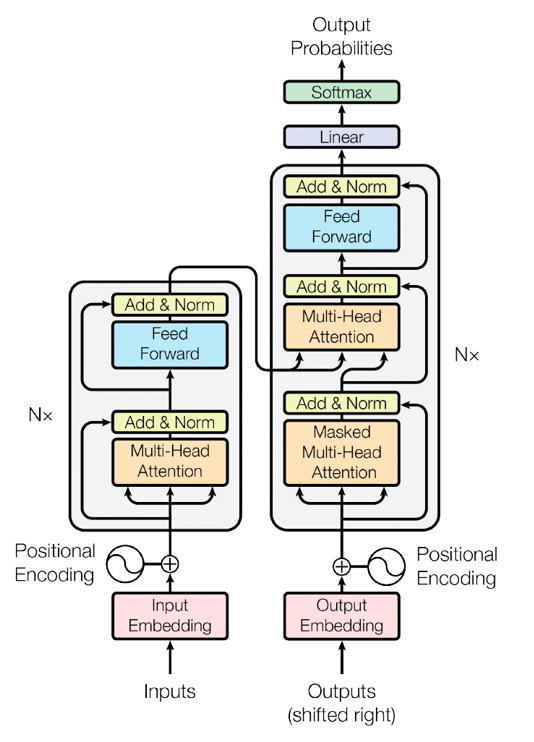
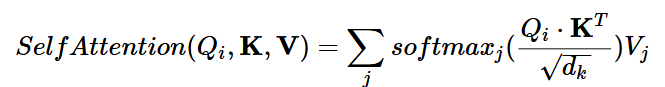


Figure 1.4 The Transformer - model architecture

* **Self-Attention Mechanism:** The self-attention mechanism allows each word in a sequence to "attend" to every other word in the sequence. It computes a set of attention scores for each word, indicating how much focus should be placed on other words in the sentence. This process enables the transformer to capture dependencies regardless of the distance between the words, making it highly effective for long sentences or paragraphs.



For each word in the sequence, the transformer calculates three vectors:

**Query (Q):** vector representing the current word, seeking relevant information from other words in the sequence.

**Key (K):** vector corresponding to each word in the sequence and serves as the source of information that can be accessed by the current word.

**Value (V):** vector containing the actual data or content associated with the word, which will be passed to the subsequent layers after attending to the relevant words in the sequence.

The attention score between two words is computed by taking the dot product of the query vector for one word and the key vector for another. This score is then normalized using a softmax function to ensure that the scores sum up to 1.

* **Multi-Head Attention:** Rather than performing a single attention calculation, the transformer uses multiple heads to capture different aspects of the relationships between words. Each head operates independently, calculating attention scores in parallel, and the results are concatenated and passed through a linear transformation.

This multi-head attention allows the model to focus on various parts of the input sequence simultaneously, improving its ability to capture complex patterns.

* **Positional Encoding :** Since transformers process all words in parallel, they lack a natural sense of the order of words in a sequence. To overcome this, transformers use positional encoding to inject information about the position of each word in the sequence. This encoding is added to the input embeddings of the words, ensuring that the model can differentiate between words that appear earlier or later in the sequence.
* **Feed-Forward Networks:** After the self-attention layers, the output is passed through a feed-forward neural network consisting of two linear layers with a ReLU activation in between. This network helps to transform the data and capture more abstract features.
* **Layer Normalization and Residual Connections:** To ensure stable training, transformers use layer normalization and residual connections. Residual connections help the model maintain information across layers, while layer normalization ensures that the inputs to each layer are on a similar scale, preventing large gradients and improving convergence.
* **Final Output :** The final output of the transformer is passed through a linear layer and softmax function to generate the predicted output for tasks like translation, text generation, or classification.

## 1.2 Deep Learning for Natural Language Processing (NLP)

### 1.2.1 Introduction to NLP

Natural Language Processing (NLP) is a field within artificial intelligence (AI) focused on enabling computers to understand, interpret, and generate human language. Language, with its complexity and nuances, has long been a challenge for machines, and NLP seeks to bridge the gap between human communication and machine understanding. [7]

NLP combines computational linguistics and statistical methods to process, analyze, and generate natural language data. It involves several tasks, ranging from text classification, named entity recognition, and sentiment analysis to more complex tasks like machine translation, diacritization, summarization, and question answering.

With the development of deep learning models like Recurrent Neural Networks (RNNs) and Transformers, NLP has made significant improvements, showing impressive capabilities in handling language data. These improvements have revolutionized applications such as chatbots, speech recognition systems, and automatic translation services.

### 1.2.2 Arabic NLP

Arabic presents several unique challenges that make NLP tasks more complex than for many other languages. These challenges stem from its rich morphology, syntactic flexibility, and the absence of diacritics in standard written form.

Arabic is a morphologically rich language, meaning that words can take on many different forms based on their roots, affixes, tenses, gender, and number.

For example, the root "كتب" can form words like:

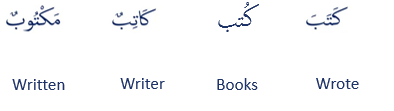


Figure 1.5 Arabic Morphology

This root-based derivation can lead to a large number of word variations, making it challenging for traditional NLP models to correctly identify and process all forms of a word.

### 1.2.3 The Task of Diacritization

##### In Arabic, diacritics (الحركات) are used to indicate pronunciation and meaning. Many words can be pronounced differently depending on the diacritics, which affect their meaning.

However, in modern written Arabic, diacritics are often omitted, making the task of understanding the exact meaning more difficult. This presents a significant challenge for automatic systems that need to infer missing diacritics based on context.

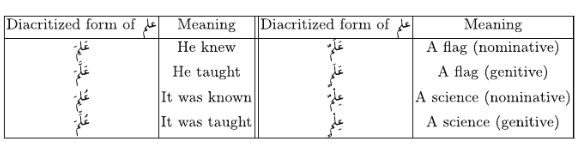


Figure 1.6 The diacritizations of علم and their meanings

Automatic diacritization is thus a crucial step in improving the quality of Arabic NLP applications, including speech synthesis and text translation.

# Chapter 2: State of the Art

## 2.1 Introduction

Arabic diacritization task has evolved significantly over the past few decades. Researchers have approached this challenge using a variety of computational strategies that reflect broader trends in NLP. Initially dominated by handcrafted linguistic rules, the field gradually embraced statistical learning methods as annotated corpora became more available. In recent years, deep learning has emerged as the dominant paradigm, enabling end-to-end models that outperform earlier approaches by automatically learning contextual patterns. In this chapter, we present a chronological narrative that traces the evolution of Arabic diacritization techniques, grouped into three broad phases: (1) early computational models encompassing both rule-based and statistical approaches, (2) hybrid systems that blend linguistic rules with statistical inference, and (3) neural models based on deep learning architectures.

Figure 2.1 Arabic Diac Model

...

Input

Output

اللُّغَةُ العَرَبيّةُ لغَةٌ جَمِيلَةٌ وَغَنِيَةٌ بِالتَّارِيخِ وَالثّقَافَةِ

اللغة العربية لغة جميلة وغنية بالتاريخ والثقافة

Arabic Diacritization Model

## 2.2 Early Computational Models: Rule Based and Statistical Approaches

In the early stages of Arabic diacritization, rule-based systems were the primary approach. These systems rely on predefined sets of rules that encode linguistic knowledge about how diacritics should be added to text based on its syntactic structure. However, with the increasing availability of annotated corpora and computational resources in the mid-2000s, statistical models began to emerge as an alternative, offering more flexible and scalable solutions.

#### **2005 – Morphological Tagging with Full Lexical Lookup** [8]

One of the earliest rule-based efforts was presented by Habash and Rambow (2005, 2007), who proposed a morphological analyzer and POS tagger that leveraged full lexical lookup tables. Their system assigned diacritics based on morphological disambiguation and probabilistic lexeme selection, effectively combining handcrafted rules with frequency-based insights.

**2005 – Cascading Weighted Finite-State Transducers (WFST)** [9]

Similarly, Nelken & Shieber (2005) introduced an innovative approach using cascading WFSTs, which modeled diacritization as a multi-stage transduction process. At each stage, a finite-state machine applied weighted rules to map unvocalized text to fully diacritized output. By combining subword and word-level rules into a unified framework, this method captured local dependencies and resolved ambiguities based on linguistic patterns

**2006 – Maximum Entropy (MaxEnt) Diacritization** [10]

The shift toward statistical learning was marked by the work of **Sarikaya et al. (2006). It** introduced one of the first purely statistical frameworks using MaxEnt models. They treated each grapheme as a state that emits diacritics, informed by a “diacritization parse tree” combining lexical, contextual, and POS information. Their model effectively resolved ambiguity without handcrafted rules, achieving moderate error rates (7–17%) on dialectal Arabic

**2010 – Trigram-based n-gram model**

Building on this trend, **Hifny (2010)** used **trigram-based n-gram models** trained on diacritized corpora to assign diacritics using dynamic programming. The model captured local contextual dependencies but remained limited by the fixed window size.

Later, **Bebah et al. (2014)** proposed a more sophisticated pipeline by integrating the AlKhalil morphological analyzer with **two-level Hidden Markov Models (HMMs)** and **Viterbi decoding**, allowing the system to disambiguate diacritic candidates based on both morphological structure and probabilistic transitions.

Collectively, these early models laid the groundwork for subsequent innovations. While rule-based systems were praised for their linguistic transparency, statistical methods provided greater adaptability and coverage. However, both approaches still relied heavily on feature engineering and struggled with long-range dependencies, paving the way for the emergence of hybrid and neural methods in later years.

## 2.3 Bridging the Gap : Hybrid Approaches

As both rule-based and statistical methods matured, researchers began to explore systems that combined the strengths of each. Rule-based models offered linguistic rigor and interpretability, while statistical approaches provided adaptability to unseen data. This convergence gave rise to **hybrid systems**—architectures that layered or integrated rule-based components with statistical models to improve performance and robustness. These systems served as transitional solutions before the wide adoption of deep learning and played a significant role in advancing the state of Arabic diacritization.

**2017 – SHAKKIL: Morphological + Syntactic Rules** [11]

SHAKKIL is a two-level diacritization framework that integrates both rule-based and statistical processing. The first level performs **morphological analysis** using over 600 handcrafted rules, including pattern recognition for prefixes, stems, and suffixes. This is followed by a **statistical disambiguation module** that employs language models (e.g., POS and stem-tag LMs) to score and resolve ambiguous candidates. The second level uses **syntactic rules** to assign case-ending diacritics based on shallow parsing. By combining handcrafted rules with statistical language models in a modular pipeline, SHAKKIL demonstrated significant improvements in both accuracy and generalizability, especially in formal written Arabic.

**2017 – Darwish et al. : Viterbi decoding + SVM Ranking** [12]

Another example is the work by **Darwish et al. (2017)**, who introduced a hybrid approach that separated the diacritization task into two sub-problems: **word vocalization** and **case-ending restoration**. The word vocalization component relied on **Viterbi decoding** to generate likely diacritized forms, while case endings were selected using a **Support Vector Machine (SVM) ranker** trained on linguistic features such as syntactic cues and morphological context. This division of labor between sequence modeling and supervised ranking reflects a hybrid design philosophy that seeks to balance statistical learning with domain-specific linguistic knowledge.

These hybrid systems illustrate a period of methodological experimentation and architectural blending. They preserved the interpretability of rule-based models while mitigating their brittleness through statistical learning. At the same time, they improved the performance limitations of purely statistical methods by injecting explicit linguistic structure. Though later surpassed by neural networks in accuracy and scalability, these systems were crucial stepping stones in the field’s evolution, demonstrating how combining symbolic and data-driven approaches could lead to better results than either alone.

## 2.4 Deep Learning Models

With the advent of deep learning, diacritization systems underwent a significant transformation. Deep learning models can automatically learn features from raw text data, eliminating the need for manual feature engineering. These models excel at capturing complex, long-range dependencies and can generalize to unseen data more effectively than statistical models. Since 2015, deep neural networks have gradually become the dominant approach in the field, achieving state-of-the-art performance across multiple diacritization benchmarks.

**2015 – Character-Level LSTM by Belinkov and Glass** [13]  
Belinkov and Glass (2015) introduced one of the first deep learning models for diacritization, employing a character-level LSTM network. This model learned contextual embeddings of characters, enabling accurate diacritization with minimal linguistic preprocessing. Their results established deep learning as a promising direction in the field.

**2015 – Feedforward Neural Network by Abandah et al.** [14]  
Abandah et al. (2015) experimented with a feedforward neural network trained on diacritized corpora. Though less expressive than recurrent models, their approach demonstrated the feasibility of neural methods even with simpler architectures.

**2015 – Multi-phase Diacritization Pipeline by Rashwan et al.**  
Rashwan et al. proposed a modular deep learning system that separated core diacritic prediction from case-ending restoration. By modeling short vowels and syntactic diacritics independently, the system improved accuracy in formal texts and demonstrated the value of task decomposition.

**2017 – Hybrid Neural Model by Darwish et al.**  
In a hybrid architecture, Darwish et al. (2017) used a deep neural network for base diacritic prediction and complemented it with a machine learning classifier (SVM) to determine case endings. This approach combined the strengths of data-driven learning with syntactic feature modeling.

**2019 – SHAKKALA: BiLSTM Encoder-Decoder Model**  
SHAKKALA, a purely neural system, used a bidirectional LSTM encoder-decoder to predict diacritics at the character level. It replaced rule-based components with a unified neural framework and achieved high accuracy on both Modern Standard Arabic (MSA) and Quranic Arabic.

**2023 – 2S-Diac: Two-Source Transformer-based Model**  
Bahar et al. (2023) introduced 2SDiac, a multi-source neural model designed to leverage both undiacritized Arabic text and a parallel stream of partially diacritized input. The architecture is built on either BiLSTM or self-attention layers and treats diacritization as a character-level classification task. A key innovation is the use of “Guided Learning,” a training scheme that introduces varying levels of masking over input diacritics, enabling the model to generalize across a spectrum of input completeness. Despite its compact size (~4.9M parameters), 2SDiac achieves state-of-the-art results on the Tashkeela and ATB benchmarks, outperforming larger transformer-based models.

**2024 – PTCAD: Pre-FineTuned Token Classification for Arabic Diacritization**  
Skiredj and Berrada (2024) proposed PTCAD, a transformer-based diacritization model that reframes the task as token classification. The system introduces a two-phase training strategy: a pre-finetuning stage using related linguistic tasks such as POS tagging and segmentation (all cast as masked language modeling tasks), followed by a dedicated fine-tuning phase for diacritic prediction. PTCAD builds upon pre-trained BERT-like models, enabling it to learn deep contextual representations specific to Arabic.

Despite their advantages, deep learning models come with challenges. They typically require large annotated datasets, demand significant computational resources, and often act as black boxes, offering limited interpretability. Nonetheless, their ability to learn rich, context-aware representations has made them the dominant paradigm in modern Arabic diacritization research and applications, including speech synthesis, digital education tools, and text-to-speech systems.

## 2.5 Conclusion

Arabic diacritization has progressed from rule-based and statistical approaches to sophisticated deep learning models. Early methods emphasized linguistic rules and statistical inference, while hybrid systems bridged the gap by combining symbolic and data-driven strategies. Today, neural models, particularly those based on transformers and pre-trained language models, represent the state-of-the-art due to their ability to capture complex contextual dependencies.

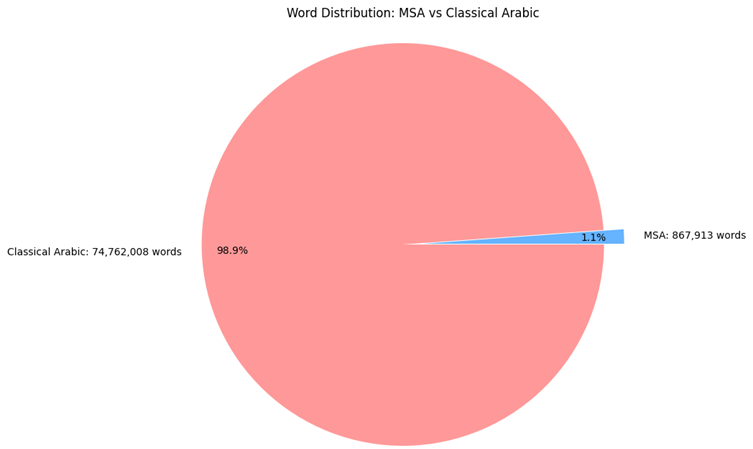
However, practical challenges remain. Many high-performing systems are not publicly available, limiting reproducibility and real-world adoption. Additionally, the need for large annotated datasets and the black-box nature of neural models present barriers to further development. Despite these challenges, deep learning continues to drive innovation in diacritization, with applications expanding into speech synthesis, digital education, and beyond.

# Chapter 3: Resources and Methodology

## 3.1 Dataset Description

1. **Tashkeela Corpus**

The Tashkeela dataset is one of the most commonly used resources for Arabic diacritization tasks. It was specifically designed for the task of automatic diacritization and contains a rich set of text from diverse domains. This dataset is made of 97 religious books written in the Classical Arabic (CA) style, with a small part of web crawled text written in the Modern Standard Arabic (MSA) style . The original dataset has over 75.6 million words, where over 67.2 million are diacritized Arabic words..



Classical Arabic : 74,762,008 words

MSA : 867,913 words

Figure 3.1 Tashkeela Word Distrribustion: MSA vs Classical Arabic

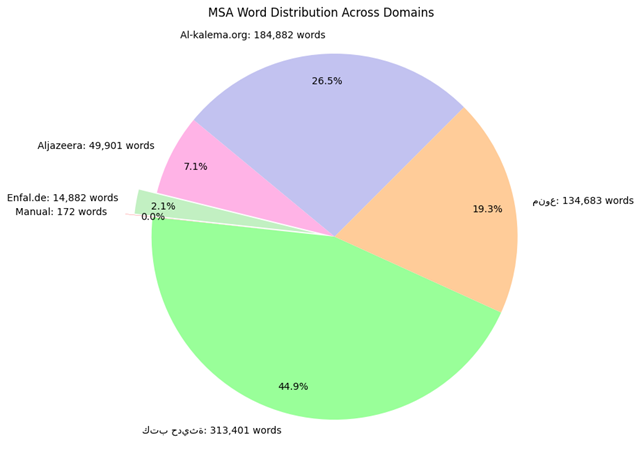
The MSA part of the dataset, even though small, spans various domains such as religious articles from the alkaleema website, news articles from al Jazeera website and some Arabic lessons and more , providing a diverse representation.

Figure 3.2 Tashkeela MSA Word Distrbiution Across Domains

The structure of the data in this dataset is not consistent since its sources are heterogeneous. Furthermore, it contains some diacritization errors and some useless entities

* Diacritization errors, such as the extra *Sukoon* on the declarative  , reversing the  + *Tanween Fath* and diacritic + *Shadda* combinations
* Ending Diacritics: The diacritic belongs to the end of the word, but is separated from it with one or more whitespaces.
* Misplaced Diacritics: Diacritics following a non-Arabic character such as a whitespace, numbers and punctuations.
* Multiple Diacritics: Multiple diacritics appear on a single character
* Inconsistency in Fathatan Placement: In most cases, for diacritizing a character with Fathatan diacritic, an extra character Alif ‘@’ must be added after the character. There are two schools for placing the diacritic, one which puts it after the Alif, and the other one puts it before the Alif (on the character itself). Both might appear in the same sentence.
* Non-Diacritized Lines: Some books of CA and many files in MSA contain lines without any diacritization.

1. **WikiNews**

The Wikinews dataset was extracted from Wikipedia and is focused on news articles. This dataset is valuable because it provides a more diverse and up-to-date set of sentences compared to Tashkeela, and includes a broad range of topics.

* Content: The dataset consists of Arabic news articles covering various subjects, including politics, economics, sports, and technology. This diversity makes it a useful dataset for training models to handle real-world applications of Arabic text.
* Size: The Wikinews dataset contains around 50,000 sentences, with over 3 million words.
* Domain: The content is less formal than Tashkeela, reflecting more conversational and journalistic styles.
* Challenges: One of the key challenges of this dataset is that it contains multiple dialects and informal language, making it more representative of the diverse ways Arabic is used in online media. However, due to its mixture of formal and informal language, it may be harder to achieve consistent results across different domains.

## 

## 

## 3.2 Preprocessing and Splitting

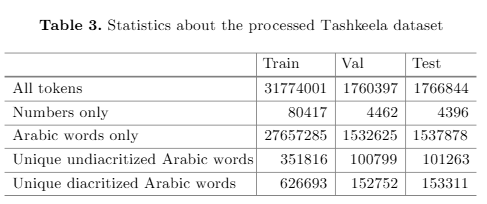
The preprocessing and splitting of datasets play a crucial role in ensuring that the model is trained and evaluated on representative and balanced data. For the **Tashkeela dataset**, we utilized different preprocessed versions and experimented with various splitting techniques to ensure the robustness of the model. Below, we describe the different versions of the Tashkeela dataset and the splitting strategies applied.

#### Abbad Tashkeela (Pre-split and Preprocessed Version)

The Abbad Tashkeela dataset is a pre-split and preprocessed version of the original Tashkeela corpus. This version of the dataset was introduced by Abbad et al. (2010) and is widely used in Arabic diacritization research.

· Processing: The Abbad Tashkeela dataset was preprocessed to remove any irrelevant text (such as punctuation marks and special characters) and standardized to ensure consistency in the diacritization.

· Split: The dataset was pre-split into training, validation, and test sets to streamline the process of training and evaluating models.

· 

The Abbad Tashkeela corpus is well-regarded for its consistent formatting and high-quality diacritization, making it a reliable resource for training models on formal Arabic text.

Fadel Split (Classical Arabic Tashkeela)

The Fadel Split is another version of the Tashkeela dataset, specifically focused on Classical Arabic texts. This version, also preprocessed and pre-split, was created to handle specific formal Arabic texts used in classical literature, science, and historical documents.

* Processing: Similar to the Abbad Tashkeela version, the Fadel Split was preprocessed to ensure consistency and high-quality diacritization. It contains only Classical Arabic, making it a specialized subset of Tashkeela.
* Split: The dataset was pre-split into training, validation, and test sets, providing a controlled environment for model training and evaluation.
* Statistics:
  + Training Set: Contains approximately 15,000 sentences (around 900,000 words).
  + Validation Set: Contains approximately 3,000 sentences (around 200,000 words).
  + Test Set: Contains approximately 2,000 sentences (around 150,000 words).

The Fadel Split is useful for training diacritization models that are specifically tailored to Classical Arabic and has been used in many studies focused on literary and historical Arabic texts.

#### Domain Stratified Split

In addition to the pre-split versions of Tashkeela, we experimented with a Domain Stratified Split, which aimed to ensure that the training, validation, and test sets contained equal percentages of each domain. This approach is important because it provides a more balanced representation of different contexts and domains, ensuring that the model is not biased toward any specific type of text.

· Process: To achieve this stratification, we first identified the domains present in the Tashkeela corpus (e.g., literature, news, scientific content, religious texts). Then, we split the data so that each domain was equally represented in the training, validation, and test sets.

· Benefits: The Domain Stratified Split helps the model learn to handle diverse contexts, which is especially important for real-world applications of Arabic diacritization, where texts can come from various domains (e.g., online articles, academic papers, social media posts).

## 3.3 Evaluation Metrics

# Chapter 4 : Offline Diacritization

## Bilstm Architectures

## Transformer Architectures

## Experiments and Analysis

# Chapter 5 : Online Diacritization

## Real-Time Inference Challenges

## Proposed Approach

## Implementation and Design

## Evaluation

# General Conclusion

References

# Bibliography

|  |  |
| --- | --- |
| [1] | I. B. Y. &. C. A. Goodfellow, Deep learning. MIT Press., 2016. |
| [2] | S. &. S. J. Hochreiter, Long short-term memory. Neural Computation, 9(8), 1735–1780, 1997. |
| [3] | M. &. P. K. K. Schuster, Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), 2673–2681, 1997. |
| [4] | A. S. N. P. N. U. J. J. L. G. A. N. K. Ł. &. P. I. Vaswani, Attention is all you need. Advances in Neural Information Processing Systems, 30., 2017. |
| [5] | J. C. M.-W. L. K. &. T. K. Devlin, BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics:, 2019. |
| [6] | A. N. K. S. T. &. S. I. Radford, Improving language understanding by generative pre-training, 2018. |
| [7] | D. &. M. J. H. Jurafsky, Speech and Language Processing (3rd ed., draft). Stanford University., 2023. |
| [8] | N. H. a. O. Rambow, Arabic Diacritization through Full Morphological Tagging, 2005. |
| [9] | R. N. a. S. M. Shieber, Arabic Diacritization Using Weighted Finite-State Transducers. |
| [10] | J. S. S. R. S. Imed Zitouni, Maximum Entropy Based Restoration of Arabic Diacritics, 2006. |