A Support System for Diacritic-aware Classical Arabic Language Processing: Integration of Speech, Text, and Vision Modalities

Adel AbdelSabour Ahmad Aly

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Committee:

Mohamed Ali, Chair

Michael Stiber

Leilani Battle

Abdeltawab Hendawi

Program Authorized to Offer Degree:
Computer Science and Systems

University of Washington

Abstract

A Support System for Diacritic-aware Classical Arabic Language Processing: Integration of Speech, Text, and Vision Modalities

Adel AbdelSabour Ahmad Aly

Chair of the Supervisory Committee:
Mohamed Ali
School of Engineering and Technology

Artificial intelligence (AI) impresses us daily, outperforming humans in complex games and tasks. Yet, AI and Large Language Models (LLMs) stumble to grasp a language that is thousands of years old. It's Arabic, where subtle diacritical marks can completely alter a word's meaning. Top language models like ChatGPT and Google's Gemini face challenges with Arabic's unique features, potentially leading to critical misunderstandings. The main obstacle is adapting successful NLP systems from other languages to Arabic without understanding its distinct nuances. This dissertation presents a support system—a **Multimodal Integration System**—for diacritic-aware Classical Arabic language processing. The system integrates speech, text, and vision modalities to address the unique challenges of Arabic's rich linguistic features, such as diacritics and linguistic styles. Arabic has multiple correct linguistic styles, each preserving the same text but with different diacritics. These variations reflect regional dialects, adding meaning, and altering grammar and rhetoric. The Holy Quran, with its 20 linguistic styles based on a single core text, serves as our ideal dataset. We developed innovative databases and models that push the boundaries of Arabic language processing. Our scalable databases store texts in 7 different Arabic linguistic styles. QR-Vision excels at recognizing precise diacritics in images. QRDiaRec adds diacritics to un-diacritical text

in various styles. QRSR and DASAM specialize in speech processing and alignment for Arabic diacritical text and voice. SemSim stands out as a dual-space similarity explorer, analyzing numeric and semantic data. Our methodology involves advanced techniques in data modeling, data quality validation, deep learning, computer vision, signal processing, and interactive data visualization.

Our findings demonstrate improvements in addressing key challenges in Arabic Natural Language Processing (NLP). *For text processing,* we created an Automated Diacritization Deep Learning model. The model supports multiple Arabic diacritical styles, a unique feature in the field. Our best-performing model achieved a 94.2% accuracy rate in adding correct diacritics to Arabic text. *In image processing,* we built a specialized Optical Character Recognition (OCR) model for diacritic-aware Arabic text. Our OCR model reached an accuracy rate of 91.67% showing an improvement over the existing models. *For speech processing,* we developed two key systems. The first system, operating at the sentence level, combines our novel FuzTPI algorithm with machine learning models. This hybrid approach achieved up to 96% accuracy in audio segmentation and text-audio classification. Our second system focuses on word-level segmentation and alignment for Arabic diacritic-based speech. It achieved R^2 values of 0.959 for word start times and 0.957 for end times. These results show an improvement over existing Arabic speech recognition technologies.

The dissertation is structured around these three modalities, with each section detailing the challenges, methodologies, and results achieved in processing Arabic with diacritics. They have far-reaching implications for applications such as machine translation, information retrieval, speech recognition, natural language understanding, educational technology, and the preservation of linguistic heritage. By addressing the unique challenges of Arabic diacritics across multiple modalities, this research paves the way for more nuanced and culturally sensitive AI applications in Arabic-speaking contexts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

Chapter 1

INTRODUCTION

Imagine that you hear some news from the Arab region saying in your language: 'The president prevents all candidates from running in the election'. Reactions and analyses follow, shaping public opinion and policy. But because computer translations or AI interpretations were used, the outcome was a complete reversal of the meaning. The truth is that 'The president is prevented from running in the election'. The sentence "منع الرئيس الترشح" without diacritics is ambiguous, with both the meanings considered possible. The first meaning, The president stopped others from running, is written with diacritics as "مَنع الرئيس الترشح". Pronounced: "mana'a al-ra'eesu al-tarashuh". The second meaning, The president was stopped from running, is written in Arabic as "مُنع الرئيس الترشح". Pronounced: "muni'a al-ra'eesu al-tarashuh". Note that the same letters are used in both sentences, with only one diacritic mark differing. The misunderstanding of the Arabic language still persists today due to the negligence of diacritics. Arabic diacritics matter and our research focuses on processing them in three areas: text, audio, and images.

This introduction will guide you through the complexities of Arabic language processing. We will start by exploring the importance of Arabic and its unique features. Then, we will dive into the challenges posed by diacritics and linguistic styles. We'll examine how current

technologies struggle with these aspects. Next, we'll introduce our system designed to address these issues. We cover text, image, and speech processing for Arabic. We'll explain why we chose the Holy Quran as our case study. Finally, we'll outline our scientific contributions and the structure of this dissertation.

Arabic, spoken as a primary language in twenty-two countries and a second language in many other countries [81], holds significant historical and cultural importance. It is ranked as the fourth most used language on the internet [37] and is spoken by over 400 million people worldwide [9]. These numbers reflect the widespread use and influence of the Arabic language. Arab scholars played a pivotal role in advancing various scientific disciplines and expanding the boundaries of knowledge. Their contributions to astronomy, mathematics, medicine, and chemistry remain a testament to their impact on human history [138]. Additionally, Arabic heritage books serve as invaluable repositories of knowledge from ancient civilizations. Although these books were meticulously handwritten, their digital transfer often received inadequate attention [23].

The styles of the spoken Arabic language are usually classified into three main categories: 1) Classical Arabic (CA), which is used for the heritage books like the Quran (the Muslim holy book) and other literary texts, 2) the Modern Standard Arabic (MSA), which is the language of formal writing and conversation and is derived from the CA [19], and 3) the Dialectal Arabic (DA), which is the regionally varying informal colloquial language [80, 68, 84]. This research focuses on CA, as it represents the most detailed linguistic level from which all other levels are derived. Moreover, the wealth of Arabic heritage literature is preserved in books that use the CA style. The CA style uses diacritics in writing, while other Arabic styles don't. However, Arabs still pronounce words as if diacritics are present. This is important for AI models, they need to learn about diacritics. Then, the models must learn to work without the diacritics, mimicking how humans naturally handle Arabic.

In addition to the aforementioned styles of spoken Arabic, there has been an evolution in the Arabic writing styles. Figure 1.1 shows a progression from ancient to modern styles in

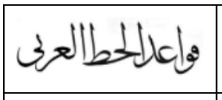
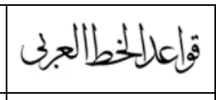
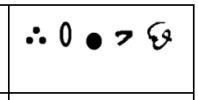
This Arabic manuscript dates back to over 1440 years ago.

the content of the same manuscript in its current form of writing

Figure 1.1: A comparison between ancient and modern Arabic writing styles.

written Arabic. Additional tiers have been incorporated over time to ensure accurate reading and to improve readability for non-native speakers [26]. Arabic writing is classified into five levels of details or annotations, with Level 5 featuring the highest amount of annotations, as presents in table 1.1. Subsequent lower levels show a decreasing level of annotations.

Table 1.1: Levels of details and annotations in the Arabic writing styles.

				
a) First drawing of writing.	b) Dotted letters.	c) Diacritic letters (tashkeel).	d) Pronunciation marks.	e) Tajwid marks

The first level, known as the First Drawing of Writing (FDW), is the simplest form and consists of 17 letter shapes without dots. The second level adds dots to letter shapes to differentiate between letters that have the same shape and, consequently, end up with an alphabet of 28 letters. Most current Arabic NLP techniques focus on the second level, which is the most commonly used. The third level includes diacritic marks (Tashkeel) to clarify meaning and pronunciation according to grammar rules. The fourth level introduces pronunciation marks that indicate variations in pronunciation, often associated with dialects. The fifth level incorporates *Tajwid* marks found in the Quran (the holy book of Muslims). This *Tajwid* marks indicate specific rules that the reader must follow to accurately pronounce the Arabic letters and to change the letter pronunciations according to the neighboring letters.

Although Arabic has a Unicode representation, the diversity of Arabic diacritics and writing levels leads to multiple Unicode representations for certain characters. For example,

The word: هب		
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

Table 1.2: Example of the orthographic and phonetic diversity for one word across different Arabic linguistic styles.

sometimes you find a character combined with a diacritic in a particular Unicode representation, while the diacritic exists separately. This complexity increases the difficulty of text comparison and processing. To address this challenge, our research proposes a model for mapping different Arabic Unicode representations, aiming to improve text search, accuracy, and consistency.

Additionally, the Arabic language has an additional layer of complexity beyond diacritics. This is called linguistic style. Linguistic style refers to correct dialects with well-known origins. These styles can change the diacritics of words. This reflects the speaker's regional background or adds new meanings to the text. In the provided table 1.2, the Arabic word "هب" 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 table illustrates that while some styles share the same form, others display notable differences.

There have been many advancements in NLP technologies in languages that use the Latin alphabet like English [177, 13]. However, the Arabic language is very special in the way it is written or even spoken. Despite these advancements, Arabic presents a new set of challenges

for NLP research [38]. This is because Arabic has its own distinct script and linguistic structure.

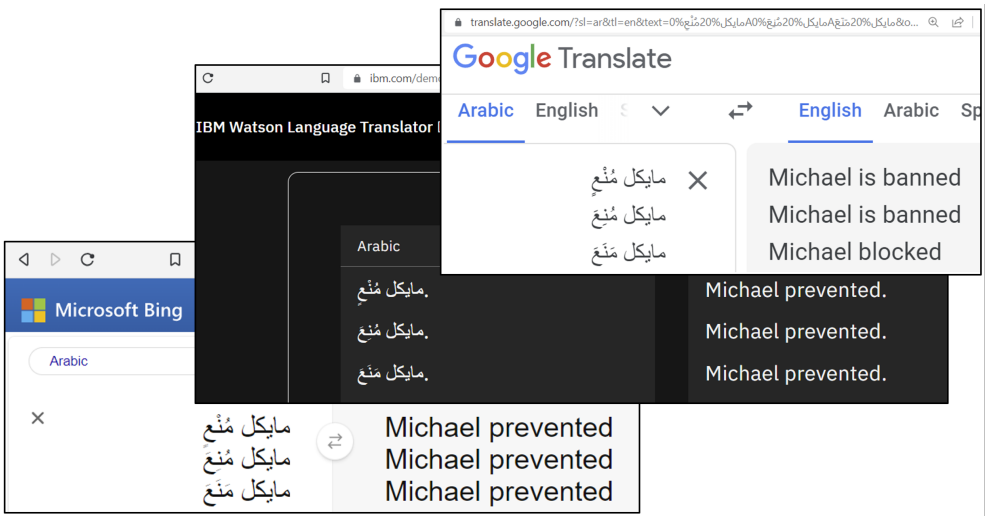


Figure 1.2: Wrong translation due to not recognizing the diacritical characters: Comparison between Google Translator, Bing Translator, and IBM Watson Translator

For example, مايكل منع written at the second level can have antonyms. With diacritics, مايكل مُنِعَ means Michael is *forbidden*. However, with diacritics مايكل مَنَعَ means Michael *prohibited something*, and مايكل مُنْعٍ means *Michael obituary (notice of death)*, someone. Figure 1.2 shows incorrect translation results from Microsoft Bing, IBM Watson Translator, and Google translation services. Figure 1.3 illustrates a challenge faced by Large Language Models (LLMs) [197]: translating Arabic text with diacritics. This unique aspect of the Arabic language can lead to inaccurate translations. This challenge is particularly relevant to my research, which seeks to improve the performance of dealing with Arabic diacritics text. GPT-4 (Generative Pre-trained Transformer 4) [140, 141], Meta's Llama 3 [132], Claude 3

Opus [24], and Google DeepMind's Gemini [76] all stumbled when translating Arabic text with diacritics into English. This failure, even on a simple task, highlights a crucial gap in the capabilities of current LLMs.

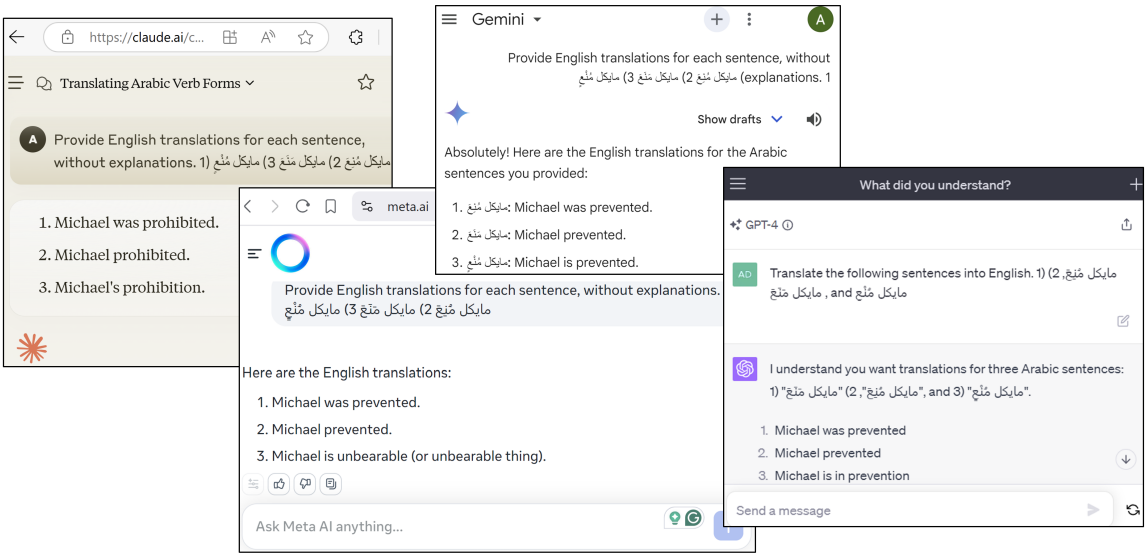


Figure 1.3: Demonstrating the limitations of current AI language models: Inaccurate translations of diacritical Arabic by GPT-4, Llama, Claude 3 Opus, and Gemini.

Recent research [13, 108, 91, 19, 6] supports this observation. It comprehensively evaluates ChatGPT's performance on Arabic NLP tasks, including Modern Standard Arabic (MSA) and dialects. The findings reveal that ChatGPT struggles significantly with Arabic tasks despite its success in other languages. This reinforces the idea that current LLMs face limitations in accurately processing Arabic, particularly when diacritics or dialects are involved.

To address the challenges in processing Arabic text with diacritics, we present a solution in Chapter 4 of this dissertation. We have developed an Automated Diacritization Deep Learning model that can add diacritics to text lacking them. This model supports multiple Arabic diacritical styles, contributing significantly to bridging the gap in processing Arabic texts with diacritics. By addressing this issue, we hope to contribute to advancing AI language processing for Arabic.

Arabic language processing faces unique challenges, especially with diacritics. These chal-

lenges extend beyond text to image processing. Optical Character Recognition (OCR) for Arabic text is particularly difficult [67]. Most of the heritage Arabic books are handwritten and need digitization. These books often retain diacritics, which are extremely small compared to letters. This adds another layer of complexity. Current OCR services struggle with Arabic diacritics. Figure 1.4 shows how Google Cloud API [46], Amazon Textract [167, 86] and Microsoft Azure Cognitive Services [134] struggle to recognize the Arabic diacritic marks. Most OCR systems focus on MSA, which lacks diacritics [67]. CA with diacritics receives less attention. Recall Michael's example from earlier. It demonstrates why supporting diacritics is crucial for correct understanding and conversion. OCR systems without diacritic support may misinterpret marks as letter dots or other features. This leads to incorrect text recognition. To address this gap, we've dedicated Chapter 6 to developing an Arabic Diacritics OCR system.

Figure 1.4: Limitations of Arabic text recognition using the current OCR technologies.

Moving from image to sound processing, we encounter new challenges in Arabic language technology. Speech recognition for Arabic requires significant improvement [152]. We tested several systems using clear, noise-free audio of a tuned Arabic voice. Figure 1.5 demonstrates that existing speech recognition systems produce errors and fail to support diacritical characters. Google Cloud Speech-to-Text [45], Amazon Transcribe [165], and Microsoft Cognitive Services [133] generated text without diacritics and with numerous letter-level mistakes.

Text-to-Speech (TTS) for Arabic also faces challenges. A recent survey [42] highlights the scarcity of Arabic TTS research, especially for classical Arabic and dialects. This indicates weak support for Arabic TTS research overall.

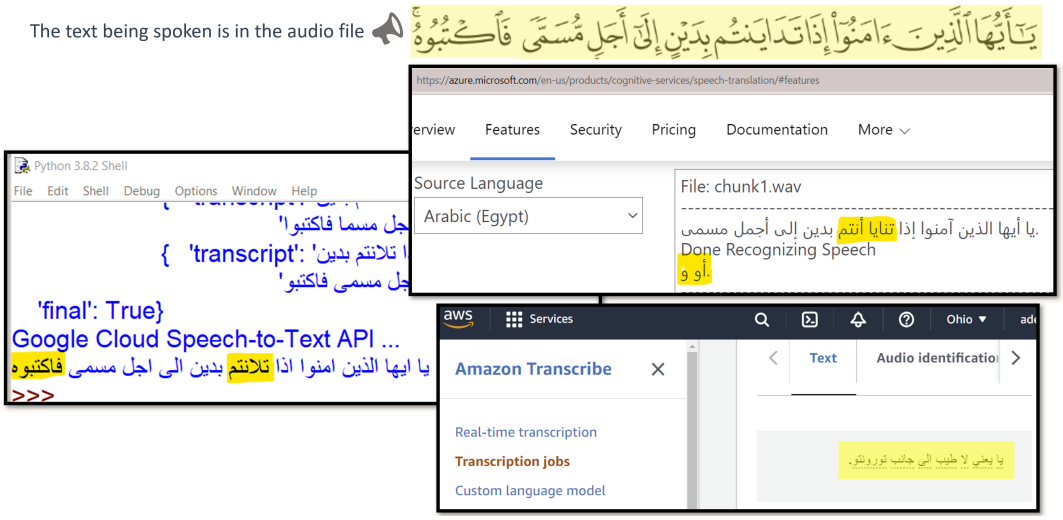


Figure 1.5: Errors and lack of support for diacritical characters in Arabic speech recognition.

Research shows that a lack of large diacritic-related datasets hinders Arabic speech processing progress [81, 152]. Such datasets are crucial for improving speech-processing systems through better training and evaluation. In Chapters 7 and 8, we present speech-processing systems to address this gap. These systems focus on linking text with diacritics to the audio at both sentence and word levels. We've developed intelligent models for segmentation and alignment, creating diacritic-aware speech and text processing in Arabic. We aim to bridge the existing gap and pave the way for enhanced Arabic speech-processing solutions.

While natural language processing (NLP) has made significant progress in English and other Latin languages, the Arabic language poses unique challenges [16]. Its complex structure includes diacritics for each letter, a rich grammatical system, and multiple styles of applying diacritics. These features make Arabic different and require special attention [150]. For example, In Arabic, many words derive from a common root, typically three or four consonants that carry the core meaning. Various related words are formed by adding vowels and other

letters around these root consonants, creating families of words with connected meanings. For example, the root k-t-b relates to writing:

- kataba (he wrote)
- kitāb (book)
- maktab (office/desk)
- kātib (writer)
- maktabah (library)

These words contain k, t, and b in various arrangements, creating "numeric similarities"—patterns in the number and arrangement of certain letters across related words. This characteristic is less common in languages using Latin script and can be leveraged as a feature in Arabic natural language processing tasks. In Arabic, these root letters are combined with specific patterns of vowels and additional consonants to form different words. The 'numeric similarities' refer to the consistent patterns in the number and arrangement of letters when forming related words from the same root. For example, the root letters k-t-b can be placed into various templates to create words with related meanings, as shown above. This systematic approach means that words sharing the same root not only have related meanings (semantic textual similarity [181]) but also share structural patterns in their letter arrangements (numeric similarity). This dual relationship between form and meaning is a distinctive feature of Arabic. Understanding and utilizing both numeric and semantic similarities can greatly enhance natural language processing tasks for Arabic by capturing these unique linguistic patterns.

However, ready-made functions imported from other languages and current exploratory methods may not always meet the unique needs of the Arabic language. This is especially true for aspects distinguishing Arabic, such as the association between repeated letters and word meanings. Standard methods often fail to effectively capture these numeric patterns in Arabic.

To address this gap, we propose **SemSim** (Semantic Similarity), a customizable system that enables users to define their functions or tailor text similarity measures, incorporating both numeric and semantic aspects. For example, users can create a custom function that specifically calculates similarity based on shared root letters and their arrangements. This allows the detection of relationships between words like 'kataba' (he wrote) and 'kitāb' (book) by focusing on their common root 'k-t-b'. Such user-defined functions enable the analysis to account for the unique morphological patterns of Arabic.

This new contribution enables users to focus on the specific characteristics relevant to their work or perspectives, enhancing the effectiveness of Arabic natural language processing tasks. By allowing customization through UDFs, our work distinguishes itself from existing projects that rely solely on standard methods, which may not capture the intricacies of the Arabic language.

The technology transfer of NLP techniques from English to Arabic is not straightforward. For example,the letters of Arabic can have multiple shapes. The same letter can have multiple shapes based on its location in the word. Moreover, the different writing styles and annotations are unique to the Arabic language. The same word can have different meanings or different derivatives based on the diacritic marks added to the word.

Modern systems cannot handle it correctly by merely adapting existing models from other languages [2]. Although transfer learning from models trained in other languages can be beneficial, it has limitations. [58, 73]. Specifically, it does not fully address the unique challenges posed by Arabic. The existing models can be effective for some linguistic properties shared across languages. However, our work focuses on four distinctive features of Arabic that pose unique challenges:

- Diacritics.
- Multiple linguistic styles for the same text.
- Numeric Similarity and its relationship to Semantic Similarities.
- Pronunciation Rules: These rules determine how long or short each letter should be

pronounced. The length of pronunciation depends on the diacritics, the letter's position, and other factors. Unlike many other languages, Arabic has clear and consistent rules for pronunciation.

These features are not prevalent in English or other Latin-based languages. While transfer learning can provide a starting point, models that don't account for these unique features may struggle to extract relationships based on these properties. Recognizing Arabic text with diacritics from images and speech introduces additional complexity. AI models must be aware of the unique properties of Arabic to process it effectively. These challenges call for additional efforts to enhance technologies for the Arabic language. In response to these needs, this research aims to provide a Multimodal Integration System as a support system for the Arabic language. The components of this system are explained in detail in Chapter 3.

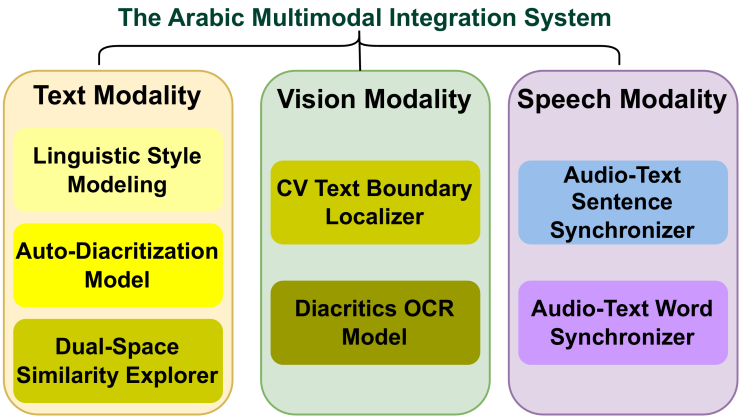


Figure 1.6: The proposed Multimodal Integration System, illustrating the integration of text, vision, and speech modalities.

The proposed multimodal integration system shown in Figure 1.6 integrates speech, text, and vision modalities. The proposed system is expected to lay a solid foundation that future NLP systems can build on top of. The proposed system is also expected to be an open-source cloud-based system where the proposed functionalities are offered as services for NLP systems to connect to and utilize. Figure 1.6 illustrates the three main modalities: Text, Vision, and Speech.

1. Text Modality: In the Text Modality, we address key challenges in processing Arabic text. We employ advanced techniques from data modeling, deep learning, and interactive data visualization. Our aim is to manage the diversity of linguistic styles and the complexity of diacritics in Arabic. The components under the Text Modality include:
   (a) **Linguistic Style Modeling:** We developed a data provenance model to record and analyze different linguistic styles within a single dataset. Implementing a data provenance tracker verifies authenticity [115] by collecting lineage metadata [72]. Our model maps relationships between narrators, their choices, and the resulting text, providing insights into the evolution of linguistic styles over time. We employed Directed Acyclic Graphs (DAGs) and interactive visualization techniques to explore how linguistic styles have evolved. This work is detailed in Chapter 4.
   (b) **Auto-Diacritization:** By exploiting the diversity of diacritics across different linguistic styles, we build an auto-diacritization models. Our system uses deep learning architectures like Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), and Transformer models. These models are trained to predict the correct diacritics and stylistic nuances in Arabic text, improving the accuracy of Arabic text processing systems. This component is also discussed in Chapter 4.
   (c) **Dual-Space Similarity Explorer:** We propose a system, called *"SemSim"* for *Sem*antic *Sim*ilarity, to assist users to explore similarities within multidimensional textual data sets. *SemSim* allows users to define entities, entity groups, and dimensions. The system then helps users explore numeric similarities across various dimensions and correlate these numeric similarities with semantic concepts provided by a knowledge graph. SemSim contributes to a deeper understanding of the relationships between the semantic and numeric aspects in Arabic text. This work is presented in Chapter 5.
2. Image Modality: In the Image Modality, we address key challenges in processing Ara-

bic text within images. We use techniques from computer vision and deep learning. Our goal is to accurately identify sentence boundaries and recognize the letters and diacritical marks in Arabic text images. The components under the Image Modality include:

(a) **CV Text Boundary Localizer:** We developed a computer vision-based boundary localization system to identify sentence boundaries in images. This system utilizes Detectron2's Faster R-CNN network as the base model. By accurately detecting sentence boundaries, we pave the way for more advanced Optical Character Recognition (OCR) applications. This work is detailed in Chapter 6.

(b) **DL-Driven Diacritics OCR:** We built a deep learning model called AraDiaOCR to recognize the small diacritics surrounding Arabic letters in images. OCR, or Optical Character Recognition, is the process of converting image text into machine-readable text. Our CNN-based (Convolutional Neural Network) OCR model specifically focuses on Arabic diacritic recognition. This work is also discussed in Chapter 6.

3. Speech Modality: In the Speech Modality, we focus on enhancing Arabic speech recognition and audio-text alignment. We use techniques in signal processing and machine learning. Our goal is to accurately synchronize audio with diacritic-annotated text at both sentence and word levels. The components under the Speech Modality include:

   (a) **Audio-Text Sentence Synchronizer:** We developed a system called QRSR (Quran Recitation Sentence Recognition) to synchronize audio recordings with their corresponding text at the sentence level. This system tackles the challenges of segmenting and aligning Arabic speech data. We proposed the Fuzzy Text Alignment and Rule-based Classifier (FTARC) approach, which combines the FuzTPI algorithm with machine learning models like Naive Bayes, Support Vector Machine (SVM), and Random Forest. This work is detailed in Chapter 7.

   (b) **Audio-Text Word Synchronizer** We created DASAM (Diacritic-Aware Seg-

mentation and Alignment Model for Arabic). DASAM focuses on word-level segmentation and alignment in Arabic speech. It's designed to work with diacritic-marked Arabic text. The model uses linguistic analysis based on intonation rules. It applies Dynamic Time Warping (DTW) to match reference audio words with unseen sentence audio. This work is presented in Chapter 8.

The scarcity of adequate systems and datasets has been a significant obstacle in Arabic NLP development [91, 30, 81]. To tackle this, we've chosen the Quran as our case study, a text preserved meticulously by Muslims for over 1400 years. The Quran serves as both a safeguard for the Arabic language and a record of its writing evolution. Its exceptional accuracy and rich meanings make it valuable for research, but what sets it apart is its 20 linguistic styles, known as "Riwayat" or "Qira'at". These styles, which vary in diacritics, dotted letters, and dialects, offer a unique opportunity for study. To our knowledge, no other dataset contains multiple linguistic styles of the same text, making the Quran ideal for our research. The Quranic dataset serves as a foundational element of high quality for model training and evaluation. Its precision and linguistic depth are remarkable, despite its relatively modest size. To balance quality with quantity, our dataset encompasses seven unique linguistic styles, expanding its scope. The Quranic dataset serves as a foundational element of high quality for model training and evaluation. Its precision and linguistic depth are remarkable, despite its relatively modest size. To balance quality with quantity, our dataset encompasses seven unique linguistic styles, expanding its scope.

The contributions of the proposed system can be categorized and summarizes as follows. The overall system's contributions:

- We build a framework supporting Arabic and Quranic research, encompassing text analysis, computer vision, speech recognition, and semantic analysis.
- We develop a scalable and accessible cloud-based database of the Quran, encompassing Arabic text, images, and audio tracks, to serve as a centralized resource available to researchers worldwide.

- We build an extensibility framework that enables developers and researchers to perform advanced queries, and codes, and publish results with accompanying documentation.
- We facilitate and integrate research endeavors by providing diverse Arabic datasets and tools, enabling researchers to explore and analyze Arabic and Quranic texts effectively.

Text Analysis Contributions:

- We compiled a database encompassing multiple linguistic styles of the Arabic Quran text, making it accessible to researchers and practitioners.
- We develop a novel model to trace the evolution of Quranic linguistic styles through narrators and their lineages.
- We design an algorithm to ensure consistent representation of Arabic characters across texts.
- We build an Auto-diacritical system for Arabic text named QRDiaRec.
- We utilize deep learning models to achieve effective Arabic diacritization.
- We build an interactive tool to visualizes narrators' relationships and linguistic styles, aiding researchers in exploring complex linguistic data.
- We achieved effective Arabic diacritization with an accuracy of 94.2% using advanced deep learning models.

Semantic and Numeric Analysis Contributions:

- We develop a model to define multidimensional entities and entity groups in an environment characterized by heterogeneity in dimensions.
- We build a system, called *SemSim* (for *Sem*antic *Sim*ilarity), that helps the user explore numeric similarities between entities that span different entity groups and, possibly, across different dimensions.
- Guided by a knowledge graph that represents the semantic similarities between entities, the system alters the user when a numeric similarity is found between entities that already show semantic similarities.

- The system detects the numeric dimensions that impact the semantic similarity and uses these impactful dimensions to further detect hidden semantic similarities
- We use the text of the holy Quran as a case study to explore the semantic and the numeric similarities over chapters, verses, and words.

Contributions to the Computer Vision of Diacritic Arabic:

- We present QR-Vision, a neural network system that detects verse markers and chapter beginnings in Quran images with high accuracy.
- We develop an algorithm to automatically localize and extract verses across different Quranic linguistic styles and printings.
- We create an OCR system capable of recognizing Arabic text with diacritics, achieving an accuracy of 91.67%.
- Facilitation the creation of a vast dataset of images containing Arabic sentences with diacritics and their corresponding text, aiding OCR systems in accurately recognizing Arabic diacritics.

Contributions to the Speech Recognition of Diacritic Arabic:

- Development of the Quran Speech Recognition (QRSR) system, utilizing Quran audio and text to advance Arabic speech recognition and contribute to natural language processing (NLP) systems.
- Application of various classification algorithms, including ML models, to classify the un-diacritic Automatic Speech Recognition (ASR) transcript into subsentences aligned with diacritic-annotated Arabic sentences.
- Alignment of segmented audio with diacritic-annotated Arabic text representations using existing Speech Recognition systems, addressing the challenge of diacritic-aware recognition not being supported by current Arabic Speech Recognition systems.
- Introduction of the FuzTPI algorithm, which enhances ML models and improves the alignment-driven classification process in Arabic speech recognition.

- Application of FuzTPI-SVM to boost the performance of the QRSR system, achieving an accuracy rate of 92.2%.
- Contribution to the development of an expanded textual audio dataset, offering potential enhancements for Arabic speech recognition on a broader scale.
- Development of DASAM (Diacritic-Aware Segmentation and Alignment Model for Arabic), a novel system for word-level audio segmentation and alignment with diacritic-annotated text.
- Implementation of a word-level alignment technique using Mel-Frequency Cepstral Coefficients (MFCC) extraction with linguistic features, enabling precise identification of word boundaries in audio.
- Achievement of superior word timing prediction accuracy compared to industry-standard ASR systems, with R^2 values of 0.959 and 0.957 for start and end times respectively.

This research advances the broader field of Arabic NLP by addressing unique challenges such as diacritization and linguistic style variations. It provides a framework for text, vision, and speech processing modalities. The development of scalable databases and models like QR-Vision, QRDiaRec, QRSR, DASAM, and SemSim, along with accessible resources, empowers researchers and practitioners. These contributions address key challenges in Arabic NLP and enhance accuracy in processing Arabic language. They pave the way for improved applications in machine translation, information retrieval, and speech recognition.

This dissertation is structured into eight chapters centralized around the study of Arabic Diacritic letters. Chapter 1 introduces the research and sets the context. Chapter 2 reviews related work. Chapter 3 outlines the architecture of our proposed support system for Arabic Natural Language Processing. Chapters 4 and 5 delve into the textual analysis; the former focuses on automated diacritization and stylistic realization, while the latter explores semantic similarities using a multidimensional approach. Chapter 6 introduces an OCR system that identifies Arabic Diacritic letters through computer vision and deep learning. Chapter 7 covers diacritic-aware speech and text processing, including segmentation, classification, and

alignment. Finally, Chapter 9 concludes the dissertation and points to future directions in the research.

Chapter 2

RELATED WORK

This section provides an overview and classification of related work aligned with the research areas discussed in this dissertation. The upcoming sections offer a comprehensive overview of the related work in diverse domains, including the potential of Arabic Language Processing, Similarity Detection in Multidimensional Heterogeneous Environments, Automated Text Boundary Localization using Computer Vision, advancements in Arabic Speech Recognition, and databases of Arabic heritage resources.

2.1 Support Systems for Arabic NLP

The research conducted by Bashir et al. (2023) [31] is a systematic review that focuses on Arabic NLP for Qur'anic research. It is considered the first comprehensive survey in this field. The researchers conducted a survey of research papers from 2005 to 2022 and selected 70 papers. They provided charts highlighting the limited availability of research in the areas of Quranic Corpus, Interface Systems, Classifiers, Translations, and other areas. The statistics presented inspired us to develop a support system that serves as an infrastructure for these areas.

Upon reviewing the Qur'anic NLP tools & resources and the associated findings, the researchers emphasized the necessity of developing specialized tools. These tools should be specifically designed for Quranic and Arabic research.

Additionally, they highlighted that generic systems and tools designed for other languages, such as English, often yield peculiar and unforeseen outcomes. This realization served as a motivation for us to develop a platform that offers specialized systems and tools specifically tailored to the Arabic language.

The researchers discussed various fields of NLP and categorized the research papers accordingly. They emphasized the need for multimodal NLP models trained on text, audio, and images. Consequently, our research addressed this gap by emphasizing the support of multimodal NLP models on diverse modalities, including text, audio, and images.

The researchers highlighted the abundance of ontological works in the field. However, they noted that these works often focus on specific and limited topics, lacking comprehensiveness. Additionally, they raised awareness of the limited presence of deep learning research in supporting Arabic NLP, despite its anticipated success. This observation motivated us to incorporate deep learning techniques in certain aspects of our own research.

The researchers noted that the majority of research directions were solely focused on one linguistic style, namely Hafs, without considering the rich variety of linguistic styles present in the Qur'an. They expressed their view that this limitation disregards valuable linguistic diversity. In contrast, our own research addressed five different linguistic styles, encompassing various aspects such as text, statistics, images, and phonetics. Moreover, different Arabic writing styles for the same text pose a challenge that hinders achieving accurate results. Our research aims to address these issues through the Text Digitizer and Analyzer Module. The module is designed to map different Arabic Unicode representations to a unified representation, providing a solution to the identified challenges.

Our system drew inspiration from the previous system developed for environmental sciences, known as AMADEUS by Hendawi et al. (2014) [87]. The authors extended the idea to the direction of Arabic and Islamic Databases. AMADEUS stands for Azure Marketplace of Applications for Diverse Environmental Use as a Service. This system can consume data from a wide variety of environmental data sources, and store them in one integrated data store. To guarantee the availability and accessibility of this data, AMADEUS hosts all data in the cloud. The system gives the users the luxury to query and analyze the underlying data sets by writing SQL queries and/or through the system's user-friendly graphical user interface. The system also gives the users the ability to study the correlation between multiple data sets

using the algorithms and data structures natively offered within the framework. The users of the system share and exchange not only datasets but also libraries of code and modules of various machine learning algorithms.

Our proposed system, QuranResearch.ORG [163], follows to some extent the footsteps of AMADEUS. It aims to provide the space and tools that ease, support and integrate the efforts of researchers in the field of Quran computational research. QuranResearch.ORG offers a public back-end database server. This server includes an indexed representation of the Quran text. Additionally, it provides a set of general-purpose tools and packages. These tools enable users to perform counting, searching, and mining of the Quran text. QuranResearch.ORG invites researchers to send their queries against our public back-end database server and to, retrieve/analyze/share their results through the system's front end. The database schema is documented and published publicly on the system's website (QuranResearch Documentation, 2022). The proposed system is designed to serve as a "wiki" style repository for all the results that researchers. It would obtain and reflect on their efforts in the Quranic research

2.2 The Potential of Arabic Language Processing: State-of-the-Art and Challenges

Starting with an in-depth survey of Arabic Natural Language Processing (ANLP) by Guellil et al. (2021) [79], 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 conducted by Marie-Sainte et al. (2018) [131] explores the involvement of machine learning techniques in developing tools for Arabic NLP. The study discusses the characteristics and complexity of the Arabic language, as well as the challenges and needs of ANLP. The authors highlight the challenge of having a gold-standard corpus with tagged text. They emphasize the lack of specialized tools for data collection, particularly in the Arabic language, which includes corpora and gazetteers.

Regarding sentiment analysis applications, the researchers point out the difficulty in identifying the sentiment direction of Arabic terms in tweets. They mention the existence of basic tools that have been developed by ANLP researchers to process Arabic text. However, due to heterogeneity and interoperability issues, researchers often have to build these tools from scratch for their specific projects. The authors stress the need for tools that are similar, interoperable, and can be used in a unified manner across projects.

In relation to our research, we align with the recommendations provided by the researchers. We aim to address the challenges faced in ANLP, including the lack of specialized tools for data collection and the need for interoperable tools. Additionally, our focus may differ from sentiment analysis. However, we have made progress in dealing with short Arabic texts through our Fuzzy Text Alignment and Rule-based Classifier (FTARC) approach.

2.3 Databases of Arabic Heritage Resources

Kadir et al. (2020) [104] conducted a study on the manuscript of the Turkish Quran. It aims to store the manuscript in a Database Management System (DBMS). The preservation of Qur'anic manuscripts holds importance. These manuscripts are not only religious texts but also valuable cultural artifacts. The history of some of these manuscripts spans over 1,400 years. Their research focuses on the storage and preservation of a specific historical copy of the Quran. In contrast, our research aims to create a comprehensive database. The

database is capable of storing both historical and modern versions of the Quran. Additionally, it facilitates comparisons between these versions.

Kadir et al. focus on preserving scanned images of Qur'anic manuscripts. We are also interested in preserving these images but with the associated texts, and phonemes. Additionally, we aim to establish connections between phonemes and texts within the scanned manuscript pages. By expanding the scope, we aim to enhance the accessibility of the Quranic texts and their interconnected components. Furthermore, our research seeks to improve the analysis and preservation of these texts.

In a study conducted by Oktaviani et al. (2019) [139], the authors designed a Quran database. Their database was an on-device database for mobile applications. In contrast, our research focuses on developing a server-side DB. Our database is accessible through an API, enabling its utilization by various types of applications. It also overcomes storage limitations on devices.

While the researchers acknowledged the presence of incomplete data and expressed their pursuit of reliable data. In our research, we have developed a web scrapping module. This module extracts data from different websites. It includes complete copies of the Quran in various linguistic styles. Furthermore, we have implemented algorithms to compare different versions, detect errors, and validate the data.

Both studies stored the translation of the Quranic text. However, in our research, we have divided these translated sentences. This division allows us to display them using different count scheme methods. The reason behind this approach is that counting systems can alter the beginnings and endings of sentences for the same text.

While the researchers mentioned the provision of audio material, they did not provide detailed information. They did not explain how the audio was stored or linked to the corresponding texts. In our research, we collected phonetic data for multiple linguistic styles of the Qur'an. Additionally, we developed audio segmentation tools that enable us to adapt the audio display based on various counting schemes.

The research conducted by ElSayed (2015) [64] focuses on designing an Arabic natural language interface system for a database of the Quran. The author's database system is designed specifically for the linguistic style of Hafs. The database is obtained from a single website without data verification or data lineage tracking. In contrast, our research is different from ElSayed's in several aspects. Firstly, we download data from multiple websites. Secondly, we implement algorithms for copy comparison, error detection, and data validation. Moreover, our database encompasses various linguistic styles of the Quran. Both ElSayed's study and our research share the goal of presenting statistics on Quranic data. However, our research goes a step further by providing statistics based on multiple linguistic styles of the Quran. Additionally, we offer a variety of chart options that allow users to choose the type of visualization that best suits their needs.

2.4 Automated Diacritization and Stylistic Realization in Arabic Textual Data

The research by Abandah and Abdel-Karim (2020) [1] 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 impressive 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) [66] 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%.

Alasmary et al. (2024) [14] introduced CATT, an Arabic Text Diacritization system leveraging character-based transformers and the Noisy-Student approach to achieve high accuracy. Their work focuses on Modern Standard Arabic, evaluating models on the WikiNews and CATT datasets.

QRDiaRec, while also an Arabic diacritization system, diverges significantly. It generates multiple correct diacritic patterns for the same word, reflecting the seven linguistic methods it is trained on, unlike CATT, which outputs a single diacritic per word. This difference stems from QRDiaRec's training on linguistic diversity.

Moreover, CATT utilizes a transformer-only architecture, while QRDiaRec integrates GRU, LSTM, and Transformer models. This multi-model approach allows QRDiaRec to evaluate different architectures comprehensively. The dataset focus also contrasts: CATT relies on one linguistic style, while QRDiaRec incorporates seven, enhancing its versatility for Arabic text processing.

The research conducted by Belinkov and Glass (2015) [33] 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) [65] 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%.

In comparison to previous works on this topic, our research utilizes character-level sequence-to-sequence models, including bidirectional LSTM, bidirectional GRU, and transformer-based neural networks. Additionally, our research goes beyond diacritization alone. We specifically address the recognition of different linguistic styles (such as dialects) in the Arabic language, which was not addressed in their research. In terms of performance, the models employed in our research achieved accuracies between 91.5% and 92.2%. However, it is important to note that achieving perfect accuracy is challenging due to the presence of multiple valid forms of diacritization in Arabic sentences. Standard evaluation metrics for diacritization may not be adequate when multiple correct diacritizations exist due to different linguistic styles. Similar to the challenges noted by ElNokrashy and Alkhamissi (2024) [63], who proposed novel metrics like Partial Diacritic Error Rate, we faced difficulties using prevailing metrics. Therefore, we adopted standard metrics such as accuracy, precision, recall, and F1-score to evaluate our models.

2.5 Automated Text Boundary Localization using Computer Vision

We overview related work along two main lines. First, we overview some research directions that utilize Machine Learning techniques. To leverage digital access and verify the authentication of the digital Quran copies. Second, we highlight the Deep Learning Models that we used in our computer vision approach to detect verse boundaries.

2.5.1 Machine Learning Techniques for Text Boundary Detection and Verification

There are several research directions [162], [113], [21], [83] that focus on validating the authenticity of the Quran verses that are published online. These directions use a form of a matching algorithm, e.g., Boyer Moore algorithm, for verification purposes. Once verification is complete, a watermarking technique is used for the purpose of tamper identification. There are many other types of research around detecting Quran words and verses from text such as [162] using support vector machines, and [61] which uses a tree-linked hash table data structure to facilitate the matching process. In [100], convolution techniques are used to detect Harfu Jar from Quran images, while in [157], convolution techniques and Bray Curtis distance are used to detect and classify the different Tajwid rules in Quran images.

2.5.2 Deep Learning Models for Computer Vision

Convolution techniques have been the backbone of recent computer vision neural networks. Convolutional neural networks (CNNs) have witnessed tremendous improvements over the last few years. One of the major improvements in this area has been the introduction of Fast R-CNN [121] and Faster R-CNN [59]. Faster R-CNN achieved state-of-the-art results on the COCO dataset [118] which is the standard dataset used in object detection tasks.

Recently, transformer-based neural networks have been increasingly used in the field of computer vision such as ViT [155] and Swin Transformer [74]. The Swin Transformer uses a shifted window with limited attention to be able to capture features at different scales to achieve better results on segmentation and detection tasks.

The research conducted by Al-Sheikh et al. (2020) [12] sheds light on the challenges involved in developing OCR systems for Arabic text. The study emphasizes the complexity faced in this task and the need for effective solutions. They emphasize the significance of datasets in shaping OCR systems and explore state-of-the-art approaches in segmentation and recognition within the Arabic domain. The review identifies a research gap in recognizing diacritical image texts, particularly in the context of the Quranic text. It emphasizes

the need for further investigation and exploration in this specific area of research. It also acknowledges the scarcity of publicly available datasets specifically designed for diacritical fonts or Quranic image text recognition. Furthermore, the study presents an Arabic OCR dataset that encompasses diverse data types and showcases its potential for automated OCR using machine learning and deep learning approaches. The effectiveness of Recurrent Neural Networks (RNNs), such as LSTM and GRU, is highlighted in Arabic text recognition.

By incorporating the insights from this review, our research builds upon deep learning techniques, specifically Automated Text Boundary Localization using Computer Vision, to develop a dataset comprising Arabic sentences with diacritics and corresponding images. This dataset serves as a valuable resource for advancing Arabic OCR systems and addressing the challenges associated with cursive scripts and diacritics in Arabic text recognition.

2.6 The Arabic OCR Systems and Models

Salah Alghyaline's (2023) [18] comprehensive study on Arabic Optical Character Recognition (OCR) over the last decade serves as an essential review for researchers in the field [18]. The review highlights the unique characteristics of the Arabic language and presents different types and stages of Arabic OCR systems, detailing researchers' contributions to each step, the metrics for OCR evaluation, and existing datasets with their attributes. A critical finding for our work is the acknowledgment that most existing Arabic OCR approaches ignore diacritics, an aspect that emphasizes the weak support for Arabic diacritic letters. The review also includes a comparison of existing methods in terms of recognition accuracy, evaluating both research methods and commercial systems like Sakhr, ABBYY, RDI, and Tesseract. Four commercial and open-source software (Tesseract, Abbyy FineReader, Sakhr, and Readiris) were particularly surveyed, and their recognition rates ranged from 70% to 80%, reflecting a need for further improvement. For example, the recognition accuracies for Sakhr, ABBYY, RDI, and Tesseract were found to be 51.56%, 75.19%, 46.00%, and 48.61%, respectively, using Arabic transparent font type. Their review shows that current commercial OCR software for printed Arabic text does not exceed 75% accuracy. Most approaches work

offline and do not recognize Arabic script in real time. The focus on printed characters or isolated words in controlled environments, rather than page-level scripts, and the negligence of diacritics, make the recognition task even more demanding. This research area remains open, and Salah Alghyaline's review can be valuable in informing enhancements to existing systems, especially those that align with our work on diacritics in the Arabic language.

The research by Aziz Qaroush et al. (2022) [148] presents a simple and robust algorithm for text-line extraction from printed Arabic text, specifically addressing challenges such as font variations, diacritics, overlapping, and touching text lines. The algorithm is divided into two stages: preprocessing and text-line extraction. It employs baselines, projection profiles, and a top-down divide and conquer technique to efficiently handle non-uniform inter-line spacing and overlapping. Our approach may utilize the deep learning approach, whereas this research focuses on a learning-free divide-and-conquer method. They deal explicitly with printed Arabic text, our research targets the handwritten and diacritics letters.

The research by Kiessling et al. (2021) [109] presents an examination of the Kraken OCR engine's performance on Arabic script, with a particular focus on achieving highly accurate Arabic OCR. The study details the use of a hybrid convolutional and recurrent neural network, avoiding the need for character-level segmentation, and emphasizes the importance of systematic training data production and technological improvements. Comparatively, our system, DiaArOCR, specifically targets the detection of diacritic marks in Arabic and employs a DDL CNN architecture. Our DiaArOCR encompasses an OCR process that includes Preprocessing, Segmentation, Recognition, Postprocessing, and Evaluation, with a focus on handwriting diacritics in Arabic text. While Kraken's study primarily revolves around the analysis of error instances and methods to enhance overall OCR accuracy, DiaArOCR's focus on Arabic diacritics detection through DDL CNN sets it apart.

Fasha et al. (2020) [69] focused on printed text, including fonts that simulate handwritten styles. In contrast, our research deals with handwritten text in Ottoman handwriting. This is more challenging because the same letter is rarely drawn by hand in the same way. Their

goal was to recognize different fonts, extending to word-level and letter-level recognition. Our focus is different. We focus on handwritten text in the Ottoman font, with a diacritic level.

Their system uses a hybrid CNN-BDLSTM model. However, no specific justification for this choice was provided. We use CNNs alone. This is due to their accuracy in analyzing spatial features, especially for handwritten text and diacritics. CNNs have also proven their accuracy and efficiency in image-based text recognition. They reported a letter recognition accuracy of 90.22% on training data. This dropped to 85.15% on test data. There were further declines at the word level, although specific percentages were not provided. Our model achieved a diacritic recognition accuracy of 91.90% on training data. This dropped slightly to 91.67% on test data, demonstrating better consistency.

The researchers Darwish et al. (2020) [50] developed a new Arabic OCR (Optical Character Recognition) model specifically for recognizing printed Arabic text. They combined Genetic Algorithm (GA) with the Fuzzy K-Nearest Neighbor classifier (F-KNN) to enhance identification accuracy. In this method, GA handled feature selection, optimizing convergence, and spreading solutions, while F-KNN was responsible for classifying ambiguous or uncertain objects. The model utilized bio-inspired feature vectors and the fuzzy KNN classifier for precise class membership. Achieving a high recognition accuracy of 98.69% for various samples in minimal time, the model emphasized reducing errors, shortening run time, and simplifying the structure. While it successfully managed overlapping Arabic words, it did not recognize handwritten text, indicating a direction for future work. The difference between this research and our work is seen at the approach level; they applied genetic algorithms, while we employ a deep learning approach. The goals diverge, as they target printed text, whereas we concentrate on handwritten text. They prioritized speed, unlike our emphasis on understanding diacritic letters. Their system's inability to recognize handwritten text underscores the significance of our work, highlighting a key area where our research contributes.

Alghamdi and Teahan's (2017) [17] work seeks to evaluate Arabic OCR systems and identify

areas for improvement, while DiaArOCR is aimed at diacritics detection within Arabic text. The study identifies challenges in Arabic OCR, including issues with diacritics and complex fonts like Diwani Letter font, and concludes that recognition accuracy is generally below 75%. They evaluated four Arabic OCR systems: Sakhr, ABBYY, RDI, and Tesseract. Interestingly, the paper indicates that two of the evaluated systems could not recognize any diacritical marks. This contrasts with DiaArOCR's specialized approach to diacritics recognition, achieving high accuracy. Their research highlights that diacritics recognition is still an open research problem.

2.7 Advancements in Arabic Speech Recognition: Current Status and Challenges

We provide an overview of speech recognition research encompassing three primary areas. Firstly, we examine research endeavors concerning Arabic text classification. This exploration is motivated by the need to address the challenge of classifying ASR transcripts containing Arabic text. Then the Arabic speech recognition, where we investigate the difficulties associated with voice recognition in the Arabic language. Furthermore, we review the latest advancements achieved in this domain. Lastly, we discuss Quranic speech recognition, which focuses on audio analysis applied to a case study similar to ours.

2.7.1 State-of-the-Art Arabic Speech Recognition

The systematic review by Alqadasi et al. (2023) [22] provides a detailed examination of Modern Standard Arabic (MSA) speech corpora. While their work centers on MSA and dialects, our research focuses on Classical Arabic (CA). Our focus on CA fills a gap not covered in their study.

Alqadasi et al. emphasize the scarcity of comprehensive Arabic speech corpora compared to other languages. This aligns with our motivation to produce high-quality Arabic corpora, particularly in Chapters 6 and 7, where we deal with Arabic phonetics and corpus development. Their literature survey, spanning from 2000 to 2023, identified only 27 pub-

licly accessible databases. To address this scarcity, we made our database publicly available through quranresearch.org.

Their work identifies a lack of diversity in voices within Arabic speech databases. To address this challenge, we applied data augmentation techniques in Chapter 7. This allowed us to generate diverse audio data, including male, female, and child voices, thereby expanding the dataset size and diversity. Additionally, while Alqadasi et al. noted that large datasets often suffer from poor quality due to recording environments, our work uses noise-free, clean recordings. Although we did not directly address noise-related challenges, we incorporated noise augmentation to improve model robustness. While their study concentrated on MSA and dialects, our exploration of CA adds a distinct dimension to Arabic corpus development.

The study by Aldarmaki et al. (2023) [15] investigates diacritic recognition in Arabic ASR systems. It addresses the problem of ASR models not producing full diacritization due to the absence of diacritical marks in existing speech corpora. The study compares the effectiveness of input diacritization and text-based diacritization as a post-processing step. The researchers fine-tune pre-trained ASR models using different diacritization conditions and evaluate diacritic recognition performance using coverage and precision metrics. The findings show that ASR diacritization outperforms text-based diacritization, especially when fine-tuned with manually diacritized data. This research is relevant to our research as it explores diacritic recognition in ASR systems and provides insights into the benefits of ASR diacritization.

Humayun et al. (2023) [94] focus on dialect classification using acoustic and linguistic features in Arabic speech. The paper proposes a classification model that combines multiple classifiers to identify different Arabic dialects. The acoustic classification utilizes ensemble classifiers and 'i-vector' features, while the linguistic classification employs transformer models pre-trained on large Arabic text datasets. The fusion of these classifiers achieves an accuracy of 0.82 for identifying five Arabic dialects, which is the highest reported accuracy on the dataset. This research is relevant to my work as it explores dialect identification using

similar techniques of combining classifiers based on different features.

Qasim, Hiba, and Huda (2022) [149] review the current state of Arabic speech recognition using deep learning methods and provide a comprehensive literature review in this area. The study highlights the importance of speech recognition for enabling computers to understand human voice feedback and emphasizes that Arabic speech recognition has received less attention compared to other languages. The paper explores deep learning techniques, including Deep Neural Networks (DNNs), RNNs, and CNNs, which have shown promising results in speech recognition.

The research by Bartelds et al. (2023) [30] explores improving low-resource Automatic Speech Recognition (ASR) systems through data augmentation techniques. Although their study focuses on non-Arabic minority languages and dialects, it addresses challenges similar to those encountered in Arabic ASR. These include the limited availability of transcribed speech data and the difficulties of building robust speech recognition systems for resource-scarce languages.

Their work emphasizes the effectiveness of data augmentation in increasing the volume of training data for low-resource languages. They demonstrate that data augmentation yields significant improvements when training data is very limited, although its effectiveness diminishes as the amount of manually transcribed data decreases. This aligns with the challenges in Arabic ASR, where the scarcity of high-quality annotated data remains a major hurdle.

While their approach involves creating synthetic data through text-to-speech (TTS) systems and self-training, we adopted a complementary strategy. In our research (Chapters 6 and 7), we focus on fine-tuning real, high-quality data to ensure it is optimized for improving Arabic ASR performance. This avoids the risks associated with synthetic data, such as overfitting to a single speaker, which they identified as a limitation. Instead, we incorporated multiple speakers for the same text, enriching the dataset with phonetic diversity, including variations in speaker age, gender, and other.

By integrating insights from their work, we emphasize the importance of carefully curating

and qualifying data, as described in Chapters 6 and 7.

2.7.2 Speech Recognition of the Quranic Research

Larbi and Ghania (2013) [114] focused on phonemic search in the Quran for Hafs's Riwayh. By extracting audio descriptors for each audio as "audio fingerprint". The audio descriptor characterizes the temporal and spectral information. Then the system receives the audio query from the microphone and performs the same audio descriptors. Then calculates the distance matrix between the two audios, called an "interference wave". Based on audio packets the similarity rate is calculated. The obtained accuracy of the results is estimated at 68%. The speed of response is 3 seconds. Reciting accuracy is important while reading the Quran. Learners make mistakes during recitation. The learner needs an expert to correct his/her recitation.

Muhammad et al. (2010) [136] proposed a solution to measure the correctness of the learner's reciting automatically. The dataset is a collection of verses from a Surah, recorded by 10 professional reciters. In the training phase, the acoustic features of each of the ten experts are extracted through Mel-Frequency Cepstral Coefficient (MFCC) technique. In the testing phase, the tested reciter reads through the microphone. Then the features of the reciter's recitation are extracted in the same way. The average value of all recitations is calculated using the features vector. The distance is calculated between the average of each expert's reciting and the tested reciter's reciting. The distance value must not exceed a given threshold value to be considered correct. The tested reciter must obtain at least 3 correct comparisons to count as a valid reciting. The system test was conducted on 30 reciters and the mean recognition ratio of reciters was 91% on average.

2.7.3 Arabic Text Classification and Speech Recognition

Aboalnaser and Sara (2019) [7] focus on text classification methods for Arabic documents. It aims to discuss the techniques used in the classification process and identify unresolved issues in Arabic text classification, while also providing future recommendations. The study

emphasizes the importance of data collection and representation, highlighting the use of machine-learning approaches and the availability of standard datasets. Text preprocessing techniques such as tokenization, removing stop words, and stemming are explored. The paper reviews existing schemes and approaches for Arabic text classification and emphasizes the need for a standard corpus and taxonomy for Arabic texts. Additionally, it addresses the challenge of classifying short texts and suggests it as an area for future research.

Our research focuses on a different aspect of text classification. While the reviewed research concentrates on general text classification, my research distinguishes itself by focusing on comparing Arabic texts with each other based on subtle differences. My objective is to classify the first text by determining if it is part of the second text, it's beginning, or if it belongs to another category entirely. This distinction in classification objectives sets our research apart, requiring a different approach and methodology. The paper acknowledges the difficulty in comparing algorithms due to different datasets used in previous studies and suggests the need for a standard corpus for Arabic texts. Additionally, it highlights the need for future investigations into the classification of short texts like my research.

Their research [7] contribution lies in providing a comprehensive review of the text classification process and techniques for Arabic texts. It emphasizes the importance of standard datasets and taxonomy for Arabic texts, facilitating better algorithm comparison. By identifying areas for improvement and suggesting future research directions, such as the classification of short texts, the paper contributes to advancing the field of Arabic text classification.

Wahdan et al. (2020) [184] present a systematic review of recent advancements in Arabic text classification using neural networks. The objective is to analyze and summarize 12 research papers, focusing on key aspects such as the types of neural networks employed, the diversity of corpora used, and the variations in efficiency measures. The review reveals extensive study of different neural network types, including RNN, CNN, FFNN, and LSTM. It also highlights the diversity of corpora utilized, with some being open source while others are author-created. The paper emphasizes the importance of the Arabic natural language

community in creating open-source corpora for testing models and emphasizes the need for standardized evaluation measures and improved reporting of neural network parameters to enhance research reproducibility and comparability in Arabic text classification.

By gaining insights into the types of neural networks employed, the challenges and limitations in the field, and the variations in corpora, efficiency measures, and neural network parameters discussed in the summarized paper, we can effectively differentiate our research approach from the studies reviewed. It is worth noting that while the reviewed papers primarily dealt with the classification of books, news, documents, hadiths, reviews, and tweets, they did not specifically focus on text-versus-text classification, which distinguishes our research. Therefore, it is expected that our results may differ from those presented in the reviewed studies due to the nature of our research objectives.

Bhogale et al. (2023) [35] propose a framework for aligning long audio segments with their corresponding transcriptions in conversational Arabic speech. In contrast, our work focuses on aligning short voice segments with diacritical text. The key difference lies in the duration of the audio segments being aligned; while their research deals with long audio files that can exceed one hour in length, we concentrate on shorter phonemes that are divided into smaller files. Additionally, our focus is on aligning diacritical text, which contains additional marks to represent Arabic phonetic information, while their work is centered around conversational Arabic, with a focus on Modern Standard Arabic (MSA).

The methodology we employ involves dividing phonemes into shorter files and aligning them with texts that contain diacritics, which enables precise alignment between the audio and text data. In contrast, their approach involves a two-pass alignment process, using a generic standard Arabic AM, biased LM, and graphemic PM for speech recognition on longer audio segments. Our work differs from their research in terms of alignment target (short voice and diacritical text versus long conversational Arabic), methodology (dividing phonemes versus two-pass alignment), and linguistic focus (diacritics alignment versus conversational Arabic alignment).

The study by Talafha et al. (2024) [177] contributes Casablanca, a dataset and models for multidialectal Arabic speech recognition. Their focus on eight dialects highlights the diversity in Arabic speech. In contrast, our research targets seven linguistic styles, emphasizing the structured nature of Classical Arabic and its diacritic-based variations. While they do not address diacritics, our system uses diacritic patterns to model linguistic differences effectively. Both works aim to fill the gap in robust Arabic datasets, yet our dataset preparation is semi-automated, unlike their fully manual process (see Chapter 6). Additionally, their work does not account for differences within each dialect, whereas our models are designed to learn and adapt to such variations, enhancing linguistic precision.

Sundus et al. (2019) [175] present a deep learning approach for Arabic text classification and compares it with the logistic regression (LR) algorithm. The goal is to evaluate the deep learning model's performance in terms of precision, recall, F1 score, accuracy, and training time. Two multi-class Arabic datasets are used to assess the effectiveness of the deep learning approach in improving classification accuracy and reducing model-building time compared to the logistic regression model. The key findings indicate that the deep learning approach outperforms the logistic regression algorithm, achieving higher accuracy, recall, F1-score, and accuracy, particularly in the "Sport" class. However, the research does not specifically address the classification of Arabic texts based on subtle differences, which distinguishes it from our research.

Our research focuses on classifying Arabic texts by comparing them to other Arabic texts to determine their relationship or belonging to the same category. This approach differs from the reviewed research, which primarily focuses on the multi-classification of general texts. While the reviewed research utilizes a deep learning model and evaluates its performance on specific datasets, my research targets the classification of Arabic texts based on subtle textual differences. This distinction in classification objectives requires a different approach and methodology, setting our research apart. Therefore, my research contributes to advancing Arabic text classification by capturing nuanced variations and relationships between texts.

2.7.4 Segmentation and Alignment Techniques for Quranic Speech

Javed et al. (2020) [101] addressed unsupervised phoneme segmentation in Classical Arabic speech. Their approach utilizes FICV features, a method that leverages the forward and inverse characteristics of the vocal tract. Evaluated on a Quranic Arabic dataset, FICV achieved a total error rate of 14.48% and an accuracy of 85.2% within a 10ms alignment error, outperforming existing hidden Markov model (HMM) based methods. This research highlights the potential of FICV features for speech segmentation in Classical Arabic, although focusing on phoneme-level rather than word-level boundaries.

Abdo et al. (2016) [4] proposed a speaker-independent, semi-automatic system for segmenting Arabic speech into syllables. Their method utilizes Mel-Frequency Cepstral Coefficients (MFCCs) and a two-step approach for segmentation: a) Maxima Extraction: The system identifies potential syllable boundaries by locating maxima in the delta function of the first MFCC coefficient. b) Template Matching: It then employs template matching techniques using reference utterances to refine the boundaries identified in step 1. This approach achieved a segmentation success rate of approximately 91.5% on a dataset of 276 utterances, divided into 2544 syllables. While focusing on syllable-level segmentation compared to your word-level with diacritics, this research aligns with yours by targeting Arabic speech segmentation. Additionally, the semi-automatic nature of their system provides a potential area for differentiation as your approach is likely fully automated.

Abdo et al. (2014) [5] introduced an algorithm for segmenting emphatic and non-emphatic sounds in continuous Arabic speech. Their method utilizes Mel-Frequency Cepstral Coefficients (MFCCs) and analyzes peaks in the delta MFCC to locate sound boundaries within the speech signal. The algorithm is designed for sub-syllable level segmentation and was evaluated on a Quran recitation corpus of 480 words, achieving up to 90% accuracy in boundary detection. However, it targets a lower segmentation level (sub-syllable) compared to your word-level focus with diacritic emphasis.

Absa et al. (2018) [8] introduced a hybrid unsupervised segmentation algorithm for Quranic

Arabic speech. Their method combines three basic speech features (entropy, zero-crossings, energy) and optimizes segmentation using a genetic algorithm. This approach achieved a 20% improvement in accuracy over traditional single-feature-based techniques. The work explores various thresholding techniques (median, mean, mode) for detecting syllable segments and discusses their strengths and weaknesses. Additionally, they propose a regression fusion scheme for robust boundary prediction. This research focuses on frame-level segmentation and emphasizes the benefits of feature fusion and genetic algorithm optimization for Quranic Arabic speech processing.

The previously described research provides valuable insights into segmentation and alignment techniques for Quranic speech. We present this comparison in Table 2.1. This table shows how DASAM relates to other research in the field. By comparing DASAM to these studies, we can see its position within the existing body of work.

This comparison 2.1 delineates the distinct methodologies and achievements of each study, illustrating DASAM's advancements in handling diacritic-sensitive segmentation.

2.8 Exploring Semantic and Numeric Similarities in Multidimensional Heterogeneous Environments

Research conducted by Wu et al. (2021) [189] introduced the SKP-LDA algorithm, which combines numerical information and semantic knowledge to analyze short texts. The algorithm leverages word frequency and co-occurrence patterns to extract numerical information, specifically through a word bag based on sentiment word co-occurrence, capturing the emotional polarity associated with the texts. Moreover, it incorporates semantic knowledge by extracting topic-specific word pairs and topic-relationship words, enhancing topic clustering and improving semantic information retrieval.

The SKP-LDA algorithm successfully integrates numerical information and semantic knowledge, enabling a comprehensive analysis of short texts and a more inclusive understanding of their content. However, there are notable distinctions between their research and our SemSim project. In our SemSim project, we adopt a different approach by providing user-

defined functions for inferring numerical information from texts instead of relying on predefined methods. Additionally, while their research primarily focuses on a specific-purpose solution for sentiment analysis, our SemSim research strives to develop a general-purpose solution applicable across diverse domains.

Research conducted by Islam and Inkpen (2006) [97] proposed the Second Order Co-occurrence PMI (SOC-PMI) method. This method aims to assess the semantic similarity of two target words using a corpus-based approach. SOC-PMI ranks important neighbor words for each target word. It does so based on Pointwise Mutual Information (PMI). The method aggregates the PMI values of the common words between the lists. It takes into account both co-occurrence and PMI to calculate the relative semantic similarity.

In contrast to their research [97], our study in this field takes a different approach to exploring semantic similarity. While their research focuses on calculating semantic similarity based on the second-order co-occurrence of neighbor words. Our work explores semantic similarity by allowing users to define the criteria for similarity calculation based on their specific needs. Users can apply any arithmetic method to the features in their dataset. While the authors contribute to the field of semantic similarity with the SOC-PMI method and predefined numeric statistics, our study extends the exploration. We incorporate user-defined numeric statistics alongside semantic similarity. This is achieved through the introduction of the SemSim System. SemSim allows users to tailor the numeric statistics according to their specific needs and preferences.

Mahapatra et al. (2012) [129] focused on contextual anomaly detection in text data by integrating statistical deviations and semantic analysis. Their objective was to identify anomalies. These anomalies may be statistically infrequent but contextually plausible. The aim was to reduce the false positive rate in existing anomaly detection systems.

The integration of statistical deviations and semantic analysis aims to enhance anomaly detection by considering the numeric and semantic aspects. The researchers employed algorithms such as clustering and topic modeling for statistical analysis in their approach to

anomaly detection. While both approaches incorporate statistical deviations and semantic analysis, our SemSim System differs from the previous research in two key aspects. Firstly, the previous research employs predefined algorithms for statistical analysis, whereas our system allows users to define their own statistical functions. Secondly, their research provides a solution specifically tailored for anomaly detection. On the other hand, our SemSim System is a general-purpose solution applicable across diverse domains.

Table 2.1: Comparison of Segmentation and Alignment Techniques for Quranic Speech

Criteria	**DASAM**	**Abdo et al. (2014)**	**Javed et al. (2020)**	**Absa et al. (2018)**	**Abdo et al. (2016)**
Segmentation Focus	Word-level alignment with diacritics focus	Emphatic and non-emphatic sound segmentation	Phoneme segmentation using cosine distances	General speech segmentation for Quran recitation	Syllable segmentation for speech recognition
Methodology and Techniques	Linguistic intonation rules with DTW	Peak detection using MFCC	Cosine distance drives phoneme boundary using FICV	Feature fusion with Genetic Algorithm optimization	Template matching with maxima extraction from delta MFCC
Algorithm Type	Advanced automatic using DTW	Automatic with predefined rules	Unsupervised	Hybrid unsupervised	Semi-automatic
Dataset & Audio Focus	Quranic Arabic	Quranic Arabic	Quranic Arabic	Quranic Arabic	Quranic Arabic
Dataset Size	20K from two reciters, 260k words with audio variation	480 words from six speakers	725 words	KACST, 5935 files from 10 reciters	2544 words, from 12 reciters
Feature Sets	MFCC	MFCC	FICV, cosine distance scores	Entropy, zero crossings, energy	MFCC
Performance Metrics	R^2 95.9% and 95.7% for start and end times	90% accuracy	85.2% accuracy	20% improvement over baseline models	91.5% accuracy

Chapter 3

THE ARCHITECTURE OF THE PROPOSED SUPPORT SYSTEM FOR THE ARABIC NATURAL LANGUAGE PROCESSING

With the widespread use of software systems and applications that serve the Arabic knowledge domain, several concerns arise. Arabic gained a lot of attention thanks to the heritage resources that have been maintained over thousands of years. The Arabic heritage books date back to a couple of thousands of years ago. Living in the Arab peninsula, featured by its large desert areas, the Arabs spent enough time on poetry as a main source of entertainment and as a channel of media and publicity at that time. The sixth century was a major milestone in the Arabic timeline with the birth of Prophet Muhammad, the prophet revealed to by God, according to Islamic belief. Revelation to Prophet Muhammad was in Arabic and the book revealed to Prophet Muhammad is called the Quran. With the embrace of Islam and the increase in the Muslim population, the Arabic language became the native language of a wide population. This population over the years has put a lot of effort to preserve the Quran, which in turn preserved the Arabic language. It is not only the text of the Quran but also the audio recitations from reciters worldwide that provide a reliable data set of an Arabic text and show how the writing of the language evolved over time.

The hard copies of the Quran and other heritage books that are handwritten have received a lot of attention over the years. The digitization of these resources has also received attention in recent years. However, according to the point of view of the authors of this dissertation, the efforts to digitize heritage books are scattered, repeated, and limited. These efforts have been carried out by enthusiastic individuals who build databases for Arabic heritage books. These databases are local on-premise databases (in contrast to cloud-based databases) and

are hardly accessible by other researchers, hence, are not fully reviewed and authenticated.

The authenticity and accuracy of these databases are questionable. Some software developers and amateur Islamic researchers, driven by enthusiasm, may make inaccurate statements or incorrect claims about numerical patterns or miracles in the Quran. Others may find it difficult to reproduce these claims, leading to a lack of trust. This uncertainty makes it unclear whether results can be replicated using the same data and methods.

Moreover, as the number of users increases, concerns about scalability—the system's ability to handle more work or to expand—and availability arise. Extensibility is also a major issue because systems may struggle to add new features without significant changes.

Properly designed systems should be extensible and reusable. They should be built on top of existing frameworks, which provide foundational structures and basic data that can be expanded and accessed. A good framework is not limited to a specific purpose; it allows integration with other systems and provides clear methods and components for development. Building separate frameworks that do not support integration and operate as black boxes offers little value, even if they provide direct services. Instead of creating each system from scratch whenever there is a new purpose, developers can build upon existing frameworks. This approach promotes efficiency, consistency, and interoperability.

In this section, we present a system that supports Arabic language research, specifically focusing on the Holy Qur'an. Our system emphasizes scalability, availability, reproducibility, and extensibility to serve Arabic database systems. We will discuss the features and the architecture of the proposed system.

The contributions of this section of the study can be summarized as follows:

- Establishment of a framework supporting Arabic and Quranic research, encompassing text analysis, vision, speech recognition, and semantic analysis.
- Creation of a scalable and accessible cloud-based database of the Quran, encompassing Arabic text, images, and audio tracks, to serve as a centralized resource available to researchers worldwide.

- Development of a dynamic expansion system, empowering developers and researchers to perform advanced queries, and codes, and publish results with accompanying documentation.
- Facilitation of research endeavors by providing diverse Arabic datasets and tools, enabling researchers to explore and analyze Arabic and Quranic texts effectively.

3.1 The Goal of the Proposed System in a Nutshell

Developing comprehensive support systems for Arabic Natural Language Processing NLP is essential for bridging the gap between Arabic texts and contemporary research methodologies. We aim to establish a framework to support the Arabic and Quranic research, focusing on Textual Analysis, Numeric Analysis, Linguistic Analysis, and Natural Language Processing (NLP). Our initiative, QuranResearch.ORG, caters to two distinct audiences. Firstly, we seek to inspire researchers proficient in software development to engage in Arabic research, particularly in textual, numerical, and linguistic analysis. These researchers are encouraged to share their findings publicly, inviting comments, feedback, and suggestions from the wider community. Secondly, we target the general public who possess a profound interest in exploring the messages, observations, and miracles embedded in the divine book, the Quran. By offering auxiliary tools and diverse Arabic datasets, our framework aims to facilitate Arabic and Quranic research endeavors, providing valuable resources for scholars in these domains.

3.2 The system Features

Our system has been meticulously designed to fulfill multiple objectives. In this section, we succinctly outline the key features of our QuranResearch.org system that contribute to the accomplishment of these objectives. While this part of our study offers a comprehensive overview of the system's features, vision, and architecture, the following sections provide the technical depth behind each component of the proposed system architecture.

3.2.1 Accessibility

The system provides a database of the Quran text that is publicly available and open to all researchers. Researchers issue their queries through the QuranResearch.ORG website and count on the backend server to run their "possibly" long-running queries, then notify the users once the results are ready. There is no need for hardware or software installation on the researcher's side.

3.2.2 Efficiency and performance

The system provides a highly indexed representation of the Quran text that speeds up the search and analysis of the Quran text. The system aims at scaling up to multiple concurrent users performing long-running queries on the Quran text. Elasticity in computation by hosting the system in the cloud enables adding servers to support query workloads at peak times.

3.2.3 Accuracy

The system aims to provide an accurate database for the entire Quran, offering precise information and statistics. This includes accommodating multiple Quran narrations and various counting or numbering methods for Ayahs, allowing for flexibility in research and analysis. Furthermore, the system seeks to support true claims about "numeric miracles in the Quran". It offers tools to verify these claims with concrete numbers. Many individuals cite intricate numerical patterns in the Quran. These patterns are often challenging to validate manually. This difficulty results in potential misinformation. The system addresses this challenge through several key features:

- Allowing users to replicate claimed calculations using direct database queries.
- Enabling transparent sharing of calculations and numerical results
- Providing a concrete and verifiable data source for testing claims.

This transparency helps distinguish between accurate claims and unsubstantiated ones. It re-

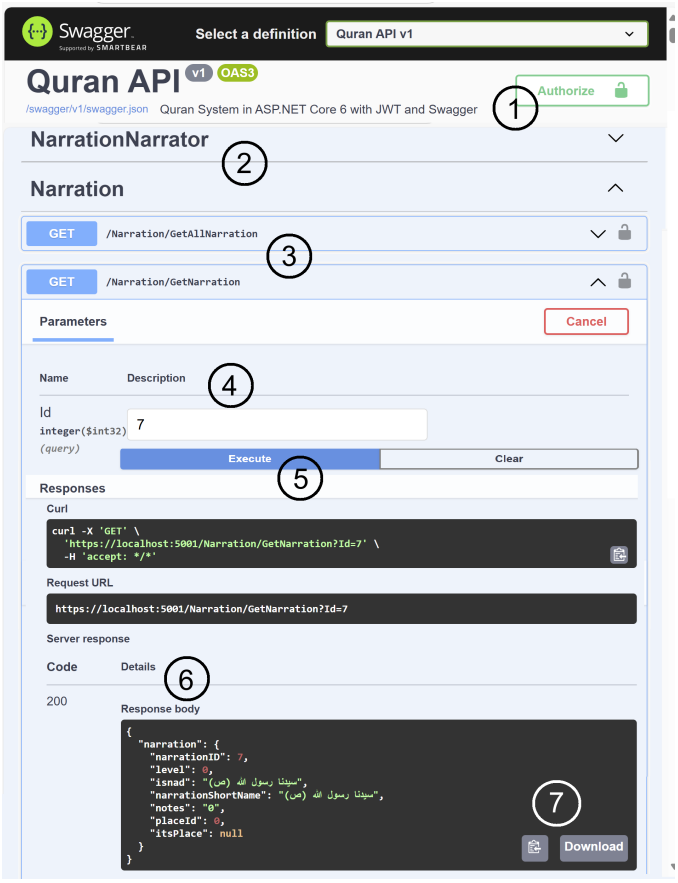


Figure 3.1: Swagger UI for QuranAPI: A Web-Based Interface for Non-Technical Users to Explore, Test, and Utilize API Functionality.

duces the spread of misleading information. By ensuring an authentic and accurate database at its core, the system serves as a critical resource. It advances credible research into numerical patterns in the Quran.

3.2.4 Application Programming Interfaces (APIs), tools and packages

The system provides an open platform for researchers to issue their queries against the backend database and retrieve results instantaneously or be notified once results are ready. The system is designed for developers who are proficient in programming skills and who are willing to use these skills in Quranic research. The system provides Application Programming Interfaces (APIs) with various settings and tuning knobs to easily search and retrieve

words/ayahs. The system provides layers of data analysis, data mining, and data visualization on top of the Quran database. QuranResearch.Org does not plan to be only accessible through its UI website but to be also accessible through its APIs that software developers can connect to and issue their queries against the system's backend.

APIs are often perceived as tools intended only for developers or technical users. However, we have designed our system to be accessible to a broader audience, including non-specialists. By utilizing Swagger, we provide a user-friendly interface that allows anyone to explore and interact with our APIs directly through their web browser, without requiring specialized knowledge.

Figure 3.1 shows the Swagger UI for our QuranAPI. The interface offers several features that make it easy for general users to navigate and use the APIs:

- **Authorization:** The "Authorize" button allows users to log in securely. Once authenticated, users can access API endpoints and receive results according to their permissions.
- **Organized API Endpoints:** The APIs are grouped into categories or sections. When a user clicks on a category, all related API endpoints are displayed, each labeled with its HTTP method (e.g., GET, POST). This organization helps users find the APIs relevant to their needs.
- **Endpoint Documentation:** Selecting an API endpoint opens a detailed view that provides information on how to use it. This includes a description of the endpoint's purpose, the parameters it accepts, and examples of expected inputs and outputs.
- **Interactive Parameter Input:** Any required parameters are clearly listed, along with their names, data types, and descriptions. Users can enter the necessary information directly into the provided fields.
- **Execution of Requests:** After entering the parameters, users can click the "Try it out" and "Execute" buttons to send a request to the server. The interface handles the technical aspects of the request, making the process straightforward for the user.

- **Displaying Responses:** The server's response is displayed in real-time within the interface. The "Response Body" section shows the data returned by the API, typically in JSON format, and the "Response Code" indicates the status of the request (e.g., 200 OK for a successful request).
- **Data Utilization:** Users have the option to copy the response data to their clipboard or download it directly from the interface. This enables them to use the information immediately, without additional steps.

By providing this interactive, web-based interface, we make the APIs accessible to users without programming experience. The Swagger UI simplifies the complexity of API interactions, allowing users to focus on exploring the data and services we offer. This approach lowers the barrier to access, allowing a wider audience to benefit from our data and functionalities.

3.2.5 Knowledge sharing and research network

QuranResearch.org is expected to be the "Wikipedia-like hub" that lists all the results from the collective efforts of the community of Quran researchers in one central place. QuranResearch.org is expected to build a network of researchers in the field, where they can communicate, provide feedback and collaborate in research. The website does not only aim to be a hub for sharing results but also to be a hub for researchers to share tools, pluggable modules and packages that researchers can use and integrate in their research. Consequently, the "value of the whole system becomes greater than the sum of the parts".

3.3 The Proposed System Architecture

This section provides an overview of the proposed system architecture and its various components. Additionally, it outlines the expected contributions and results, if any. The proposed architecture is shown in Figure 3.2 as an integrated multimodal Arabic processing ecosystem. The following sections discuss each component at a high level. The technical details are left to the subsequent chapters of this dissertation.

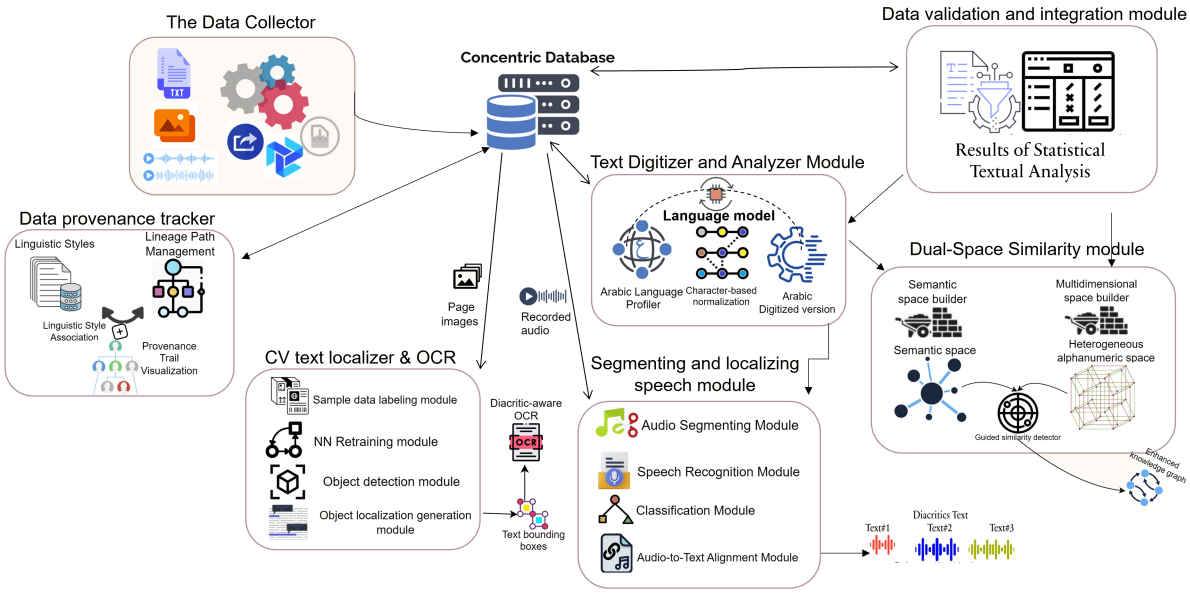


Figure 3.2: The Proposed System Architecture.

3.3.1 The data collector

The data collector performs the ETL (Extract, Transform, Load) functions. We extract text, audio, and image data using web scraping tools. We aim at collecting Arabic and Quran data from multiple resources, compare and contrast data against each other, and reach a cleaned-up and integrated version from the scraped sources. To deal with heterogeneous data from various sources in different formats, we convert it into a unified format and export it to our database.

3.3.2 Text Digitizer and Analyzer Module

In Arabic, a character can have multiple Unicode representations. Users who have used an Arabized version of operating systems (e.g., Microsoft Windows) are requested to choose the type of Arabic dialect to be installed based on the region or country the user is in. With the data coming from different sources and/or regions, unifying the format is crucial so that we can compare and contrast the data sets. To achieve this, we developed a Text Digitizer and Analyzer Module, which can map different Arabic Unicode representations to a unified representation used by our system. This module identifies different representations

and creates a unified digitized version of the data, making the comparison between text documents easier.

3.3.3 The Data Validation and Integration module

This module accepts the unified format of the data, as given by the Text Digitizer and Analyzer Module. The module integrates data copies of the same text from multiple sources and cleans up and validates these data copies by contrasting these copies against each other. The module ensures the quality and consistency of the input data in an attempt to increase the trustworthiness and the adoption of the proposed system.

We aim to create a model for mapping different Arabic Unicode representations. This model 1) improves the search in the Arabic and Quranic text, 2) increases the accuracy of discovering differences between copies of the same text, 3) creates consistent statistical results, 4) digitizes the older books from scratch, and 5) integrates existing sources to attain a conflated data set from several sources, where we can track the data lineage and provenance. For digitizing older books, this is done through the OCR introduced in Chapter 6. The OCR reads the handwritten text from the images, and the model unifies its representations as Unicode.

3.3.4 The Concentric Database

At the core of the proposed system, comes the database. The database stores heterogeneous types of data. The most important data types can be categorized into three primary themes: text, images of heritage books that contain text, and audio tracks that contain the audible recitation of the text. More specifically, if we consider the Quran as a data set, then we can summarize its associated data as follows:

Mushaf Images. The term 'Mushaf' refers to a copy of the Qur'an book, which represents the Quran according to a specific linguistic style (riwayh) expressed in Arabic. Each linguistic style varies in letter writing, character set, and dialect. Each Mushaf is preserved according to a chain of narrators who carried this linguistic style over the years from one generation to another. Each Mushaf is basically a separate printed Quran book. There are 20 linguistic

styles agreed on by the Muslim populations. We store the page images of each Mushaf. Each Mushaf is in the range of 600 pages. Expert calligraphers write Mushafs, which are checked at the fifth level of Arabic. Pages are scanned and kept as images in the proposed system's database.

Text and Metadata. We store the Quran text of the 20 Mushafs in a Unicode textual representation. In addition to the text, we store metadata about the stored text. This metadata, for example, stores the verse (also called the Ayah or the complete sentence) bounding boxes of each verse in the scanned pictures to allow users to interact with the text by displaying text, translation, and statistics on averse by verse level. This metadata also allows for the comparisons between different Mushafs and different chains of narrators associated with each Mushaf. All Mushafs are divided into 114 Surahs, with each Surah having a number of verses or Ayahs that varies subject to a counting scheme. Although the Quran text is the same across the 20 Mushafs, the division of the text into verses can vary based on the counting scheme used in a Mushaf. The database stores the various counting schemes and their relationships to each verse's start and end positions as part of its metadata.

Audio tracks. The database maintains the audio tracks, which are the audible representation of the Quran text, and aligns the audio to the text. The audio tracks are saved for each verse separately and for each reciter in each chain of narration. Maintaining this data as part of the database is crucial to research in Arabic speech recognition. Because it aligns audio with Arabic text. This alignment supports training accurate speech recognition models and improves phonetic understanding.

3.3.5 A Computer-Vision Based Text Localizer

The Computer Vision (CV) Text Localizer is responsible for linking each text to its corresponding part of the image in scanned pages. It uses a CV approach with a Faster RCNN neural network to analyze the images. The network is trained on annotated images to determine the location of each sentence on the image. Then, the localizer generates the dimensions of the boundaries for each sentence.

The main goals of the text localizer can be summarized as follows:

- To improve the accuracy of text recognition in scanned Arabic images by identifying the text regions and the sentence boundaries in both printed and handwritten Arabic text.
- To enhance the performance of Arabic Optical Character Recognition (OCR) through the detection of text boundaries.
- To develop a large dataset of images along with their corresponding text bounding boxes, which can be used for further research in Arabic NLP.
- To develop techniques that can extract features of letters and diacritical marks that are specific to Arabic writing.

3.3.6 The Speech Localization, Segmentation, and Alignment Module

The Speech Localization, Segmentation, and Alignment Module is designed to enhance Arabic Speech Recognition and to align speech to its corresponding text. To utilize machine learning and deep learning techniques in the process of *ASR*, we need to prepare a training data set of audio tracks along with its matching text pieces. We started our efforts along the direction of *ASR* by preparing a ground truth dataset that ties the Arabic textual audio data with its corresponding text narratives for training and evaluation purposes. We have investigated various machine learning and deep learning models. We trained, tuned, and tested these models to associate audio files with their corresponding text sentences. The process of associating audio tracks to their corresponding text is called audio-to-text alignment. The proposed ASR system aims to achieve accurate alignment of the Arabic languages taking into consideration the diacritic marks that are usually ignored by current audio-to-text services.

The Speech Localization, Segmentation, and Alignment Module consists of several sub-modules, each serving a specific purpose. These sub-modules can be summarized as follows:

- **The Audio Segmentation Module**, which segments audio files into individual sentences to facilitate the analysis and processing of sentence-level speech recognition and

alignment.

- **The Speech Recognition Module**, which utilizes the existing speech-to-text technologies to convert audio files into text. We note that existing services do not provide much support for diacritic marks and higher levels of annotations featured in the Arabic language. Subsequent Modules help to overcome the challenges and errors related to diacritic marks.
- **The Classification Module**, which incorporates machine learning models and classifiers to improve the accuracy of the speech recognition output (obtained from the previous step) and to provide support for the diacritic marks. This module adjusts the low-accuracy diacritic-oblivious recognized text to generate a high-accuracy diacritic-aware output, called the adjusted text.
- **The Audio-to-Text Alignment Module**, which takes on the responsibility of aligning each sentence or even a word from the audio files to its corresponding text in the transcript.

  This alignment operation produces a diacritic-aware audio-to-text dataset, where each audio segment aligns with a diacritic-annotated Arabic sentence.

The Speech Localization, Segmentation, and Alignment Module serves as an infrastructure to enhance the accuracy and performance of Arabic Speech Recognition systems. It tackles challenges related to diacritics, error correction, and alignment between audio and text. This module leads to notable improvements in the accuracy of speech recognition for various Arabic language applications. It also paves the road for future research to enable a better understanding and analysis of the spoken Arabic content

3.3.7 Semantic Similarity Detection module

The extraction of numeric features from text has a substantial impact on multiple domains. The numeric similarity allows for valuable insights into the semantic aspects of texts. Additionally, the integration of these numeric features with semantic similarity further enhances

their significance. By leveraging the combination of numeric and semantic similarity, researchers have achieved notable progress in multiple fields. These fields include sentiment classification [196], topic modeling [169, 204, 82], anomaly detection [129], and other related fields. This approach has proven to be fruitful in advancing the understanding and analysis of textual data. For this purpose, this module aims to enhance knowledge discovery by uncovering hidden associations and similarities among entities in multidimensional data sets. The module helps users explore the numeric and semantic similarities in heterogeneous environments. The module provides a variety of tools to construct numeric and semantic spaces. These spaces facilitate the exploration of relationships between entities at both the numeric and semantic levels. This exploration can lead to a deeper understanding and enrichment of knowledge. For instance, in cases where there are similar concepts at the semantic level, the module examines the numerical similarity or equality between their respective features. To illustrate this, consider the following Table 3.1, which contains a group of Arabic words that share the first letters (ع) and (ل) as prefixes, all related in meaning:

Table 3.1: Arabic Words Sharing Prefixes (ع) and (ل): Illustrations of Semantic and Numeric Relationships

Arabic Word	**Pronunciation**	**Meaning**
عَلَّمَ	*'allama*	Taught someone
عَلِمَ	*'alima*	Knew something
عُلِّم	*'ullima*	Educated person
عِلم	*'ilm*	science
عُلُوم	*'ulūm*	sciences
عَلاَّمة	*'allāmh*	a scholar well-versed in science
عَلمانيّ	*'almanī*	Secular
عليم	*'alīm*	All-knowing expert
علماء	*'ulamā'*	scientists

These words share numerical similarity through their common prefix letters, reflecting their

related meanings. On the other hand, When numerical similarities are uncovered among these features, it suggests the possibility that other concepts sharing similar features may also have semantic similarities. This discovery contributes to the enrichment of knowledge by revealing previously unknown semantic relationships.

The Semantic Similarity Detection Module offers tools to create a numeric space and a semantic space and, then, explores similarities across the two spaces. The multidimensional space builder tool performs Co-occurrence Analysis on the data by examining entities such as chapters, sentences, words, and letters to build a numeric space. This co-occurrence analysis generates numeric values that capture the relationships between text items (e.g., chapters, sentences, words, and letters), reflecting their composition and structure. These numbers are then represented in a multidimensional space, assigning specific features to each word based on the obtained co-occurrence analysis results. In addition to the numeric space tools, the module also provides tools to construct and manage an ontology-based semantic space. Furthermore, the module has a semantic-numeric similarity discoverer. The examination is done in both semantic and numeric spaces. The semantic-numeric similarity discoverer notifies the user of any entities that are similar along both the numeric and semantic spaces. This module provides a basis for semantic analysis and relationship discoveries among entities.

In the proposed research, we developed a system, called *SemSim* for Semantic Similarity, to leverage semantic similarities (given by a knowledge graph) and to correlate these similarities with detected numeric similarities. Such a correlation can help discover new patterns and, consequently, improve the knowledge graph. The proposed system is discussed in Chapter 5.

3.4 Integration of System Components

Having detailed each component individually, we now illustrate how these modules interact within the system to achieve our objectives. The diagram 3.2 presents the interconnected architecture of our proposed system, showcasing how each component collaborates to enhance Arabic language processing. The Data Collector initiates the process by extracting text,

audio, and image data from various sources. This raw data is fed into the Database.

The Data Validation and Integration Module accesses this data from the Database to perform cleansing, ensuring consistency and accuracy. The Data Provenance Tracker interacts continuously with the Database, maintaining records of data lineage to ensure authenticity and reliability throughout the system.

The Text Digitizer and Analyzer Module both retrieves from and updates the Database. It standardizes various Arabic Unicode representations into a unified format, preparing the text for further analysis. This module also supplies processed data to the Semantic Similarity Detection Module, the Computer Vision Text Localizer and OCR Module, and the Segmenting and Localizing Speech Module. The Semantic Similarity Detection Module utilizes the unified text to explore numeric and semantic similarities among entities, uncovering hidden associations and enriching knowledge discovery.

The Segmenting and Localizing Speech Module uses the processed text from the Text Digitizer and Analyzer Module along with audio data from the Database. It performs speech segmentation and localization, aligning audio segments with the corresponding text. Simultaneously, the Computer Vision Text Localizer and OCR Module leverages the text data to accurately identify sentence boundaries in images and recognize Arabic text along with diacritical marks in scanned images.

This cohesive interaction among modules facilitates efficient data processing across text, image, and speech modalities. Each component contributes uniquely but works synergistically towards the common goal of advancing Arabic language processing.

3.5 The System's Front End

This section explains how to navigate through the QuranResearch.org frontend, describes the functionality provided by each page, and shows how the backend features are surfaced to the end user.

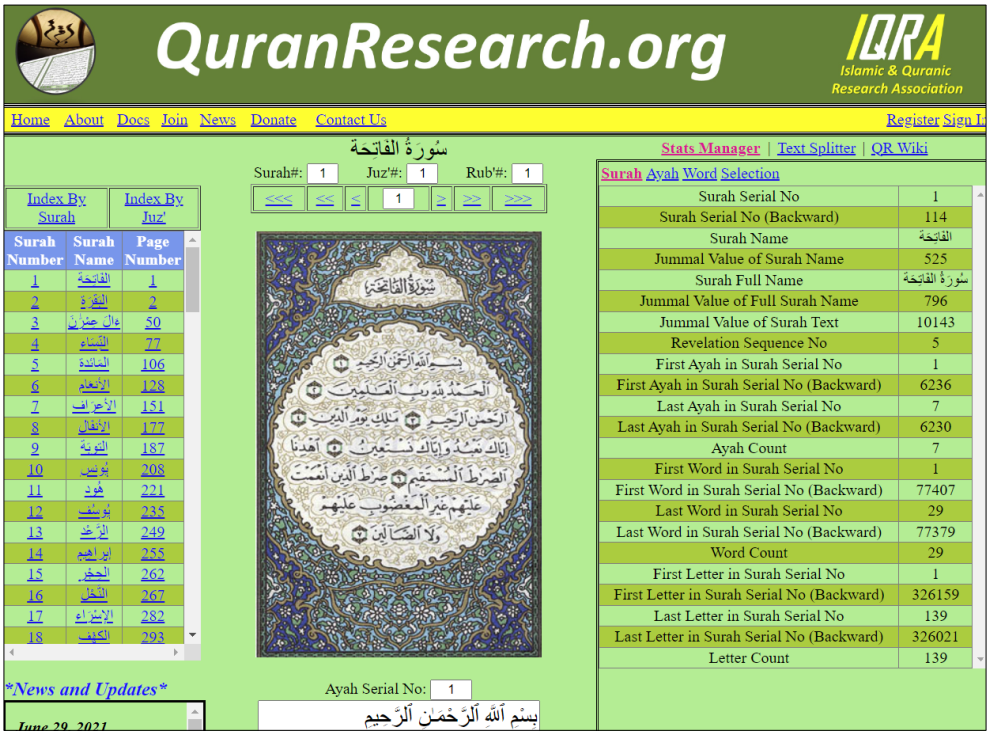


Figure 3.3: Homepage of QuranResearch.org.

3.5.1 The System's UI structure and the Basic Navigational Functionalities

As shown in Figure 3.3, the user interface is divided horizontally into three sections. The leftmost section and the top part of the middle section provide the "basic navigational controls". It allows the user to navigate to the desired place in the Quran by surah number, chapter (juz') number, page number, etc. Moreover, some additional navigational controls allow the user to navigate forward and backward by one, ten or hundred pages with every click. Figure 3.3.

Homepage of QuranResearch.org. The middle section displays the Quran page that is navigated by the user. Clicking an ayah on the displayed page will populate a textbox at the bottom of the middle section with the text and the ayah number of the ayah that has been clicked. This textbox (at the bottom of the middle section) is referred to as the "Selected Ayah Textbox". The middle section is basically the Quran page and, more specifically, the selected ayah that subsequent queries will be considering. Note that the Quran page and the

selected ayah (displayed in the "Selected Ayah Textbox") are navigated through the leftmost section, the navigational buttons at the top of the middle section, and by clicking on ayah in the displayed Quran page. Moreover, the Quran page and the selected ayah can also display (as explained in the next paragraph) the page and ayah linked to a row in the query result of the query being executed in the rightmost section of the page. The rightmost section of the user interface is dedicated to the operations and queries being executed over the Quran text. All text operations, numerical operations, analysis queries, and visualizations are displayed in the rightmost section of the interface. These operations are expected to be executed over the selected page, ayah, part of an ayah, word, or even part of an ayah that is highlighted in the "Selected Ayah Textbox". The rightmost section is featured by the Statistical Manager, the Text Splitter, and the Quran Wiki modules of the system. These modules are described in the following section.

3.5.2 Stats Manager, Text Splitting and Quran Wiki

In the rightmost section of the UI, comes the statistical and textual analysis components of the Quran. There are three tabs in the interface, namely:

Stats Manager: As shown in Figure 3.4, this subpage displays statistics related to the current surah, ayah, word, or selected text of the Quran. It gives the serial numbers and counts of Ayahs/Words/Letters, counting forward/backward (from beginning or end) relative to the beginning/end of the whole Quran, current Surah/Ayah/Word. The statistics can be computed according to four granularities: a full surah, an ayah, a word, or a selection (which is the text highlighted by the user in the Selected Ayah Box).

Text Splitter: The text splitter (Figure 3.5) splits the current surah, ayah, or word into its constituting letters or words. The user has the option to either consider Tashkeel (diacritics) or not while splitting. The user can also group similar words/letters and get a count of how many times the word/letter is repeated in the currently selected Surah/Ayah. The text splitter helps identify how the words and letters are counted by various queries by showing the constituting parts of each ayah or word.

Stats Manager | Text Splitter | QR Wiki

Surah Ayah Word Selection

Surah Serial No	1
Surah Serial No (Backward)	114
Surah Name	الفاتِحَة
Jummal Value of Surah Name	525
Surah Full Name	سُورَةُ الفَاتِحَة
Jummal Value of Full Surah Name	796
Jummal Value of Surah Text	10143
Revelation Sequence No	5
First Ayah in Surah Serial No	1
First Ayah in Surah Serial No (Backward)	6236
Last Ayah in Surah Serial No	7
Last Ayah in Surah Serial No (Backward)	6230
Ayah Count	7
First Word in Surah Serial No	1
First Word in Surah Serial No (Backward)	77407
Last Word in Surah Serial No	29
Last Word in Surah Serial No (Backward)	77379
Word Count	29
First Letter in Surah Serial No	1
First Letter in Surah Serial No (Backward)	326159
Last Letter in Surah Serial No	139
Last Letter in Surah Serial No (Backward)	326021
Letter Count	139

Figure 3.4: The Statistics Manager.

Stats Manager | Text Splitter | QR Wiki

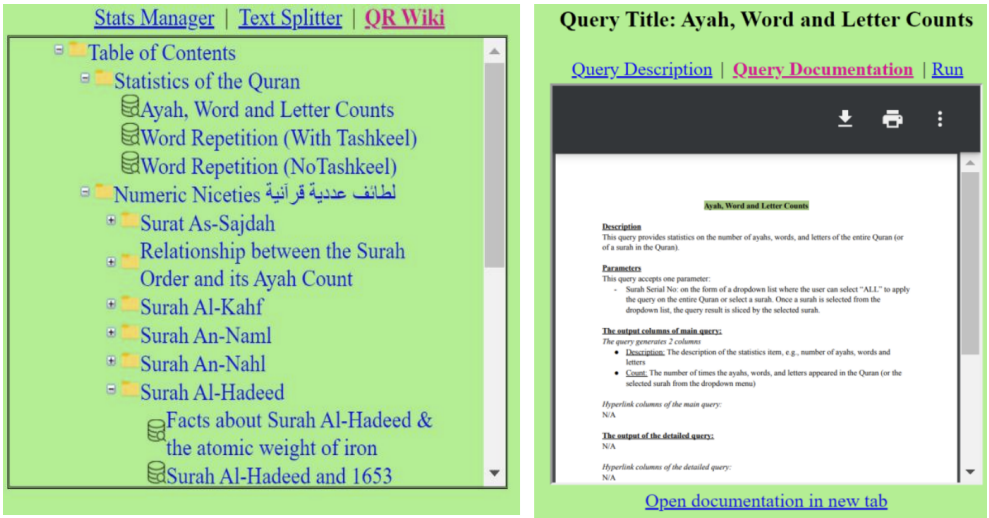
Split current Surah into Letters

Without Tashke and Without Groupir

Row#	Letter
1	ب
2	س
3	م
4	أ
5	ل
6	ل
7	ه
8	أ
9	ل
10	ر
11	ح
12	م
13	ن
14	أ
15	ل
16	ر
17	ح
18	ي

Figure 3.5: The Text Splitter.

QR Wiki: This component makes the system dynamic and extensible. Researchers and developers can write their queries in SQL through the Developer's Dashboard (see Section 3.6). The system supports real-time updates to queries. Developers can modify their queries in real-time. These updates are immediately processed by the system. However, the updated results are not instantly published on the site. All modified queries undergo a review process. The system administrator must approve the changes before they become publicly visible under the QR Wiki section. QR Wiki serves as a "Wikipedia-like hub" that centralizes the results from the community of Quran researchers. As shown in Figure 3.6, QR Wiki provides a table of content (Figure 3.6(a)) that organizes queries into a tree structure based on topics and sub-topics. When a query is selected, the query description, documentation, SQL source code, and results are displayed through tabs at the top of the query interface (Figure 3.6(b)). As the research community grows, QR Wiki will expand to include more valuable queries. It facilitates discussions among researchers and fosters a healthy environment for sharing ideas, reviewing results, and suggesting improvements.



(a) Table of Contents *(b) The Query Documentation*

Figure 3.6: The QR Wiki.

3.6 The Developer's Dashboard

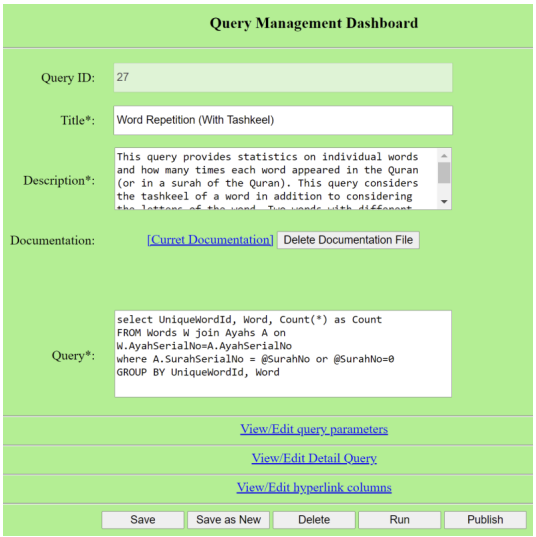


Figure 3.7: The query title, description, and main SQL query.

As mentioned earlier in this section of the dissertation, the proposed system has two types of audiences:

(1) The developers who have the programming skills and can write queries to extract insights and statistics from the Quran text and (2) the general public who would consume the result

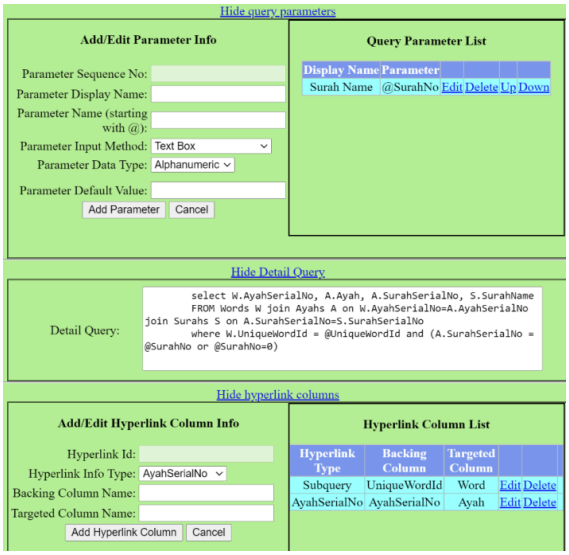


Figure 3.8: The query parameters, detail query SQL, and hyperlink columns.

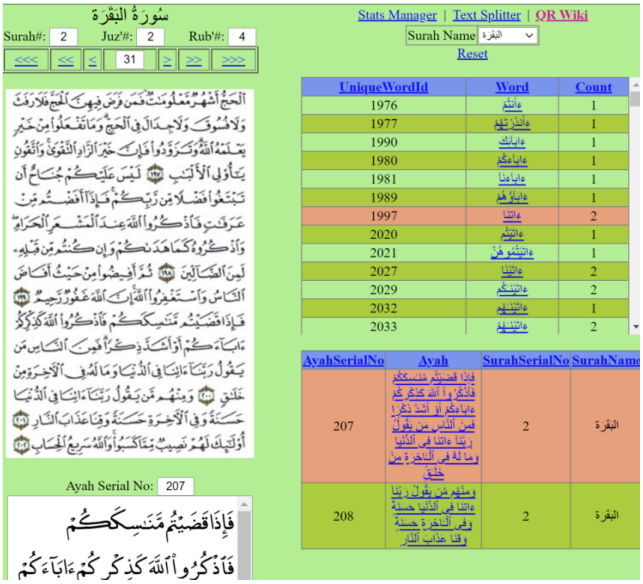


Figure 3.9: An example query result.

of the queries. The developer's side of the system is crucial to the extensibility of the system. Unlike existing applications that provide statistics on the Quran, QuranResearch.Org provides a platform for developers all over the world to envision their queries and to inspire their creativity about what type of insights to extract from the Quran text. This section provides the interface of the developer's dashboard and the culture behind it.

The developer will author a query in a system-accepted language. As of now, the system accepts SQL as its language. However, plans include the possibility of accepting multiple languages to query the Quran database, with the possibility of a graphical editor to compose queries. To write a query, the user would provide (through the developer's dashboard as in figures (3.7, 3.8, and 3.9) the following information:

- Title, which is a descriptive identification of the query.
- Description, which documents the query in a paragraph or two to highlight the main idea of the query.
- Documentation, where the user uploads a PDF file that documents the query details.
- Query (or main Query), which is an SQL query that generates the query results.

- Parameters, where the query can accept various sorts of parameters that slice the query result, for example, by surah, chapter (juz'), or ayah.
- Detail Query, which is another query that is executed once a row from the Main Query result is clicked. The Detail Query is linked to the Main Query through a hyperlink column. Once the hyperlink column in a row (in the Main Query result) is clicked, the value of that cell is passed to the Detail Query as a parameter. The result of the Detail Query is displayed under the main query result.
- Hyperlink columns, which are used to provide hyperlinks in the query results. These hyperlink columns serve as a linking mechanism between the Main Query result and the Detail Query. Also, the hyperlink columns serve as links in the query result so that the user can transition to the page and/or ayah that is referenced in the query result.

Figure 3.7 shows the developer's dashboard of a typical query with the title, description, documentation, and Main Query sections while Figure 3.8 expands the query parameters, the Detail Query and the hyperlink columns sections. The QuranResearch.Org website provides a series of videos on how to author queries and fill in all the required/optional sections to achieve the query result as desired by the developer. Once the query is written, the developer can test the query by clicking on the run button.

Figure 3.9 displays the user interface that shows the results of executed queries. When a developer runs a query, the system presents the data in a table format. If the query includes sub-queries, clicking on a record in the main table displays the sub-results in another table. Additionally, if the query results reference specific pages, chapters, or verses, clicking on these records displays the corresponding content on the opposite side of the interface. When the developer is satisfied with the query results, they can submit a request for publication. The system administrator reviews the query and, upon approval, makes it publicly available under the appropriate topic in the QR Wiki.

This interactive design enhances data exploration by allowing users to navigate through related data seamlessly. The interface employs event-driven actions to provide a dynamic

user experience. It is important to note that this page is also accessible to non-specialist users. They can run existing queries and view the results without needing any knowledge of SQL. They interact with the data by clicking on records to see detailed results, without dealing with the technical aspects behind the scenes.

3.7 Query System Security Protocols

To ensure the secure and reliable operation of the query system, several measures have been implemented. These measures protect against misuse while providing a smooth experience for legitimate users.

- **User Authentication:** Every user has a verified account. This verification ensures accountability for all actions within the system.
- **Role-Based Access Control:** Users have different access levels based on their roles. This limitation reduces the potential for unauthorized actions.
- **Query Rate Limiting:** The system restricts the number of queries a user can make within a specific time frame. Each user receives a default time quota each day. Every time a query is made, this quota decreases. If the quota is exhausted, the user must wait until the next day for a new quota. This mechanism prevents overloading the system or conducting automated attacks. It also stops resource-intensive queries that could slow down the system.
- **Audit Logging:** All user actions are logged. This logging allows for tracking any suspicious activities. It also aids in forensic analysis if needed.
- **Approval Workflow:** All queries are reviewed before publication. The system administrators perform this review. This step adds an extra layer of security and quality control.
- **Input Validation:** All user inputs are thoroughly checked for potential malicious code or SQL injection attempts. These precautions safeguard the integrity of the database. *Malicious code* is any code inserted into a system with harmful intent. It can damage,

disrupt, or gain unauthorized access to a computer system. *SQL injection* is a type of attack where attackers try to insert malicious SQL code into application queries. If successful, they can view, change, or delete data they should not access. They might even run commands on the database server. Input validation was implemented using:

1. **Input Screening:** The system checks all user inputs against a list of allowed characters. This process helps prevent unexpected inputs that could be part of an attack.
2. **Parameterized Queries:** We use prepared statements with parameterized queries. This approach separates the SQL code from the data. The database treats the input as data, not as part of the SQL command. This method makes it much harder for attackers to inject malicious SQL.
3. **Regular Expression Filtering:** We use regular expressions to check input patterns. The system rejects any input that does not match these patterns. This technique can catch many attempts to inject malicious code.

These measures work together to create a secure and reliable query system. They protect against misuse while ensuring a smooth user experience for legitimate users.

3.8 Summary

I worked as the Lead Developer for QuranResearch.org, a system that encompasses the various research aspects discussed in this dissertation. The proposed system aims to improve the performance of existing language models in Arabic and Quranic research. By providing a framework and diverse resources, it enhances the training and evaluation of language models on Arabic text, audio, and images.

The system includes features like the Text Digitizer and Analyzer Module, the Computer Vision Text Localizer, and the Semantic Similarity Detection Module. These modules contribute to standardizing Arabic data, improving text recognition accuracy, and uncovering hidden semantic relationships among entities. This standardized and enriched data can be

used to fine-tune existing language models. This enables them to better understand and process Arabic text, audio, and images.

Additionally, the Speech Localization, Segmentation, and Alignment Module addresses challenges related to diacritics and error correction in Arabic speech recognition. It aligns audio and text. This improves the accuracy of speech recognition systems. It also facilitates better training of speech recognition models on Arabic speech data.

Our scalable and accessible cloud-based database empowers researchers to explore and analyze Arabic datasets. They can write SQL statements directly on the database and run additional code. To ensure the system's integrity and security, we have implemented measures. These include user authentication, role-based access control, query rate limiting, and thorough input validation. This security framework maintains the reliability and trustworthiness of our research platform.

Researchers can also use the system's APIs as a backend for their own applications, taking advantage of our services. The APIs come with various settings and tuning options. This makes it easy for researchers to search and retrieve Quran text. This enables them to conduct in-depth Arabic and Quranic research. It contributes to advancing the performance and capabilities of existing language models in the Arabic language domain.

We will continually expand and refine the system's capabilities to support a wide range of NLP operations. This includes enhancing existing modules and incorporating new ones to handle tasks such as sentiment analysis, topic modeling, text summarization, language translation, and more. We will also leverage the latest advancements in NLP research. We will integrate state-of-the-art language models, such as transformer-based models. This will empower the system with improved language understanding and generation abilities. Our goal is to make the system a comprehensive and cutting-edge resource. We aim to cater to the diverse needs of Arabic and Quranic research while advancing the capabilities of NLP in the domain.

Chapter 4

AUTOMATED DIACRITIZATION AND STYLISTIC REALIZATION IN ARABIC TEXTUAL DATA: ADVANCING ARABIC LANGUAGE UNDERSTANDING

In Arabic, diacritics are essential for determining pronunciation and meaning [168]. Without understanding diacritical letters, it is challenging to grasp word meanings, sentences, and context. This chapter introduces an auto-diacritization solution for Arabic text called Automatic **Dia**critization and **Dia**lect System for Arabic **Qu**r**a**n Text **Rec**ognition (QRDiaRec). This solution serves as a fundamental component in enhancing the comprehension of the Arabic language. It focuses on understanding diacritical letters and recognizing various Arabic linguistic styles. Arabic features correct dialects with well-known origins, known as linguistic styles or "Riwayah" and "Qira'ah" in Arabic [68]. These styles encompass various aspects of grammar, rhetoric, and pronunciation. In this context, the Qur'an is an Arabic diacritic-annotated text with 20 authentic linguistic styles [137]. Currently, only seven of these styles have been digitized, while the remaining 13 are still in handwritten form. We aim to create a database of the Qur'an texts with the available linguistic styles, tracking the origins of each style's data. We designed a Process-Oriented Provenance Model [144] to trace the evolution of Quranic linguistic styles through narrators and their lineages. Similar to the work of Kang et al. [105], who employed provenance tracking for synthetic data, our model tracks the provenance of linguistic styles in real texts. However, unlike their focus on synthetic text inspection with tools like INSPECTOR [105], we concentrate on human-authored texts to understand how different styles originated and evolved. Our model ensures that the data's lineage is well-documented, providing a reliable foundation for linguistic analysis and research. This effort prepares a dataset containing non-diacritic Arabic text and

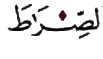
The word: الصراط			
Linguistic Styles	**Word**	**Pronunciation**	**Unicode**
Qonb and Ruwais		a**s-zi**raatta	**0627** 0671 **0644** **0635** 0651 0650 (06F0) **0631** 064E 0656 **0637** 064E
Khalf and Khallaad		a**s-sii**raatta	**0627** 0671 **0644** **0635** 0651 0650 (06E3) **0631** 064E 0656 **0637** 064E
The other styles with Khallaad		a**s-sa**raatta	**0627** 0671 **0644** **0635** 0651 0650 (null) **0631** 064E 0656 **0637** 064E

Table 4.1: Example of the orthographic and phonetic diversity for one word across different Arabic linguistic styles.

the corresponding diacritic-annotated text, aiding AI models in the automated conversion process. To visualize the complex relationships among different linguistic styles and their origins [164], we developed interactive visualization tool [142]. These tool allow researchers to explore the connections between narrators and styles, enhancing understanding of how linguistic features evolved over time.

Transferring NLP techniques from languages like English to Arabic is not straightforward due to the unique features of Arabic script and the significance of diacritics [68]. The same word can have different meanings based on the diacritic marks added to it. For example, the word "الصراط" is pronounced differently across various linguistic styles, as shown in Table 4.1. While the basic shape of the word remains consistent, variations in diacritics alter its pronunciation. These differences might add meaning or reflect regional pronunciations. Some

linguistic styles share identical forms, while others exhibit distinct variations. Notably, the Khallad style allows for two correct pronunciations of the same word, demonstrating the complexity and nuance of Arabic linguistic styles. The Unicode column in Table 4.1 highlights the Unicode representations of the Arabic letters and diacritics. Letters are represented in bold, while diacritics are displayed in regular font. Differences in Unicode encodings across the linguistic styles are emphasized with a yellow background for easy identification. Managing such complexity requires robust tools and methodologies. To address these challenges, we developed a system that combines data modeling, interactive visualization, and advanced deep learning techniques.

To address these challenges, we developed character-level sequence-to-sequence models [198] for auto-diacritization. This model utilizes the bidirectional LSTM (Long Short-Term Memory) [77], achieving an accuracy of 92.1%, while the bidirectional GRU (Gated Recurrent Unit) [3] achieved 94.2%. The transformer-based model, known for its ability to model long-range dependencies, achieved an accuracy of 92.1%. This system enables us to interpret Arabic text, understand word meanings, sentences, and context, and explore various linguistic styles and dialects. This system holds significant potential to benefit several areas, including NLP, sentiment analysis, machine translation, speech recognition and synthesis, information retrieval, text-to-speech systems, and Arabic computational linguistics.

The contributions of this section of the study can be summarized as follows:

- **Process-Oriented Provenance Model:** Developed a novel model to trace the evolution of Quranic linguistic styles through narrators and their lineages.
- **Provenance Tracking of Linguistic Styles:** A system to track the provenance of Quranic styles through narrators and their lineages.
- **Arabic Character Normalization Algorithm:** Designed an algorithm to ensure consistent representation of Arabic characters across texts.
- **Interactive Narrators' Data Visualization Tools:** interactive tool to visualizes narrators' relationships and linguistic styles, aiding researchers in exploring complex

linguistic data.

- **Establishing a database:** Compiled a database encompassing multiple linguistic styles of the Arabic Quran text, making it accessible to researchers and practitioners.
- **Building an Auto-diacritical system:** Developed QRDiaRec, an auto-diacritization system for Arabic text.
- **Utilizing deep learning models:** Achieved effective Arabic diacritization with an accuracy of 94.2% using advanced deep learning models.

4.1 The Proposed Process-Oriented Provenance Model for Quranic Linguistic Styles

In our research, we have created a comprehensive database of the Quranic text that includes multiple linguistic styles. Each linguistic style has been preserved by a group of individuals over time and across different geographical areas. We refer to each of these individuals as a Narrator.

These linguistic styles represent the same text but differ in certain aspects. They have variations in diacritics and different interpretations of where sentences or words begin and end. For example, one linguistic style might consider two sentences as one indivisible unit. Another style might divide those same sentences differently. Similarly, a style might interpret a word as consisting of four letters, followed by a six-letter word. Another style might split these letters into two five-letter words. This can lead to new meanings that add to the original without conflicting with it. This variation occurs because, in the Arabic language, the boundaries between words not always a space.

Although the text remains the same, the tenth word in one linguistic style does not necessarily correspond exactly to the tenth word in another style. The problem is that there is no existing system, model, or framework that combines these styles. Each existing style is often treated separately. To address these issues, we developed a provenance model that establishes relationships between the narrators, the choices that shape the linguistic style,

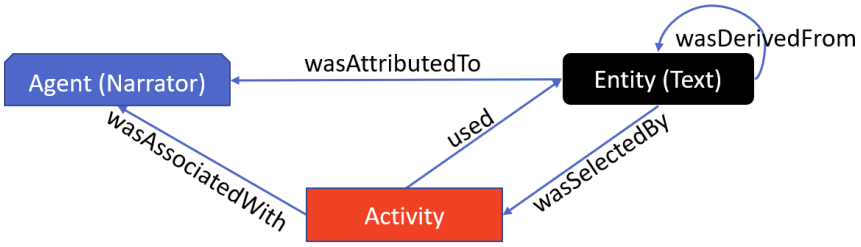


Figure 4.1: Diagram of the Process-Oriented Provenance Model Tracing Quranic Style Evolution Through Narrators.

and the linguistic styles.

This model serves as the foundation upon which our database is built. It creates an interconnected system that will enable more extensive scientific research in the future. Researchers can explore differences between styles at the text level, image level, phonetic level, and even at the level of modern dialects. These modern dialects can be

However, we acknowledge the existence of various dialects during the Quran's revelation. The Prophet Muhammad (PBUH) allowed the use of certain words with specific dialects, following established guidelines. Narrators, within these guidelines, exercised some choice in style selection. Narrators refer to individuals who have memorized the entire Qur'an with one or more linguistic style and are qualified to teach others. This flexibility could involve choosing a dialect, method, or style most familiar to them. Over time, this practice resulted in the emergence of the twenty known and defined linguistic styles. It is important to emphasize that narrators did not modify the Quranic text. Their choices involved selecting from the options permitted by the Prophet (PBUH). A narrator's style preference could be transmitted to their students, potentially leading to a lineage of narrators using the same style.

Lineage data is metadata that documents the narrators who preserved and transmitted the linguistics styles. This metadata includes relationships between narrators, their predecessors, and successors, as well as the historical paths through which linguistic styles were passed down. By tracing the origin and transmission paths of a specific linguistic style, we can confirm its authenticity. A recognized chain of narrators strengthens the credibility of a style,

while a break in the chain—such as a narrator whose origins or students are unknown—can indicate potential issues. These insights are visually represented in figure 4.3.

The lineage data has practical applications, such as:

- Validation of Styles: Tracing the chain of narrators confirms the originality and consistency of a linguistic style.
- Error Detection: Identifying inconsistencies in the transmission chain highlights potential transcription errors.

Our Process-Oriented Provenance model sheds light on the dynamic process of Quranic transmission among the narrators and the development of diverse linguistic styles. It traces how Quranic linguistic styles evolved and were passed down through narrators. It also examines the choices narrators made when transmitting these styles, providing valuable insights. This research follows the development and transmission of Quranic linguistic styles through narrators, shedding light on their transmission choices.

1. **Activity:** This component represents the narrator's choice, which shapes the linguistic style.
    - An arrow labeled 'wasAssociatedWith' connects the Activity to the Agent (Narrator), indicating the narrator's involvement in the activity.
    - An arrow labeled 'used' connects the Activity to the Entity (Text), specifying the text used in the activity.
    - An arrow labeled 'wasSelectedBy' leads from Entity (Text) to Activity, denoting the text selections during the activity.
2. **Agent (Narrator):** This represents individuals (narrators) who have transmitted the Quranic text.
    - An arrow labeled 'wasAttributedTo' connects Entity (Text) to Agent, attributing the text selections to the narrator.
3. **Entity (Text):** This pertains to the Quranic text or the linguistic style.

- An arrow labeled 'wasDerivedFrom' loops from Entity to itself, reflecting the derivation of a specific linguistic style from the text.

This provenance model is the foundational framework we followed to create our database. It effectively captures the interconnections between Quranic narrators and linguistic styles. This framework is crucial for understanding the historical and stylistic evolution of the Quranic text. It confirms the systematic approach we used to document and analyze the development of linguistic styles within the Quran.

4.2 QRDiaRec System: Key Operations

The main objective of QRDiaRec is to create an automated solution for diacritization and dialect recognition in Arabic text. The system begins with the data collection process, which involves extracting text data from various sources and transforming it into a unified format. This data is then stored in the database. This database provides us with datasets for training and evaluation of the AI models employed in the system. To handle the challenges posed by different Unicode representations of Arabic characters [168], the Text Digitizer and Analyzer Module is employed. This module maps various representations to a unified format, facilitating easier comparison and analysis of the text. The Data Validation and Integration module ensures the quality and consistency of the integrated data by validating and contrasting multiple copies of the same text. Finally, the QRDiaRec system utilizes deep learning models for Arabic diacritization.

4.2.1 The Data Collector

In this module, we utilize ETL techniques to extract, transform, and load data into a suitable format for processing within the system. Due to the lack of open-source databases for the Quran with more than one linguistic style, the development of web scraping mechanisms became essential for data extraction. The system incorporates web scraping algorithms that automatically extract Quran text from various online sources [56]. These algorithms are task-specific and vary based on the structure of the data source. While the system

also encompasses algorithms for extracting audio data and images, this particular part of the study concentrated solely on textual data. The text data is scraped from a range of websites, documents, and databases, each with its unique structure. We exclusively utilized publicly available, non-proprietary Quranic texts, ensuring that our data collection process does not involve any privacy-sensitive information. As a consequence, dedicated algorithms were implemented to handle the extraction process for each specific data source. The system records the data from each source and the corresponding acquisition timestamps, ensuring that only new data is collected during subsequent extraction runs. Once the data is obtained, it undergoes a transformation process to unify its format. After the data acquisition, a transformation process is applied to convert this heterogeneous data into a unified format. This step incorporates established data science practices [95] like the standardization of data cleansing. We use deduplication algorithms to eliminate duplicates. The organization is based on database normalization principles [107]. These practices facilitate rigorous analysis and validation. Furthermore, advanced tools are developed to partition and distribute the transformed data efficiently within our database structure. By implementing this module, the system minimizes manual effort and reduces errors. It achieves automation in data collection, allowing for consistent and up-to-date data gathering from diverse sources.

4.2.2 Linguistic Styles' Provenance Tracker

The data includes multiple linguistic styles with potential variations from different sources. To differentiate version errors from style differences, we developed the Linguistic Styles' Provenance Tracker [147]. The tracker aids data lineage analysis that leads to understanding the interdependencies among linguistic styles. Lineage metadata [170] provides insights into the individuals involved in moving the linguistic style data from its original source to its final state. The Tracker is important for maintaining data integrity. It helps us distinguish between genuine linguistic variations and potential errors. This tracker records the journey of each linguistic style from its source to its current state. By doing so, we can trace back any discrepancies to their origin. It records who handled the data, and when it was modified.

This detailed history allows researchers to study how different Arabic dialects have evolved over time. It also helps in identifying and correcting any unintended alterations to the original texts. The tracker ensures that our system provides reliable and verifiable data for Arabic language studies.

Each linguistic style has a group of narrators who have preserved and passed it down through the generations. The linguistic style is named after the most renowned individual who mastered it, as well as his teacher's name. In Table 4.2, the teacher's name is referred to as the "Main Narrator". The most famous expert in the linguistic style is referred to as the "Narrator". Usually, dialects appear in one region and then spread to neighboring regions. We determined the region where the linguistic style appears according to the location of the "main narrator". Region data is shown by the "Place" column that defines the main region, and the "City" column.

The origin of the Quranic linguistic style can be traced back to the time of Prophet Muhammad PBUH, with individuals directly taught by him being assigned a lineage level of 1. Those who learned from anyone from level 1 are assigned a lineage level of 2, and so on. This lineage level is denoted as "Lineage" in the table. The "Typed" column indicates whether or not we have access to an electronically typed version. The "Scanned pages" column represents the number of pages scanned from the respective Quran book. Quran book is known as Mushaf. The column "Counting Scheme" specifies the name of the Scheme used to determine the start and end of verses.

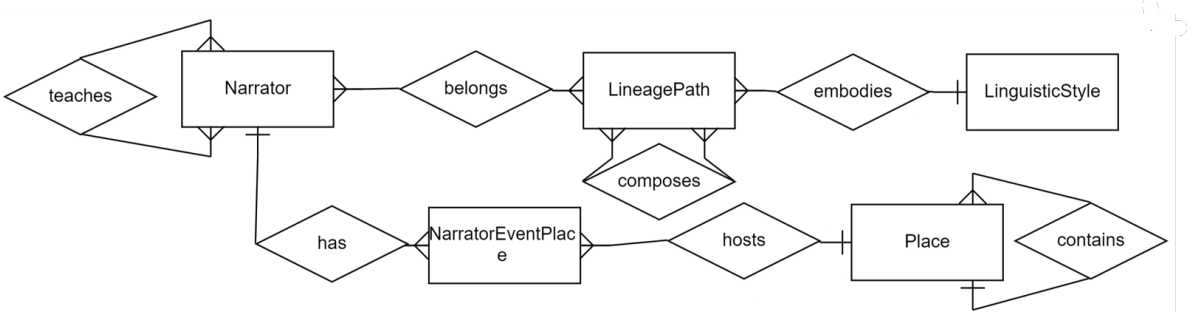


Figure 4.2: ER diagram for narrators, linguistic styles, and data lineage in the Quranic context.

Table 4.2: Data of the Quran and its associated linguistic styles.

Place	City	Main Narrator	Lineage	Counting scheme	Narrator	Lineage	Typed	Scanned pages
Hejaz	Almadina	Nafie	4	Madani Awal	Qalun	5	Yes	559
			4	Madani Akhir	Warsh	5	Yes	604
		AboJaafr	3	Madani Awal	Abn-Wardan	4	No	559
			3		Abn-Jammaz	4	No	604
	Makkah	Ibn-Kathir	3	Makki	Al-Bazi	6	Yes	604
			3		Qonbl	7	Yes	604
Iraq	Basra	AbuAmr	4	Basri	Al-Dawri	6	Yes	604
			4		Al-Suwsi	6	Yes	521
		Ya'qub	4		Ruwais	5	No	604
			4		Rauoh	5	No	604
	Kufa	Hamza	4	Kufi	Khalf	6	No	604
			4		Khallaad	6	No	604
		AlKisayiy	5		Abul-Harith	6	No	604
			5		AlDawri	6	No	604
		Assem	3		Hafs	4	Yes	604
			3		Shueba	4	Yes	604
		Khalaf	6		Ishaq	7	No	604
			6		Iddris	7	No	604
Sham	Damascus	IbnAmer	2	Damascus	Hisham	5	No	604
					Ibn-Thakwan	5	No	604

Figure 4.2 illustrates a part of database design that captures information about narrators' data and linguistic styles of the Quran.

- Each narrator learns from one or more other narrators. Basic data about each narrator is recorded, along with some data related to events that occurred for each narrator.
- Event data related to each narrator, including their birth, transition, death, etc., is recorded in the entity 'NarratorEventPlace.' The 'Place' entity stores data about the location of each event. The temporal and spatial data of the narrators are recorded to verify their presence at the same place and time as the narrator who taught them, ensuring the authenticity of the linguistic style.
- Narrators transfer linguistic styles through 'lineage paths,' which start with the origin (the first narrator) and end with a specific narrator. Each lineage path is assigned a 'lineage level.' For example, a narrator learning directly from the origin is at level 1, while a narrator learning from a level 1 narrator is at level 2, and so on. Lineage paths can also include multiple other lineage paths
- Each narrator can be associated with multiple lineage paths, and vice versa. A linguistic style is specifically associated with one lineage path. The last narrator in that lineage path is considered the most renowned master of the particular linguistic style.

This part of the database can be regarded as a data-oriented provenance model. This model captures information about narrators, lineage paths, and linguistic styles within the Quranic narrators' network. By providing a provenance trail, it offers insights into the lineage, authenticity, and evolution of linguistic styles.

To visualize the tracking process, a directed acyclic graph (DAG) can be employed [176]. In a DAG, each edge has a specific direction, indicating a unidirectional flow. Importantly, a DAG is acyclic, meaning it does not contain any loops. Graphically, vertices are used to represent narrators, and edges symbolize the flow between them.

Figure 4.3 illustrates one lineage path associated with a linguistic style. The origin (first narrator) is depicted as the initial point on the left side. Vertices connected to the right

of the origin represent the subsequent levels of students. When a narrator is selected, the corresponding vertex is displayed in black. Narrators who taught the selected narrator are shown on the left in blue, while narrators who are students of the selected narrator are displayed on the right in red. A more comprehensive visualization of the interconnected network of narrators can be found on our website associated with this research project [163].

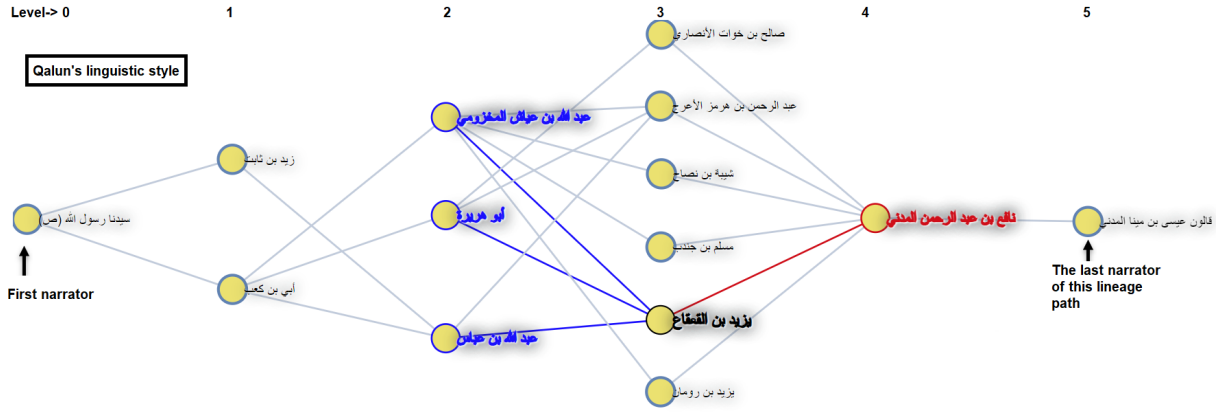


Figure 4.3: The lineage path of a linguistic style (Qalun) in the Quranic narrators' network.

4.2.3 Text Digitizer and Analyzer Module

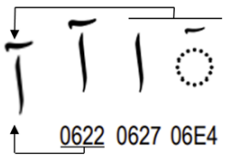


Figure 4.4: Illustration of Diverse Unicode Representations for a Arabic Character

The Text Digitizer and Analyzer Module addresses the challenge posed by multiple Unicode representations for Arabic characters, resulting in variations in the encoded text. In Arabic, certain characters can have different Unicode representations, resulting in variations in the encoding of the same text (See figure 4.4). These variations can be attributed to different font styles, regional preferences, or historical conventions.

Algorithm 1: Arabic Character-based Normalization Algorithm: Unified Unicode Representation

Input : *ArText:* Arabic text in any Unicode representation.
LingStyle: The type of linguistic style used in the *ArText*.
Output: UnifiedArText: The unified Arabic text

```
Initialize ArabicCharacterNormalizationMapping data structure;
Initialize CharacterEncodingAlternatives data structure;
function NonStandardUnicodeEncodingDetection(ArText,LingStyle);
begin
    Initialize CharacterConversionPositionsList as an empty list;
    forall PossibleUnicodeEncoding in CharacterEncodingAlternatives(LingStyle) do
        if PossibleUnicodeEncoding is found in ArText then
            Push characterId, position to CharacterConversionPositionsList;
    Return CharacterConversionPositionsList;
function EncodingConversionMapper(ArText, LingStyle,
 CharacterConversionPositionsList);
begin
    forall CharacterId and position in CharacterConversionPositionsList do
        Retrieve TargetUnicodeEncoding based on CharacterId, LingStyle from
         ArabicCharacterNormalizationMapping;
        Transpose PossibleUnicodeEncoding to TargetUnicodeEncoding in ArText;
    Return ArText;
Main:
Read ArText in any Unicode representation;
Read LingStyle that is used in ArText;
Call NonStandardUnicodeEncodingDetection with ArText and LingStyle;
Obtain CharacterConversionPositionsList;
Call EncodingConversionMapper with ArText, LingStyle, and
 CharacterConversionPositionsList;
Obtain the UnifiedArText after encoding conversion;
```

For example, the letter آ *(alef with madda)* can be represented by Unicode code point 0622 or by combining Unicode code points 0627 *(alef)* and 06E4 *(madda above)*. Given the diverse origins and sources of the data, achieving a unified format becomes essential for meaningful comparison and analysis of the datasets. The Linguistic Styles' Provenance Tracker enables this module to incorporate linguistic style-specific rules and considerations, to apply appropriate normalization techniques. To tackle this, we have developed an algorithm known as the Arabic Character-based Normalization Algorithm: Unified Unicode Representation 1. Character-based normalization is a process of transforming characters in the text to a standardized or unified form. This algorithm 1 is an integral part of the Text Digitizer and Analyzer Module and plays a pivotal role in achieving a standardized representation of Arabic text within our system. This module 1) improves the search in the Arabic and Quranic text, 2) discovers differences between copies of the same linguistic style,2) increases the accuracy of discovering differences between different linguistic styles, and 4) creates consistent statistical results.

Algorithm 1 addresses the challenge posed by variations in Unicode encodings for Arabic characters, which can lead to inconsistencies and hinder the analysis and processing of the Arabic text. The algorithm utilizes several key data structures to facilitate the normalization process. Firstly, the "ArabicCharacterNormalizationMapping" data structure contains all Arabic characters. For each character, the structure includes its CharacterId, CharacterDescription, and the target Unicode representation based on each linguistic style referred to as "TargetUnicodeEncoding." The Unicode encoding can be a composite of multiple Unicode codes required to represent a character.

Moreover, the "CharacterEncodingAlternatives" data structure records all possible Unicode representations for each character based on the linguistic style except TargetUnicodeEncoding. Since a character can have multiple valid Unicode representations, this data structure accounts for the various possibilities.

The normalization process begins by parsing the input Arabic text. The text undergoes

an evaluation within the NonStandardUnicodeEncodingDetection function. This function checks for the presence of any PossibleUnicodeEncoding within the CharacterEncodingAlternatives data structure based on the linguistic style. Whenever a PossibleUnicodeEncoding is found, the algorithm records its position in the text, creating a list known as "CharacterConversionPositionsList." This list accumulates the CharacterId and the respective position of each identified PossibleUnicodeEncoding within the text.

Following the detection of non-standard Unicode encodings, the EncodingConversionMapper function comes into play. This function maps the PossibleUnicodeEncoding found in the input text to their corresponding TargetUnicodeEncoding based on the associated CharacterId and the linguistic style. By performing this transposition, the algorithm ensures that the text adopts the unified Unicode representation specified in the ArabicCharacterNormalizationMapping data structure. The resulting transformed text is then returned as the output of the function.

Algorithm 1 is not merely traversing a mapping structure. It relies on the ArabicCharacterNormalizationMapping and CharacterEncodingAlternatives. Its main contribution is providing a systematic method to normalize Arabic text at the character level across different linguistic styles. Initially, we manually created these mappings for common representations. We then used automated tools to analyze various data sources. These tools helped us discover additional representations. We added these new representations to the alternatives. Over time, the data structures stabilized. We carefully accounted for the variations present in the data. This normalization is essential for accurate text processing and analysis. It ensures consistency across different sources and linguistic styles. The algorithm enables easy comparison between texts collected from various origins. It also handles texts written with different Unicode representations. Without such normalization, finding relationships or correlations between texts would be challenging. Different Unicode encodings would create significant barriers to comparison. Therefore, Algorithm 1 is fundamental for unifying diverse texts into a consistent dataset.

By applying the Arabic Character-based Normalization Algorithm, the Text Digitizer and Analyzer Module achieve a standardized and digitized representation of Arabic text. This unified format enables the comparison and analysis of diverse datasets, regardless of the variations in Unicode encodings.

4.2.4 The Data Validation and Integration Module

Algorithm 2: Stylistic Conflation and Text Validation Algorithm

```
Input  : ConflatedWords, WordsToConflation
Output: ConflatedWords
1  while WordsToConflation.hasUnConflatedBooks do
2      WordsToConflation.InitializeNextBook();
3      while true do
4          BestMappingCase ←
            WordConflationMapper(ConflatedWords, WordsToConflation);
5          if BestMappingCase.AreSimilar then
6              InConflationMapping.add(ConflatedWords, WordsToConflation)
7          else
8              if ConflatedWords.hasWord and WordsToConflation.hasWord then
9                  InConflationMapping.add(ConflatedWords, null);
10                 InConflationMapping.add(null, WordsToConflation);
11             else if not ConflatedWords.hasWord and
                WordsToConflation.hasWord then
12                 InConflationMapping.add(null, WordsToConflation);
13             else if ConflatedWords.hasWord and not
                WordsToConflation.hasWord then
14                 InConflationMapping.add(ConflatedWords, null);
15             else if not ConflatedWords.hasWord and not
                WordsToConflation.hasWord then
16                 break;
17             else
18                 SaveUnexpectedCase (ConflatedWords, WordsToConflation);
19         ConflatedWords ← InConflationMapping;
20 return ConflatedWords;
```

This module accepts the unified format of the data, as given by the Text Digitizer and

Analyzer Module. This module takes advantage of Linguistic Styles' Provenance Tracker to differentiate between different styles and between multiple versions of the same style. The module integrates data copies from multiple sources and validates these data copies by contrasting these copies against each other. The Data Validation and Integration Module is designed to enhance the trustworthiness [34] and adoption of the system by ensuring the quality and consistency of the input data.

One of the algorithms within this module is the Stylistic Conflation and Text Validation Algorithm 2. Its main objective is to facilitate the mapping of words among different books that contain the same text but may differ linguistic style, or contain typos. This algorithm plays a critical role in identifying variations in word representation across different linguistic styles used in the books and detecting misspellings, missing words, or extra words in the text.

The proposed algorithm aims to facilitate the mapping of words. The primary objectives of this mapping process are: To identify variations in the representation of each word across different linguistic styles used in the books. To detect misspellings, missing words, or extra words in the text. The algorithm operates as follows:

Initially, the words from the first book are loaded into the ConflatedWords, which serves as the base mapping for all subsequent comparisons. For each additional book, the words are loaded into WordsToConflation, representing the words to be mapped. The algorithm iterates through the process until all the books have been compared. The wordConflationMapper function is called, which returns the BestMappingCase representing the similarity between WordsToConflation and ConflatedWords. If BestMappingCase indicates that the words are similar, the similarity information between WordsToConflation and ConflatedWords is stored in InConflationMapping. If BestMappingCase does not indicate similarity, the algorithm evaluates five conditions to determine the case and perform the appropriate mapping:

- If there are words in both WordsToConflation and ConflatedWords, InConflationMapping adds the data from WordConflationMapping and ConflatedWords. The process

then moves on to the next words for comparison.

- If ConflatedWords has no more words to map and there are still words in WordsToConflation, InConflationMapping adds the remaining data from WordsToConflation without corresponding words.
- If WordsToConflation has no more words and there are still words in ConflatedWords, InConflationMapping adds the remaining data from ConflatedWords without corresponding words.
- If both WordsToConflation and ConflatedWords have no more words to map, the mapping process is completed, and the algorithm exits.
- If none of the expected cases occur, the algorithm adds the situation as an unexpected case.

The "hasWord" function returns true if one or more words exist in a given set, while the "hasWords" function returns true if more than one word exists in the set. Finally, the data in InConflationMapping, representing the updated mapping between the words, is saved as the new ConflatedWords, ready to be used for subsequent comparisons

The Word Mapping Algorithm 3 is a part of the Stylistic Conflation and Text Validation Algorithm. Algorithm 3 is specifically designed to map two sets of words, namely ConflatedWords and WordsToConflation. The algorithm leverages the concept of Levenshtein similarity, which quantifies the degree of similarity between words based on the minimum number of operations required to transform one word into another [199]. The similarity calculation is similar to the Levenshtein similarity used for English words. However, Arabic diacritics are treated as distinct Unicode characters, effectively making them separate 'letters'. This means that differences in diacritics result in greater divergence in similarity scores. It is important to note that the comparison between the second and next words is indeed accounted for in the iterative process of the Stylistic Conflation and Text Validation Algorithm. The WordConflationMapper is called within a loop (line 4) of Algorithm 3, which advances cursors through the words. This ensures that comparisons involving the sec-

ond next words occur in subsequent iterations as the cursors progress. The Word Mapping Algorithm aims to determine the best mapping case between the words.

Algorithm 3: The Word Mapping Algorithm

Input : ConflatedWords, WordsToConflation
Output: BestMappingCase

```
1  Function WordConflationMapper(ConflatedWords, WordsToConflation):
2      DecisionList ← EmptyList;
3      if WordsToConflation.hasWord and ConflatedWords.hasWord then
           // Case 1
4          SimilarityResult ← LevensteinSimilarity(WordsToConflation.nextWord,
             ConflatedWords.nextWord);
5          DecisionList.push(Case1, SimilarityResult);
6      if WordsToConflation.hasWord and ConflatedWords.hasWords then
           // Case 2
7          SimilarityResult ← LevensteinSimilarity(WordsToConflation.nextWord,
             ConflatedWords.nextTwoWords);
8          DecisionList.push(Case2, SimilarityResult);
           // Case 4
9          SimilarityResult ← LevensteinSimilarity(WordsToConflation.nextWord,
             ConflatedWords.secondNextWord);
10         DecisionList.push(Case4, SimilarityResult);
11     if WordsToConflation.hasWords and ConflatedWords.hasWord then
           // Case 3
12         SimilarityResult ← LevensteinSimilarity(WordsToConflation.nextTwoWords,
             ConflatedWords.nextWord);
13         DecisionList.push(Case3, SimilarityResult);
           // Case 5
14         SimilarityResult ←
             LevensteinSimilarity(WordsToConflation.secondNextWord,
             ConflatedWords.nextWord);
15         DecisionList.push(Case5, SimilarityResult);
16     Return DecisionList.BestDecision();
```

The Word Mapping Algorithm 3 proceeds by evaluating five distinct cases of word collocations. These cases dictate how the words from WordsToConflation correspond to those from ConflatedWords, aiming to establish associations and alignments between the two sets of words. The five cases are as follows and illustrated in figure 4.5:

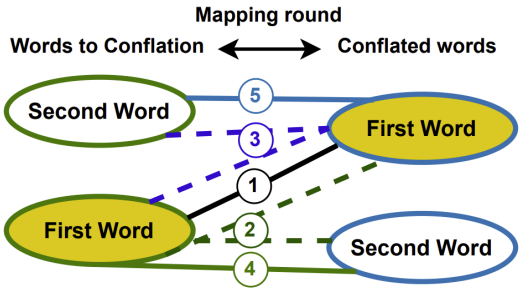


Figure 4.5: Visual Representation of Diverse Word Collocation Scenarios for Arabic Text Mapping Algorithm

- **Case 1:** The algorithm examines whether the next first word in WordsToConflation corresponds to the next first word in ConflatedWords.
- **Case 2:** This case considers situations where the next first word in WordsToConflation corresponds to the next two words in ConflatedWords.
- **Case 3:** In this case, the algorithm examines whether the next first and second words of WordsToConflation correspond to the next first word in ConflatedWords.
- **Case 4:** The algorithm evaluates scenarios where the next first word of WordsToConflation corresponds to the next second word in ConflatedWords.
- **Case 5:** This case addresses situations where the next second word of WordsToConflation corresponds to the next first word in ConflatedWords.

The algorithm aims to determine the highest similarity, prioritizing the case order. Ultimately, the algorithm returns the best mapping case, encompassing information such as the case number, and the similarity.

4.2.5 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 (Refer Section 7.1.4 and Algorithm 7). 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 [198]. 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 can generate sequences of variable length [161], making it suitable for handling the varying lengths of Arabic words and sentences.

Table 4.1 displays the Unicode representation of Arabic letters and diacritics across different linguistic styles. The table highlights the differences in Unicode encoding. These differences enable deep learning models to identify and represent diacritics accurately. Additionally, the models learn to differentiate between linguistic styles and understand that a word can have multiple valid representations depending on the style. By utilizing bidirectional LSTM or GRU layers, the model can effectively capture both past and future context [194], enabling it to make informed decisions about appropriate diacritics based on the surrounding characters. The deep learning models employed in our study learn to predict diacritics through the following mechanisms:

- **Contextual Learning:** Capturing the relationships between characters and diacritics based on the surrounding context within the text.
- **Pattern Recognition:** Identifying patterns where specific letters are commonly as-

sociated with certain diacritics or sequences of diacritics.

- **Representation Learning:** Developing dense vector representations for diacritics that encapsulate their semantic and phonetic properties, allowing the model to generalize to unseen data.

Through training on data from multiple linguistic styles, the models develop the ability to handle variations in diacritic usage, enhancing their ability to diacritize text across different styles.

LSTM and GRU are different types of recurrent neural network (RNN) architectures used for sequence modeling [161]. 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 [198]. 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 [54]. LSTM has a more complex structure than GRU, with more parameters, which allows it to capture longer-term dependencies in the input sequence [194]. GRU has a simpler structure with fewer parameters, making it computationally more efficient and easier to train [198].

The transformer-based model is employed for diacritizing Arabic text, offering a broader range of alternative approaches [178]. This model leverages self-attention mechanisms to model long-range dependencies and capture contextual relationships within the text [55]. Inspired by the influential "Attention Is All You Need" architecture [183], our model excels at handling large-scale datasets and effectively captures global dependencies across the entire sequence [183]. 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 4 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.

Algorithm 4: Training a Character-Level Language Model

Require:

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}

Ensure:

Trained character-level language model M

function PreprocessData
    Convert sentences in D_{train} and D_{test} into character lists
    Convert character lists into integer representations
    Pad the sequences to ensure the uniform length
function InitializeModel
    // The implementation of this function varies depending on the model used.
function CompileAndTrainModel
    Compile the model M using the sparse categorical cross-entropy loss function and the Adam optimizer
    Train the model M on the preprocessed training data, iterating over N_{epochs} epochs with a batch size of B_{size}
    Evaluate the model's performance on the validation data
    Save the trained model M for future use
PreprocessData
InitializeModel
CompileAndTrainModel

Our models, implemented using the Keras framework, are powerful deep-learning algorithms for capturing and predicting patterns in sequential data [135]. The algorithmic steps for training the models are summarized in Algorithm 4. 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 [60]. 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 [60]. 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 [48]. This loss function is suitable for multi-class classification tasks like diacritization, where each character can have multiple possible diacritics. We can express the sparse categorical cross-entropy loss function with the following equation:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log p_{i,y_i}$$

Explanation of the equation:

- L is the average loss over all samples in the training batch.
- N is the total number of samples (characters) in the batch.
- y_i is the true class label for the i-th sample. In our case, it represents the correct diacritic class for the i-th character.

- p_{i,y_i} is the predicted probability that the model assigns to the true class y_i for the i-th sample. This probability comes from the model's output after applying the softmax activation function.

Understanding the loss function components:

- For each character in the input sequence, the model predicts a probability distribution over all possible diacritic classes.
- The term $\log p_{i,y_i}$ calculates the natural logarithm of the predicted probability for the correct diacritic class.
- By taking the negative logarithm, $-\log p_{i,y_i}$, we measure how well the model predicts the true class. A higher predicted probability for the correct class results in a lower loss, which is desirable.
- Summing over all samples and dividing by N gives us the average loss per sample.

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 [200].

Regarding the InitializeModel function, we can combine the idea of implementing LSTM and GRU into Algorithm 4. 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 [130]. 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 [127] 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 5: Initialize the GRU and LSTM Models

Require:
Number of LSTM/GRU units $U_{\text{LSTM/GRU}}$

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 $U_{\text{LSTM/GRU}}$ units
4: Create a dense layer with softmax activation

Algorithm 6 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 [187]. 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 [55]. 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 [49]:

- 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 4 and 6 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 4. By incorporating these algorithms, the implementation process for the Automated Arabic Diacritization Module is accomplished, encompassing all requisite steps.

Algorithm 6: Initialize the Transformer Model

Require:
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

4.3 Integrating Linguistic Styles and Diacritization Dataset: Building an Arabic Multi-Style Database Model

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 organizes different Quran Arabic linguistic styles at various levels, such as chapters, verses/sentences, words, letters, and diacritics. The database serves as the foundation for the dataset. The dataset is derived from this database, prepared for various learning models, and divided into training and evaluation subsets to improve diacritization accuracy.

4.3.1 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.

Figure 4.6 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 144 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 characters. 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

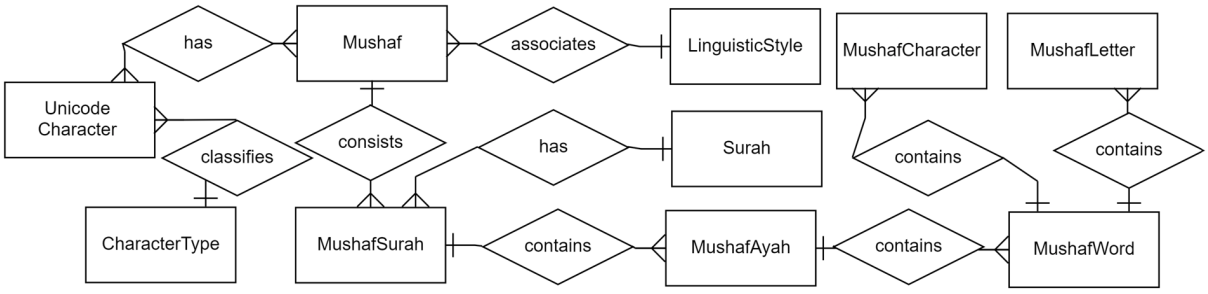


Figure 4.6: Database Schema for Modeling Linguistic Styles in the Qur'an

4.3.2 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 4.3 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 4.3: Auto Diacritization Dataset Statistics

Number of Linguistic Styles	7
Number of Tokens/Words	542,020
Number of Characters without Diacritics	2,283,177
Number of Diacritics	2,203,148
Number of Characters in the dataset	4,486,325

Table 4.4, 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 sentence without diacritics and the same sentence with diacritical letters. This pairing facilitates the training and evaluation of diacritization models. 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 4.4 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

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 4.4: Dataset Statistics: Multistyle Dialects for Automated Arabic Diacritization

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.

The deep learning models employed in this study (LSTM, GRU, and Transformer networks) learn to predict diacritics through the following mechanisms:

Contextual Learning: Capturing the relationships between characters and diacritics based on the surrounding context within the text. Pattern Recognition: Identifying patterns where specific letters are commonly associated with certain diacritics or sequences of diacritics. Representation Learning: Developing dense vector representations for diacritics that encap-

sulate their semantic and phonetic properties, allowing the model to generalize to unseen data. Through training on data from multiple linguistic styles, the models develop the ability to handle variations in diacritic usage, enhancing their ability to diacritize text across different styles.

4.4 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 [198].

4.4.1 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.

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.

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

Table 4.5: Performance Metrics of LSTM Algorithm in Diacritizing Arabic Text

Table 4.5 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.

Assessment of LSTM Model Generalization: Training vs. Validation Accuracies

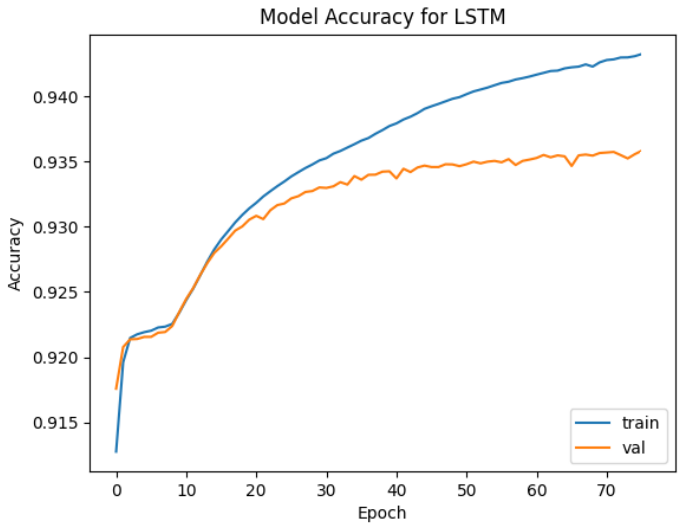


Figure 4.7: Accuracy Comparison: Training and Validation Results for the LSTM Model

Figure 4.7, 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 [99]. 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.

Analyzing Loss in LSTM Model: Training vs. Validation Results

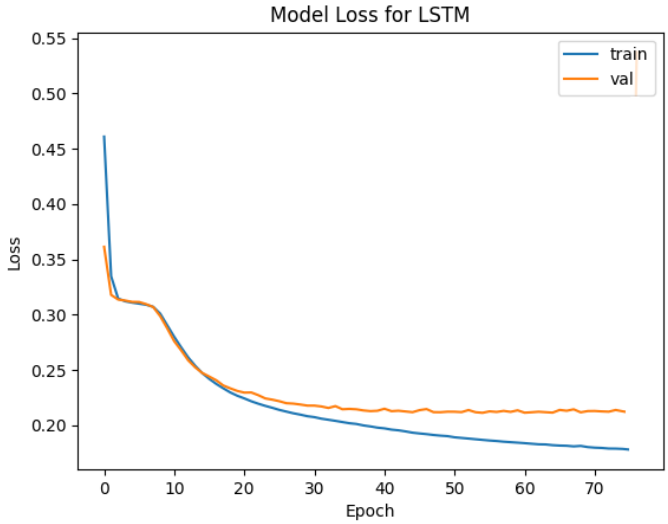


Figure 4.8: Loss Comparison: Training and Validation Results for the LSTM Model

Figure 4.8, 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.

4.4.2 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.

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 4.6 provides an evaluation of the performance of

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

Table 4.6: Performance Metrics of GRU Algorithm in Diacritizing Arabic Text

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.

Assessment of GRU Model Generalization: Training vs. Validation Accuracies

Figure 4.9: Accuracy Comparison: Training and Validation Results for the GRU Model

Figure 4.9, 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 [156].

- 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.

Analyzing Loss in GRU Model: Training vs. Validation Results

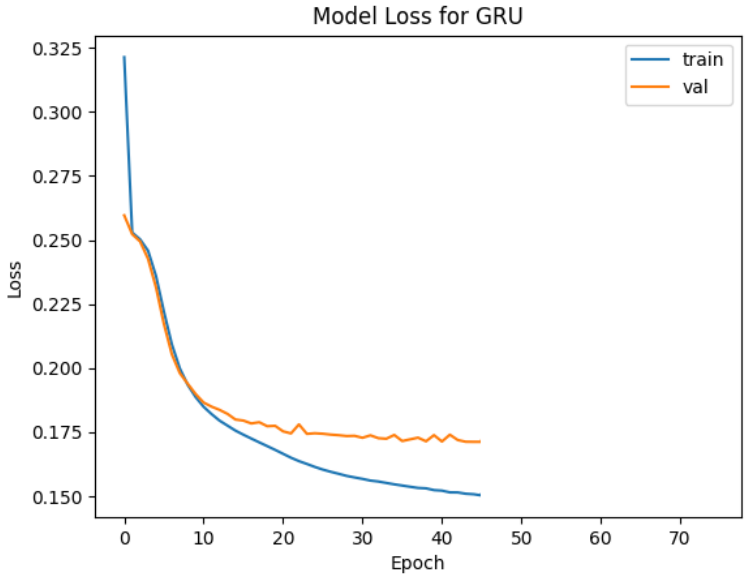


Figure 4.10: Loss Comparison: Training and Validation Results for the GRU Model

Figure 4.10, 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 perfor-

mance 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.

4.4.3 Experimental Results of the Transformer Algorithm

The Transformer model is a state-of-the-art architecture that has revolutionized NLP tasks, including language modeling [96]. This section presents the outcomes of the experiments conducted using the Transformer model and examines its effectiveness in capturing patterns and making accurate predictions.

Performance Analysis of the Transformer Algorithm: Metrics and Results

The experimental results of the Transformer algorithm for diacritizing Arabic text are summarized in Table 4.7. 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%. Figure 4.11, visualizes the training and validation accuracies achieved during the evaluation of the Transformer model.

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

Table 4.7: Performance Metrics of the Transformer Algorithm in Diacritizing Arabic Text

These accuracies are computed over multiple 75 epochs and can be summarized as follows:

Assessment of Transformer Model Generalization: Training vs. Validation Accuracies

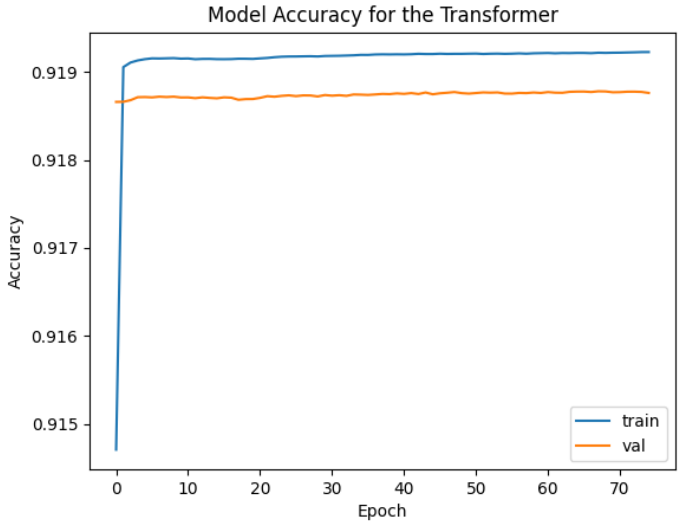


Figure 4.11: Accuracy Comparison: Training and Validation Results for the Transformer Model

- 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.

Analyzing Loss in Transformer Model: Training vs. Validation Results

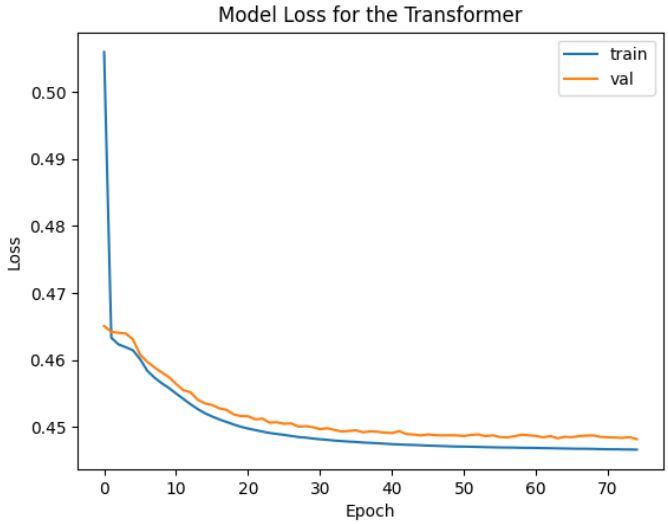


Figure 4.12: Loss Comparison: Training and Validation Results for the Transformer Model

Figure 4.12, 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.

4.5 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.

4.5.1 Evaluating Performance Metrics

The comparison of performance metrics among the Transformer, GRU, and LSTM models is visualized in Figure 4.13. 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.

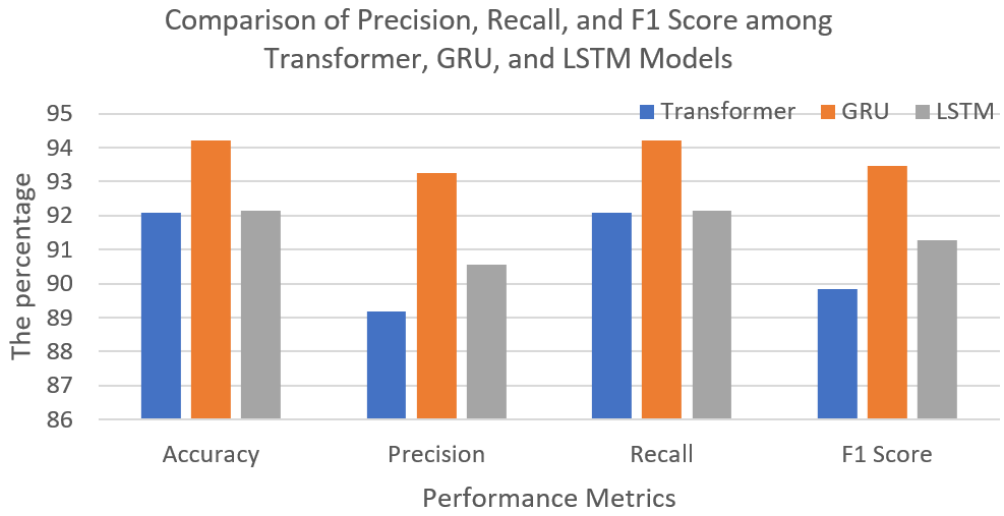


Figure 4.13: Comparison of Precision, Recall, and F1 Score among Transformer, GRU, and LSTM Models

4.5.2 Analysis of Training Accuracy Trends

Figure 4.14 compares the Training Accuracy of the Transformer, GRU, and LSTM models over 75 epochs. The key observations:

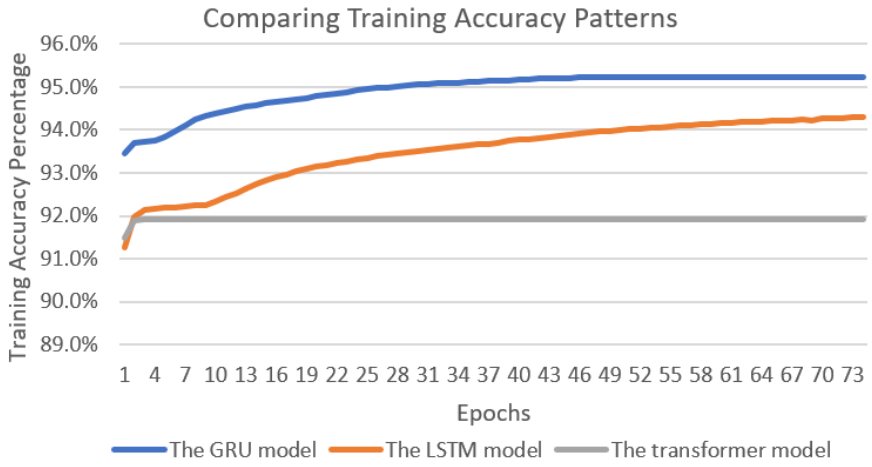


Figure 4.14: Comparing Training Accuracy Patterns: Transformer, GRU, and LSTM Models

- 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.

4.5.3 Analysis of Evaluation Accuracy Trends

Figure 4.15 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

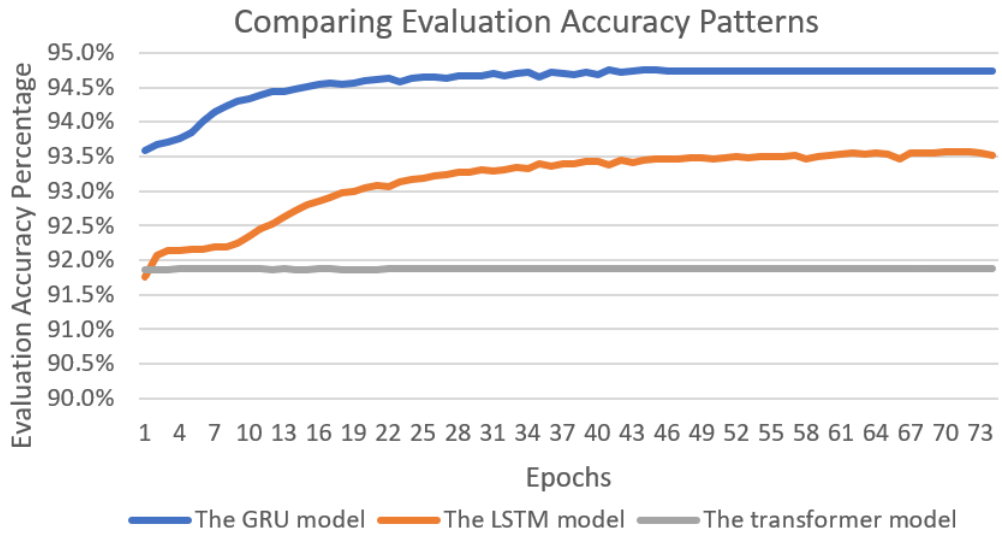


Figure 4.15: Comparing Evaluation Accuracy Patterns: Transformer, GRU, and LSTM Models

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.

4.5.4 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 4.16 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

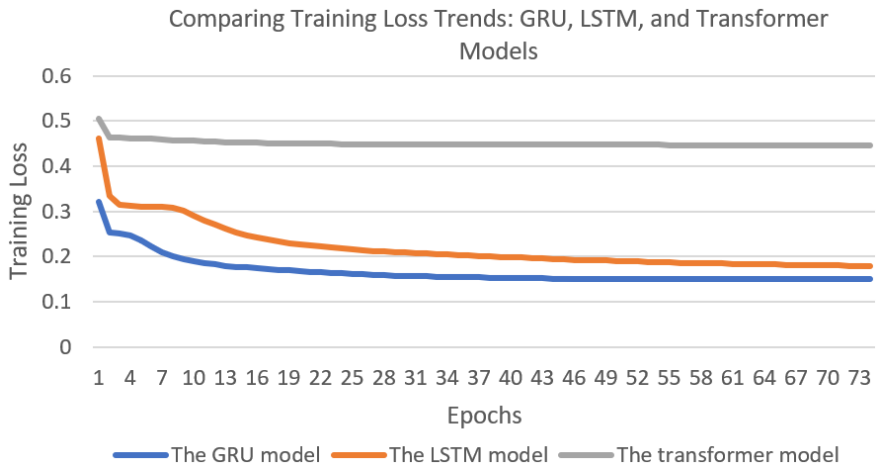


Figure 4.16: Comparing Training Loss Patterns: Transformer, GRU, and LSTM Models

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.

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.

4.5.5 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.

Figure 4.17 displays the evaluation loss patterns of the three models:

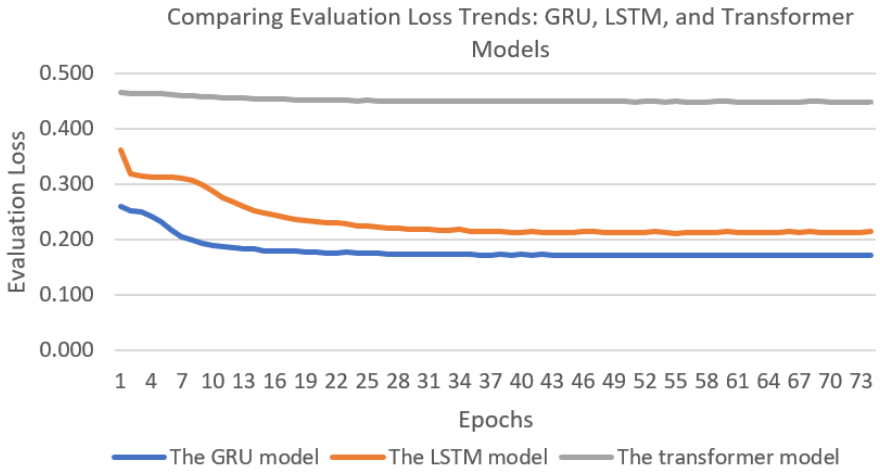


Figure 4.17: Comparing Evaluation Loss Patterns: Transformer, GRU, and LSTM Models

- 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.

4.5.6 Ahadith Dataset Performance Metrics: Uncovering Model Generalization

In addition to the training, validation, and testing datasets composed of Quranic text, we incorporated a dataset containing Ahadith, which are conversations attributed to the Prophet Muhammad (PBUH) found in various books.

The Ahadith dataset, encompassing 8046 records, that serves a specific purpose in testing the model's generalization capacity beyond Quranic text. Unlike the Quranic text, the Ahadith dataset does not adhere to a specific linguistic style. Instead, it represents a collection of

mixed styles, with Ahadith following linguistic patterns from Madinah, Makkah, or the surrounding areas. The variegated nature of this dataset, reflecting diverse linguistic styles, may contribute to its comparatively lower accuracy when compared to the training, validation, and testing datasets rooted in the Quranic context.

In the evaluation of the Ahadith dataset, our model achieved an accuracy of 90%, precision of 87.1%, recall of 90%, and an F1 score of 88.1%. These metrics provide insights into the model's performance when applied to the Ahadith dataset, enabling an understanding of its capabilities in a broader context. To raise this slight decrease, we have to train the model on data that expresses the mixed styles used in cases such as Ahadith dataset.

4.6 Summary

In this part of the study, we have addressed the task of advancing Arabic language understanding through auto-diacritization and stylistic realization in Arabic textual data. Our focus has been on developing the QRDiaRec (Automatic Diacritization and Dialect System for Arabic Quran Text Recognition) system, which serves as a fundamental component in enhancing the comprehension of the Arabic language, specifically in understanding diacritical letters and recognizing various Arabic dialects. The QRDiaRec system incorporates deep learning models, including bidirectional LSTM, bidirectional GRU, and transformer-based models, to achieve accurate diacritization of Arabic text. Through these models, we have achieved a high accuracy rate 94.2%. This system provides researchers and practitioners in Arabic linguistics and related disciplines with valuable tools for analyzing and interpreting Arabic text. Furthermore, we have established a comprehensive database that encompasses multiple linguistic styles for the Arabic Quran text. This database serves as a valuable resource, enabling the exploration of different linguistic styles and their origins. By incorporating this database into the QRDiaRec system, we enhance the system's ability to recognize and analyze various Arabic dialects, further contributing to the understanding of Arabic language variation.

The contributions of this study lie in the development of the QRDiaRec system, the creation of the comprehensive linguistic styles database, and the utilization of deep learning models for accurate diacritization. These advancements pave the way for further research and application in the field of Arabic language processing.

Chapter 5

EXPLORING SEMANTIC AND NUMERIC SIMILARITIES: A MULTIDIMENSIONAL APPROACH WITH SEMSIM

Undoubtedly, semantic textual similarity [181] is crucial in various applications. It plays a vital role in information retrieval, natural language processing, knowledge representation, and understanding the relationships between concepts and entities, among other areas[41]. The similarity of numeric features is of great significance in various domains, facilitating information retrieval [124], data clustering [128], pattern identification [10], and text similarity [173], among other domains. By leveraging the synergistic potential of both numeric and semantic similarity, researchers have made significant advancements across various domains. These fields encompass topic modeling [169, 204, 82], sentiment analysis [196], anomaly detection [129], and other related fields. Semantic similarity captures the meaning and context of the text, while numerical similarity provides a quantitative representation of the data. Together, they offer a richer understanding of the relationships between texts, enabling a more nuanced analysis of the textual content. This integration provides insights into the underlying patterns in textual data in NLP. Some of the benefits include:

- **Text Clustering:** The primary objective is to create clusters where texts within each cluster share a semantic similarity. We use several techniques to capture semantic relationships between words or sentences. These include:
  - *Word embeddings:* These are numerical representations of words. Popular models like Word2Vec and GloVe turn words into vectors [192]. **Word2Vec** learns word associations from a large text corpus. **GloVe** creates word vectors based on how often words appear together.

  - *Topic modeling:* This technique discovers abstract topics in a collection of documents.

  These methods help us understand the meaning behind words and sentences. We also use numerical similarity measures to improve the clustering process. These include: **Co-occurrence analysis:** This looks at how often words appear together in texts [103]. **Vector representations:** These turn texts into numerical vectors for easy comparison [185]. By combining semantic and numerical approaches, we can group similar texts more effectively.

- **Word Representations:** Numeric similarity, often derived from word embeddings like Word2Vec [44] or GloVe [143], captures the distributional patterns of words in a corpus. These embeddings encode semantic information, meaning that similar words have similar numeric representations [70]. This aids in measuring the similarity between words based on their context and usage.
- **Recommendation Systems:** By leveraging both semantic and numerical similarities, recommendation systems can offer more relevant and diverse suggestions to users. Content similarity (semantic) [52] and user preferences (numerical) are taken into account, leading to more accurate and personalized recommendations [180].
- **Machine Learning:** Integrating both semantic and numerical similarities can enhance machine learning models [106]. For instance, using semantic similarity as an additional feature can improve the performance of sentiment analysis or text classification tasks. It enables models to capture the underlying meaning of the text, leading to better predictions and insights.

In this part of the dissertation, our aim is to provide an approach that integrates semantic and numeric similarity, offering a unifying umbrella that brings them together. This integrated approach can provide more analysis and uncover hidden insights and patterns, contributing to advancements in different fields.

We implement this approach within a system, called *SemSim* (for Semantic Similarity),

which we have built to assist users explore similarities in heterogeneous environments of multidimensional datasets. The system helps the user (a) define entities, entity groups, and dimensions of entities, (b) detect numeric similarities among entities either across the same dimensions or across different dimensions, (c) correlate the semantic similarities given in a knowledge graph [146] with the detected numeric similarities, and then (d) use the detected numeric similarities to enhance the knowledge graph by exploring and mining for other hidden semantic similarities.

5.1 Overview

A car is an entity. The car, as an entity, has multiple attributes (or dimensions). Example dimensions include the engine size in cubic centimeters, the fuel consumption in miles per gallon, and the number of seats. All car entities (e.g. car makes and models) fall under an entity group that may be called *the car entity group* . Similarly, other entity groups can be defined to include other types of entities like airplanes, bikes, books, and movies, to name a few examples.

Numeric similarities can be defined and measured across the multidimensional space of each entity group. Cars can have similar engine sizes or fuel consumption rates in miles per gallon. There is a body of research that studies and introduces various algorithms to detect numeric similarities between points in a multidimensional space (Hjaltason and Samet, 2003) [89] using nearest neighbor queries (Roussopoulos et al., 1995) [159], range queries (Gaede and Günther, 1998)[71] and clustering techniques (Rai et al., 2010) [153].

Semantic similarities between entities (in contrast to numeric similarities) are human-judged or evaluated through various machine-learning techniques. Two cars may be identified as semantically similar in being popular car models. Two movies may be identified as semantically similar in being violent, comic, or drama. Two books can be identified as semantically similar because they both address the same issues.

So far, we have highlighted two types of similarities **(numeric and semantic similarities)**

and we have introduced the concept of an entity group that represents a group of multi-dimensional entities of the same type (or the same dimensions). We have also introduced the possibility of comparing entities numerically or semantically as long as (a) we compare entities from the same entity group and (b) we compare entities along the same dimensions. For example, we compare two car models (as two entities in the same car entity group) along the mileage per gallon dimension (which is the same dimension for the two entities).

In this section, we question the possibility of comparing two entities from different entity groups across the same/different dimensions, possibly after normalizing the numeric value (Nazanin et al.,2016) [182]. For example, we may compare the mileage per gallon and the engine size of specific airplanes to the mileage per gallon and the engine size of cars, after normalizing the magnitude of the numeric values so that the comparison is feasible. Moreover, we question the value of comparing entities from different entity groups across different dimensions and correlating the numeric similarity of these entities (across different dimensions) to the semantic similarities between these entities. For example, two cars can be popular (which is an observable semantic similarity) although these cars have no similarities across the same numeric dimensions. These two cars are both popular because one of them is the most economical car (as measured by the mileage per gallon) and the other one is the most luxurious (as measured by the engine size). Therefore, the numeric similarity across different dimensions (i.e., having the highest values over the fuel consumption and engine size dimensions) made these two cars similar over the popularity semantic dimension.

Based on the example above, we believe that semantic similarities and numeric similarities are tied together. However, the numeric similarities that result in semantic similarities do not have to be along the same dimension. Numeric similarities across heterogeneous dimensions may be the cause of semantic similarity. A combination of numeric similarities across different dimensions may stand behind the human-observed semantic similarity. In this part of the dissertation, we bridge the gap between similarities in the numeric space and the semantic space. Building upon concepts from cross-knowledge graph representation learning [90, 39],

Arabic	Phonemic Transcription	English Translation
كَتَبَ	kataba	he wrote
كَتَّبَ	kattaba	to dictate
كَتْب	katb	it was written
كُتُب	kutub	books
مكتبة	maktabh	library
كَاتِب	katib	writer
كُتَّاب	kuttab	Writing School
مَكَاتبُ	makatib	offices
كُتَيِّب	kutayyib	booklet

Table 5.1: Words Derived from the Linguistic Root 'KTB'

we aim to project entities into a unified space where both semantic and numeric similarities are considered. This unified approach can enhance the detection of hidden patterns that may not be apparent when considering semantic or numeric data alone. Guided by semantic similarities (as given by a knowledge graph), we build a system that helps the user explore semantic similarities between various combinations of numeric heterogeneous dimensions. We understand that there can be an enormous number of possible combinations of the numeric dimensions. We mine for the *impactful* combinations of numeric dimensions that stand behind semantic similarities. After we detect the impactful combinations of numeric dimensions, we use these dimension combinations to detect additional semantic similarities that may be hidden in the semantic space and, consequently, improve the knowledge graph.

5.1.1 Harmony of Semantic and Numerical Similarities in Arabic

In Arabic, a unique harmony prevails - a symphony of semantic and numerical similarities. The linguistic roots produce a diverse range of words that are interconnected in meaning and share the same letters. This captivating interplay between numeric and semantic similarities beckons us to build a unified approach for analyzing numeric and semantic similarities in text under the umbrella of NLP. This approach seeks to uncover the relations between semantic and numeric features in Arabic texts.

In English, we can observe similar phenomena, where related words share the same letters and meaning, guided by linguistic roots. For instance, words like "Write," "Written," "Wrote," "Writing," and "Writer" all share the linguistic root (W, R, and T). This insight leads us to question whether a relationship exists between the repetition of letters and semantic meaning. Analyzing co-occurrence text patterns reflects numeric similarities. These numeric similarities for words could shed light on the semantics of these words. Imagine if the words in the English language are built on the feature of linguistic roots. That will encourage the researchers to discover the relationship between the repetition of letters and the semantic meaning of it.

The Arabic language is different, built upon linguistic roots just like other Semitic languages such as Aramaic, Tigrinya, Amharic, and Hebrew. Through the use of three letters, a diverse array of words is formed, all semantically related to one another and sharing the repetition of these letters. Table 5.1 presents examples of Arabic's linguistic root 'KTB'. In Arabic, the letters used in words can reflect their underlying meanings, and the shared letters between words can contribute to their meanings. The table 5.1 presents words derived from the linguistic root 'KTB' in Arabic, and it showcases a remarkable linguistic phenomenon. These words are not only semantically related in meaning but also share the same letters, which gives rise to the numeric repetition between them. This unique interplay between semantic and numeric similarities highlights the inherent connection between the meanings of these words and the presence of shared letters. The table 5.1 exemplifies how linguistic

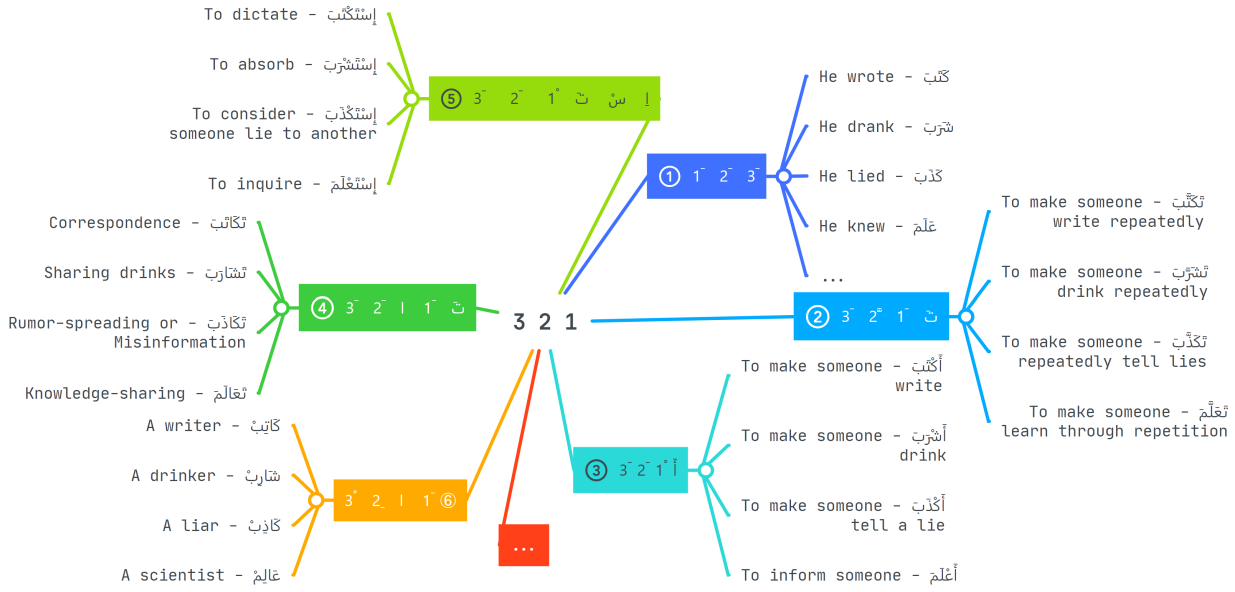


Figure 5.1: The Interplay between Letter Repetition and Meaning in Arabic.

roots can produce a diverse range of words that not only convey related meanings but also exhibit numerical patterns, providing valuable insights into the structure and content of the Arabic language.

What's interesting is that each combination still carries the general meaning of the root. This shows that the repetition of letters and their appearance in words are connected to their meaning. In other words, there's a clear and simple link between the structure of the Arabic language and its meaning. The connection between letter repetition and word meaning isn't limited to linguistic roots. The Arabic language follows consistent patterns. These patterns reveal underlying templates that guide letter appearance, repetition, pronunciation, and meaning association.

The diagram 5.1 shows four words across six patterns. This diagram highlights how letter repetition is linked to meaning, including:

- Extra letters and their positions within words
- Additional meanings according to the patterns
- Diacritics and their connection to letters, pattern repetition, and meaning

In figure 5.1, we can represent any three-letter root with placeholders: 1 for the first letter, 2 for the second, and 3 for the third. This applies to most Arabic words derived from three-letter roots. We chose four words as examples:

- كَتَبَ (kataba) – write
- شَرَبَ (sharaba) – drink
- كَذَبَ (kadhaba) – lie
- عَلِمَ ('alima) – know

We applied six common patterns to these words. Each pattern modifies the words in a consistent way to create new words with related meanings. Added letters appear in the same positions across different words, and they change the meaning in regular ways. Diacritic marks also follow consistent patterns, affecting pronunciation and meaning. Looking at the "Pronunciation" column in table 5.2, we'll see that the root letters always appear in their corresponding words (marked in bold). Interestingly, extra letters added to these roots according to the pattern also appear in the same positions across all words. Each pattern adds a new layer of meaning to a word, clarifying its meaning without changing its original sense:

- **Pattern 1: Past Actions** All actions in this pattern describe completed events in the past, focusing on the outcome or result.
- **Pattern 2: Repetitive Influence** All actions in this pattern involve influencing someone to repeat an action, emphasizing the process of repetition.
- **Pattern 3: Causative Actions** All actions in this pattern describe causing someone

to perform an action, highlighting the causal relationship.

- **Pattern 4: Directive Actions** All actions in this pattern involve directing or guiding someone's actions, emphasizing control or guidance.
- **Pattern 5: Interactive Processes** All actions in this pattern describe interactive processes between individuals, involving exchange, sharing, or mutual influence.
- **Pattern 6: Characteristic States** All actions in this pattern describe permanent or characteristic states of being, defining someone's profession, trait, or identity.

This combination of root letter repetition and added letters creates many regular numerical patterns. These patterns reveal a strong link between the meaning of words and their numerical structures in Arabic. As well as between the sentences that are naturally made up of these words. This example shows that Arabic has a harmonious system where the repetition and arrangement of letters are directly linked to meanings. This is different from English and other Latin languages. In Arabic, numeric patterns like letter repetition are closely tied to semantic meaning. This demonstrates the need for models and algorithms that focus on these unique features of Arabic. Our SemSim system aims to explore the relationship between numeric correspondences and semantic correspondences in Arabic. This is just the beginning, opening the door to deeper research in Arabic natural language processing.

As a case study, we use this system to detect the relationship between the semantic space and the numeric space using the holy Quran text. Chapters, verses, and words are entity groups. Each entity group has multiple numeric dimensions like the number of letters, number of words, number of verses in a chapter, the chapter serial number, the word serial number from the beginning of the Quran, the word number from the beginning of the verse or the beginning of the chapter. On the semantic level, chapters, verses, and words are associated with topics and, therefore, semantic similarities can be observed among these entities and entity groups. The proposed system helps researchers detect the correlation between the semantic similarities and the numeric similarities in the holy book of the Quran.

The contributions of this part of the study can be summarized as follows:

Pattern	Root	Arabic	Pronunciation	Meaning
Pattern 1	k-t-b	كَتَبَ	kataba	He wrote
	sh-r-b	شَرَبَ	sharaba	He drank
	k-th-b	كَذَبَ	kathaba	He lied
	a-l-m	عَلِمَ	’alima	He knew
Pattern 2	k-t-b	تَكَتَّبَ	takattaba	To make someone write
	sh-r-b	تَشَرَّبَ	tasharraba	To imbibe
	k-th-b	تَكَذَّبَ	takaththaba	To become a liar
	a-l-m	تَعَلَّمَ	ta’allama	To learn
Pattern 3	k-t-b	أَكْتَبَ	aktaba	To make someone write
	sh-r-b	أَشْرَبَ	ashraba	To make someone drink
	k-th-b	أَكْذَبَ	akthaba	To prove someone a liar
	a-l-m	أَعْلَمَ	a’lama	To inform someone
Pattern 4	k-t-b	اِسْتَكْتَبَ	istaktaba	To dictate
	sh-r-b	اِسْتَشْرَبَ	istashraba	To absorb
	k-th-b	اِسْتَكْذَبَ	istakthaba	To consider someone a liar
	a-l-m	اِسْتَعْلَمَ	ista’lama	To inquire
Pattern 5	k-t-b	تَكَاتَبَ	takātaba	To correspond
	sh-r-b	تَشَارَبَ	tashāraba	To share drinks
	k-th-b	تَكَاذَبَ	takāthaba	To exchange lies
	a-l-m	تَعَالَمَ	ta’ālama	To share knowledge
Pattern 6	k-t-b	كَاتِب	kātib	Writer
	sh-r-b	شَارِب	shārib	Drinker
	k-th-b	كَاذِب	kāthib	Liar
	a-l-m	عَالِم	’ālim	Scholar

Table 5.2: Arabic Patterns: Exploring the Relationship between Semantic Meaning and Numeric Structures.

- We develop a model to define multidimensional entities and entity groups in an environment characterized by heterogeneity in dimensions.
- We present an integrated approach that combines semantic and numeric similarity under a unifying umbrella.
- We build a system, called *SemSim* (for *Sem*antic *Sim*ilarity), that helps the user explore numeric similarities between entities that span different entity groups and, possibly, across different dimensions.
- Guided by a knowledge graph that represents the semantic similarities between entities, the system alters the user when a numeric similarity is found between entities that already show semantic similarities.
- The system detects the numeric dimensions that impact the semantic similarity and uses these impactful dimensions to further detect hidden semantic similarities
- We use the text of the holy Quran as a case study to explore the semantic and the numeric similarities over chapters, verses, and words.

5.2 The model for the numeric and semantic spaces in a heterogeneous environment

In this section, we define our notion of a heterogeneous numeric multidimensional space as well as the semantic space. We define entities, entity groups and dimensions of these entities. We provide a formalization for the similarity problem under investigation and its associated query types. This section provides a generic formalization of the problem while section 5.2.5 provides an explanation of the dataset used and its representation in numeric and semantic spaces.

5.2.1 The heterogeneous multidimensional numeric space

A heterogeneous domain is a domain that has multiple entities and entity groups [92]. For example, a heterogeneous domain includes entities from the *car* entity group as well as entities from the *aeroplane* entity group. All entities are multi-attribute or multidimensional.

Each entity group is characterized by the same set of dimensions. All cars have the same dimensions, all aeroplanes have the same dimensions and, similarly, all entities coming from the same entity group have the same dimensions. However, dimensions are expected to be different across entity groups. Cars and aeroplanes may share several dimensions but not all dimensions. There are some attributes (or dimensions) that are unique and specific to cars or aeroplanes. Our proposed multidimensional numeric space, as given by Definition 1, is composed of the union of all dimensions in all entity groups. Therefore, entities are expected to have numeric values over dimensions that are defined within their entity group and have null values otherwise (or across dimensions that are not defined for the entity's entity group).

5.2.2 The numeric space

Definition 1 *The numeric space is composed of E, a set of K-dimensional entities that has n entities. There are also m overlapping entity groups (each group is a subset of E). Each entity e_i where $1 <= i <= n$ belongs to g entity groups where $0 <= g <= m$. Each entity e_i has k_i non-null dimensions and, consequently,$K - k_i$ null valued dimensions, where $k_i << K$. The number of non-null dimensions (k_i) is variable across entity groups. For each entity e_i, and for each non-null dimension k, the entity has a numeric value v_{i_k}.*

5.2.3 The semantic space

Consider a knowledge graph [120] $KG(E, W)$, where entities in E, the set of K-dimensional n entities, represent the vertices of the graph and $W = w_{i,j} \,|w_{i,j} = SemSim(e_i, e_j)$ is the set of edge weights between entities. The weight $W_{i,j}$ represents a semantic similarity measure between entities e_i and e_j.

Figure 5.2 presents the proposed integration between the numeric and semantic spaces. The left side of the figure presents a knowledge graph over entities, where a node $(e_1, e_2, ...e_n)$ is an entity and an edge between two nodes is the semantic similarity weight between these two entities $(SS_{v1}, SS_{v2}, ...SS_{vn})$. These edges can be between entities (nodes) within the same entity group or they may extend across different groups $(g_1, g_2, ...g_m)$. On the right side is

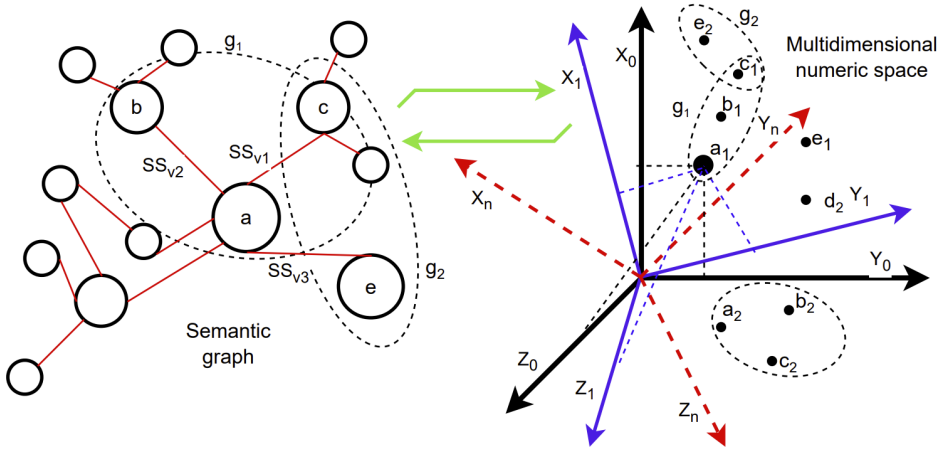


Figure 5.2: Proposed model for the overlap between numeric and semantic similarities in a multidimensional space.

the multidimensional space where each entity is represented as a point in a multidimensional space.

5.2.4 Formalization of Similarity Queries in the Numeric and Semantic Spaces

In this section, we establish a formalization of a proposed set of similarity queries that *SemSim* is expected to support. These similarity queries detect similarities over the same dimensions or over different dimensions in the numeric spaces and correlate these similarities with the semantic domain. The proposed queries are defined in the remainder of this section.

Single Value Equality over Same Dimension Similarity Query.

Query 5.1 represents a query designed to retrieve entity pairs (e_i, e_j) that share the same value (v) across a specific dimension (k). The purpose of this query is to identify entities that possess a common value for a given dimension. By comparing the numeric values $v_{i,k}$ and $v_{j,k}$ of entities e_i and e_j respectively, this query selects entity pairs that exhibit numeric similarity in terms of a particular dimension.

$$\{(e_i,\ e_j, k, v)\ \ | v = \quad v_{i_k} = v_{j_k}\ \ \} \tag{5.1}$$

In this Query 5.1, the entities are denoted as (e_i, e_j), where e_i and e_j represent two distinct entities. The variable k represents the specific dimension being considered, and v represents the shared numeric value for that dimension. The query states that for a given entity pair (e_i, e_j), if the numeric values $v_{i,k}$ and $v_{j,k}$ are equal, then they satisfy the condition of having the same value for dimension k.

Semantic Similarity Threshold Query.

Query 5.2 retrieves entity pairs that have a semantic similarity score greater than or equal to the threshold.

$$\{(e_i,\ e_j) \mid SS(e_i, e_j) \geq \theta\} \tag{5.2}$$

In Query 5.2, $SS(e_i, e_j)$ represents the semantic similarity between entities e_i and e_j, and θ is the threshold value.

Integrating Numeric and Semantic Similarity Query.

By combining both Query 5.1 and Query 5.2, we can consider both numeric and semantic similarities in our queries, allowing us to retrieve entity pairs that share the same value across a specific dimension and exhibit a certain level of semantic similarity.

$$\{(e_i,\ e_j, k, v)\ \mid v = \quad v_{i,k} = v_{j,k},\ SS(e_i, e_j) \geq \theta\} \tag{5.3}$$

In query 5.3, the additional condition $SS(e_i, e_j) \geq \theta$ is included, where $SS(e_i, e_j)$ represents the semantic similarity between entities e_i and e_j, and θ is a threshold value. This condition ensures that only entity pairs with a semantic similarity score greater than or equal to the threshold are considered. By including semantic similarity, the query filters entity pairs with shared values and semantic similarity.

Combining Distance-based Numeric and Semantic Similarity for Entity.

One of the objectives of the SemSim system is to discover the numerical similarity between

semantically similar entities. By combining the distance-based numeric similarity and semantic similarity conditions, Query 5.4 enables the identification of entities that not only exhibit close numeric similarity within dimension k but also share a certain level of semantic similarity.

$$\{(e_i,\ e_j, k, v_{i,k}, v_{j,k}) \mid dist(v_{i,k}, v_{j,k}) < \varepsilon, SS(e_i, e_j) \geq \theta\} \tag{5.4}$$

The Query 5.4 represents a set of entities $(e_i, e_j, k, v_{i,k}, v_{j,k})$ that satisfy two conditions. First, the distance between their numeric values $v_{i,k}$ and $v_{j,k}$ for dimension k must be less than a threshold ε, as measured by the function $dist$. Second, the semantic similarity between entities e_i and e_j denoted by $SS(e_i, e_j)$ should be greater than or equal to a threshold θ.

Enhancing Semantic Knowledge through Numeric Similarity Discovery

As we mentioned, the system explores the numerical similarity among entities that share a semantic similarity. Additionally, the system aims to identify other entities that may not possess semantic similarity but exhibit the same numerical resemblance as semantically similar entities. This approach can potentially uncover previously undiscovered semantic knowledge by leveraging the shared numerical patterns across entities. Therefore, the system aims to enhance our understanding of semantic relationships beyond the boundaries of traditional semantic similarity.

We define Query 5.5 to incorporate the condition of lacking semantic similarity. Query 5.5 captures entity pairs (e_i, e_j) where the numeric values $v_{i,k}$ and $v_{j,k}$ for dimension k are equal, their semantic similarity $SS(e_i, e_j)$ is below a certain threshold θ, and there exists no other entity e_m that has the same numerical similarity and a higher semantic similarity value. In other words, we are querying for the entity that is most semantically similar to e_i (even if this semantic similarity is below the threshold) among all entities that are numerically similar to e_i.

$$\{(e_i, e_j, k, v_{i,k}, v_{j,k}) \mid v_{i,k} = v_{j,k}, \\ SS(e_i, e_j) < \theta, \nexists e_m : v_{i,k} = v_{m,k},\ SS(e_i, e_m) > SS(e_i, e_j)\} \tag{5.5}$$

By utilizing Queries 5.4 and 5.5 together, we can effectively identify the entities that exhibit both numerical and semantic similarity, and then search for other entities that possess the same numerical similarity but lack semantic similarity.

Numeric Clustering: Discovering Consistent Numeric Values across Entities

Query 5.6 enables the discovery of groups of entities based on their numeric values, resulting in a form of numerical clustering. Each entity within a group shares the same numeric value for its respective dimension. By considering multiple dimensions, the query facilitates the identification of entities that exhibit consistency in their numeric values across different aspects or features. This approach allows us to explore the consistency of numeric attributes across entities and gain insights into their shared characteristics or behaviors in relation to different aspects or features.

$$\{(S, K, v) \mid S = \{e_1, e_2, ..., e_n\}, K = \{k_1, k_2, ..., k_n\}, v = v_{i,k_i}, \forall e_i \in S, \forall k_i \in K\} \tag{5.6}$$

The query represents a set (S, K, v), where S denotes a group of entities $\{e_1, e_2, ..., e_n\}$, K represents a set of dimensions $\{k_1, k_2, ..., k_n\}$, and v signifies the shared numeric value v_{i,k_i} for each entity e_i and its corresponding dimension k_i within the group S.

Dual Clustering: Integrating Numerical and Semantic Aspects for Cluster Formation.

By combining both numerical and semantic clustering, Query 5.7 enables the formation of clusters that possess both numerical and semantic cohesion. This new type of clustering allows for a more comprehensive understanding of the data, uncovering meaningful relationships that arise from the integration of both numerical and semantic aspects. Query 5.8 represents a set (S, K, v) where entities are clustered based on both numerical and semantic

similarity. It builds upon the numerical clustering concept from Query 5.6 and incorporates the additional condition of semantic clustering.

$$\begin{aligned}&\{(S, K, v) \mid S = \{e_1, e_2, ..., e_n\}, K = \{k_1, k_2, ..., k_n\},\\ &v = v_{i,k_i}, \forall e_i \in S, \forall k_i \in K, SS(S) \geq \theta, NS(S) \geq \phi\}\end{aligned} \tag{5.7}$$

Similar to Query 5.6, S denotes a group of entities $\{e_1, e_2, ..., e_n\}$ that exhibit numerical similarity for their respective dimensions $K = \{k_1, k_2, ..., k_n\}$ as represented by the shared numeric value v_{i,k_i} for each entity e_i and its corresponding dimension k_i within the group.

In addition, two new conditions are introduced. The first condition $SS(S) \geq \theta$ ensures that the entities within the cluster share a minimum level of semantic similarity denoted by $SS(S)$. This condition ensures that the cluster comprises entities that are semantically related.

The second condition $NS(S) \geq \phi$ indicates that the cluster should also exhibit a certain level of numerical similarity, represented by $NS(S)$. This condition ensures that the cluster contains entities that not only share semantic similarity but also display consistent numerical patterns across their dimensions.

Customized Grouping: Exploring Entities with Targeted Weighted Numeric Values.

Query 5.8 allows us to find groups of entities where a custom function f applied to the numeric values across dimensions yields a specific target value v. This query provides flexibility in capturing similarity by enabling the use of different functions to evaluate the combination of numeric values. It allows for the exploration of various similarity measures and customized assessments based on the specific requirements of the problem or domain.

$$\{(S, K, v) \mid S = \{e_1, e_2, ..., e_n\}, K = \{k_1, k_2, ..., k_m\}, f_{j=1}^{m} v_{i,k_j} = v, \forall e_i \in S, \forall k_m \in K\} \tag{5.8}$$

By introducing a function f that operates on the numeric values. It represents a set

(S, K, v) where S denotes a group of entities $\{e_1, e_2, ..., e_n\}$, K represents a set of dimensions $\{k_1, k_2, ..., k_m\}$, and the function f applied to the numeric values v_{i,k_j} results in the value v.

Weighted Numeric and Semantic Similarity: Customizing Query for Entity Grouping.

To incorporate semantic similarity, we can modify Query 5.8 to include a semantic similarity measure or score. We can introduce weights for the semantic similarity scores [146] and incorporate them in the query. The modified query would then capture similarity based on both the weighted numeric values and the weighted semantic similarity scores.

For example, we can extend Query 5.8 as follows:

$$\begin{gathered}\{(S, K, v) \mid S = \{e_1, e_2, ..., e_n\}, K = \{k_1, k_2, ..., k_m\}, \\ f(\sum_{j=1}^{m} w_{num,j} \cdot v_{i,k_j}) + g(\sum_{j=1}^{m} w_{sem,j} \cdot s_{i,k_j}) = v, \forall e_i \in S, \forall k_m \in K\}\end{gathered} \tag{5.9}$$

In Query 5.9, we have introduced two functions: f represents the function applied to the weighted numeric values, and g represents the function applied to the weighted semantic similarity scores. The weights $w_{num,j}$ and $w_{sem,j}$ determine the importance of each dimension's numeric value and semantic similarity score, respectively. The query captures similarity by evaluating the combination of the weighted numeric values and the weighted semantic similarity scores, which should sum up to the target value v.

By incorporating semantic similarity in Query 5.9, we can explore similarity measures that consider both numeric and semantic aspects. The weights assigned to the numeric values and semantic similarity scores allow for fine-tuning the relative importance of each component. This flexibility enables customized assessments and the exploration of different alternatives for capturing similarity based on the specific requirements and characteristics of the problem or domain.

The primary goal of Query 5.9 is to identify entities that exhibit a balance of both numeric

and semantic similarities based on specific criteria. By combining these similarities into a single target value v, we create a unified similarity measure. This approach is grounded in the following principles:

1. Composite Similarity Measures: In many fields, composite similarity measures are used to integrate multiple aspects of similarity into a single metric. This facilitates ranking, clustering, and retrieval tasks where a holistic assessment of similarity is desired. The application of composite measures in various fields, such as economic [36], social determinants of health [85], and surgical quality [154].
2. A Reverse Similarity Query: is a type of query that retrieves data entities based on a target value [122]. In other words, it starts with a known result and searches for the data that satisfies the condition. Our query enables the retrieval of entity groups that satisfy a specific condition, where the weighted numeric values and weighted semantic similarity scores sum up to a target value v. Instead of searching for entities based on specific attributes or values.
3. Targeted Retrieval: Setting the combined similarity to equal a target value v enables the formulation of specific queries that retrieve entities meeting precise similarity criteria. This is particularly useful in data mining and pattern discovery, where researchers may be interested in entities that satisfy certain balanced conditions.
4. User Experience: From a user's perspective, having a single similarity score simplifies decision-making processes. Users can interpret and act upon a unified metric rather than juggling multiple separate scores.

This approach facilitates the discovery of meaningful patterns and correlations between numeric and semantic attributes, potentially revealing insights that contribute to the advancement of knowledge in the field. This aids linguists and researchers in exploring the intricate relationships between word forms and meanings, potentially contributing to the development of more sophisticated Arabic natural language processing tools.

These queries provide a foundation for understanding and measuring both numeric and

Numeric Space

Group (E)	Entity (e)	Dimension (k)	Value (v)
Surah	Verse	No. of words, verse no., No. of letters, etc	Value of k

Semantic Space

Vertices (e)	Edge (k)	Weight (w)
Concepts	Similarity	percentage of similarity

Table 5.3: An example of the distribution of the problem elements in Quran case study.

semantic similarity within a heterogeneous environment, enabling the proposed model to capture the essence of similarity across multiple dimensions and entity groups.

5.2.5 Quranic Data Representation in Semantic and Numeric Spaces

The purpose of our case study is to utilize the proposed system *SemSim* to discover and elicit knowledge from the Quranic text. Finding relationships and patterns between words, verses, and chapters. As shown in Table 5.3, the surah is an entity group that has verses as its entities. Example dimensions could be the number of words, the verse number, the surah number, and the number of characters per verse. Note that *SemSim* provides a user interface that gives the flexibility to the user to define entities, entity groups, dimensions, and dimension values dynamically and, then, to interactively explore the numeric and semantic similarities between entities, as described later in section 5.3. Using NLP techniques that analyze the context and the topics addressed in each surah or verse an ontology [160] can be constructed to capture the semantics of these entities. Moreover, a knowledge graph, with entities represented as vertices, can capture the semantic similarities among these entities. Such similarity scores become the edge weights of the graph to reflect the semantic relationships between the surahs, verses, or words.

5.3 The SemSim system

The *SemSim* system consists of three sub-systems designed to detect numeric and semantic similarities among entities in multidimensional datasets. These sub-systems are an ontology-building [25] sub-system (OBS), a multidimensional numeric representation [158] sub-system

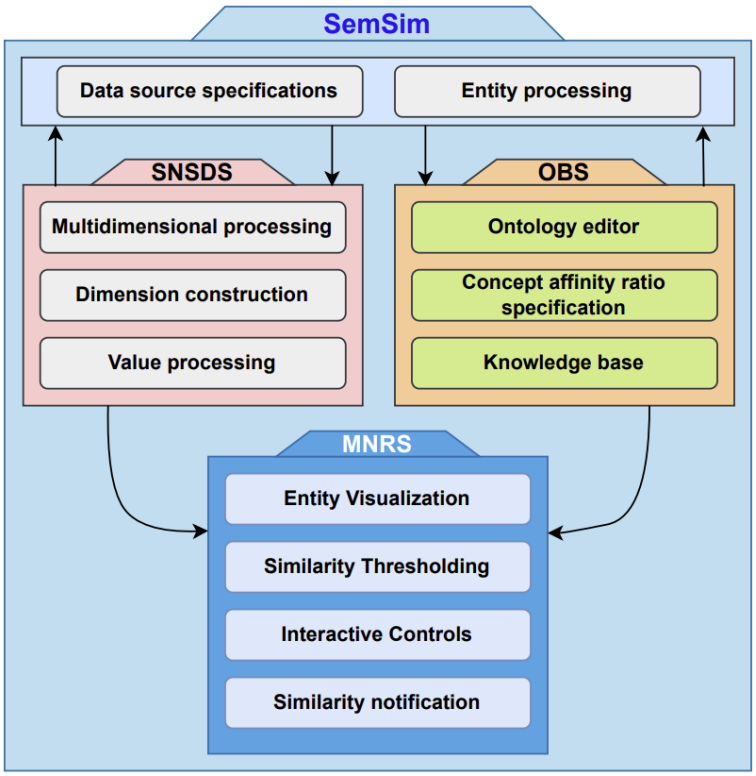


Figure 5.3: The Proposed System Architecture of SemSim: A System for Multidimensional Numeric and Semantic Similarity Detection.

(MNRS), and a semantic-numeric similarity detection sub-system (SNSDS). The OBS and Entity Processing provide an in-depth implementation of the semantic space mentioned in Figure 5.2. Similarly, the SNSDS and Entity Processing are responsible for implementing the numeric space represented in Figure 5.2. The MNRS component is shared between both spaces and offers tools to determine the similarities between numeric and semantic representations. In this section, we will present an overview of the proposed system's structure, followed by an explanation of each subsystem in the context of the entire system.

5.3.1 The Proposed System Architecture

Figure 5.3 depicts the system, its three subsystems, and the components of each subsystem. The **MNRS** is a sub-system that processes numeric data in multidimensional datasets and has six components. *Data Source Specification* allows users to specify the data source. While

Entity Processing component creates and processes entities, *Multidimensional Processing* constructs dimensions and processes entity values, and *Dimension Construction* constructs dimensions based on entity properties. Finally, *Value Processing* processes entity values to create a multidimensional representation.

The **OBS** builds and manages ontologies and shares two components with the MNRS (Data Source Specification and Entity Processing). It also has four components of its own. *Ontology Editor* allows users to create and modify ontologies, while *Concept affinity ratio specification* determines the strength of relationships between concepts. *Knowledge Base* stores and manages ontology-related knowledge.

The **SNSDS** detects similarities among entities in multidimensional datasets, attempts to find a relationship between the results of OBS and MNRS, and has five components. *Entity Visualization* provides a visual representation of entities and their relationships. *Similarity Thresholding* sets the similarity detection threshold, and *Interactive Controls* allow users to modify similarity settings. Finally, the *Similarity Notification* notifies users of similarities.

5.3.2 The multidimensional numeric representation sub-system (MNRS)

In this section, we discuss how the MNRS operates in two steps. The first step, which is *entity processing*, creates entities from a specified data source and classifies these entities into entity groups. The second step, which is the multidimensional processing, constructs the dimensions of the multidimensional space and processes the values of each entity across these dimensions. The following sections provide an overview of these two steps.

Entity processing

Entity Processing has three sub-steps: a) identifying the data source, which is the underlying database(s) that the system will import the data from, as shown in Figure 5.4; b) creating entity groups that act as containers for the entities that are to be created in the following step; and c) creating the entities and assigning them to the entity groups. Entity groups and entities are created through SQL statements that query the underlying data sources, as shown in Figure 5.6. Figure 5.7 shows the created entities along with their entity groups and

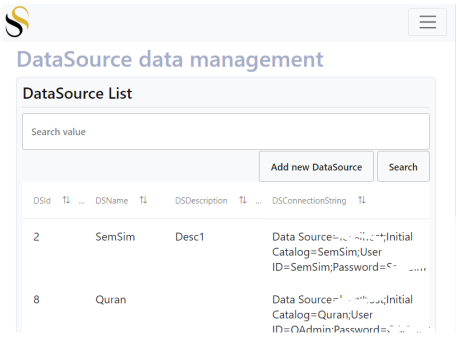


Figure 5.4: Identifying the underlying data sources.

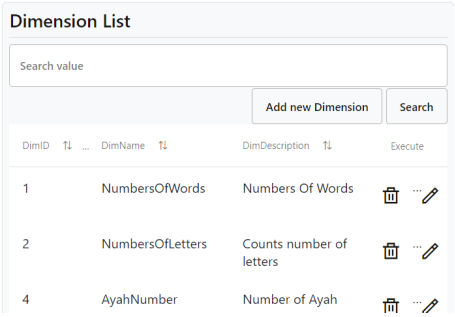


Figure 5.5: Creating the list of expected dimensions.

gives the user the ability to edit and delete entities and entity groups.

Multidimensional processing

As discussed earlier, the system accepts entity groups where each group has a set of dimensions. A dimension can be common to multiple entity groups or can be unique to one entity group. For example, the *number of words* dimension is common to the surah and the verse entity groups. However, the *surah number* is unique to the surah entity group.

The multidimensional processing step includes three sub-steps: a) Defining the dimension list of all dimensions across all entity groups, as shown in Figure 5.5, b) assigning dimensions to entity groups, and c) querying the values of each dimension in each entity from the underlying data source. Sub-steps (b) and (c) are done using SQL statements, as shown in Figure 5.8.

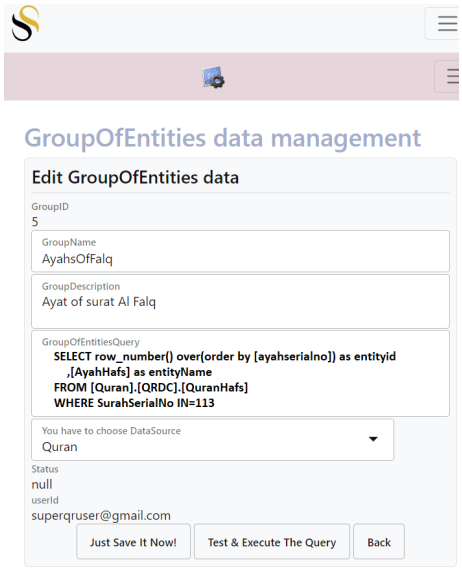


Figure 5.6: Creating entity groups by querying the underlying data source.

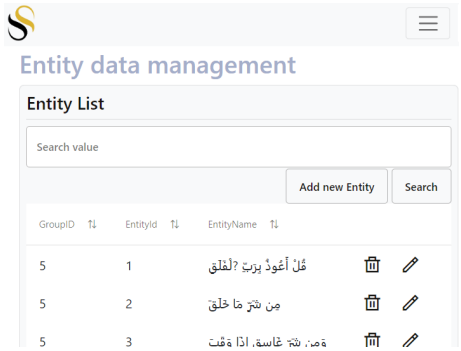


Figure 5.7: Creating entities by querying the underlying data source.

5.3.3 The ontology-building sub-system (OBS)

The Ontology Building System *OBS* module enables the user to specify the structure and relationships between different entities in the ontology. The system offers multiple ways for users to specify concept affinity ratios, including manual input, SQL statements, and the option to import existing ontologies. Furthermore, the *OBS* displays a list of relationships between entities along with their corresponding percentages of proximity, as shown in Figure 5.9.

5.3.4 The Semantic-numeric similarity detection sub-system (SNSDS)

SNSDS enables the user to explore relationships between entities across both the numeric and semantic spaces. Its visual representation of entities and their similarities elevates the ability to understand complex relationships between entities. Furthermore, users can set numeric and semantic similarity thresholds.Then, the system enables the user to "play" with various knobs and sliding bars to discover new insights and relationships between entities. As the user explores various knobs and changes various parameters, the system continuously

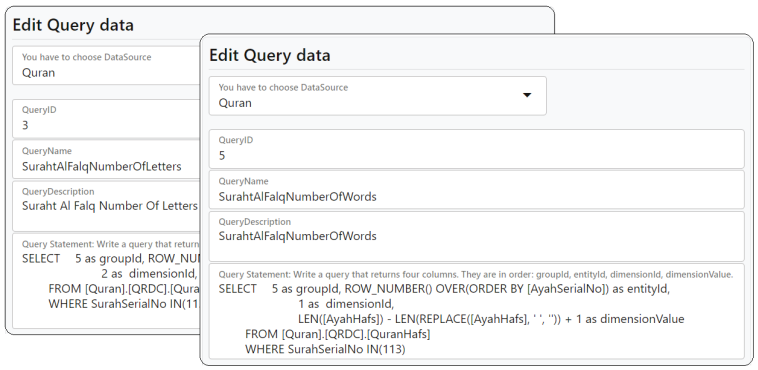


Figure 5.8: Dimension Values Filling via SQL Statements.

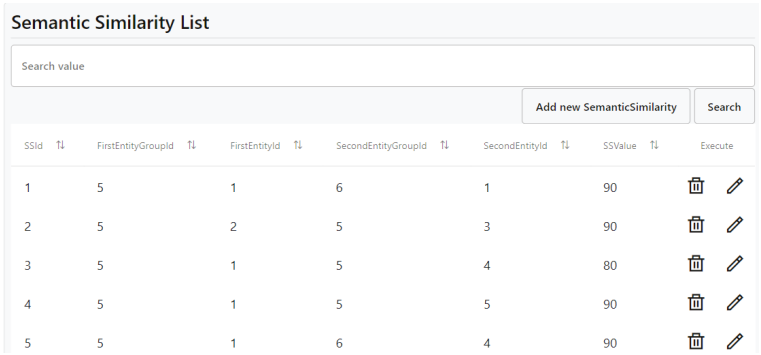


Figure 5.9: List of Entity Relationships and Proportions in a Semantic Space.

notifies the user once the similarity between two entities falls below both the semantic and the numeric thresholds. SNSDS semi-automates the process of identifying relationships and saves the user a lot of time and effort to manually analyze large data sets.

As shown in Figure 5.10, the system gives the user a visual representation of the multidimensional numeric space with each entity represented as a point. The lines connecting two points represent the degree of numeric similarity between the entities. The sliding bars (on the left side) give the user the option to adjust the weights of various dimensions in the space to reflect the importance of this dimension in the similarity function. The graph clusters entities based on the numeric similarity between entities. The closer the entities on the graph, the more similar they are in the numeric space. The user sets (as shown in the figure) two thresholds, one for numeric similarity and another one for semantic similarity. As the user explores the domain by (a) adding/removing entities, entity groups, and dimensions,

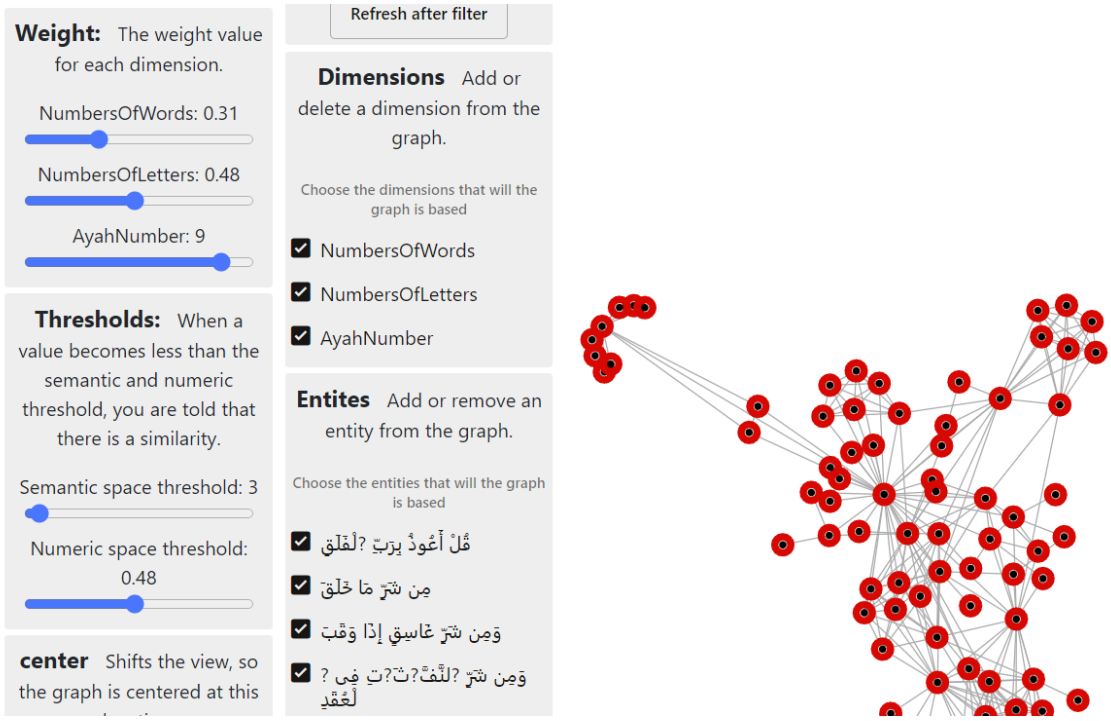


Figure 5.10: Multidimensional Numeric Space Visualization with Entity Clusters and Relationship Dimensions.

(b) changing the weights of dimensions and (c) changing the numeric and semantic similarity thresholds, the system continuously alerts the user of any pair of entities, where both their numeric and semantic similarities fall below the specified thresholds. Therefore, the tool helps the user explore the similarities across the numeric and semantic spaces at the same time.

5.4 Semantic and Numeric Dataset in SemSim

SemSim is a versatile system used to explore relationships between semantic and numerical similarities. We made a dataset for studying these semantic and numerical relationships in the texts of the Holy Quran for chapters 2 and 3 and to test SemSim's results.

The dataset includes pairs of texts (verses or sentences): a first text and a different text that is semantically similar to it. To measure semantic similarity, we used NLP embeddings [186] which are numerical representations of the sentences that capture semantic meaning. This

process maps the sentences from their original text form into dense numerical vectors, where similar sentences are placed closer together in the high-dimensional space. This representation allows the system to understand and measure the similarity between different sentences.

The similarity between sentence embeddings is calculated using cosine similarity. This measure is used to determine the similarity between two vectors in a high-dimensional space. It calculates the cosine of the angle between the two vectors, and the result lies between -1 and 1. A cosine similarity of 1 indicates that the two vectors are identical, while a value of 0 means they are orthogonal (completely dissimilar). We employ the Universal Sentence Encoder (USE) model from TensorFlow Hub [93], specifically the multilingual large model, to generate NLP embeddings [195]. This model is the "universal-sentence-encoder-multilingual-large", which is designed to work with sentences in multiple languages, including Arabic [195]. The USE is trained to capture semantic relationships between sentences based on large datasets before these calculations. It learns to associate similar meanings with closer embeddings. This means that sentences with similar semantic meanings are mapped to numerically close vectors. The model encodes this understanding into embeddings, which are high-dimensional numerical vectors. Calculations compare meaning:

- Cosine similarity measures the angle between two vectors.
- Smaller angles mean the sentences are (semantically closer).

The NLP embeddings capture semantic meanings like semantic relationships, synonymy, paraphrasing, thematic similarity, etc.

For each sentence in the dataset, 6 numeric dimensions are added. The choice of these dimensions was driven by our goal to represent the most popular calculations. These dimensions allow us to capture the most significant patterns while maintaining a balance between complexity and computational efficiency. In general, the dimensions are created by the user and can be direct numbers or numbers derived from equations depending on the user's point of view. The purpose is to represent different numerical aspects of the sentences. We didn't use many numeric dimensions to avoid increasing the chances of accidental matches.

The summary of the dimensions used in the analysis:

- Word Count: The number of words in the sentence. This dimension provides information about the length and complexity of the sentences. Longer sentences might convey more detailed information or ideas, while shorter sentences could be more concise and straightforward.
- Letter Count: The total count of letters in the sentence. Longer sentences with more letters might indicate more complex or elaborative expressions.
- Unique Letter Count: The count of unique letters in the sentence. This dimension indicates the diversity of characters used in the sentences. Sentences with a higher count of unique letters may represent a wider range of vocabulary and potentially convey more nuanced meanings.
- Arithmetic Operation: We performed a simple arithmetic operation involving division with the number 7. We chose 7 as an example but could have tested with other numbers too. We performed this arithmetic operation for the first three dimensions. The choice of number 7 is significant for religious reasons, considering the context of analyzing the texts from the Holy Quran. It might reveal patterns or relationships that align with certain numeric properties.
- Count of Arabic Letters: A binary representation where each Arabic letter occurrence in the sentence is represented as 1, and non-occurrence is represented as 0. This dimension helps identify letters that frequently appear in sentences, providing insights into common linguistic patterns.
- Count of Repeated Arabic Letters: The count of each Arabic letter that appears in the sentence, considering the repetition of letters. This dimension highlights the importance of certain letters that are consistently used in the text.

When two texts have the same numeric value on a specific dimension, it's considered a numeric match with a value of 1. We categorized the semantic similarity into ten groups based on percentage semantic similarity. The categories range from less than 10% similarity

Table 5.4: Distribution of Numeric Matches and Sentence Pairs Across Semantic Similarity Ranges

Semantic Similarity Range	Total Numeric Matches	Sentence Pairs Count
10%	2,758.00	6,381.00
20%	7,226.00	19,680.00
30%	12,077.00	32,968.00
40%	12,488.00	32,456.00
50%	7,149.00	17,620.00
60%	2,786.00	5,999.00
70%	657	1,220.00
80%	90	143
90%	23	22
100%	7	4
Totals	**45,261.00**	**116,493.00**

(10% category), then from 10% until less than 20% similarity (20% category), and so on to 100% similarity (100% category). Sentences that are 100% identical were excluded from the analysis, as well as sentences with zero similarity.

The table 5.4 presents a description of the dataset along with its distribution across semantic similarity classes. The dataset consists of 116,493 records, and it includes numerical similarities in eight dimensions totaling 45,261.00.

Table 5.4 shows the distribution of total numeric matches and sentence pairs count across different semantic similarity ranges. For instance, in the 10% semantic similarity range, there are 2,758.00 total numeric matches found in 6,381.00 sentence pairs. As the semantic similarity range increases, the number of total numeric matches generally decreases, indicating a possible relationship between semantic similarity and numeric matches.

5.5 Results and Discussion

By analyzing table 5.4, we can gain insights into the relationship between the semantic and numerical aspects of sentences, and how different similarity ranges impact the occurrence of numeric matches. Table 5.4 shows that numeric matches are more prevalent when semantic similarity is weak. However, this perception is deceptive, as the number of non-similar

Table 5.5: Numeric Match Ratios Across Semantic Similarity Ranges

Semantic Similarity Range	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Numeric match ratio	43%	37%	37%	38%	41%	46%	54%	63%	105%	175%

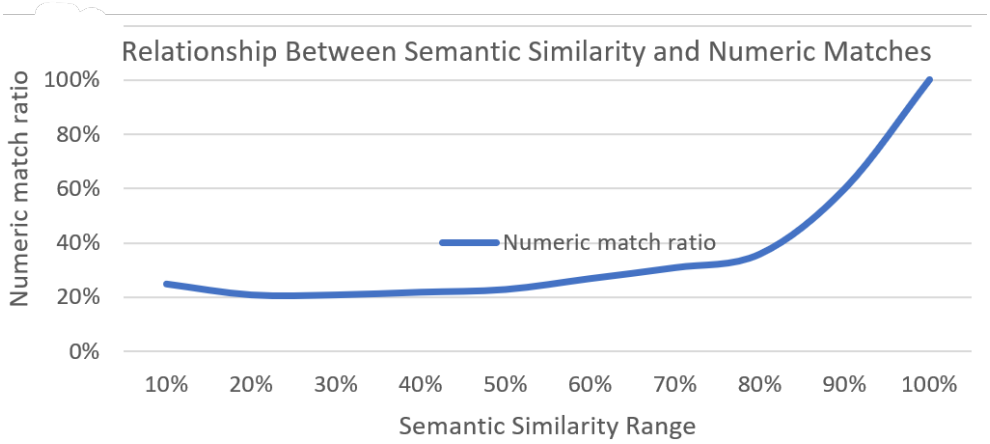


Figure 5.11: Numeric Matches Trend with Increasing Semantic Similarity.

sentences is naturally much greater than the linguistically similar sentences, as indicated in the column 'Sentence Pairs Count.'

For example, in the 90% similarity category, there are only 22 sentence pairs, including 23 numeric matches, which results in a very high numeric match ratio. To address this, we created a Numeric Match Ratio (Total Numeric Matches / Sentence Pairs Count) to reflect the correct ratio for numeric matches.

The table 5.5 displays the semantic similarity ranges and their corresponding numeric match ratios. It becomes apparent from the data in 5.5 that the numeric match ratio generally increases with semantic similarity, except for the 10% similarity range.

Furthermore, the line chart in figure 5.11 illustrates that in strong semantic similarity cases (80% or more), numeric matches increase. This correlation seems logical, as closely related meanings often involve the use of similar letters and words.

The analysis suggests that there is a relationship between semantic and numerical similarities

in sentences. As semantic similarity increases, the numeric match ratio generally tends to increase as well, implying that closely related meanings often correspond to similar linguistic patterns. However, there are interesting exceptions, such as the 10% similarity range may contain sentences that are diverse and have varying degrees of linguistic differences. This noise and variability in sentence content could lead to unexpected numeric matches, creating deviations from the general pattern.

The Pearson correlation coefficient is a statistical measure that quantifies the strength and direction of a linear relationship between two variables. It is used to assess how closely two sets of data are related to each other in our case the two sets are the "Semantic Similarity Range" and the "Numeric match ratio". The coefficient ranges from -1 to +1, where -1 indicates a perfect negative linear relationship, +1 indicates a perfect positive linear relationship, and 0 indicates no linear relationship.

The Pearson correlation coefficient is calculated using the covariance between the two variables and their respective standard deviations. The formula 5.10 for the Pearson correlation coefficient is:

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}} \tag{5.10}$$

The obtained Pearson correlation coefficient of 0.7674 indicates a strong positive correlation between the "Semantic Similarity Range" and the "Numeric match ratio." This means that as the semantic similarity between sentence pairs increases, the numeric match ratio also tends to increase.

With a correlation coefficient of 0.7674, we can conclude that there is a significant and positive linear relationship between the two variables in the data. The strong positive correlation suggests that higher semantic similarity is associated with a higher numeric match ratio, and vice versa. This finding is valuable as it confirms that the numeric matches are influenced by the semantic similarity between sentences and vice versa.

The Spearman correlation coefficient is a statistical measure that assesses the strength and direction of a monotonic relationship between two variables. In our case, the two vari-

ables are the “Semantic Similarity Range” and the “Numeric match ratio.”

Unlike the Pearson correlation coefficient, which measures linear relationships, Spearman correlation focuses on monotonic relationships, where the variables tend to change together but not necessarily at a constant rate. The Spearman correlation coefficient ranges from -1 to +1, with -1 indicating a perfect negative monotonic relationship, +1 indicating a perfect positive monotonic relationship, and 0 indicating no monotonic relationship. To calculate the Spearman correlation coefficient, we first rank the data points for each variable and then compute the Pearson correlation coefficient for the ranked data.

The obtained Spearman correlation coefficient of 0.875 indicates a strong positive monotonic relationship between the “Semantic Similarity Range” and the “Numeric match ratio.” This suggests that as the semantic similarity between sentence pairs increases, the numeric match ratio tends to increase as well, and vice versa.

With a correlation coefficient of 0.875, we can conclude that there is a significant and positive monotonic relationship between the two variables in the data. This finding further confirms that higher semantic similarity is associated with a higher numeric match ratio, and vice versa, providing valuable insights into the relationship between the semantic and numeric aspects of sentences.

The Scatter Plot and regression equation show a strong positive correlation between Semantic Similarity Rank and Numeric Match Rank. As Semantic Similarity Rank increases, the Numeric Match Rank tends to increase as well. The R-squared (R^2) value of 0.7664 indicates that approximately 77% of the variation in the Numeric Match Rank can be explained by Semantic Similarity Rank.

The Scatter Plot 5.12 and the equation $y = 0.8727x + 0.7$ reveal a strong positive correlation between Semantic Similarity Rank (x) and Numeric Match Rank (y). The equation $y = 0.8727x + 0.7$ represents a linear relationship between the two variables. The coefficient 0.875 (approximately 0.87) is the slope of the line, indicating that for every unit increase in Semantic Similarity Rank, the Numeric Match Rank tends to increase by approximately

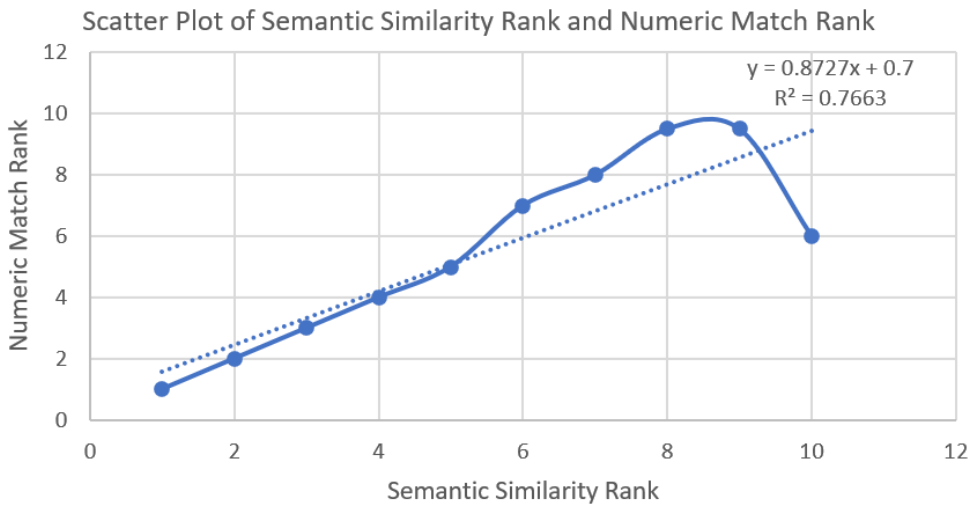


Figure 5.12: Correlation between Semantic Similarity and Numeric Matches.

0.87 units. The term 0.7 represents the y-intercept, which is the Numeric Match Rank value when Semantic Similarity Rank is 0.

The R^2 value of 0.7663 indicates that about 76% of the variability in the Numeric Match Rank can be explained by the linear relationship with Semantic Similarity Rank. This means that the regression line fits the data points well and provides a reasonable representation of the association between the two ranks.

However, there is an interesting observation in the Scatter Plot: after reaching Semantic Similarity Rank 8, the Numeric Match Rank starts to decrease with Semantic Similarity Rank 9 and further decreases with Semantic Similarity Rank 10. This suggests that the relationship might not be strictly linear at higher ranks and could be influenced by other factors or potential outliers in the data. This is due to outliers, which are present in the 10% category, along with stability in the 20% and 30% categories.

Our system employs three key strategies to handle conflicting similarity measures:

- **Threshold-based Filtering:** We use separate thresholds for semantic and numeric similarities. When both values fall below their respective thresholds, the system identifies a similarity. This dual-threshold approach allows for nuanced detection of similarities according to the user's presepective.

- **Weighted Multi-dimensional Analysis:** Each dimension in our numeric space is assigned a unique weight. For instance, the number of words might carry a different weight than the number of letters. This weighting system allows for the varying importance of different numeric features.
- **Interactive Visualization:** We represent each entity as a point in a multi-dimensional space. The position of each point is determined by combining its dimensional values and semantic similarity. Users can explore this visualization interactively (See Figure 5.10). The system notifies users when entities fall within their defined threshold of interest.

In this part of our study, we utilize various statistical measures and visualizations, to examine the relationship between numeric matches and semantic similarities. The Spearman correlation, the Pearson correlation coefficient, the Scatter Plot with regression analysis, and other measures were employed to verify this relationship. Notably, all of these methods consistently confirm the existence of a strong and significant association between numeric matches and semantic similarities. By understanding the interplay between numerical matches and semantic similarities, we gain valuable insights into the structure and content of sentences. This opens up new possibilities for utilizing numeric matches to infer semantic similarities and vice versa, potentially enhancing various natural language processing applications. It is important to acknowledge that our results are specific to the dataset of the Holy Qur'an. Further investigations are necessary to explore this relationship in the context of the Arabic language in general or even in other languages. Different languages may exhibit unique characteristics that could impact the nature of the relationship.

5.6 Summary

In this part of the dissertation, we have introduced the SemSim (Semantic Similarity) system that integrates semantic and numeric similarities to explore multidimensional spaces. This integration allows us to uncover hidden insights and patterns in diverse domains. Specifically, we have developed a model to define multidimensional entities and entity groups in a het-

erogeneous environment. Formal modeling has been applied to different cases to effectively utilize the combined numeric and semantic similarities.

The SemSim system consists of three sub-systems: the ontology-building sub-system (OBS), the multidimensional numeric representation sub-system (MNRS), and the semantic-numeric similarity detection sub-system (SNSDS). These sub-systems work together to detect numeric and semantic similarities among entities in multidimensional datasets. The MNRS sub-system processes numeric data by creating entities, constructing dimensions, and generating a multidimensional representation. The OBS sub-system builds and manages ontologies, allowing users to define concepts and relationships between entities. The SNSDS sub-system leverages the results of the OBS and MNRS to detect similarities between entities. The system provides visual representations and interactive controls to facilitate the exploration of relationships. We have developed a model to define multidimensional entities and entity groups to formalize the concept of similarity. We provided a framework for exploring similarities within multidimensional datasets. The system currently implements basic data structures and algorithms to execute similarity queries.

We analyzed the connection between numeric matches and semantic similarities using a range of statistical measures and visualizations. The Spearman correlation, Pearson correlation coefficient, Scatter Plot with regression analysis, and other techniques were employed to confirm this relationship consistently. These methods collectively reveal a strong and meaningful association between numeric matches and semantic similarities. Understanding this interplay offers valuable insights into sentence structure and content. It also opens new possibilities for utilizing numeric matches to infer semantic similarities and vice versa, enhancing natural language processing applications. Future work will focus on enhancing performance by developing data structures and algorithms that better serve these similarity queries and, consequently, optimize the system's performance.

Chapter 6

COMPUTER VISION-BASED BOUNDARY LOCALIZATION AND DEEP LEARNING-DRIVEN ARABIC DIACRITICS OCR

OCR systems face significant challenges in detecting Arabic text, especially in handwritten form. Diacritical and Tajwid marks add complexity. Additionally, the Ottoman font used in current Mushafs, or the Quran as a book, has been in use for more than 1400 years and is particularly difficult to detect. This font represents a form of calligraphy where scribes showcase their creativity while adhering to specific writing rules.

For effective training of OCR systems, it is essential to have Unicode text as output linked to corresponding handwritten images as input. However, linking entire pages to Unicode text often reduces accuracy. Smaller units, like verses or sentences, yield better results. Ideally, we would link each word, letter, and diacritic mark, but this presents significant challenges. The artistic nature of calligraphy leads to overlap between words, making automatic separation difficult. In many cases, spaces do not clearly separate words, and letters may connect in intricate ways. Figure 6.1 illustrates this challenge, showing a sentence of four words so tightly intertwined that they cannot be automatically separated. Therefore, identifying the beginnings and ends of sentences or verses and linking them to annotated Unicode text with diacritics is our practical solution. This approach helps create a dataset that aids in building our OCR model.

The Mushafs come in different editions produced by multiple printing companies. These

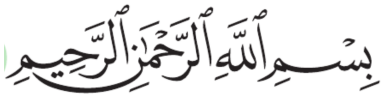


Figure 6.1: Handwritten Arabic calligraphy: Four words seamlessly connected, highlighting the difficulty of automated word separation.

Figure 6.2: The verse bounding boxes of the same page across different prints and different Mushaf linguistic styles.

editions vary in colors and formats but all fall under specific linguistic styles, such as Hafs or Warsh. Each linguistic style employs a unique system for separating and counting verses, which further complicates the OCR process. Therefore, we need to take into account that each Mushaf, regardless of its linguistic style has 1) a different design for the verse marker 2) different verse positions, and 3) different counting schemes that decide the number of verses in each chapter.

Figure 6.2 shows an example of three prints of the same page. Two subfigures (a and b) present two different prints of Hafs Mushaf, and subfigure c presents a print of Warsh Mushaf *(Warsh is another Quran linguistic style)*. From the figure 6.2, we can see a difference in the verse positions across two different editions of the same linguistic style of Hafs. For example, the positions of verses 2 and 7 are different across the editions. Warsh follows a different counting scheme than Hafs. Therefore, the position of verse 1 appears completely different between Hafs and Warsh. The positions of verses 1, 2, and 6 are also different across all editions regardless of the linguistic style. Such differences show the importance of localizing and extracting the verses for each linguistic style automatically.

Figure 6.3: Verse boundary detection enables user interaction at the verse level.

Current websites and apps use manual data entry to store the verse boundaries. The difficulty with manual performance is that for each new edition, the boundaries need to be manually entered again. This manual process is very time-consuming and prone to inaccurate boundaries, not to mention errors. As a result, the current apps usually support only the most widely circulated linguistic style, called Hafs.

The diverse linguistic styles, counting systems, and printing formats of Quranic texts necessitate an automated system for verse localization and extraction. To meet this need, we introduce QR-Vision, a neural network system that uses computer vision to detect verse markers and boundaries automatically. This significantly reduces the time and manual effort required to process various Quranic styles and printings. QR-Vision's ability to identify verse boundaries enables the creation of a dataset pairing Arabic sentences (with diacritics) to their corresponding images. This dataset is essential for training AraDiaOCR, our optical character recognition model specialized in Arabic text with diacritics. In addition to aiding OCR, QR-Vision has other applications. It enhances user interaction with digital Quran pages. Users can highlight verses and access related text, statistics, and audio tracks at the verse level (See figure 6.3).

In this section, our contributions can be summarized as follows:

- Presented QR-Vision, a neural network system that detects verse markers and chapter beginnings in Quran images with high accuracy.

- Developed a method to automatically localize and extract verses across different Quranic linguistic styles and printings.
- Created an OCR system capable of recognizing Arabic text with diacritics, achieving an accuracy of 91.67%.
- Facilitation the creation of a vast dataset of images containing Arabic sentences with diacritics and their corresponding text, aiding OCR systems in accurately recognizing Arabic diacritics.
- We publish our training datasets, trained model, inference results, and code online to make it available to advance research in this area (GitHub, 2022).

6.1 The QRVisionLocator System Architecture

The QRVisionLocator system architecture in figure 6.4 represents an ambition to digitize and analyze the handwriting diacritics Arabic pages, encompassing three main steps:

Detecting Verse Markers and Chapter Beginnings: This step involves pinpointing the location of chapter beginnings and verse markers in various book images. This is the foundation for processing, enabling the system to identify the structural components within the pages, thus facilitating further analysis.

Verse Boundaries Identification and Image Segmentation: After detecting markers, the system outlines the borders of each verse and divides the page into smaller images, each representing a single verse. This segmentation is paramount for isolating individual verses for specific analysis and allows the processing of each verse in isolation, thus enhancing the accuracy and efficiency of the OCR.

OCR Feeding and Text Recognition: This step feeds each segmented verse image into a specialized OCR system, pairing it with the corresponding diacritics-annotated Arabic text. This phase transforms handwritten text into digital form, thus bridging the gap between manual and digital Arabic scripts.

The architecture of the proposed system in figure 6.4 can be broken down into the following

primary modules:

6.1.1 Data Collection and Labeling Module:

This module forms the basis for the system by curating a dataset. It includes:

- Data Collection: Collects 305 images of Mushaf pages to form the training dataset.
- Data Annotation: Utilizes the CVAT labeling tool [179] to annotate the dimensions of chapter beginnings and verse boundaries. The boundaries of verses are represented as a set of bounding boxes around each line that constitutes the verse.
- Data Formatting: Processes the annotated data and exports it in the COCO format for further usage.

In the data collection phase, a systematic ETL (Extract, Transform, Load) process is implemented. We utilize web scraping as the method for extraction. This process extracts Quran page images from various online sources, encompassing different editions and styles of the Mushaf. The text for each Mushaf is collected in the same method, to prepare the dataset for the OCR Model. Following extraction, the data is transformed into a standardized format suitable for analysis. Finally, the transformed data is loaded into a centralized database for further processing. Figure 6.5 details the flow of information from the original web sources to the final database, illustrating the stages of web scraping and ETL.

For labeling, each verse marker and chapter beginning in the 305 training images is annotated manually using the *CVAT* labeling tool [179], as shown in Figure 6.4. The dataset is exported in the Common Objects in Context (COCO) format [118]. The images are stored as raw data after the extraction process, and the CVAT tool is used for transformation, enabling annotation of chapter beginnings and verse dimensions.

Figure 6.6 demonstrates a sample output, containing a list of boxes for each verse, representing the detected boundaries. Each box specifies the (x,y) coordinates of the top left corner, height, and width. This structured output facilitates subsequent processes in the OCR system.

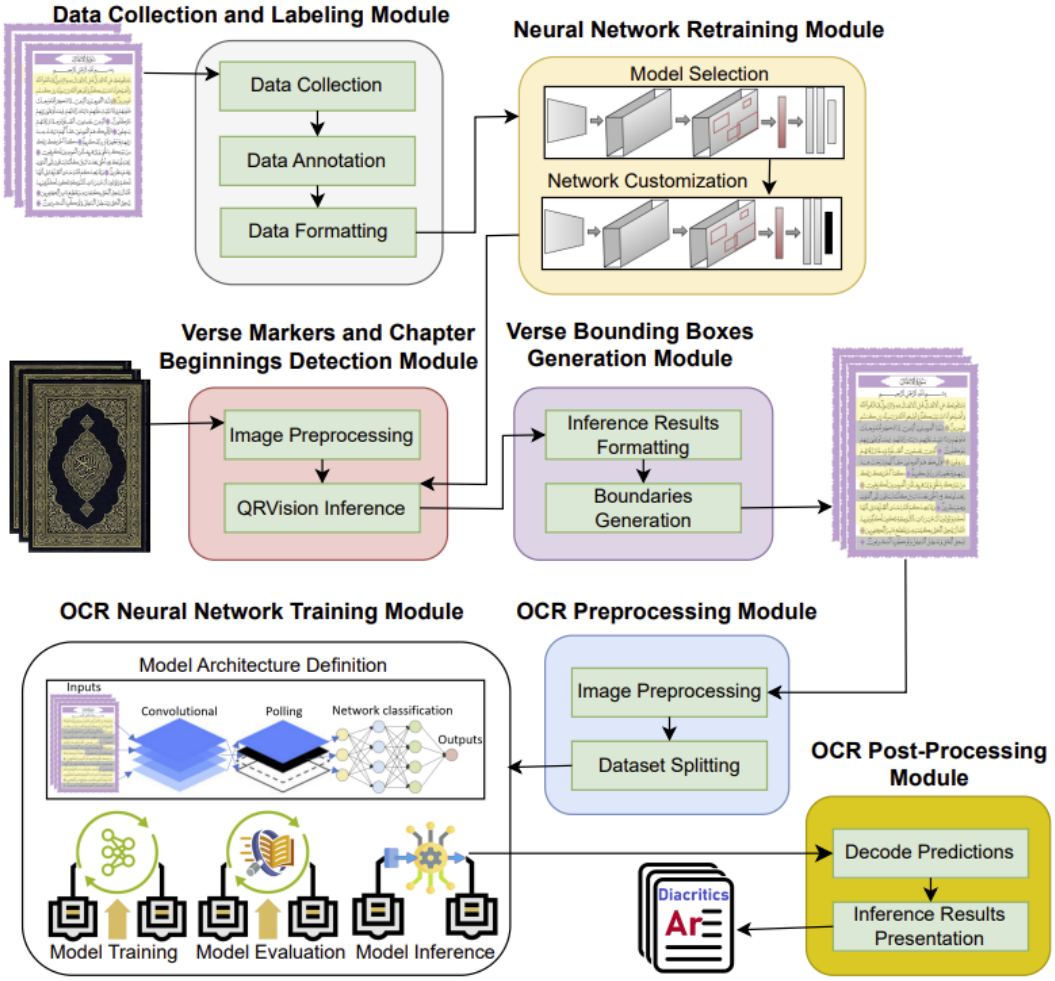


Figure 6.4: The QR-Vision System Architecture.

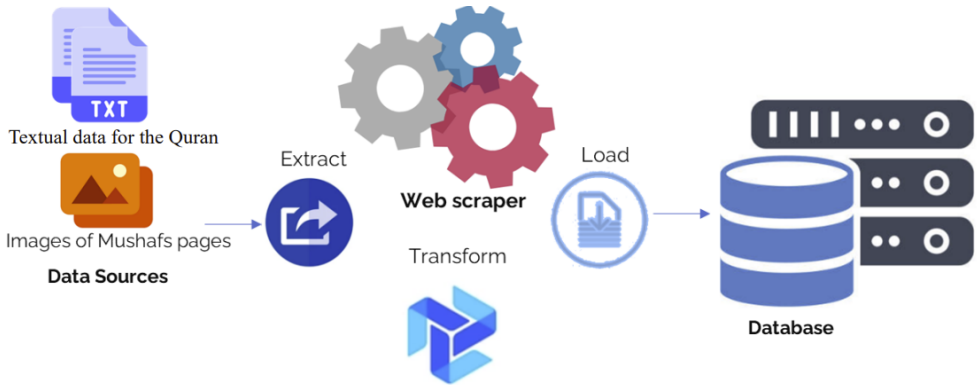


Figure 6.5: Collection of the Quran Text and Page Images ETL Process.

	A	B	C	D	E	F	G	H	I
1	StyleID	PageNo	AyahSerialNo	BoxNoWithinAyah	BoxNo	x	y	w	h
2	2	1	1	1	1	808	1099	729	119
3	2	1	2	1	2	590	1099	218	119
4	2	1	2	2	3	712	1218	825	119
5	2	1	3	1	4	590	1218	122	119
6	2	1	3	2	5	1095	1337	442	119
7	4	1	1	1	1	840	1199	697	132
8	4	1	2	1	2	590	1199	250	132
9	4	1	2	2	3	1267	1331	270	132
10	4	1	3	1	4	796	1331	471	132
11	6	1	1	1	1	785	1199	753	132

Figure 6.6: An Example Tabular Format of the Detected Verse Bounding Boxes.

6.1.2 Neural Network Retraining Module:

Tailoring the model to the specific task of detecting verse markers and chapter beginnings, this module includes:

- Model Selection: Utilizes Detectron2's [190] Faster-RCNN network as the base model.
- Network Customization: Constructs the QRVisionLocator Detection Neural Network by retraining the last layer of the base model. This customization allows the network to specialize in detecting verse markers and chapter starts.

This module employs a convolutional neural network to extract the visual properties of Quran images as a handwritten Arabic text in the highest level of the language accuracy. That aiming to mark verse boundaries and generate boxes representing the area in the image corresponding to each verse separately. The process begins with detecting and localizing verse markers and the beginnings of chapters. Once the markers are identified, separating the boundaries for each verse becomes straightforward. The task of detecting verse markers and chapter beginnings is an objective that help us to build our Arabic diacritics OCR. Detectron2 [190], a library offering collections of Faster R-CNN neural networks, serves object detection [119] and we will use it for detection the verse and chapter markers.

Table 6.1 provides a comparison of different text bounding box detection libraries. These libraries exclusively specialize in text bounding box detection, and none function as OCR tools. The table enumerates various libraries, detailing their respective applicability.

Bounding Box libraries	Reason for Use
EAST Detector	Specialized for scene text detection, efficient and accurate [203].
TextBoxes	Fast detection of arbitrary-shaped text instances [116].
Detectron2	Utilizes Fast R-CNN and Faster R-CNN for object detection, widely used and maintained [190].
YOLO	Popular can be adapted for text bounding box detection [57].
OpenCV Text Detection	Computer Vision open-source library provides various algorithms for text region detection [75].
CRAFT	Can detect individual characters and their bounding boxes [193].
TextSnake	Can predict both text regions and instance representations [126].

Table 6.1: Comparison of Text Bounding Box Detection Libraries

Reasons for choosing Detectron2 as a suitable solution for our research problem:

- Versatility: Detectron2 provides a wide range of object detection models, which can be adapted and tuned for different tasks [43]. This flexibility allows us to customize the model based on the characteristics of the Quran images and the verse marker shapes.
- Established Framework: Detectron2 is a well-established and actively maintained library by Facebook AI Research (FAIR).
- Performance: Detectron2's models are known for their high performance in object detection tasks.

Figure 6.7 illustrates the high-level architecture of Faster R-CNN. This method introduces a separate network to learn and predict region proposals, which are then fed into the detection network for object detection. This strategy not only accelerates the process but also makes it suitable for re-training on different classes (verse and chapters), as the region proposal network can be re-trained.

To adapt the model to our specific needs, we prepare a new training dataset by labeling Quranic images, freezing all neural network layers except the last fully connected one. The training dataset helps us retrain the network, tuning the weights of this last layer according to the new dataset and the specific classes to be detected. Two classes of objects are targeted: verse markers and the beginnings of chapters (surahs). Figure 6.7, elucidates the re-training architecture. After refining and tuning, the trained model is utilized to perform batch inference on all Quranic pages to detect verse markers and chapter beginnings.

Table 6.2 presents the distribution of data from a vision dataset, which is derived from a

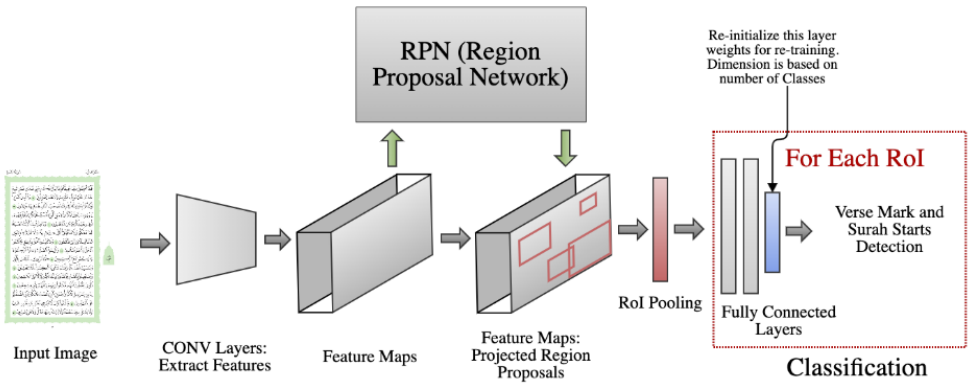


Figure 6.7: Retraining The Faster R-CNN Architecture at a High Level.

Table 6.2: Text localization Dataset: Sentences and Boxes in Training, Validation, and Testing

Text localization dataset	Training	Validation	Testing	Totals
Sentences	2413	623	1007	4043
Boxes	5338	1223	1502	8063

single book comprising 604 pages. This dataset is for text boundary detection only. We use different dataset to train the OCR model. This is because the goal at this stage is completely different from OCR and therefore the training datasets are different. For text boundary dataset, approximately 50% of the book, which is 305 pages, is used. The data is further split into three subsets: training, validation, and testing. Each subset contains a specific number of sentences and corresponding boxes.

The "Training" column shows the number of sentences and boxes used for training. There are 2413 sentences and 5338 boxes in the training subset. The "Validation" column represents the data used for validating process. This subset consists of 623 sentences and 1223 boxes. The "Testing" column contains the data used to evaluate the performance of the trained model. It includes 1007 sentences and 1502 boxes. The "Totals" row provides the overall count of sentences and boxes present in the entire dataset, which is 4043 sentences and 8063 boxes.

6.1.3 Verse Markers and Chapter Beginnings Detection Module:

Engaged in preprocessing and analyzing images to detect chapter beginnings and verse boundaries, this module encompasses:

- Image Preprocessing: Responsible for resizing, normalizing, and adjusting the images to be compatible with the Detectron2 library, ensuring optimal conditions for inference.
- QRVisionLocator Model Inference: Applies the trained model to new images of the Mushaf pages. The module analyzes these images and returns the coordinates of detected chapter beginnings and verse boundaries.

Detectron2's preprocessing pipeline usually involves transformations such as resizing to a fixed size and normalizing the pixel values with a standard mean and standard deviation. This uniform preprocessing makes the images suitable for training and inference, allowing the neural network to work efficiently with diverse input data. This module feeds the testing dataset Quran images, for a Hafs linguistic style, into the neural network for batch inference. The neural network processes all pages and detects verse markers and chapter beginnings in all pages and generates boundary boxes around them. When evaluating the QRVisionLocator model for object detection on the testing dataset, it achieved an accuracy of 100%. This perfect accuracy can be attributed to several factors:

- **Clarity of Markers:** Both the verse and chapter markers within the Quranic images are distinct and clear. Their uniqueness aids in precise detection.
- **Detectron2 Efficiency:** The application of Detectron2 for object detection lends itself to the model's high accuracy.

The goal of the project extends beyond understanding just a single book or linguistic style. Different books may share the same linguistic style but differ in formatting due to variations in the time of writing or printing techniques. Moreover, the project aims to recognize different linguistic styles, each of which may possess slight differences in formatting and appearance, as shown in Figure 6.10. Figure 6.10 displays some corner cases that appear in the pages

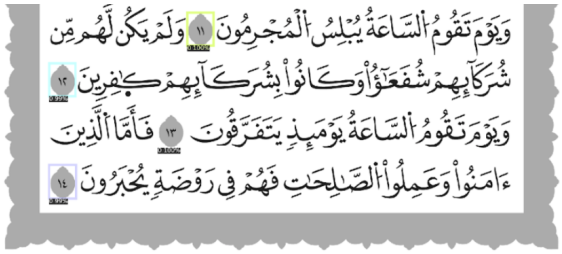

وَيَوْمَ تَقُومُ ٱلسَّاعَةُ يُبْلِسُ ٱلْمُجْرِمُونَ ١١ وَلَمْ يَكُن لَّهُم مِّن
شُرَكَآئِهِمْ شُفَعَٰٓؤُا۟ وَكَانُوا۟ بِشُرَكَآئِهِمْ كَٰفِرِينَ ١٢
وَيَوْمَ تَقُومُ ٱلسَّاعَةُ يَوْمَئِذٍ يَتَفَرَّقُونَ ١٣ فَأَمَّا ٱلَّذِينَ
ءَامَنُوا۟ وَعَمِلُوا۟ ٱلصَّٰلِحَٰتِ فَهُمْ فِى رَوْضَةٍ يُحْبَرُونَ ١٤

Figure 6.8: Verse Markers Detected by The Model.


خَٰلِدِينَ فِيهَا ۚ رَضِىَ ٱللَّهُ عَنْهُمْ وَرَضُوا۟ عَنْهُ ۚ أُو۟لَٰٓئِكَ حِزْبُ
ٱللَّهِ ۚ أَلَآ إِنَّ حِزْبَ ٱللَّهِ هُمُ ٱلْمُفْلِحُونَ ٢٢
سُورَةُ الْحَشْرِ
بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ
سَبَّحَ لِلَّهِ مَا فِى ٱلسَّمَٰوَٰتِ وَمَا فِى ٱلْأَرْضِ ۖ وَهُوَ ٱلْعَزِيزُ
ٱلْحَكِيمُ ١ هُوَ ٱلَّذِىٓ أَخْرَجَ ٱلَّذِينَ كَفَرُوا۟ مِنْ أَهْلِ ٱلْكِتَٰبِ مِن

Figure 6.9: Chapter Beginnings Detected by The Model.

of the Quran. For example, Surah At-Tawba has no Basmallah (no normal beginning), and Surah Al-Fatihah has a unique layout across linguistic styles. Despite these exceptions, the model still marks the verses correctly.

The method employed in determining the positions of verses and chapters is shared across various books and linguistic styles. As such, the model is trained, taking these widespread considerations into account, ensuring that it is adaptable to different formats and styles. After adopting the results from QR-Vision, we applied the system to five different linguistic styles available on our website https://quranresearch.org [163]. Currently, we are focusing on the King Fahd Printing Press [47] for the Holy Quran. This application demonstrates the versatility of our approach across various Quranic texts. Accurate determination of sentence or verse boundaries is highly beneficial for creating datasets used in training and testing OCR. Figures 6.8 and 6.9, showcasing examples of Verse Markers and Chapter Beginnings detected by the model, provide a sense of the accuracy of the model.

6.1.4 Verse Bounding Boxes Generation Module:

Converting the detected markers into usable formats, this module includes:

- Boundaries Generation: Derives and stores the boundaries of each verse in the dataset, translating the dimensions detected by the neural network into a usable format.
- Inference Results Formatting: Transforms the inference results from the COCO format to the case of displaying borders around the verses.

Figure 6.10: Visualization of Some Corner Cases in the Detection of Verse Bounding Boxes.

Figure 6.11: Verse Bounding Boxes Visualization on Page of Quran.

Through simple logic, it draws boxes around the content of each verse. The content associated with each verse is situated between the current verse marker and the preceding one. In the case of the first verse in a chapter, the beginning of the chapter mark is utilized to detect the starting boundary.

The boundary of each verse may consist of multiple boxes. The combination of these boxes between two verses forms the bounding box for a given verse. Figure 6.11 demonstrates the boundary of verse 10, represented by a list of three boxes. This logic is applied iteratively to all verses, accurately marking the boundaries. Figure 6.10 highlights specific corner cases appearing in the Quran's pages, such as the unique layout of Chapter At-Tawba and Surah Al-Fatihah. Despite these nuances, the model still marks the verses precisely. The resulting output serves multiple purposes. It can be a standalone output for further use in OCR systems or be integrated into various applications and websites, adding to its adaptability

and utility. By clearly delineating the boundaries of verses, this module paves the way for more advanced applications such as OCR.

6.1.5 OCR Preprocessing Module:

Preparing images of individual verses for OCR, this module includes:

- **Image Preprocessing:** Following the detection of markers and the outlining of verse boundaries, this step resizes, pads, and normalizes the segmented verse images to enhance the accuracy and efficiency of OCR.
- **Dataset Splitting:** This step divides the preprocessed images and corresponding text labels into training, evaluation, and testing sets.

The core of the image preprocessing involves a two-level hierarchy, focusing on pages and verses, respectively.

- **Pages Level:** Every individual page is processed, accommodating the unique layouts and structures within.
- **Verses Level:** On each page, verses are grouped and then cropped into separate images. An empty verse image is created with specific dimensions, and each box (part of the verse) is cropped from the page image and pasted onto the verse image. The entire image of the verse is then saved with the corresponding Arabic annotated-diacritics text.

Table 6.3: Splitting of the OCR Dataset

Set	**Percentage**	**Number of Images**
Total	100%	6,237
Training Set	70%	4,366
Evaluation Set	15%	935
Testing Set	15%	936

The dataset used for the OCR is the output of the previous four modules. Table 6.3 provides an overview of the distribution of images across the different sets:

- **Training Set:** Comprising 70% of the total images, or 4,366 images, this set is utilized

for training the model.

- **Evaluation Set:** Containing 15% of the images, or 935 images, this set is used for tuning hyperparameters and evaluating the model's performance during development.
- **Testing Set:** Also containing 15% of the images, or 936 images, the testing set is reserved for assessing the model's final performance, simulating how it might perform on unseen data.

6.1.6 OCR Neural Network Training Module:

This module is responsible for constructing and training the OCR model and consists of:

- Model Architecture Definition: Defines the CNN model architecture for Arabic text with diacritics.
- Model Training: Trains the CNN model on the preprocessed images.
- Model Evaluation: Evaluates the trained model on the evaluation and test sets, calculating accuracy and other performance metrics.
- Model Inference: Utilizes the trained CNN model to recognize Arabic text in given images, preserving diacritics.

Figure 6.12 illustrates a schematic overview of the training architecture used to recognize Arabic Diacritics through OCR, emphasizing key components and data flow.

1. **RGB Input Image:** The input layer encodes the Arabic verse's visual information into a numerical format, preserving the spatial structure essential for understanding Arabic diacritics.
2. **CNN (Convolutional Neural Network):** Processes the input image through convolutional, activation, and pooling layers.
    - **Convolutional Layer:** Extracts features like edges, curves, and patterns, crucial for recognizing characters and associated diacritic marks.
    - **Activation Function:** Implements ReLU to introduce non-linearity, enabling

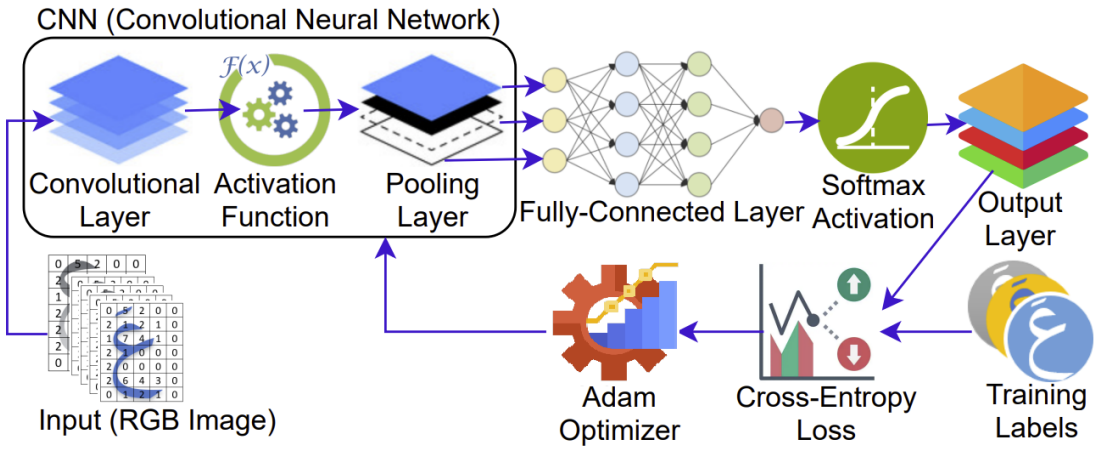


Figure 6.12: CNN-Based OCR Model for Arabic Diacritic Recognition: AraDiaOCR Training Process.

the model to learn complex relationships and patterns vital in Arabic diacritics OCR.

- **Pooling Layer:** Utilizes Max Pooling to reduce feature map dimensionality, retaining crucial information while boosting computational efficiency.

3. **Fully Connected (Dense) Layer:** Transforms the high-level features into a classification-ready format, further discerning the relationships between the features and the desired Arabic text with diacritics.
4. **Softmax Activation:** Converts the model's raw output into probability distributions over different classes, aiding final decision-making.
5. **Output Layer:** Delivers the final prediction of the Arabic verse with diacritics.
6. **Training Labels (Arabic Text with Diacritics):** Serves as the ground truth, enabling the model to learn patterns corresponding to correct Arabic text and diacritics.
7. **Cross-Entropy Loss:** Quantifies the difference between predicted probabilities and actual target values, guiding the optimization of diacritic recognition.
8. **Adam Optimizer:** Adjusts the model's weights to minimize loss, efficiently tuning the performance in recognizing Arabic diacritics.

6.1.7 OCR Post-Processing and Storage Module:

Translating OCR results into accessible formats, this module includes:

- Decode Predictions: Converts the model's predictions into human-readable Arabic text with diacritics.
- Inference Results Presentation: Transforms the recognized text into a suitable format for further analysis, displaying, involving formatting that enhances readability, and adding metadata like page, verse, and so on.

These modules collectively form an integrated pipeline that streamlines the complex task of translating the handwritten Arabic pages into a digitized format and preserving the intricate diacritics of the Arabic text. The resulting system is adaptable to various linguistic styles and serves as an advancement in the digitization of heritage books.

6.2 Results and Discussion

In this section, we present and analyze the performance of our Arabic Diacritics OCR model, AraDiaOCR. We compare its performance with Microsoft's Azure OCR to highlight the effectiveness of our approach in recognizing Arabic text with diacritics. We also discuss the training dynamics and provide visual representations of the results. Finally, we summarize our findings.

6.2.1 Performance of AraDiaOCR

We evaluated our OCR model using standard performance metrics: accuracy, precision, recall, and F1 score. These metrics were calculated for the testing, training, and evaluation datasets.

Performance Metrics

Table 6.4 summarizes the performance metrics of AraDiaOCR. The model demonstrates high accuracy across all datasets, exceeding 91%. Precision, recall, and F1 score are also

consistently high, indicating the model's effectiveness in recognizing Arabic diacritical text.

Table 6.4: Performance Metrics for Arabic Diacritics OCR

Metric	**Testing**	**Training**	**Evaluation**
Accuracy	0.9167	0.9190	0.9162
Precision	0.8950	0.8980	0.8940
Recall	0.8970	0.8995	0.8965
F1 Score	0.8960	0.8987	0.8952

From Table 6.4, we observe that the testing accuracy is 91.67%, closely matching the training and evaluation accuracies. The precision and recall values are also balanced, resulting in F1 scores around 89.6%. These results indicate that the model performs reliably across different datasets and is not overfitting.

Training and Validation Loss

Figure 6.13 shows the training and validation loss over epochs. The training loss starts at 4.4 and decreases to 3.5, while the validation loss decreases from 4.5 to 3.6. This downward trend suggests that the model is learning effectively during training.

The AraDiaOCR Training and Validation Loss

The convergence of training and validation loss curves indicates that the model is generalizing well and not overfitting to the training data. The slight gap between the two curves is acceptable and shows consistent learning.

Training and Validation Accuracy

Figure 6.14 illustrates the training and validation accuracy over epochs. The training accuracy increases from 88% to 91.9%, while the validation accuracy rises from 90.7% to 91.6%. These improvements reflect the model's ability to learn and generalize from the data.

The close alignment of the training and validation accuracy curves suggests that the model maintains consistent performance on unseen data. The steady increase in accuracy demonstrates the effectiveness of our training process.

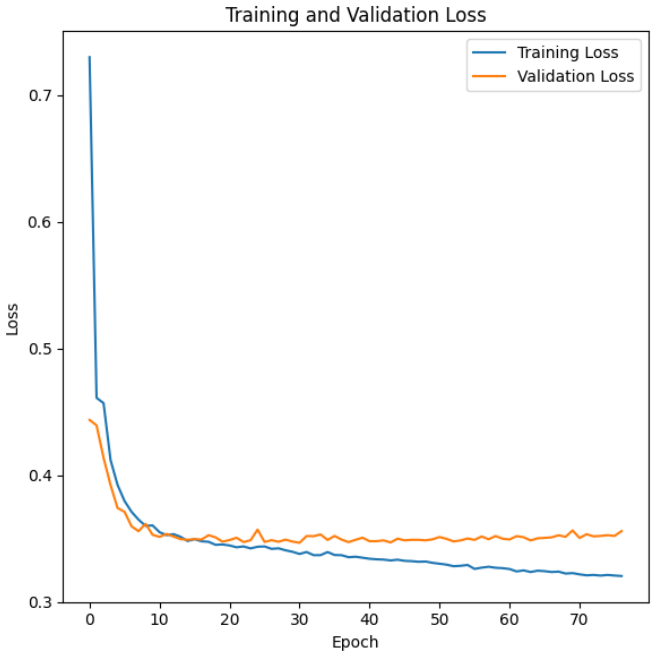


Figure 6.13: Training and Validation Loss over Epochs

6.2.2 Comparison with Microsoft Azure OCR

To evaluate our model's performance relative to existing solutions, we compared AraDiaOCR with Microsoft Azure OCR. We tested Azure OCR using the same datasets.

Performance Metrics of Azure OCR

Table 6.5 presents the performance metrics for Microsoft Azure OCR when it works with the Arabic with diacritic marks. The accuracy, precision, recall, and F1 score are all around 74% to 76%, significantly lower than those of our AraDiaOCR model.

Table 6.5: Performance Metrics for Microsoft Azure OCR

Metric	**Testing**	**Training**	**Evaluation**
Accuracy	0.7600	0.7619	0.7596
Precision	0.7400	0.7420	0.7395
Recall	0.7420	0.7435	0.7415
F1 Score	0.7410	0.7427	0.7405

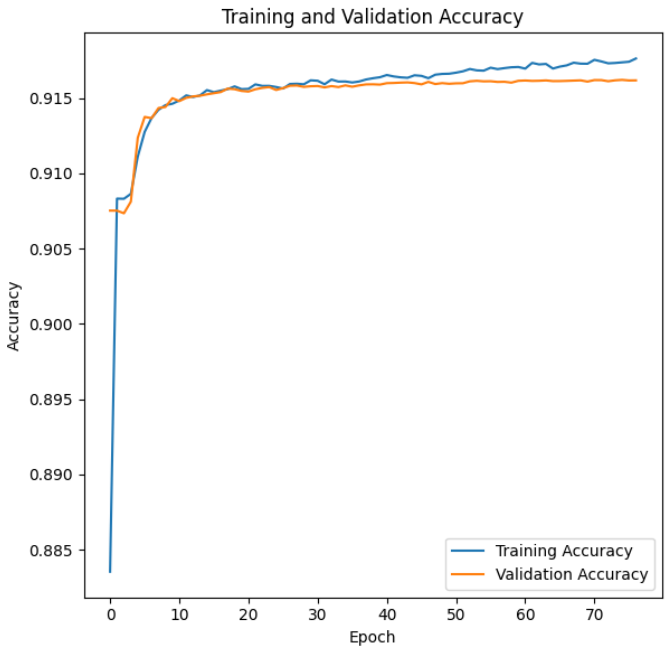


Figure 6.14: Training and Validation Accuracy over Epochs

Microsoft Azure OCR achieves an accuracy of 76% on the testing dataset, which is substantially lower than the 91.67% accuracy of our model. The precision and recall values are also lower, resulting in a lower F1 score.

Discussion of Results

Our model, AraDiaOCR, outperforms Microsoft Azure OCR by a significant margin. The higher accuracy and balanced precision and recall indicate that our model is more effective in recognizing Arabic text with diacritics. We achieved this by creating the system that links text images to their matching diacritic-annotated text. This approach allowed us to build a dataset. We then used this dataset to train our model, resulting in high accuracy. Azure OCR, for instance, does not officially support Arabic diacritics [134], which explains its lower performance on our dataset.

Visual Comparison of Performance Metrics

Figure 6.15 provides a grouped bar chart comparing the performance metrics of AraDiaOCR and Azure OCR across the testing, training, and evaluation datasets.

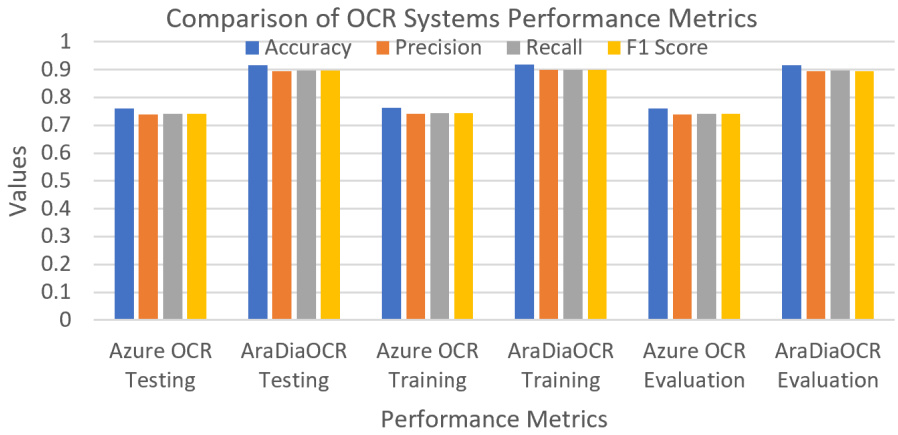


Figure 6.15: Comparison of OCR Systems Performance Metrics

From the chart 6.15, it is evident that AraDiaOCR consistently outperforms Azure OCR across all metrics and datasets. The differences in accuracy, precision, recall, and F1 score highlight the effectiveness of our model in handling Arabic diacritic text.

Figure 6.16 presents a radar chart visualizing the performance metrics of both OCR systems on the testing dataset.

The larger area covered by AraDiaOCR in the radar chart emphasizes its superior performance. Each axis represents a performance metric, and our model shows higher values across all metrics compared to Azure OCR.

6.2.3 Summary

Our OCR model, AraDiaOCR, achieves high accuracy in recognizing Arabic text with diacritics, outperforming existing commercial solutions like Microsoft Azure OCR. The model demonstrates robust learning capabilities, as evidenced by the training and validation loss and accuracy curves.

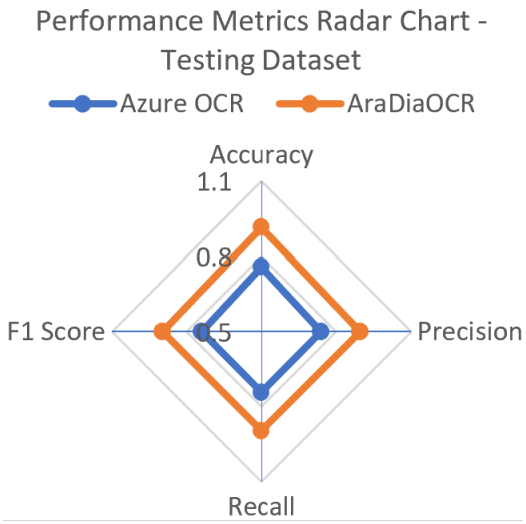


Figure 6.16: Performance Metrics Radar Chart - Testing Dataset

The significant improvement over Azure OCR underscores the importance of specialized models for languages with unique characteristics like Arabic. Our model addresses the challenges associated with Arabic diacritics, filling a gap left by general-purpose OCR systems.

These results indicate the potential for further advancements in Arabic OCR and encourage the development of language-specific solutions to improve text recognition accuracy.

6.3 Summary

This chapter introduces QR-Vision and AraDiaOCR, two systems designed to address the challenges of Arabic text recognition with diacritics. QR-Vision focuses on detecting verse markers and chapter beginnings in Quranic texts, while AraDiaOCR is an OCR system specifically tailored for Arabic text with diacritics. We created a dataset of 6,237 images, representing various linguistic styles of Arabic text. They developed novel preprocessing techniques and a specialized CNN architecture to handle the complexities of Arabic script. The AraDiaOCR system achieved an impressive 91.67% accuracy in recognizing Arabic diacritical characters, significantly outperforming existing commercial solutions. This achievement addresses a crucial gap in current OCR technologies, which often struggle with or ignore Arabic diacritics. The chapter provides detailed insights into the system architecture,

data preprocessing, model training, and evaluation processes. Overall, this research makes a contribution to the field of Arabic OCR, particularly in handling diacritics and different linguistic styles. It paves the way for improved digitization of Arabic heritage texts and better processing of classical Arabic in various applications.

Chapter 7

DIACRITIC-AWARE SPEECH AND TEXT PROCESSING IN ARABIC: SEGMENTATION, CLASSIFICATION, AND ALIGNMENT

Speech recognition is an essential component of Natural Language Processing (NLP) that converts spoken language audio into written text. On the other hand, Text-to-Speech (TTS) synthesizes spoken language from written text. These speech components involve audio segmentation, where the soundtrack is split into phonemes, words, and sentences. Additionally, audio-to-text alignment aims to synchronize the audio files with the corresponding text, creating a reliable audio-text dataset. Various models and systems have shown promising results in the domain of Arabic speech recognition; however, their focus does not encompass diacritics [177]. This limitation is primarily attributed to the lack of accessible datasets containing diacritized Arabic texts. Hence, the primary motivation behind our research endeavors is to address this challenge and explore potential solutions.

Adding diacritics is important for Arabic text comprehension and has a significant impact on phonetics. For instance, a single Arabic letter can have over 12 diacritical marks, each corresponding to a distinct sound. We conducted experiments to investigate how diacritics influence phonetics in Arabic. First, we compared the same letter with the same diacritic across two sounds. This established a baseline measurement of natural variation. The baseline shows typical differences that occur even when the diacritic does not change. Then, we compared the same letter with different diacritics. This comparison measures how diacritics influence pronunciation.

We calculated the distance between formant frequencies of the sounds. Formants are resonance frequencies of the vocal tract. We used the Formant Euclidean Distance (FED) to

measure the difference between two sounds. FED calculates the Euclidean distance between their formant frequencies. Formants are essential in characterizing vowel sounds. In phonetics and linguistics, analyzing formant frequencies helps understand speech sounds and their variations.

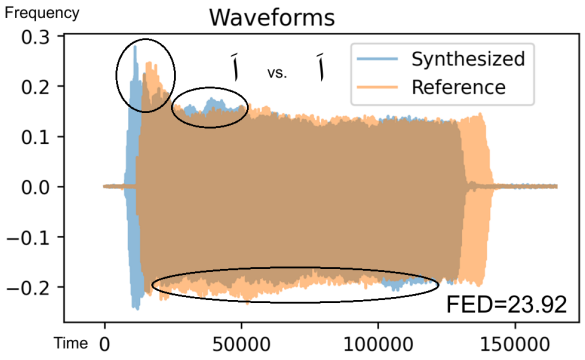


Figure 7.1: Comparison of two sounds for the letter Alif with Fatha diacritics

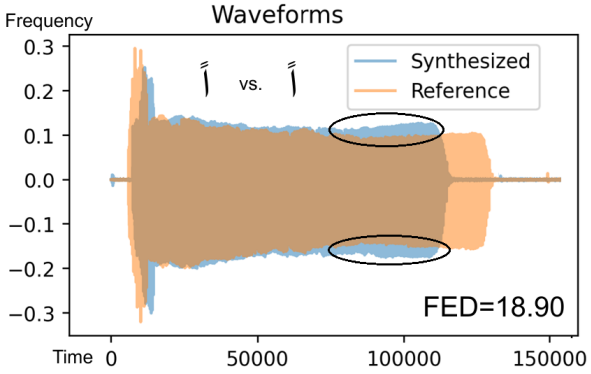


Figure 7.2: Comparison of two sounds for the letter Alif with Tanween Fatha diacritics

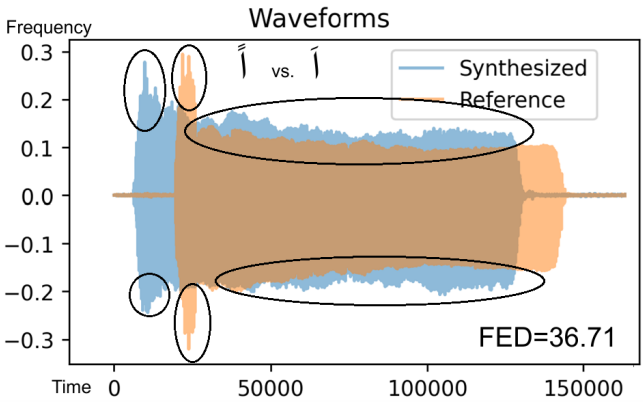


Figure 7.3: Comparison of sounds for Alif with Fatha and Alif with Tanween Fatha diacritics

We created overlaid waveforms, with the first sound in blue and the second in transparent orange. This visual representation helps see differences between any two sounds in the audio signals over time and frequency. Sometimes a sound starts or ends before another sound. One sound may be slightly longer or shorter than the other. These are natural differences that don't significantly affect the pronunciation comparison. The two letters are supposed to take the same time, but this difference is a human estimate. FED typically uses average

formant values over the duration of a sound, which helps mitigate the impact of small timing differences.

Figure 7.1 compares two sounds of the Arabic letter Alif with the Fatha diacritic (اَ and اَ). The FED between these two sounds is 23.92. Figure 7.2 shows a comparison of two sounds of Alif with the Tanween Fatha diacritic (اً and اً). The FED value here is 18.90. This value is lower than the previous comparison. It suggests that the two sounds are almost identical, with very slight differences. Figure 7.3 compares the sounds of Alif with Fatha and Alif with Tanween Fatha (اَ and اً). The overlaid waveforms reveal many differences in the phonetic signals. These differences appear both at the beginning and throughout the sounds. The FED value for this comparison is 36.71. This value is significantly higher than the previous values. It indicates that the diacritics cause noticeable changes in pronunciation.

These findings highlight the importance of accurate diacritization in Arabic text-to-speech (TTS) and speech recognition systems. Diacritical marks change the phonetic representation and pronunciation of a letter. This change leads to distinct speech sounds. Therefore, ensuring precise diacritization is essential for producing natural and intelligible speech synthesis. It is also crucial for accurate speech recognition in Arabic language applications.

This section of the dissertation introduces the Quran Speech Recognition (QRSR) system. Its name is derived from the Qur'an, as it utilizes audio recitations of the Qur'an in Arabic as the foundation for its operations. The QRSR focuses on enhancing Arabic Automatic Speech Recognition (ASR) and TTS through the preparation of a diacritic Arabic text-audio dataset. The idea revolves around using the VAD (Voice Activity Detection) algorithm to divide audio files based on areas of silence. The content of these audio files is then processed using avail-

able Speech Recognition systems. The available Arabic ASR systems cannot handle more than the second level of Arabic (see Figure 1.5). After obtaining ASR transcripts, alignment is done with fifth-level Arabic text that includes diacritical marks. However, comparing texts with different levels of Arabic and containing errors poses challenges for the classification process (determining the position of sound in relation to the text) and the alignment process (identifying each audio file and its affiliation with the text). The goal is to provide Audio files aligned with Arabic diacritic-annotated text that supports Speech Recognition and Text-to-Speech (TTS) systems. This support aims to handle Arabic diacritic-annotated text, which is not addressed by existing Arabic ASR systems. To address these challenges, we proposed a Fuzzy Text Alignment and Rule-based Classifier (FTARC) approach. This approach offers a practical solution for segmenting audio files, aligning texts, and classifying based on predefined labels. This approach achieves an accuracy rate of up to 90.78% according to our testbed. To further improve accuracy, we integrate the FuzTPI algorithm, initially used in FTARC, with Machine Learning (ML) models. We examined the effectiveness of several Machine Learning (ML) algorithms, including Naïve Bayes [110], Support Vector Machine (SVM) [123], and Random Forest [174], for enhancing the classification of Arabic texts. Integrating FuzTPI with ML models significantly enhances the efficiency of the alignment-driven classification process. Among the four models implemented, the FuzTPI-Random Forest model achieves the highest accuracy, reaching 96%. The proposed approach, which combines fuzzy text alignment, and integration with ML models, aims to improve Speech Recognition and TTS systems to support the Arabic diacritic-annotated text. In this part of the research, the Qur'an serves as a case study, utilizing the availability of Quranic diacritic-annotated text and audio recitations. However, the generalization of the work to the general Arabic text is one of our goals.

The contributions of this section of the dissertation can be summarized as follows:

- Development of the Quran Speech Recognition (QRSR) system, utilizing Quran audio and text to advance Arabic speech recognition and contribute to natural language

processing (NLP) systems.

- Application of various classification algorithms, including ML models, to classify the un-diacritic Automatic Speech Recognition (ASR) transcript into subsentences aligned with diacritic-annotated Arabic sentences.
- Alignment of segmented audio with diacritic-annotated Arabic text representations using existing Speech Recognition systems, addressing the challenge of diacritic-aware recognition not being supported by current Arabic Speech Recognition systems.
- Introduction of the FuzTPI algorithm, which enhances ML models and improves the alignment-driven classification process in Arabic speech recognition.
- Application of FuzTPI-Random Forest to boost the performance of the QRSR system, achieving an accuracy rate of 96%.
- Contribution to the development of an expanded textual audio dataset, offering potential enhancements for Arabic speech recognition on a broader scale.

7.1 QRSR System: Methodology and Key Operations

The QRSR System objective is to segment and align Arabic audio with diacritic-annotated text. This process aims to create a substantial dataset of diacritics in Arabic audio-text pairs. The resulting dataset will serve as a valuable resource to enhance and improve diacritic-aware ASR and TTS systems. The system is comprised of six modules, as depicted in Figure 7.4. The upcoming six subsections will provide detailed explanations of these modules. However, the system's overall concept can be summarized as follows.

We start by preparing our dataset that has *diacritic-annotated text* (at the 5^{th} level of Arabic annotation) along with its corresponding audio recitations as *audio files*. This dataset serves as a resource for model training and evaluation purposes. Audio files are usually of long duration, which hinders processing and aligning based on sentences. The audio files are segmented into smaller audio clips based on detected periods of silence by using *Voice Activity Detection*, or *VAD*, algorithm [53]. Each audio clip can represent a sentence, part of

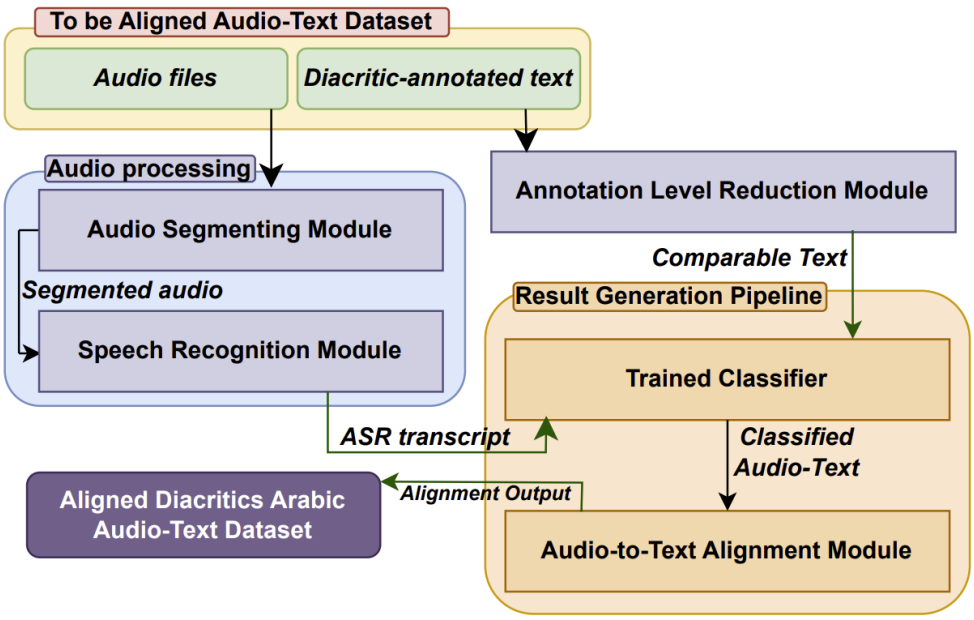


Figure 7.4: The architecture of Arabic speech to diacritic-annotated text alignment system.

a sentence, or several sentences the reciter connected together in one connected recitation. Each audio clip is processed by one of the available ASR systems, specifically the Google Cloud Speech-to-Text API, to generate a textual transcript referred to as "ASR Transcript". The *ASR transcript* produced by current ASR services is featured by two things: (a) the ASR transcript is in the second level of annotation and (b) has a low degree of accuracy and shows several errors in the Arabic speech (as shown in figure 1.5). We align the generated ASR transcript to the original text (i.e., *diacritic-annotated text*) that we have for the audio files. A set of models is trained to determine the alignment between the diacritic-annotated sentence and the ASR transcript. The architecture of the QRSR System is illustrated in Figure 7.4 and is discussed in the following subsections.

7.1.1 To be Aligned Audio-Text Dataset

Arabic audio files and their corresponding texts are collected to establish a dataset. Quranic texts and phonetics are chosen for their accurately written diacritical letters and precise pronunciation. The dataset comprises 9,596 audio files and records. All audio files utilized in the dataset are high-quality recordings without any noise interference. The noise interference

is not within the scope of our study. The label indicates the alignment of the ASR transcript of the audio file with the verse. The labels specify the position of the reading within the verse, such as the beginning, middle, or end. The dataset is divided into a 70% training set (6,715 records), a 15% test set (1,442 records), and a 15% validation set(1,439). We used the validation dataset to fine-tune the settings that control our models, which we refer to as hyperparameters. The process of adjusting these hyperparameters is important in achieving the best performance from our models. The division of the dataset is performed separately for each label to ensure a fair representation of each label in both the training and test data. This approach ensures that each label is adequately represented in both sets.

Table 7.1: Labels alignment and unalignment cases between audio and text.

#	Label	Explanation
1	FAFV	**FullAudioFullVerse:** The entire audio file corresponds to a complete verse.
2	VPOA	**VersePartOfAudio:** The audio contains a complete verse along with other verses.
3	VPOEA	**VersePartOfEndAudio:** The complete verse is at the end of the audio, which also includes other verses before it.
4	PVLP	**PartialVerseLastPartRecited:** The audio contains a portion of the verse, specifically the last part of it.
5	PVNF	**PartialVerseNotFinished:** The audio contains a portion of the verse but does not include the last part of it.
6	Uncertain	**Uncertain:** The text extracted from the audio is unclear regarding its association with a verse. However, there is still a low similarity ratio.
7	ANR	**AudioNotRecognized:** The audio file content is unidentifiable and therefore excluded from the classification process. It was not included in the dataset from the beginning.
8	UA	**UnspecifiedAudio:** The audio could not be linked to the verse.
9	ANAV	**AudioNotAVerse:** This audio does not represent a specific verse. It may contain opening or closing words unrelated to any verse. As a result, it is excluded from the classification process since it requires comparison with all verses, rather than a single classification.
10	MA	**MissingAyah:** We could not establish a link between this verse and any audio files.

The sentences or verses display variations in length. Longer sentences may be read in parts,

where one long sentence corresponds to multiple audio clips separated by periods of silence. On the other hand, shorter sentences may be recited together without periods of silence, resulting in one audio clip. This variation introduces challenges in segmenting and aligning the audio at the sentence level. To address this, during the dataset preparation phase, we carefully label each audio clip based on whether it contains a complete sentence, a partial sentence, multiple sentences, or falls into other categories. This labeling process involves a manual review of both the audio clips and their corresponding text. The resulting labels are summarized in Table 7.1.

To gain insights into the label representation in the collected dataset, we assess their distribution, and a bar chart is given in Figure 7.5. The x-axis represents the class labels and the y-axis represents the frequency of each label in the dataset.

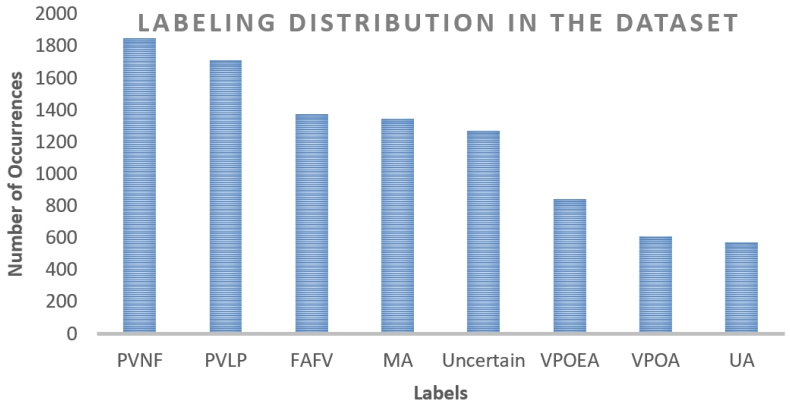


Figure 7.5: Labeling Distribution in the Dataset.

The preparation of the training and evaluation dataset involves a semi-automatic process of comparing and labeling the ASR transcript with the comparable text. The dataset preparation involves several automated steps:

1. The Audio Segmentation Module automatically segments phonemes.
2. The Speech Recognition Module generates an ASR transcript.
3. An automated process links the ASR transcript to the corresponding phonemes.
4. The Annotation Level Reduction Module processes diacritical-rich text. It removes

diacritical characters and makes other changes to create "comparable text" that can be easily compared with the ASR transcript.

5. The system automatically assigns some straightforward labels. For example, it labels audio files as "AudioNotRecognized" when the content cannot be recognized.

Each of these automated steps is explained in detail in the following sections of this chapter. After these automated steps, manual data labeling is conducted for the remaining labels that require human judgment.

During manual data labeling, we encountered 10 labels, which are summarized in Table 7.1. These labels demonstrate the alignment of the text with the corresponding audio content. To create the training and evaluation data set, we select some sentences and audio clips from the full data set such that the labels are equally represented in the training data set. That is to prevent the models from being more inclined to predict the majority label than the minority labels [27].

7.1.2 Audio Segmentation Module

We obtain the audio files of the Quran recitation from various sources on the web. These audio files are recorded over the years for the purpose of listening to the Quran recitations and not for the purpose of computer-aided audio processing. The widely-available audio files contain the recitations of the chapters of the Quran. Such lengthy audio files need to be segmented into smaller audio clips that can be matched and aligned to sentences of the corresponding text.

The goal of the Audio Segmentation Module is to divide the audio file into smaller clips based on detected periods of silence in the audio track. The segmentation process is accomplished through *Voice Activity Detection* (*VAD*) algorithms that detect periods of silence [53]. In this module, we use a segmentation algorithm [125] that distinguishes between silence and non-silence segments by analyzing the amplitude levels. The algorithm dynamically utilizes a variable amplitude threshold value to differentiate silence from non-silence. This

segmentation algorithm is particularly suitable for audio with natural pauses, as observed in Quranic recitation. The output of this module is segmented into small audio clips obtained by splitting long audio files according to periods of silence.

7.1.3 Speech Recognition Module

We utilize speech recognition technologies, specifically the Google Speech-to-Text API, to convert audio files into text only, and we refer to the output as the ASR transcript. The ASR transcript that is generated by current systems is not perfect as it lacks diacritics and contains conversion errors. However, it is considered one step in the direction of converting speech into text. This ASR transcript is then used to create the ground truth and is also sent to the Audio-To-Text Alignment module for additional processing. The workflow of the process is depicted in Figure 1.5.

7.1.4 Annotation Level Reduction Module

The Quranic text is written in annotation fifth-level Arabic, while the ASR transcript is in second-level Arabic. The difference in writing annotation levels poses challenges to comparing these texts. Additionally, the Quranic text follows the Ottoman Drawing Style (ODS) in its writing. This ODS style results in variations in certain characters compared to the standard Arabic writing. To address this, we developed a module that reduces the annotation level for comparisons. The module generates a *comparable text*, a second-level Arabic text derived from the Quranic diacritic-annotated text. By using the *comparable text*, it becomes easier to compare the original diacritic-annotated text (after reducing its annotation level from the fifth level to the second level) with the ASR transcript at the same level of annotation. Text processing methods are employed to adjust the annotation level for improved comparability. The following steps are taken:

- **Tokenization:** The text was divided into individual sentences/verses.
- **Text cleaning:** The diacritical letters are removed and adjustments are made to the Ottoman text to match standard texts.

- **Noise removal:** Unpronounceable signs, such as verse numbering, are removed from the texts for better suitability in comparisons.

Algorithm 7: Annotation Level Reduction Algorithm

```
Input  : ArabicFifthLevelDiacriticText: Diacritic-annotated text at the fifth level
Output: ComparableSentences: List of comparable text reduced to the second level
//Initialization:
ArabicAnnotationLevelMap ← Structure that holds Ottoman Drawing Style (ODS) letters
ArabicDiacriticMarks ← List of Arabic diacritic marks
ODSPatternCleanupExpressions ← List of regular expressions for cleaning ODS text
//Processing:
foreach text in ArabicFifthLevelDiacriticText
    //Step 1: Transform ODS to CA Fifth Level
    foreach character in text
        if character exists in ArabicAnnotationLevelMap then
            Append corresponding CA fifth-level letter to ConvertedCAText
        else
            Append character as it is to ConvertedCAText
    //Step 2: Tokenization
    TokenizedDiacriticSentence ← DiacriticSentenceTokenizer(ConvertedCAText)
    //Step 3: Remove diacritical letters
    foreach mark in ArabicDiacriticMarks
        // Replace all occurrences of mark in TokenizedSentence with an empty string
        TokenizedSentence ← TokenizedDiacriticSentence.replace(mark, '')
    //Step 4: Noise removal
    foreach pattern in ODSPatternCleanupExpressions
        // Remove all matches of pattern from TokenizedSentence using regular expressions
        ComparableSentence ← RegExp.Replace(pattern, '', TokenizedSentence)
    Append ComparableSentence to ComparableSentences
Output:
        Return ComparableSentences
```

The algorithm 7 addresses the challenge of comparing diacritic-annotated Quranic text written in annotation fifth-level Arabic with ASR transcripts written in second-level Arabic. To facilitate this comparison, a module has been developed that reduces the annotation level

for better comparability. The algorithm consists of four steps.

Step 1: Transforming ODS to Classical Arabic Fifth Level In this step, the algorithm converts the Quranic text from Ottoman Drawing Style (ODS) to Classical Arabic (CA) fifth level. The algorithm iterates through each character in the input text and checks if it exists in the ArabicAnnotationLevelMap, which holds the mapping between ODS letters and CA fifth-level letters. If a match is found, the corresponding CA fifth-level letter is appended to the ConvertedCAText. Otherwise, the character is appended as it is. This process ensures that the text is transformed to the desired annotation level.

Step 2: Tokenization The algorithm tokenizes the diacritic-annotated text into individual sentences/verses. It uses the DiacriticSentenceTokenizer function to split the text into tokenized diacritic sentences, which are stored in the TokenizedDiacriticSentence list.

Step 3: Removing Diacritical Letters In this step, the algorithm removes diacritical letters from the tokenized sentences. It iterates through each sentence in the TokenizedDiacriticSentence list and removes any characters present in the ArabicDiacriticMarks list. This ensures that only the base letters remain, making the text suitable for comparison. The resulting cleaned sentences are stored in the TokenizedSentence list.

Step 4: Noise Removal The algorithm performs noise removal to improve the suitability of the text for comparison. It iterates through each sentence in the TokenizedSentence list and applies a series of regular expression-based cleanup patterns. These patterns target specific noise elements present in the text, such as verse numbering, and remove them. The cleaned sentences are stored in the ComparableSentence list.

Finally, the algorithm outputs the ComparableSentences list, which contains the reduced, cleaned, and comparable text in the second-level Arabic format. This output can be easily compared with the ASR transcript for further analysis or evaluation.

7.1.5 Trained Classifier

The classification process categorizes and labels data. It analyzes alignment patterns between the ASR transcript and comparable text. The ASR transcript represents written speech, while the comparable text represents diacritic-annotated text in simple form. Ultimately, the classification is applied to the audio and diacritic annotated text, leveraging the alignment patterns between the ASR transcript and the comparable text.To pair the text with the ASR transcript, we take the following steps:

- FuzTPI processes the ASR transcript and comparable Arabic text as inputs (Section 7.2.1).
- FuzTPI generates fuzzy numbers that measure how well different parts of the texts match, such as the beginning, middle, and end, as well as other aspects.
- These fuzzy numbers indicate where the corresponding text is located in the ASR transcript and show how the two texts align.
- The fuzzy numbers are fed into the ML model, which produces labeling predictions based on these inputs (Section 7.3).
- This integration ensures that the ML model focuses only on fuzzy-number patterns, making it robust to unexpected errors in the ASR transcript.

This process allows the classifier to identify the corresponding text for the ASR transcript. We aim for the classifier model to learn from the provided data, including the erroneous text, and make predictions based on it. Hence, we refrained from correcting the linguistic errors present in the ASR transcript. We deliberately chose not to correct ASR transcript errors for three key reasons:

1. **Realistic Simulation:** ASR is a fixed module within our system. Current ASRs inherently produce errors in recognizing Arabic speech. To ensure real-world effectiveness, we must test the system in a mirror environment.
2. **Data Consistency:** Correcting errors would yield misleading accuracy. Our system

must process imperfect ASR outputs.

3. **Robustness:** Training and testing on uncorrected transcripts develops a robust system, capable of handling real-world speech recognition imperfections.

This approach assesses the system's true performance and adaptability to current ASR challenges in Arabic speech recognition. The classifier is expected to account for the presence of errors and still perform effectively in our QRSR system. Table 7.2 provides examples of challenges encountered during the classification process caused by the ASR errors.

Table 7.2: Some Cases that Reflect the Challenges in Text Classification.

Text (Diacritics omitted)	**ASR transcript**	**Class**
أوءاباؤنا الأولون	اواب جاءنا الاولون	1
Note: Fragmented and misspelled words make the text difficult to detect.		
قل إنما أنا بشر مثلكم يوحى إلي أنما إلهكم إله وحد فاستقيموا إليه واستغفروه وويل للمشركين	بس تلقيم اليه واستغفروه	6
Note: The text contains both correct and incorrect parts, and it is essential to clarify that it does not represent the beginning or end of the full text.		
لهم من فوقهم ظلل من النار ومن تحتهم ظلل ذلك يخوف الله به عباده يعباد فاتقون	ياعبادي فتتكون	7
Note: The text lacks coherence, even though an Arabic speaker might expect it at the end of the text.		

During the inference phase, the data is obtained from the comparable text from the Annotation Level Reduction Module and the ASR transcript from the Speech Recognition Module. Data is classified based on the labels provided in table 7.1. The first five labels in Table 7.1 demonstrate an alignment relationship between the audio and text, showcasing different types of alignment. These labels represent various alignment relations, such as the full audio file corresponding to a full verse (FAFV) and the verse being a part of the audio file (VPOA), among others. Figure 7.6 provides a visual representation of the first five labels. The last five labels in Table 7.1 demonstrate different cases for an unalignment of the text and the

ASR transcript. These labels involve cases like audio content not recognized (ANR), audio not corresponding to a verse (ANAV), and others.

Two significant label types may be absent based on the preceding discussion. Nonetheless, the subsequent descriptions illustrate how these cases are inherently accounted for.

1. A complete verse at the beginning with other verses after it: This case is handled using the VersePartOfAudio label:
    - The systems recognizes that the audio contains more than one verse.
    - It moves to the next text without skipping the audio.
    - The text for the full verse and any following verses is saved with the corresponding audio clip.
    - The system then moves to the next audio clip.
2. A verse in the middle with other verses: This case is also handled using the VersePartOfAudio label.
    - The system retains the text from the previous verse.
    - It adds the text of the current verse to the ongoing transcription.
    - It continues processing the same audio clip, adding each verse sequentially.
    - When the clip ends, the full text for the clip is saved alongside the audio.
    - The system then moves to the next audio clip.

7.1.6 Audio-to-Text Alignment Module

The module has two purposes: aligning Arabic speech to diacritic-annotated text and preparing a dataset for enhancing Arabic Speech Recognition. The process begins with a trained classifier. This classifier matches audio files to their corresponding text. The classifier sends these audio-text pairs to the alignment algorithm. The inputs for the alignment algorithm include the ASR transcript, the comparable text, and the predicted labels. These inputs are provided by the trained classifier. The Alignment Module retrieves the actual audio files cor-

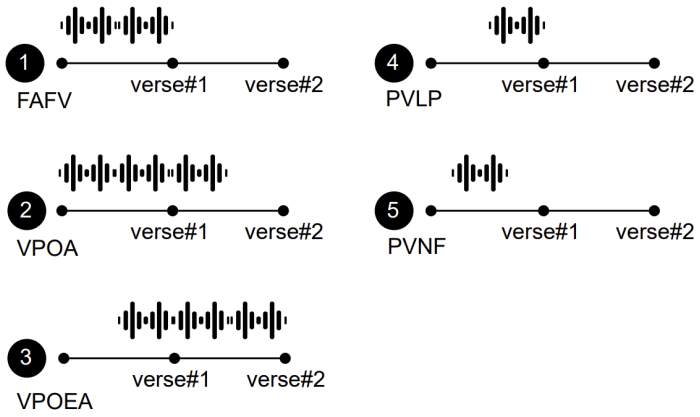


Figure 7.6: Visual representation of alignment relationships between audio and text.

responding to the ASR transcript. It also retrieves the Arabic diacritical text corresponding to the comparable text. If necessary, the audio files are merged to match a full sentence. The alignment algorithm processes these classified audio-text pairs systematically. Finally, the module combines the audio files with their corresponding diacritical text. This process results in a comprehensive diacritical Arabic audio-text dataset. The final dataset includes precisely matched audio files and their corresponding diacritical texts.

Algorithm 8: Audio-to-Text Alignment Algorithm

Inputs : ClassifiedAudioTextPairs

Output: Diacritic Arabic audio-text dataset

```
Initialize empty lists: audio_list, text_list
foreach pair in ClassifiedAudioTextPairs do
    label ← pair.label
    ASRTranscript ← pair.ASRTranscript
    ComparableText ← pair.ComparableText
    audio_files ← fetchAudioFiles(ASRTranscript)
    diacritical_text ← fetchDiacriticalText(ComparableText)
    push diacritical_text into text_list
    combined_file ← combineAudioFiles(audio_files, label)
    push combined_file into audio_list
Return mergeAudioAndText(audio_list, text_list)
```

The Audio-to-Text Alignment Algorithm 8 consists of several steps to collect system inputs,

align audio and text, and generate the final output.

Step 1: Algorithm Inputs The algorithm takes one input: *ClassifiedAudioTextPairs*, where each pair includes the ASR transcript, comparable text, and the label assigned by the classifier. These inputs serve as the foundation for aligning audio and text.

Step 2: Retrieving Audio Files and Diacritical Text The algorithm retrieves the actual audio files corresponding to each *ASRTranscript* by calling the *fetchAudioFiles* function. Similarly, it retrieves the Arabic diacritical text corresponding to each *ComparableText* by calling the *fetchDiacriticalText* function.

Step 3: Building Audio and Text Lists The algorithm initializes two empty lists: *audio_list* and *text_list*. For each pair of *ASRTranscript* and *ComparableText*, it pushes the corresponding diacritical text into the *text_list*. Similarly, it combines the audio files by calling the *combineAudioFiles* function, which merges the audio files if necessary. The label is an input for the *combineAudioFiles* function with the *audio_files*. This function merges audio files in specific cases. It does so when the label indicates that the audio is part of the text, not the other way around. This merging occurs in only two cases: PVLP (PartialVerseLastPartRecited) and PVNF (PartialVerseNotFinished). Then, pushing the resulting *combined_file* into the *audio_list*.

Step 4: Generating the Diacritic Arabic Audio-Text Dataset The algorithm returns the diacritic Arabic audio-text dataset by calling the *mergeAudioAndText* function, which takes the *audio_list* and *text_list* as inputs. This function combines the audio files with their corresponding diacritical text, resulting in a diacritic Arabic audio-text dataset ready for further analysis, training, or evaluation. By aligning audio and text, the algorithm facilitates various downstream tasks and applications, such as training speech recognition models, improving language understanding, and enabling audio search and retrieval.

7.2 Fuzzy Text Alignment and Rule-based Classifier (FTARC)

To overcome the changes that faced the ML models in our problem, we have developed a Fuzzy Text Alignment and Rule-based Classifier (FTARC). FTARC is composed of two algorithms. The first algorithm is a Fuzzy Text Position Inference (FuzTPI) Algorithm 9. This algorithm utilizes fuzzy matching techniques to generate numerical indicators. These numerical indicators provide insights into alignment patterns and the presence of text segments within each other. The second algorithm is a rule-based classifier, which leveraging from the Fuzzy numerical indicators to classify the texts. We aim to improve the accuracy of the classification process by utilizing FTARC.

In this section, we will discuss two key components: the FuzTPI Algorithm and the Rule-based Classifier. We will explore how these algorithms contribute to the task at hand. Furthermore, we will present and analyze the results obtained from FTARC.

7.2.1 Fuzzy Text Position Inference (FuzTPI) Algorithm

The proposed FuzTPI algorithm facilitates the comparison of two texts of varying length and content by leveraging fuzzy matching techniques [40, 88]. The algorithm provides numerical indicators that reflect the text alignment patterns. This algorithm depends on two main functions to determine these numerical indicators, fuzzy full-text matching and fuzzy partial-text matching.

The *fuzzy full-text matching* function, denoted as `Fuzzy(longerText, shorterText)`, can be defined as equation(7.1). This equation calculates the similarity between a longer text and a shorter text [111]. It considers every possible pair of characters between the two texts and computes the membership value (μ) indicating their degree of similarity. The numerator sums up these membership values for all character pairs, capturing the overall similarity. The denominator normalizes the result by dividing the numerator by the maximum length between the longer and shorter texts. This normalization accounts for differences in text length when comparing them.

The *fuzzy partial-text matching* function, denoted as `PartialFuzzy(longerText, shorterText)`, can be defined as equation(7.2). This equation focuses on the similarity between a longer text and a shorter text when the shorter text is a part of the longer text. Similar to the equation(7.1), it calculates the membership value (μ) for each pair of characters between the longer and shorter texts. The numerator sums up these membership values, indicating the overall similarity. However, in this case, the denominator is the length of the shorter text itself. This normalization allows for a direct comparison of the similarity between the shorter text and the corresponding part of the longer text [111].

$$\text{Fuzzy}(\text{longerText}, \text{shorterText}) = \frac{\sum_{i=1}^{|\text{shorterText}|} \sum_{j=1}^{|\text{longerText}|} \mu(\text{longerText}[j], \text{shorterText}[i])}{\max(|\text{longerText}|, |\text{shorterText}|)} \tag{7.1}$$

$$\text{PartialFuzzy}(\text{longerText}, \text{shorterText}) = \frac{\sum_{i=1}^{|\text{shorterText}|} \sum_{j=1}^{|\text{longerText}|} \mu(\text{longerText}[j], \text{shorterText}[i])}{|\text{shorterText}|} \tag{7.2}$$

where:

- `longerText`$[j]$ represents the j-th character in `longerText`.
- `shorterText`$[i]$ represents the i-th character in `shorterText`.
- $\mu($`longerText`$[j],$ `shorterText`$[i])$ represents the membership value indicating the degree of similarity between `longerText`$[j]$ and `shorterText`$[i]$.
- $|$`longerText`$|$ represents the total number of characters in `longerText`.
- $|$`shorterText`$|$ represents the total number of characters in `shorterText`.

By using these equations(7.1, 7.2), the FuzTPI 9 provides numerical indicators that reflect the alignment patterns between texts, capturing the degree of similarity between them. Initially, the algorithm identifies the longest text and determines the alignment indicator, which determines whether the text is part of the audio or vice versa. The algorithm measures the matching degree of the shorter text at the beginning, end, and middle of the longer text,

Algorithm 9: Fuzzy Text Position Inference Algorithm.

Input: ASRTxt, ArTxt

Specify the alignment indicator
    alignmentIndicator ← (length(text1) >length(text2)) ? 1:2
Determine the longest and shortest text
    longerText ← (length(ASRTxt) >length(ArTxt)) ? ASRTxt : ArTxt
    shorterText ← (length(ASRTxt) >length(ArTxt)) ? ArTxt : ASRTxt
End-of-text matching ratio
    shortenedLongerText ← longerText[length(shorterText):]
    EndOfTextMatching ← `Fuzzy(`*shortenedLongerText, shorterText*`)`
Matching percentage of the beginning of the text
    shortenedLongerText ← longerText[:-length(shorterText)]
    BeginOfTextMatching ← `Fuzzy(`*shortenedLongerText, shorterText*`)`
Middle text matching ratio
    trimmedLongerText ← CutToMatchTheMiddle(longText, length(shorterText))
    MiddleOfTextMatching ← `Fuzzy(`*trimmedLongerText, shorterText*`)`
Last Text Part in ASR Transcript Verification
    lastPart ← " ".join(ArTxt.split()[-n:]) or ArTxt
    IsLastPartInASR ← `PartialFuzzy` (lastPart, ASRTxt) >threshold
Measure fuzzy text matching
    FuzzyTextMatching ← `Fuzzy(`*ASRTxt, ArTxt*`)`
Measure fuzzy partial matching
    FuzzyPartialMatching ← `PartialFuzzy(`*longerText, shorterText*`)`
Count occurrences of the short text in the long text
    OccurrencesCount ← `CountOccurrences(`*longerText, shorterText*`)`
Return fuzzy numbers
    return *All fuzzy numbers that have been calculated*

providing insights into the relative positioning of the texts. Furthermore, the algorithm determines the full and partial matching degree between the two texts. Additionally, the algorithm counts the number of occurrences of the shorter text within the longer text, providing further information about their presence and potential repetitions. By offering these insights the fuzzy algorithm equips the classifier with essential alignment data.

7.2.2 The Rule-based Classifier

The rule-based classifier starts by retrieving a list of Arabic sentences, along with their corresponding audio files and ASR transcripts. The system then initializes relevant variables and iterates through the lists. To ensure consistency, it applies text normalization techniques to the input texts. Additionally, it utilizes the numerical outputs generated by the Fuzzy Algorithms 9. Based on predefined cases outlined in Table 7.3, the rule-based classifier classifies the data using these numbers.

Table 7.3: Rule-Based Conditions and Return Values.

Condition	Return Value
ASRText is None	ANR
FuzzyFullTextMatching > Threshold	FAFV
FuzzyPartialMatching > Threshold and alignmentIndicator == 1 and IsLastPartInASR	VPOEA
FuzzyPartialMatching > Threshold and alignmentIndicator == 1 and not IsLastPartInASR	VSNF
FuzzyPartialMatching > Threshold and alignmentIndicator == 2 and IsLastPartInASR	PVLP
FuzzyPartialMatching > Threshold and alignmentIndicator == 2 and not IsLastPartInASR	PVNF
IsDataClassifiedBefore and NextTextFuzzyPartialMatching > Threshold	Uncertain
IsDataClassifiedBefore and PreTextFuzzyPartialMatching > Threshold	StepBackPreText()
IsDataClassifiedBefore and PreASRFuzzyPartialMatching > Threshold	StepBackPreASR()
IsDataClassifiedBefore and IsPreASRPartOfText	Uncertain
IsDataClassifiedBefore and PreTextFuzzyPartialMatching > Threshold and IsLastPartInASR	PVLP
IsDataClassifiedBefore and PreTextFuzzyPartialMatching > Threshold and not IsLastPartInASR	PVNF
IsDataClassifiedBefore and PreTextFuzzyPartialMatching < Threshold	IncreaseSkippedASRCount(), UA
not IsDataClassifiedBefore and FuzzyPartialMatching < Threshold	IncreaseSkippedASRCount(), ANAV
IsASRLimitExceeded	MA

The rule-based classifier, enhances the classification process, enabling the generation of aligned diacritized Arabic texts and audio files. The purpose of this classifier is to assign appropriate classifications to the data based on predefined rules and numerical fuzzy outputs generated by the FuzTPI Algorithm 9.

Initially, the classifier retrieves a list of Arabic sentences along with their corresponding audio files and ASR transcripts. To ensure consistency, the system applies text normalization techniques to the input texts. The first set of conditions on table 7.3 focuses on evaluating the availability of the ASR transcript and the degree of similarity between the two texts. If the ASR transcript is not available or the full text and transcript exhibit a high degree of similarity, the classifier returns specific labels indicating the alignment status. Subsequent conditions take into account the alignment of partial text, considering factors such as the alignment indicator and the presence of the last part of the text in the transcript. Additional conditions build upon the previous classifications and consider the degree of similarity between adjacent texts and ASR transcripts. These conditions aim to capture nuances in the alignment process and ensure accurate classification. The final set of conditions handles cases where the ASR transcript is ignored beyond predefined limits or when the alignment falls below the specified threshold. These conditions further refine the classification process, accounting for specific alignment scenarios. Based on the classifications received from the rule-based classifier, the system merges the audio files into complete sentences that align with the list of Arabic sentences. As a result, the system generates a structured output consisting of diacritized Arabic texts accompanied by their corresponding audio files. The threshold used to accept the agreement between ASR transcripts and Arabic text plays an important role in the FTARC algorithm. Through experimentation on our dataset, we observed that different threshold values affect the alignment accuracy differently. When the threshold is set above 80%, the algorithm tends to lose sentences that are actually almost complete and similar. This is due to ASR transcript errors, causing the convergence percentage to fall below 80%. On the other hand, when the threshold is set at 70% or lower, the algorithm starts to agree on similar sentences, but some words are missing, leading to incomplete alignments.

After conducting extensive experiments, we found that a threshold of 75% strikes the right balance between capturing similar sentences and maintaining completeness.

7.2.3 FTARC Performance

The performance evaluation of the FTARC showed an accuracy of 90.77%, precision of 91.96%, recall of 90.77%, and an F1 score of 90.59%. The high values for these metrics indicate that the FTARC achieved a good level of accuracy in its classification results when compared to the manually entered results.

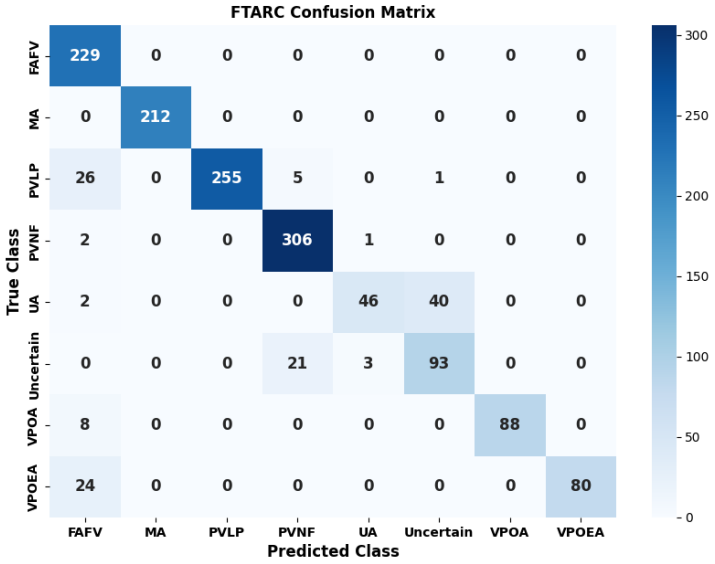


Figure 7.7: Confusion Matrix of FTARC Algorithm.

Analyzing the confusion matrix in figure 7.7, it is evident that the FTARC classifier exhibited accuracy in recognizing the majority of the classes (which appeared in the diagonal elements). It demonstrated moderate accuracy in recognizing UA. This result reflects its accuracy, which is 90.7%. These findings highlight the algorithm's overall success in classification, while also indicating potential areas for improvement in accurately distinguishing between certain classifications.

7.3 Fusion of FuzTPI Algorithm and Machine Learning Models for Improved Arabic Speech Classification

FTARC, the Fuzzy Textual Alignment Rule-based Classifier, emerged as the top-performing model with an accuracy exceeding 90%. The high accuracy of FTARC is attributed to the FuzTPI Algorithm, which effectively addresses typographical errors in the Arabic ASR transcript by generating fuzzy numbers that indicate the degree of text present in the second text. While ML models outshine rule-based models in their generalizability and applicability across different classification tasks, rule-based models offer domain-specific solutions based on predefined rules and conditions. In this section, we aim to leverage the strengths of both approaches by integrating the FuzTPI Algorithm with ML models. This integration involves generating fuzzy numbers using FuzTPI for the textual data and allowing ML models to learn from these fuzzy numbers and their corresponding taxonomy, bypassing the need to directly handle the ASR transcript with its errors. Figure 7.8 illustrates the integration of the FuzTPI algorithm with the ML model. FuzTPI takes the ASR transcript and Arabic comparable text as input, generating fuzzy numbers as output. The ML model solely uses fuzzy numbers as input and produces labeling predictions. This integration helps safeguard the ML model from unexpected errors in the Arabic ASR transcript. Integration is done through a method for serializing and deserializing called pickle [171]. This means the FuzTPI is "bundled" with the ML model in a way that it can be deployed as one unit. By integrating FuzTPI with the ML model, we eliminate the need to program or manage the fuzzy inference engine separately. Everything is handled within the FuzTPI-ML Integration Unit. Text classification has gained significant attention due to using it in a wide range of applications [102, 112, 7]. In the classification task, ML models are widely utilized in various applications [112, 7]. Popular ML models for text classification tasks such as Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF). In this part, we will present the results of applying the NB, SVM, and RF models when they learn directly from the output of FuzTPI. By examining the performance of these ML models with fuzzy numbers as input, we aim to assess their effectiveness in enhancing the alignment-driven classification of Arabic speech.

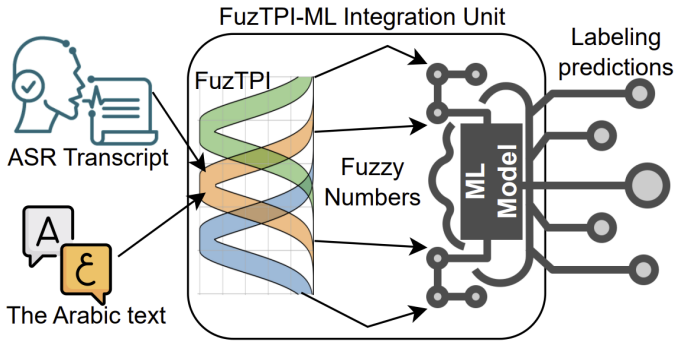


Figure 7.8: Integration of FuzTPI with ML Model.

7.3.1 FuzTPI-Driven Naive Bayes

Naïve Bayes is a widely used and versatile classification algorithm known for its simplicity and efficiency [7]. By applying Bayes' theorem, it determines the most probable class for a given instance, making it particularly valuable in scenarios with limited training data [78]. We utilize the MultinomialNB variant of Naive Bayes for text classification tasks [191]. MultinomialNB is particularly effective in handling discrete features such as word frequencies and counts. In our algorithm, we further optimize the performance of MultinomialNB by employing GridSearchCV. The GridSearch is a technique that systematically explores the hyperparameter space to identify the best combination of parameters for the classifier [32]. This allows us to enhance the accuracy and effectiveness of the text classification process.

FuzTPI-Driven Naive Bayes Performance:

The Naïve Bayes is trained on the data generated from FuzTPI. The model performance is evaluated on the testing dataset, and it achieved the performance measures: Accuracy Ratio of 55.13%, Precision of 49.24%, Recall of 55.13%, and F1 Score of 50.55%. These metrics assess the algorithm's classification performance across our labels.

The confusion matrix in Figure 7.9 reveals that the FuzTPI-NB classifier struggles to accurately recognize the "Uncertain" label, as indicated by the scattered predictions across the matrix. Additionally, it completely fails to distinguish the MA class, classifying all instances as PVNF. However, the classifier demonstrates relatively higher accuracy in identifying the

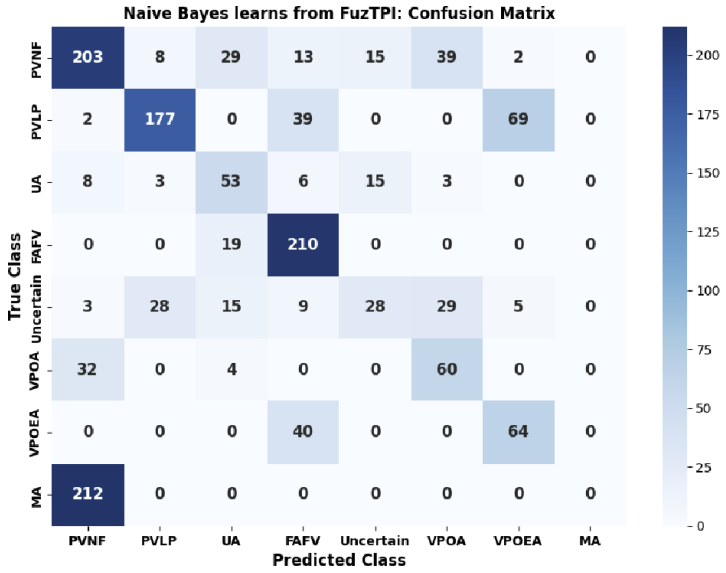


Figure 7.9: Confusion Matrix of FuzTPI-Driven Naive Bayes Algorithm.

remaining classes compared to the average performance. Overall, the FuzTPI-NB classifier achieves an average accuracy of 55%.

7.3.2 FuzTPI-Driven SVM

SVM is a widely used supervised ML algorithm for classification tasks. The main objective of SVM is to maximize the margin between different categories, creating a clear separation between them [7, 112]. SVM identifies support vectors, which play a crucial role in determining the boundaries between classes [112]. SVM can be applied to features derived from textual data, such as word frequencies, n-grams, or other linguistic characteristics. Therefore, we chose SVM to apply to the problem of Arabic diacritic-annotated text. SVM performs well on datasets of small to medium sizes [51] and is particularly suitable for datasets with a moderate number of samples. We utilized TF-IDF (Term Frequency-Inverse Document Frequency) vectorization [98], to transform the text data into numerical features, allowing for analysis and modeling using SVM [117].

FuzTPI-Driven SVM Performance:

The SVM is trained on the data generated from FuzTPI. The mode performance is evaluated

on the testing dataset, and it achieved the performance measures: Accuracy Ratio of 95.42%, Precision of 95.78%, Recall of 95.42%, and F1 Score of 95.51%. These metrics assess the algorithm's classification performance across our labels.

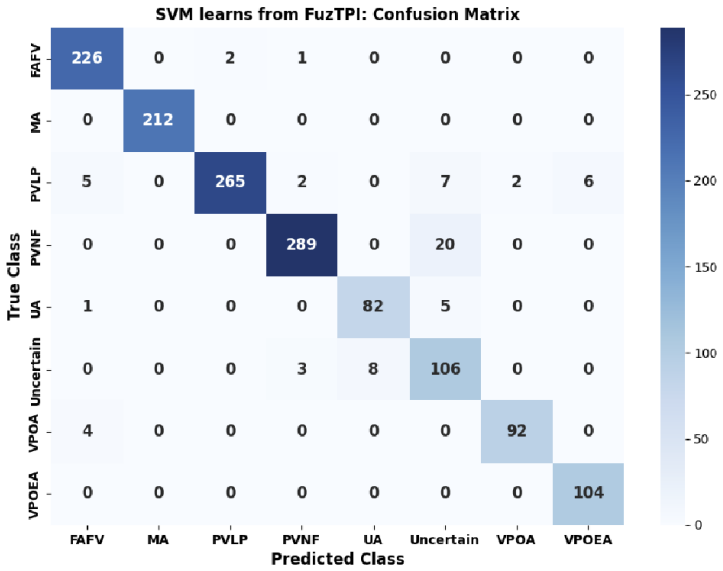


Figure 7.10: Confusion Matrix of FuzTPI-Driven SVM Algorithm.

The confusion matrix of the FuzTPI-SVM classifier in figure 7.10 reveals a distinct diagonal line, indicating the high accuracy of its predictions. However, the classifier encounters challenges in recognizing the "Uncertain" class. It also shows some errors in correctly identifying the PVNF class. On the other hand, the classifier performs well in recognizing the remaining classes, resulting in an accuracy rate exceeding 95.4%.

7.3.3 FuzTPI-Driven Random Forest

The Random Forest algorithm is a powerful learning algorithm widely used for classification tasks. It leverages the predictions of multiple decision trees to enhance accuracy [112]. We implemented the Random Forest algorithm, and employed grid search, a technique that systematically explores different parameter combinations. Through grid search, we identified the best parameters that resulted in the optimal configuration for the Random Forest algorithm [151].

FuzTPI-Driven RF Performance:

The RF is trained on the data generated from FuzTPI. The mode performance is evaluated on the testing dataset, and it achieved the performance measures: Accuracy Ratio of 95.90%, Precision of 96.51%, Recall of 95.90%, and F1 Score of 96.02%. These metrics assess the algorithm's classification performance across our labels.

The confusion matrix of the FuzTPI-RF classifier exhibits a diagonal line pattern, indicating its high accuracy in classification. However, the classifier encounters challenges in correctly recognizing the "Uncertain" class, as shown in figure 7.11. Additionally, it demonstrates some errors in identifying the PVNF and PVLP classes. Nevertheless, the classifier performs well in recognizing the remaining classes, resulting in an accuracy rate approaching 95.9%.

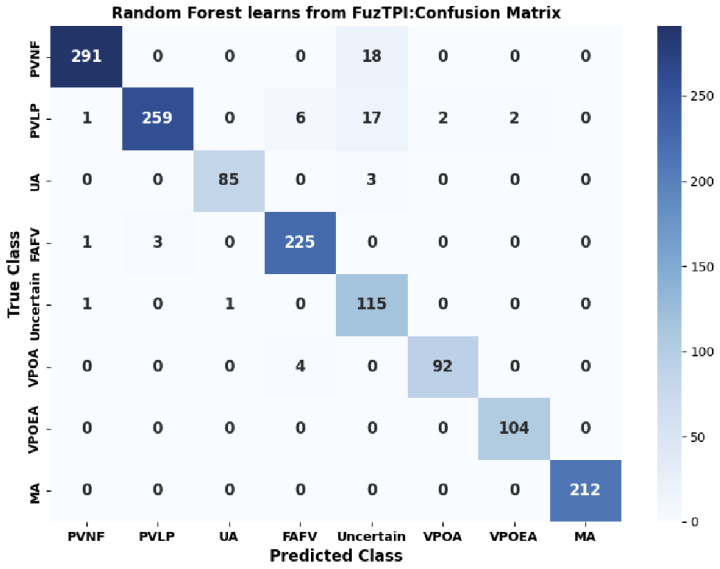


Figure 7.11: Confusion Matrix of FuzTPI-Driven RF Algorithm.

7.4 Results and Discussion

This section presents the performance evaluation results and a discussion of the implemented classifiers. We compare the classifiers, analyze their performance, and discuss the implications of our findings. Additionally, the performance metrics of the Alignment process.

Analyzing Performance Measures: We present an analysis of the performance measures

according to the testing dataset, including Accuracy, Precision, Recall, and F1 Score, for a range of models used in Arabic speech classification. Analyzing the table 7.4, we observe variations in the performance measures across the different models.

Classifier	Accuracy	Precision	Recall	F1 Score
FuzTPI-NB	55.13%	49.24%	55.13%	50.54%
FTARC	90.78%	91.97%	90.78%	90.60%
FuzTPI-SVM	95.42%	95.78%	95.42%	95.51%
FuzTPI-RF	**95.90%**	**96.51**	**95.90.16%**	**96.02%**

Table 7.4: Performance Comparison of Arabic Speech Classifiers

The FuzTPI-NB achieved an accuracy of 55.13%. The FTARC classifier achieved an accuracy of 90.78% and demonstrated high precision, recall, and F1 Score. The FuzTPI-SVM model exhibited further improvement, with an accuracy of 95.42% and high precision, recall, and F1 Score. Notably, the FuzTPI-RF model showcased the highest performance among all classifiers, achieving an impressive accuracy of 95.90%. It also displayed superior precision, recall, and an F1 Score of 96.02, indicating its robustness in accurately classifying Arabic speech data. Note that the results of the FuzTPI-SVM and the FuzTPI-RF models are very close, so there is no clear winner model between them.

The accompanying figure, Figure 7.12, visually represents the comparative analysis of classifier performance. The bar chart reinforces the observations made in the table, clearly illustrating the variations in performance measures among the models. Overall, the results highlight the effectiveness of the FTARC, FuzTPI-RF, and FuzTPI-SVM models in improving the classification accuracy of Arabic speech. These models leverage advanced techniques and algorithms to enhance the alignment-driven classification process, resulting in superior performance compared to traditional ML classifiers.

Class Difficulty Analysis:

In the analysis of classifiers' performance, it is important to take into consideration the challenging classifications encountered. This information is presented in Figure 7.13, which provides insights into the number of correct and incorrect predictions made for each class

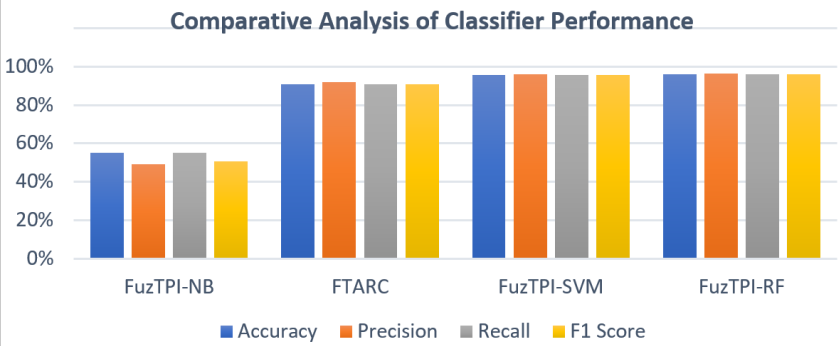


Figure 7.12: Comparative Analysis of Classifier Performance.

across all utilized classifiers.

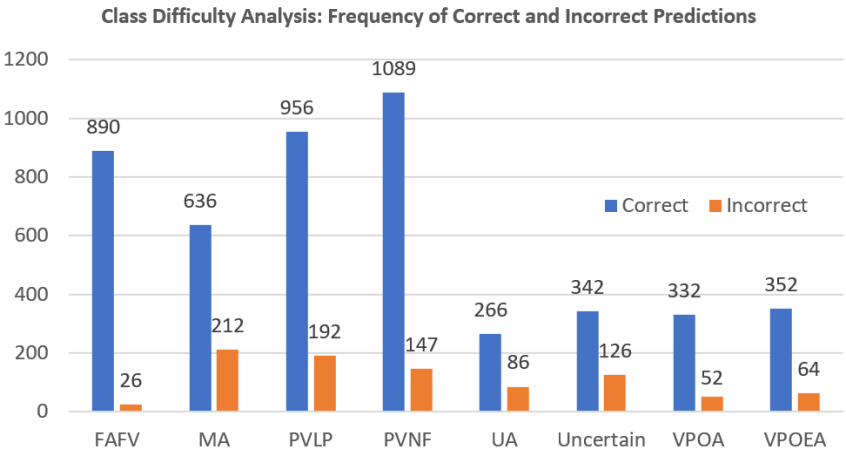


Figure 7.13: Class Difficulty Analysis: Frequency of Correct and Incorrect Predictions.

Among the classes, FAFV, PVLP, and PVNF stand out as the easiest to classify, as they have a high frequency of correct predictions and a low frequency of incorrect predictions. This suggests that these classes possess distinct features that enable accurate classification by the classifiers. On the other hand, the UA, and Uncertain classes presented significant challenges, resulting in a higher number of incorrect predictions compared to correct predictions. These classes are closely related to the texts with exhibit limited similarity. For MA, after reviewing the data, it is found that the incorrect predictions are all from FuzTPI-NB, and this is a point to improve the performance of this model. Enhancing the detection and classification of these classes could lead to substantial improvements in overall performance.

Classifier	FuzTPI-NB	FTARC	FuzTPI-SVM	FuzTPI-RF
Correct	795	1309	1376	**1383**
Incorrect	647	133	66	**59**

Table 7.5: Comparison of Successful and Unsuccessful Predictions by Classifier.

Analyzing Successful and Unsuccessful Classifications:

An analysis is conducted to assess the success and failure rates of the classifiers. Figure 7.14 provides clear visualizations of the distribution of successful and unsuccessful predictions.

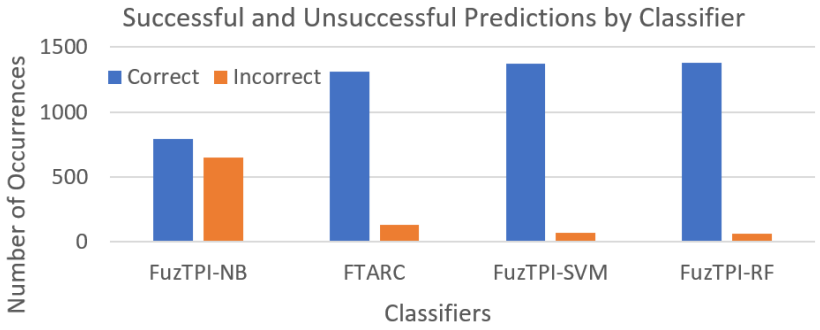


Figure 7.14: Graphical Analysis: Successful and Unsuccessful Classifications.

Based on the data in Table 7.5, we can observe that the FuzTPI-RF classifier achieved the highest number of correct predictions with a total of 1,383. It is followed by the FuzTPI-SVM classifier with 1,376 correct predictions and the FTARC classifier with 1,309 correct predictions. The lowest number of correct predictions recorded by the NB classifier with 795.

Audio-Text Alignment Performance:

As in 7.4 of System Architecture, after obtaining the best Trained Classifier, we move to the last stage, which is Audio-to-Text Alignment. The Alignment module takes input from the winner-trained classifier, in our case FuzTPI-RF. Based on these classifications, it begins to align the audio files against the texts, potentially combining audio files to form complete verses. The final audio file is then linked to its Arabic diacritic-annotated text.

This module is based on the algorithm 8. To evaluate it, a dataset consisting of 1550 audio files is created. Each file is annotated to identify whether it corresponds to a complete verse, a part requiring merging, or the end of a verse, etc. This is essential to measure the

accuracy of the algorithm and facilitate error tracking, improvement, and measurement. In the alignment process, the QRSR system achieved an accuracy of 95.61%, with an alignment count of 1482 correct states and 68 incorrect states. The detailed performance measures for alignment are summarized in Table 7.6.

Table 7.6: Performance Measures for Alignment

Alignment Performance	Accuracy	Precision	Recall	F1 Score
Value	95.61%	86.88%	96.42%	90.80%

The table 7.6 illustrates the overall effectiveness of the alignment module. An accuracy of 95.61% indicates that the vast majority of audio files are correctly aligned. The high recall value of 96.42% suggests that most of the relevant audio files are correctly identified, while the precision of 86.88% shows the proportion of true positive alignments among the total positive alignments. The F1 score, a harmonic mean of precision and recall, provides a comprehensive view of the model's balance between precision and recall.

7.5 Summary and Future Directions

This section of the dissertation presented the QRSR system, its objectives, and the challenges it addresses. We explored several ML algorithms and proposed the FTARC approach to enhance Arabic speech recognition. The evaluation results and discussion shed light on the performance of the implemented classifiers, identified challenging classifications, and highlighted the potential of incorporating the FTARC algorithm into other ML techniques. The study introduced the Fuzzy Text Alignment and Rule-based Classifier (FTARC) approach is proposed, achieving an accuracy rate of up to 90% in segmenting and aligning audio files. We combined the FuzTPI algorithm and Machine Learning (ML) models. This combination led to audio segmentation, text-audio classification, which achieved up to 96% accuracy, and text-audio alignment, which achieved 95.61%. The research used different classifiers, including Naïve Bayes, Support Vector Machine (SVM), and Random Forest. Among them, the FuzTPI-Random Forest model had the highest performance. The results between FuzTPI-

SVM and FuzTPI-RF are close, indicating both models were effective.

This research contributes to the advancement of Arabic NLP systems, Arabic speech recognition systems, and segmentation/localization techniques, particularly in the context of Quranic studies. Our research contributes also to the development of an expanded textual audio dataset that can have a broader impact on Arabic speech recognition systems.

Chapter 8

A WORD-LEVEL SEGMENTATION AND ALIGNMENT MODEL FOR ARABIC DIACRITIC-BASED ANALYSIS

The Arabic language features a diverse array of linguistic styles. Each style expresses a dialect and may carry additional meanings and clarifications to the text. Unlike the dialects, which might contain linguistic errors, these styles are free from such inaccuracies and often enhance the text's meaning. To the best of the authors' knowledge, no dataset other than the Qur'an has a single Arabic text written more than once in multiple linguistic styles. The Qur'an is unique in containing the same text in 20 linguistic styles, all authenticated by linguists. These texts maintain identical letter shapes but differ in diacritic marks, altering pronunciation and sometimes clarifying additional meanings.

The gap in resources poses a significant challenge in analyzing these variations comprehensively. While audio recordings exist for all 20 styles, only seven currently have corresponding digitized text representations. In the field of Natural Language Processing (NLP), Automatic Speech Recognition (ASR) is a technology that translates spoken language into written text. Developing an ASR system for Arabic that accommodates diacritics could bridge this gap. While various models have achieved success in Arabic speech recognition, they often overlook the importance of diacritics in the language. Arabic diacritics are used to clarify pronunciation, maintain the accent, and define intended meanings. Without diacritics, many words remain ambiguous or carry multiple potential meanings. Unfortunately, leading ASR platforms like Google STT [45], Microsoft Azure Speech Services [133], and Amazon Transcribe [166] typically do not support Arabic diacritics.

Our related work identified a scarcity of Arabic datasets annotated with diacritics at the word level [7, 175]. Most available audio datasets are at the sentence level, which limits

training effectiveness for an ASR model. Focusing on individual words allows models to learn the specific phonetic patterns associated with each word and its diacritics [202]. This lack of suitable datasets motivated our pursuit of a solution.

Audio segmentation and audio-to-text alignment are fundamental steps in ASR processes [172]. Audio segmentation involves dividing the audio track into smaller units such as phonemes, sentences, and words. The audio-to-text alignment synchronizes audio files with their corresponding textual representations. This synchronization ensures the creation of accurate and reliable audio-text datasets.

This research introduces the Diacritic-Aware Segmentation and Alignment Model for Arabic (DASAM). DASAM transcends basic dataset generation by establishing a method for segmenting and aligning audio recordings with their corresponding written texts. This chapter builds on the work presented in Chapter 7 on the QRSR system. The key difference lies in their targets. QRSR associates diacritic-rich Arabic text with sentence-level audio clips. In contrast, DASAM determines the precise start and end times for each word in the audio and aligns it with its corresponding diacritic-annotated text. This change in target level—from sentence to word—requires different internal operations. QRSR employs a combination of speech recognition tools and machine learning models to align audio with text at the sentence level. DASAM, on the other hand, relies on signal processing, linguistic rules, and dynamic time warping (DTW) to achieve word-level alignment. Accurate segmentation isolates individual words, while alignment ensures that audio segments match the corresponding written words with diacritics. This alignment enables ASR models to learn the nuanced relationship between pronunciation and diacritics in Arabic.

DASAM focuses on pinpointing the location (start and end time) of a specific word within a spoken sentence. It achieves this by analyzing and comparing sound characteristics, making it adaptable to various audio sources without requiring training data. DASAM relies on direct audio analysis, making it independent of training data and adaptable to various audio sources. DASAM utilizes a technique called Mel-Frequency Cepstral Coefficients (MFCC)

extraction to pinpoint word location. MFCCs capture the essential characteristics of a sound [201]. DASAM compares the MFCCs of an individual word (the "first sound") with the corresponding segment within a complete sentence (the "second sound"). This meticulous comparison allows DASAM to determine the word's precise start and end time within the sentence audio.

DASAM is not an ASR and can't be directly comparable to ASR. DASAM can be compared with the word alignment feature offered by some ASR systems, like Google STT. Although these systems have different goals (ASR for speech-to-text conversion, DASAM for word alignment), they share a similar result of identifying word boundaries. Our comparison targets the specific word timing results produced by both systems. This comparison allows us to assess the accuracy of DASAM's word boundary detection against an established industry standard. The Qur'an has a unique role in this process. With its well-defined text and established pronunciations, the Quran serves as a reliable "ground truth" for DASAM. DASAM uses the Quran's established timings as a benchmark to assess its word segmentation and alignment accuracy.

DASAM's ability to accurately predict the start and end times of words within audio recordings is notably superior, with R^2 values of 0.959 and 0.957 for start and end times respectively, compared to 0.870 and 0.849 for Google STT. This chapter illustrates DASAM's strengths through detailed statistical analysis, establishing its potential as a highly effective model for word-level alignment in Quranic diacritics tasks.

DASAM's contributions extend beyond dataset creation. It provides a foundational tool for advancing NLP technologies by improving the accuracy of Arabic speech analysis. These include improved machine translation, enriched text-to-speech systems, and more effective speaker diacritization, and automatic speech summarization by recognizing nuanced linguistic features. Our model pave the way for in-depth linguistic analysis and more robust NLP applications.

8.1 From Raw Audio to Rich Data: The DASAM Dataset

We describe the development of the dataset, starting with raw audio data and progressing towards a resource enhanced for model evaluation.

8.1.1 The Raw Audio Data:

The starting point for DASAM was a collection of sentence-level audio files. Each audio file was paired with a separate file containing timestamps. The timestamp files consisted of values representing the words' sequence and starting, and ending positions of words in the audio. The actual words for these timings are merged with the data later in the pipeline.

The audio files were assessed by listening. We curated a final reviewed dataset of 10,000 audio files from two different Quran reciters. The dataset's timing information was accurate enough, obviating the need for manual adjustments. This dataset forms the foundation for the evaluation of DASAM alignment model.

8.1.2 Our Enhanced Dataset:

To enhance the raw audio data, we performed several steps:

1. **Segmentation:** Automated processing splits each audio file into word-level clips, creating a dataset of audio segments.
2. **Text Alignment:** Each word and its corresponding sentence were meticulously linked to their Arabic text with diacritics. This creates a comprehensive linguistic dataset.
3. **Feature Extraction:** We extracted MFCC's audio features for each segmented word. These features act as reference points when comparing unseen audio to identify words and their timing. The model compares the extracted features from the unseen audio to the features of these reference words to determine if a particular word exists in the unseen audio and pinpoint its starting and ending points within the recording.
4. **Audio Variations:** To increase the model's robustness to different speech variations, we created 13 audio variations. These variations were generated by modifying the

original audio (e.g., changing pitch to resemble a child's voice or adding noise). A detailed explanation of audio variations and their impact on the model is presented in sections 8.4 and 8.5.2, respectively.

Enhanced Dataset Summary:

The final enhanced dataset includes the original audio files, the corresponding Arabic texts with diacritics, and a rich set of extracted audio features. We used the original 10,000 files for feature extraction. With the addition of audio variations, the total dataset size reaches 260,000 audio files (10,000 original files used for feature extraction multiplied by 13 variations for two reciters).

8.2 DASAM Model Overview

This section delves into the core functionalities of the Diacritic-Aware Segmentation and Alignment Model for Arabic (DASAM). DASAM tackles the challenge of segmenting and aligning words with diacritics within Arabic speech. The methodology employs a multi-stage process to achieve this. We'll explore each stage in detail, referencing the DASAM's Word-Audio Localization Process diagram (Figure 8.1) to visualize the overall workflow.

1. **Baseline Data Preparation:** We leverage existing ground truth data for the Holy Quran, containing the start and end times of each word for specific sentence-level audio files. These recordings were automatically segmented into individual word clips. Each clip is linked to its corresponding Arabic text with diacritics. These segments are visualized as "Reference Word Audios" in Figure 8.1.

2. **Feature Extraction:** In speech processing, audio analysis often involves converting the raw audio signal into a numerical representation that captures its essential characteristics. This simplifies sound comparison and matching tasks. To achieve this, we extract features from the audio data.

   We opted for MFCCs that capture and provide representations of the time-varying spectral characteristics for the audio speech [201]. We extract these features from both

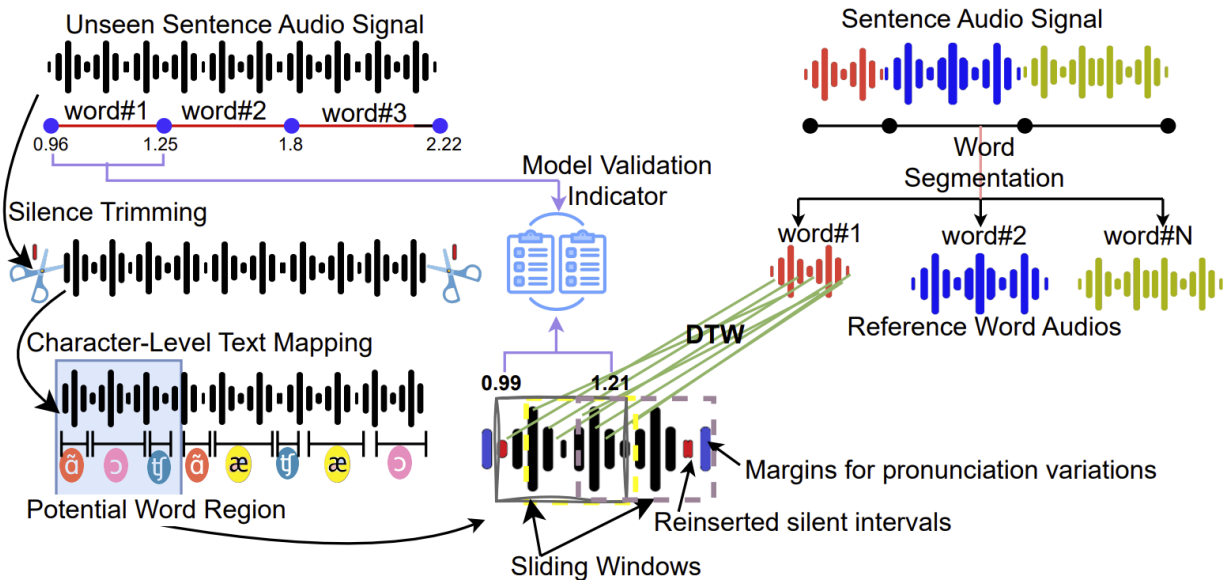


Figure 8.1: DASAM's Word-Audio Localization Process.

the reference words and unseen phonetic phrases within the audio data.

3. **Silence Trimming:** Many audio recordings contain periods of silence without speech. To achieve accurate alignment, it's crucial to identify and remove these silent intervals. This process, visualized in Figure 8.1 as "Silence Trimming," ensures the alignment focuses solely on the spoken content within the unseen sentence audio signal.

   We employ a technique called Voice Activity Detection (VAD) to achieve silence trimming. VAD algorithms segment audio files by identifying periods of silence [53]. We utilize a segmentation algorithm based on amplitude analysis, as described in [125]. This algorithm effectively distinguishes between speech and silence by dynamically adjusting a silence threshold based on the audio amplitude.

   Silence removal is important because its presence can negatively impact the subsequent step of Character-Level Text Mapping. Silence could result in letters being assigned to incorrect spoken words or extending into silent gaps.

4. **Character-Level Text Mapping:**

   This stage maps characters and diacritics in the text to their corresponding audio seg-

ments. It considers the rules of Arabic phonetics [62], where letters can vary slightly in duration. Weights are assigned based on average pronunciation lengths (e.g., vowels are typically longer than consonants). This helps distribute characters across the audio, estimating each word's position within the sentence (see "Character-Level Text Mapping" in Figure 8.1).

A key challenge in Arabic word-level alignment is the presence of similar-sounding words. This arises from the inherent characteristics of the language, where some letters share close pronunciations [152]. Due to these phonetic similarities, directly comparing entire words within the audio can be inefficient.

Character-Level Text Mapping tackles this challenge by leveraging knowledge of Arabic language structure. It acts as a pre-processing step before acoustic analysis. This process essentially "distributes" the words across the audio based on estimated pronunciation lengths for each character.

By performing this mapping, we significantly reduce the search space for word-level alignment. Instead of searching for entire words within the whole sentence audio, the system focuses on smaller, more manageable segments corresponding to individual characters. This approach significantly increases the efficiency of the word-level alignment process.

Table 8.1 shows an example of the number of moras required to pronounce each letter in a word. Mora is a unit of relative duration in spoken language, typically corresponding to the time it takes to pronounce a letter or a syllable. Although this is fixed and standard, it varies according to the reciter's estimates. Estimates vary, but the reciter should maintain the same proportionality between the number of moras for all letters regardless of the speed.

The example in Table 8.1 shows a word that takes 22 moras to pronounce correctly. For example, if each mora takes a second, the reciter takes 22 seconds to pronounce this word. And if the reciter performs the phoneme in half a second, then it will take

Word	كَـــــــــــهـــــــــــيـــــــــــعَـــــــــــضْ				
letter	كَـــــ	هـــــ	يـــــ	عَـــــ	ضْـــــ
#Moras	6	6	2	2	6

Table 8.1: Moras Distribution for Arabic Word Pronunciation.

11 seconds to pronounce it. The number of moras has several bases for its calculation.

Character-Level Text Mapping Pipeline:

This pipeline details (Figure 8.2) the process of mapping characters and diacritics in Arabic text to their corresponding audio segments within an audio file. It considers the rules of Arabic phonetics, where letters can have slightly varying durations.

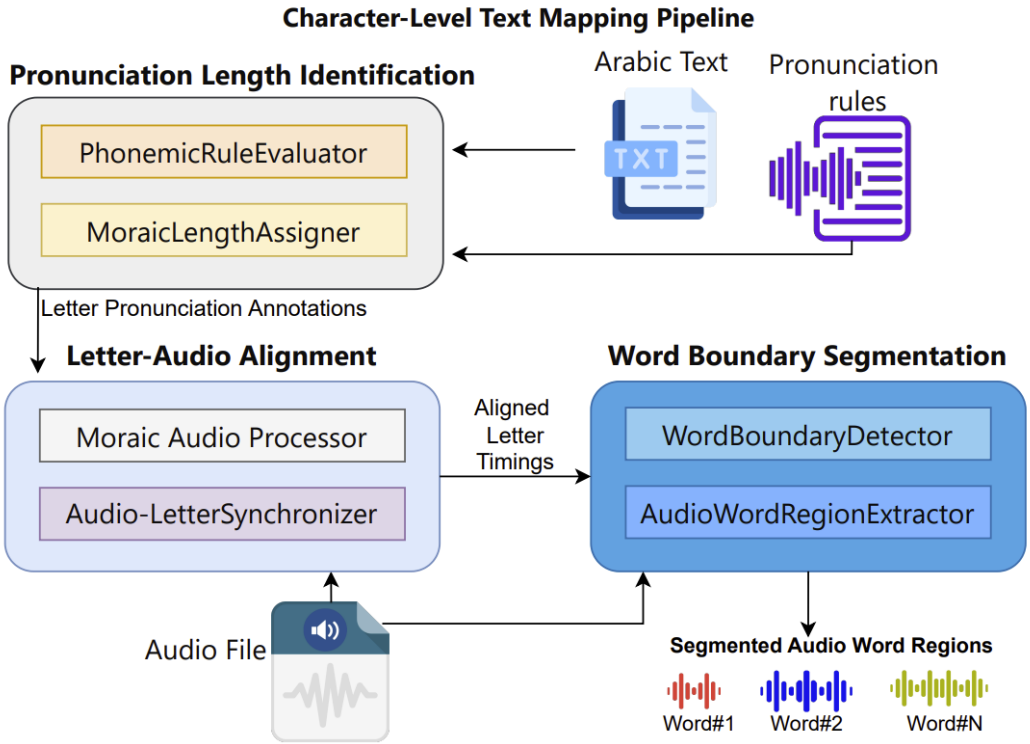


Figure 8.2: Character-Level Text Mapping Pipeline.

- **Inputs:**
  (a) **Arabic Text with Diacritics:** The input text contains Arabic characters and diacritics.

(b) **Pronunciation Rules:** These rules govern how the length of pronunciation varies for each letter based on its position within a word (beginning, middle, end), neighboring characters, and diacritics [29]. These rules are typically derived from Tajwid, a science focused on the correct pronunciation of Arabic, especially in the Quran [11].

(c) **Audio File:** The audio file containing the spoken Arabic text for which word boundaries need to be identified.

- **Modules:**

(a) **Pronunciation Length Identification:** This module determines the pronunciation length of the text. It consists of two components: PhonemicRuleEvaluator and MoraicLengthAssigner. These work together to identify relevant pronunciation rules, analyze letter and diacritic interactions, and assign time units to each letter or phoneme.

- **PhonemicRuleEvaluator:** employs a combination of text parsing algorithms and rule-based analysis to detect diacritics and letter positions in the text. Determines the phonetic duration of Arabic script components in our model. Uses Contextual Phonetic Mapping to analyze letter and diacritic marker interactions. Scans Arabic text to identify pronunciation duration based on Pronunciation Rules. These rules account for diacritic adjacency and letter position in determining phonetic duration. Assign time units to each phoneme according to these rules. For example, a letter with a specific diacritic mark has a set pronunciation duration in a time unit.

- **MoraicLengthAssigner:** Based on the identified rules, the MoraicLengthAssigner assigns a "mora" value to each letter or phoneme. A mora is a unit of relative pronunciation length, roughly corresponding to the time it takes to pronounce a letter or syllable.

Example: The Arabic letters "م" (m) and "ن" (n) followed by the diacritic sign *shaddah* like "مّ" and "نّ" require a prolonged pronunciation, equivalent to 2 moras. According to the rules, the PhonemicRuleEvaluator identifies this rule and triggers the MoraicLengthAssigner to assign 2 moras, accurately reflecting the required duration. However, the actual timing in seconds that corresponds to a mora can vary depending on the speaker's pace, though the relative pronunciation length (in terms of moras) remains consistent.

Despite English not having fixed rules for mora timing we still can see examples that can express the idea. For example, the word "TV" has two letters. Each letter takes the same time to pronounce. This period is neither long nor short. This period is called a mora. We cannot lengthen the "T" by 2 mora. Doing so would make the pronunciation incorrect. It could also result in another word. Some letters naturally take 2 moras. For example, the word "example" is pronounced "ig'zampel". The letter "x" is pronounced "g'z," taking 2 moras. This is twice the time of pronouncing "g" or "z". The letter "e" after "l" is not pronounced. Its mora is zero.

The table 8.2 shows an example in English. It highlights differences in mora length in the simple sentence "Cute Express", compared to the International Phonetic Alphabet (IPA) for each letter.

Table 8.2: Mora Length Comparison in English

English	c	u	t	e	x	p	r	e	s	s
IPA	k	yoo	t	i	k's	p	r	e	s	
Mora	1	3	1	1	2	1	1	1	1	0

(b) **Letter-Audio Alignment:** Aligns textual phoneme durations with audio data, ensuring synchronization of spoken content.

- **Moraic Audio Processor:** This module first obtains the total number of moras from the text analysis. It then extracts audio features (MFCCs)

from the audio file and calculates the total audio length (excluding silence). Finally, it calculates the "Moraic Frame Rate" (MFR) by dividing the audio feature length by the total number of moras. This MFR represents the number of audio frames (time slices) per mora. The MFR is determined by the equation (8.1).

$$\text{MFR} = \frac{\text{Total Audio Length (excluding silence)}}{\text{Total Number of Moras}} \tag{8.1}$$

where *Total Audio Length* is the length of the audio file excluding any periods of silence, and *Total Number of Moras* is obtained from the text analysis of the Arabic script. Consider a practical example to illustrate the calculation of the MFR. Assume a sentence requires ten moras for proper pronunciation. If a reader pronounces this sentence in 20 seconds, the MFR, according to equation (8.1), would be 2 seconds per mora. In another scenario, if a different reader speaks more slowly and takes 30 seconds to pronounce the same sentence, the MFR would then be 3 seconds per mora. These examples demonstrate how the MFR can vary with the speaking rate, even though the number of moras required remains constant. This variability emphasizes the adaptability of the DASAM model to different speech rates while maintaining synchronization between text and spoken audio.

– **Audio-Letter Synchronizer:** This module uses MFR and frame rate information to convert theoretical mora boundaries into practical start and end times. This conversion is done for each letter within the audio file. The process involves two steps: (a) Calculating the temporal position of each mora using the MFR and frame rate. (b) Determining the start and end times for each letter by multiplying the duration of one mora by the starting and ending mora indices of that letter.

Calculation Example: Assume the MFR indicates there are X frames per mora, and the audio frame rate (frames per second) is Y. Then the time duration T in seconds for one mora is calculated as in (8.2) :

$$T = \frac{X}{Y} \quad \text{(seconds per mora)} \tag{8.2}$$

Given this duration T, the start and end times for any letter spanning from mora index a to b are calculated as: Given this duration T, the start and end times for any letter can be determined by multiplying T by the starting and ending mora indices of that letter. If a letter spans from mora index a to b, then the calculation will be as in (8.3 and 8.4):

$$\text{Start Time} = a \times T \tag{8.3}$$

$$\text{End Time} = b \times T \tag{8.4}$$

Application Example: Consider a scenario where a specific letter spans the third to the fifth mora in the text. The MFR is calculated to be 100 frames per mora with an audio frame rate of 1000 frames per second. The time duration T for one mora would thus be: The time duration T for one mora is given by eq(8.5)

$$T = \frac{100}{1000} = 0.1 \text{ seconds per mora} \tag{8.5}$$

Thus, for this letter:

$$\text{Start Time} = 3 \times 0.1 = 0.3 \text{ seconds}$$

$$\text{End Time} = 5 \times 0.1 = 0.5 \text{ seconds}$$

This calculation helps us determine the exact timing of each letter in the audio file. It transforms theoretical mora assignments into precise timing annotations. The Audio-Letter Synchronizer ensures correct alignment of each letter's pronunciation within the spoken sentence. This enhances the accuracy of subsequent word boundary segmentation.

(c) **Word Boundary Segmentation:** Segments the audio into discrete words based on the synchronized letter timings.

- **WordBoundaryDetector:** This module takes the letter timings and the original text as input. It identifies word boundaries based on the letter timings and their corresponding text characters. This results in a list of words, each with its start and end time within the audio file.
- **AudioWordRegionExtractor:** This module extracts the corresponding audio segments for each identified word based on their start and end times.

- **Output:**

  The final output consists of a list, where each element represents a word from the original text along with its corresponding audio segment extracted from the audio file. This pipeline effectively addresses the challenge of similar-sounding words in Arabic by leveraging pronunciation rules. It facilitates the identification of word boundaries within the audio file.

5. **Potential Segment Identification:** The possible segment is determined based on the linguistic distribution along the spoken phoneme of the sentence. This helps in reducing the length of the sound within which you need to search for the position of each word in the sentence. After silent intervals are reintegrated, segments are aligned with the original audio to maintain correct timing from the ground truth data. Margins around each word segment are adjusted for potential pronunciation variations, in preparation for sound matching and alignment. As depicted by "Reinserted silent

intervals" and "Margins for pronunciation variations" in Figure 8.1.

6. **Refined Word Alignment:** Dynamic Time Warping (DTW) is a technique that helps align two sequences that may have different speeds or lengths [145]. Here, it compares the features of the potential word region with the reference word audio to find the best match. As shown by "DTW" in Figure 8.1, where it tries to find the closest location in the Potential Segment to the Reference Word Audio. Word boundary segmentation based on linguistic rules provides optimal pronunciation timings, but human variations and errors occur. Meanwhile, DTW may struggle with similar-sounding or repeated words. To overcome this, we combine linguistic segmentation with DTW. Linguistic rules determine the optimal pronunciation location, and we add margins for phonetic distortion. DTW then refines the result within these limits, preventing large errors and improving overall accuracy. By integrating both systems, we achieve greater improvement and flexibility with human estimates.
7. **Timing Data Extraction:** A sliding window technique pinpoints the precise start and end times of each word within the sentence-level audio [188]. By moving the window across the unseen sentence audio and comparing the features within the window to the reference word audio, the model determines the most likely start and end times of the word. This technique is visualized with "Sliding Windows" in Figure 8.1, comparing the times deduced by the model to the ground truth.
8. **Evaluation:** The effectiveness of the DASAM model is assessed by the Model Validation Indicator which compares the model's predicted word timings with the actual word boundaries to assess the model's accuracy. Then we compare this accuracy to Google STT alignment's results for the same audio data. The evaluation primarily focuses on the accuracy of detecting word start and end times.

This structured, multi-stage approach ensures that DASAM not only generates datasets but also enhances understanding and processing capabilities for Arabic speech recognition technologies.

8.2.1 DASAM's Operational Algorithm:

The Algorithm 10 provides an overview of the DASAM Model's workflow. It streamlines the detailed operations by omitting preliminary data preparation and final result comparison procedures. Focusing on the main processes facilitates a deeper understanding of the model's inner workings.

The first step involves feature extraction (lines 1, 2). The MFCCs are extracted from both the sentence audio and reference word audio segments. Subsequently, silence removal is performed on the unseen sentence audio (line 3). This step ensures that the linguistic alignment focuses solely on the spoken content.

Lines 4-12 address Character-Level Text Mapping. The algorithm analyzes the Arabic text with diacritics and pronunciation rules. This analysis estimates the pronunciation length for each character based on its position within a word and surrounding letters. This information is then used to define potential word regions within the sentence audio.

Lines 13-17 expand potential word regions. Silence intervals are re-inserted to align predicted times with standard times for comparison. Additionally, margins are added to account for variations in pronunciation.

Algorithm 10: Audio-Text Alignment for Pronunciation Assessment

Input : text (diacritic-annotated),
phonetics (word-level),
audio (unseen sentence audio),
rules (pronunciation rules).

Output: FinalResults: List of words with aligned audio timestamps

1 $mfcc\_sentence \leftarrow$ ExtractMFCC(*audio*, level="sentence")
2 $mfcc\_words \leftarrow$ ExtractMFCC(*phonetics*, level="word")
3 $audio\_trimmed \leftarrow$ TrimSilence(*audio*)
4 $potential\_regions \leftarrow \emptyset$
5 **foreach** *(word in text)*
6 $pronunciation\_length \leftarrow$ PhonemicRuleEvaluator(*text*, *rules*)
7 $moras \leftarrow$ MoraicLengthAssigner(*pronunciation_length*, *audio*)
8 $mfr \leftarrow$ MoraicAudioProcessor(*moras*, *mfcc_sentence*)
9 $letter\_timings \leftarrow$ AudioLetterSynchronizer(mfr, *mfcc_sentence*)
10 $word\_boundary \leftarrow$ WordBoundaryDetector(*letter_timings*, *text*)
11 $audio\_region \leftarrow$ AudioWordRegionExtractor(*word_boundary*, *audio*)
12 Add to *potential_regions* the pair (*word*, *audio_region*)
13 $segmented\_regions \leftarrow \emptyset$
14 **foreach** *(region in potential_regions)*
15 Add reinserted silence to region
16 Add margins for pronunciation variations
17 Add to *segmented_regions*
18 $final\_results \leftarrow \emptyset$
19 **foreach** *(word in segmented_regions)*
20 **foreach** *(sliding_window_word in word)*
21 $dtw\_distance \leftarrow$ DTW(*phonetics*, *sliding_window_word.audio*)
22 $min\_distance \leftarrow \infty$
23 **if** $dtw\_distance < min\_distance$ **then**
24 $min\_distance \leftarrow dtw\_distance$
25 $start\_time\_msec \leftarrow$ CalculateStartTime(*sliding_window_word*)
26 $end\_time\_msec \leftarrow$ CalculateEndTime(*sliding_window_word*)
27 Add (*sliding_window_word.text*, *start_time_msec*, *end_time_msec*) to *final_results*
28 **RETURN** *final_results*

Lines 18 to 27: A sliding window technique iterates through each potential word region. The DTW calculates the distance between the reference word audio segment and each window

inside the potential word region to find the most similar match. Based on the best match, the algorithm extracts the precise start and end times for the word within the sentence audio. The final output is a list containing each word with its corresponding start and end times within the unseen sentence audio.

This algorithm highlights the key steps involved in segmenting and aligning words within Arabic speech using DASAM.

8.3 DTW Impacts on Speech Timing Accuracy

This section evaluates the impact of DTW on speech timing accuracy. We compared two methods: Word Timing Estimates based on text-audio linguistic distribution and DTW-Refined Word Timings. Both methods were evaluated against the Ground Truth baseline. These comparisons highlight the effectiveness of DTW when integrated with the linguistic distribution in the DASAM algorithm. The analysis is based on our dataset from 20000 audio files, read by two reciters.

8.3.1 Timing Optimization with DTW

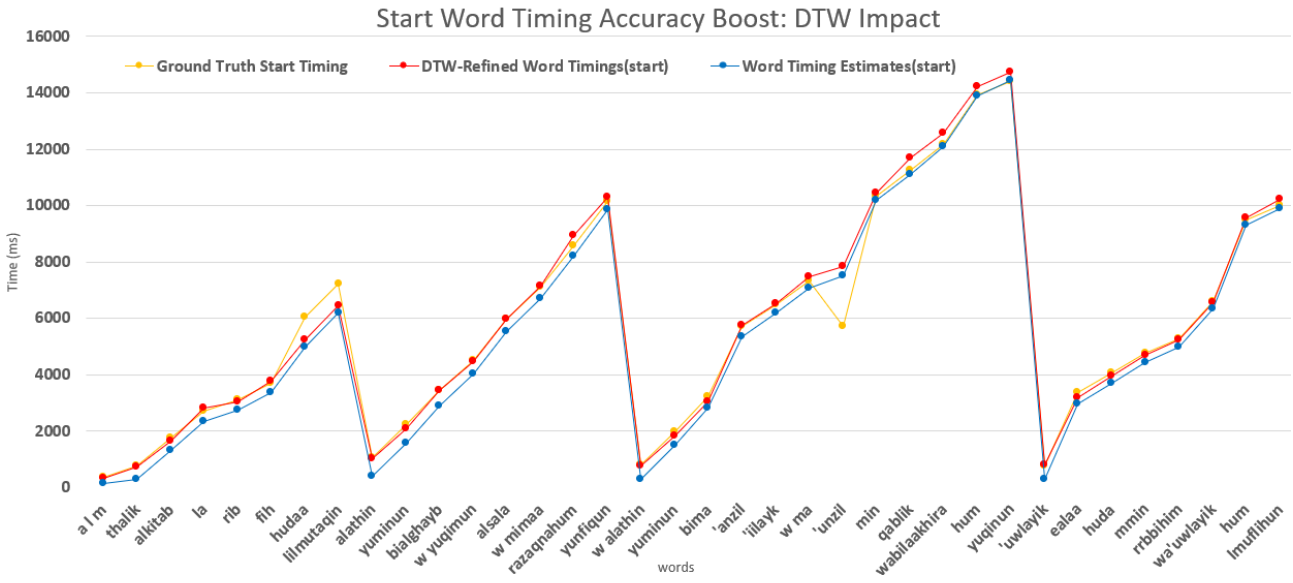


Figure 8.3: Star Timing Accuracy: Before and After DTW Refinement.

Figure 8.3 depicts the start timings of a word sample. Ground Truth Start Timing is the baseline for comparison. Word Timing Estimates based on text-audio linguistic distribution show minor deviations from Ground Truth, suggesting room for improvement. DTW-Refined Word Timings align closely with the baseline, indicating effective refinement. The figure shows that the lines consistently start at nearly 0ms for each audio sentence. As the sentence progresses, the lines steadily rise with each word. The lines reach their highest point at the last word. Then, they reset to 0ms or near 0ms, indicating the start of a new audio file/sentence, which represents a new sentence.

In the accuracy of DTW Refinement, the DTW-Refined Word Timings consistently display increased precision. For example, for the word "alm," DTW refinement adjusted the start time from an estimated 129.63 ms to 319.15 ms, closer to the ground truth of 350 ms. Figure 8.3 demonstrates modest enhancements in detecting word starts. Despite the subtlety of these improvements, they contribute meaningfully to longer audio sequences.

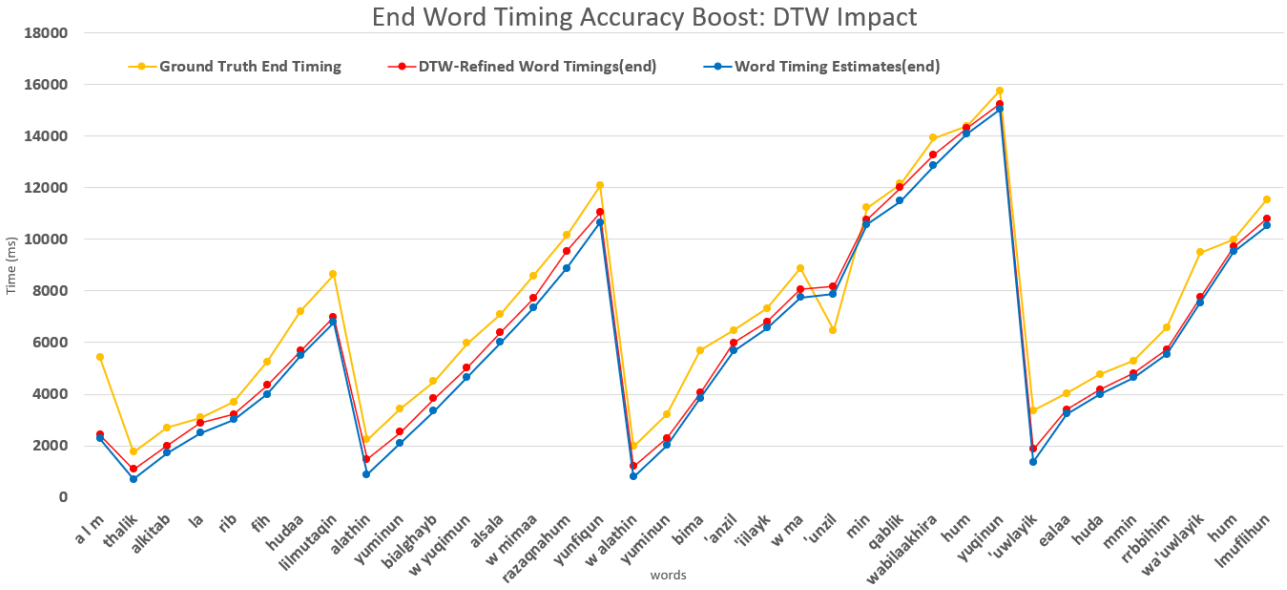


Figure 8.4: End Timing Accuracy: Before and After DTW Refinement.

Figure 8.4 presents end-of-word timings with marked improvements post-DTW Refinement, aligning more closely with Ground Truth End Timings than earlier estimates. These refine-

ments underscore the consistency and potential of DTW adjustments in audio processing tasks. For example, for the word "qablik," DTW refinement adjusted the start time from an estimated 11488 ms to 11999 ms, closer to the ground truth of 12150 ms.

8.3.2 Quantitative Error Analysis in DTW Refinement

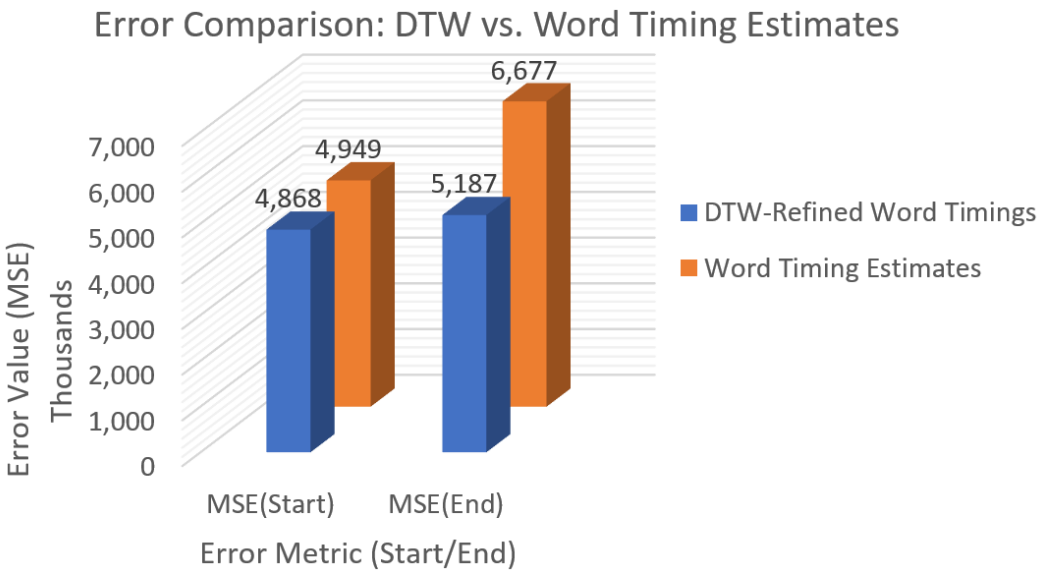


Figure 8.5: Illustrating MSE Improvements with DTW-Refined Word Timings.

Figure 8.5 illustrates the Mean Squared Error (MSE) for start and end word timings. MSE measures the average squared difference between the predicted word timings (start and end times) and the actual timings data. Lower MSE values indicate better alignment accuracy, as smaller differences translate to less error. MSE is typically expressed in milliseconds squared (ms^2). An MSE of 0 ms^2 would indicate perfect alignment, where all predicted timings perfectly match the reference. Smaller values represent greater accuracy. The MSE for DTW-Refined Word Timings and Word Timing Estimates is presented in thousands. For start times, DTW refinement reduced MSE from 4,949 to 4,868. For end times, the reduction is more pronounced, from 6,677 to 5,187. The data demonstrate significant improvements with DTW. The more substantial reduction in end-time errors highlights DTW's effectiveness. This analysis confirms the robustness of DTW in refining word timing predictions, aligned

with linguistic distributions over audio.

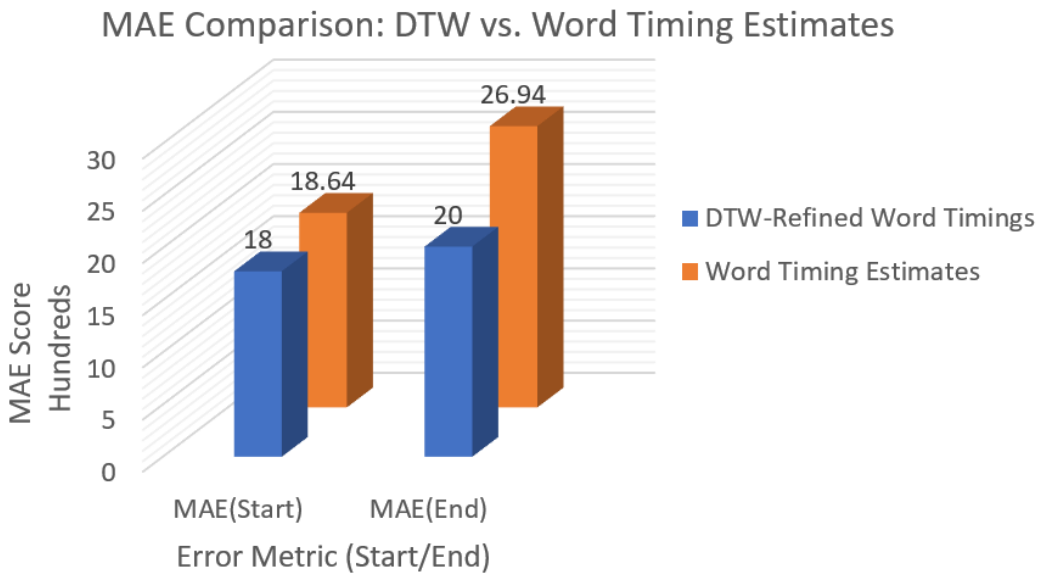


Figure 8.6: Effectiveness of DTW in Minimizing Mean Absolute Errors in Speech Timing.

Figure 8.6 displays the Mean Absolute Error (MAE) for start and end word timings, comparing DTW-Refined Word Timings to Word Timing Estimates. MAE calculates the average absolute difference between the predicted and actual word timings. It provides a simpler interpretation of error compared to MSE, focusing on the magnitude of the errors without considering their direction (positive or negative). MAE is also expressed in milliseconds (ms). Similar to MSE, lower MAE values indicate better alignment. For start times, MAE decreased from 18 to 17 due to DTW refinement. The end times showed a more significant reduction, from 26.94 to 20.

Across all presented metrics, DTW-Refined Word Timings consistently show improvements over basic Word Timing Estimates. The consistent positive results across different evaluations demonstrate the robustness of adding DTW to our process.

8.4 Handling Audio Variations

This section explores the concept of audio variations and their role in enhancing the robustness of the DASAM model. Audio variations are synthetic speech recordings created by modifying original audio to simulate real-world listening environments [20]. These variations introduce factors like changes in pitch, background noise, and speaker characteristics [28]. To improve the model's generalizability and adaptability to real-world scenarios. We used the audio variations with DASAM for several reasons:

- **Mimicking Real-World Speech:** They help us test DASAM under conditions that resemble real-world speech scenarios (noise, different genders, etc), making the model more adaptable.
- **Enhancing Dataset Diversity:** Variations effectively enlarge the dataset size, providing a wider range of speech samples for model evaluation [20]. This leads to more reliable and generalizable results.
- **Building Robust Models:** By incorporating audio variations, we can assess DASAM's ability to handle speech variations, ensuring its effectiveness in diverse listening conditions.

8.4.1 Mathematical Model: Addressing Speech Variations

Phonetic variations can affect the ground truth of word-level text-to-audio alignment. The impact depends on the type of variation method used:

- **Non-Time-Altering Methods:** Some phonetic variations do not change the audio signal's duration. These include changes to spectral characteristics or subtle articulation. They do not alter the timing information. Therefore, the ground truth of word-level alignment remains unchanged.
- **Time-Altering Methods:** Other phonetic variation techniques involve stretching or compressing the audio. These directly impact the pronunciation and length of words and sentences. In such cases, the ground truth for word-level alignment must be

adjusted. The amount of time shift can be calculated. It is based on the expansion or contraction factor applied by the chosen variation method. Each subsequent word in the aligned text needs a shift by the same amount. This maintains accurate alignment with the modified audio.

For audio variations that involve speed adjustments (acceleration or deceleration), we apply a mathematical model. This model calculates the adjusted word-level timing. Our model operates on two principles during speed adjustments:

- **Equal Displacement:** When the sentence speed is altered, every word is uniformly affected. That is, the start and end times of all words shift proportionally to the change in sentence duration.
- **Equal Rate of Displacement:** The rate at which the timings of each word are displaced is consistent across the sentence. This means the relative duration of each word compared to the sentence remains constant, regardless of the speed adjustment.

This approach ensures that the sequential integrity of words is preserved. Each word's relative duration within the sentence remains constant across all speed variations. Our confidence in this method stems from its alignment with the inherent properties of linear time scaling in audio processing. Here is how the model is formulated:

Let S_o represent the original sentence duration, and S_m denote the modified sentence duration after speed adjustment. The original and modified timings for each word, represented by $t_{start,o}$, $t_{end,o}$ for the original, and $t_{start,m}$, $t_{end,m}$ for the modified, are related by the equations (8.6 & 8.7):

$$t_{\text{start},m} = t_{\text{start},o} \times \frac{S_m}{S_o} \tag{8.6}$$

$$t_{\text{end},m} = t_{\text{end},o} \times \frac{S_m}{S_o} \tag{8.7}$$

This approach ensures that the temporal relationship between words remains consistent. It

preserves the linguistic integrity of the sentence across variations.

Example: Consider a sentence with an original duration of 20 seconds. Within this duration, the word "quick" is pronounced from the 2nd to the 4th second. If an audio variation involves halving the sentence's duration to 10 seconds, the start and end times for "quick" adjust proportionally:

- Original Duration (S_o): 20 seconds
- Modified Duration (S_m): 10 seconds
- Original Timing for "quick": 2 to 4 seconds
- Adjusted Timing for "quick": 1 to 2 seconds

This example illustrates the application of the mathematical model. It adjusts word-level timings in response to speed variations. This enables us to derive accurate ground truth annotations for each audio variation.

8.5 Discussion and Comparative Analysis

This section discusses the performance of DASAM in word-level alignment. Alignment involves determining the precise start and end times for each word within an audio file. Here, we evaluate DASAM's ability to perform this task by comparing its results with those from Google Speech-to-Text (STT), a well-established industry standard.

We provided both DASAM and Google STT with the same audio files. Both systems returned a list of words along with their corresponding start and end times within the audio file. This common output format allows for a direct comparison of performance between the two systems.

8.5.1 Metrics Comparison: Understanding Key Performance Metrics

This section delves into the key performance metrics used to evaluate the word-level alignment capabilities of DASAM and Google Speech-to-Text. These metrics provide quantitative insights into how accurately each system identifies word start and end times within audio

Metric	Value	
	DASAM	Google STT
MSE (Start Time) (ms^2)	61.4M	238M
MSE (End Time) (ms^2)	65.4M	277.3M
MAE (Start Time) (ms)	14.1K	27.6K
MAE (End Time) (ms)	15.9K	30.7K
R^2 (Start Time)	0.959	0.870
R^2 (End Time)	0.957	0.849
Pearson Correlation (Start Time)	0.981	0.936
Pearson Correlation (End Time)	0.982	0.927

Table 8.3: Summarizing the performance of DASAM and Google STT on the key metrics

recordings. We will explain each metric and present the results in a table 8.3 for easy comparison.

The key performance metrics employed in this analysis:

1. **Mean Squared Error (MSE):**

   MSE measures the average squared difference between the predicted word timings (start and end times) by the system and the actual timings in the reference data. DASAM reports MSE of 61.4M/65.4M for start and end times respectively, while Google STT exhibits a higher MSE of 238M/277.3M, indicating DASAM's superior accuracy in error minimization.

2. **Mean Absolute Error (MAE):**

   MAE calculates the average absolute difference between the predicted and actual word timings. DASAM's MAE values stand at 14.1K/15.9K, which are significantly lower than Google STT's 27.6K/30.7K, showcasing DASAM's closer predictions to true values.

3. **R-squared (R^2):**

   R^2 is a statistical measure that reflects the proportion of variance (spread) in the reference timings that can be explained by the predicted timings from the system. It essentially indicates how well the model's predictions capture the overall pattern of

the actual timings. R^2 ranges from 0 to 1, with higher values closer to 1 signifying a stronger correlation between the predicted and actual timings. An R^2 value of 1 indicates that the model perfectly explains all the variance in the reference timings, while a value of 0 suggests no explanatory power.

With DASAM achieving 0.959/0.957 and Google STT at 0.870/0.849 for start and end times respectively, DASAM is shown to account for more variance in the dataset.

4. **Pearson Correlation Coefficient:**

   The Pearson Correlation Coefficient measures the linear correlation between the predicted and actual word timings. It indicates the strength and direction of the linear relationship between these two variables. Values range from -1 to +1.

   - A coefficient of +1 indicates a perfect positive linear correlation, meaning that as the predicted timings increase, the actual timings also increase proportionally.
   - A coefficient of -1 indicates a perfect negative linear correlation, where higher predicted timings correspond to lower actual timings.
   - A value of 0 suggests no linear correlation between the predicted and actual timings.
   - In the context of word-level alignment, values closer to +1 are desirable, indicating a strong positive correlation between the predicted and actual timings.

Both systems score highly, but DASAM slightly edges out with 0.981/0.982 against Google STT's 0.936/0.927, affirming the model's stronger predictive alignment.

8.5.2 Statistical Insights: Unveiling the Details in the Charts

This section presents a statistical charts analysis used to evaluate the DASAM model. These charts help illustrate the model's performance in handling audio variations compared to Google Speech-to-Text in word-level alignment. By examining these charts, we can gain deeper insights into the strengths and weaknesses of each system.

1. **Boxplot of Errors:** A boxplot is a visualization tool that summarizes the distribution

of errors in a dataset. It depicts several key elements:

- Center line: This line represents the median error value, which indicates the middle point when all errors are ordered from smallest to largest.
- Box: The box encompasses the middle 50% of the data, with the bottom and top edges representing the first and third quartiles (Q1 and Q3), respectively. A narrower box signifies a tighter concentration of errors around the median.
- Whiskers: These lines extend from the box and capture the remaining data points (outside the middle 50%). Longer whiskers indicate a wider spread of errors in the dataset.

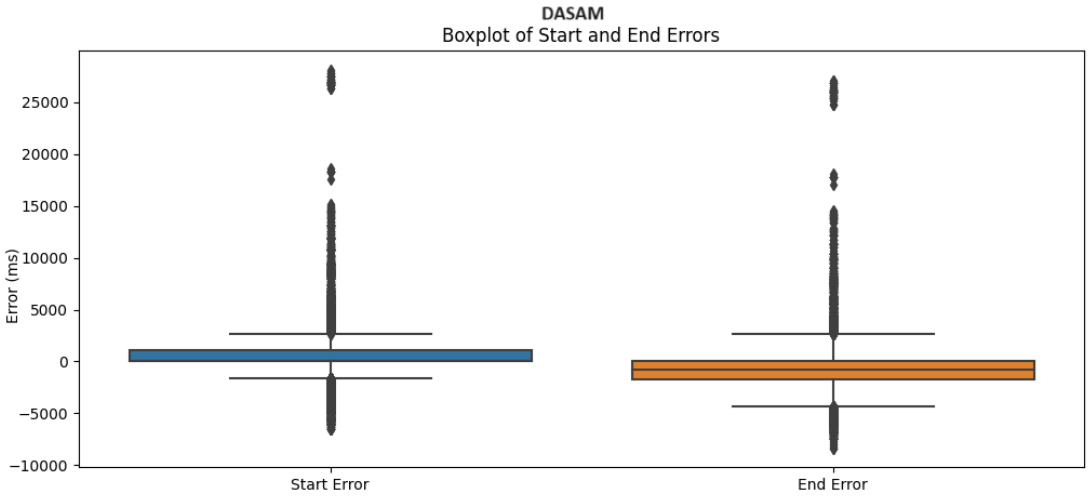


Figure 8.7: Distribution of Alignment Errors for DASAM: Showcasing the concentration of errors within a narrower range.

Figure 8.7, a boxplot of DASAM's errors, shows that errors are mostly concentrated between -10,000ms and 25,000ms. This tighter range suggests that DASAM's predictions are generally precise. Google STT has errors ranging from -20,000ms to 80,000ms (Figure 8.8). This wider spread indicates greater variability in prediction accuracy. A narrower boxplot for DASAM indicates more consistent performance compared to Google STT, particularly for start times.

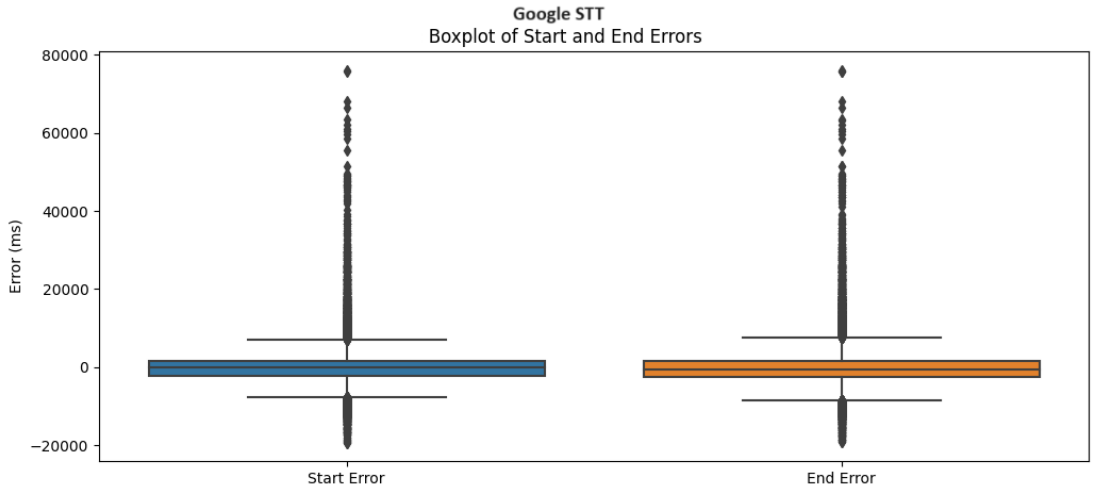


Figure 8.8: Distribution of Alignment Errors for Google STT: Illustrating wider variability in alignment errors.

2. **Visualizing Error Distribution with Histograms:**

   The histogram visualizes the distribution of errors on the x-axis (error values) and the frequency of each error value on the y-axis (number of occurrences). A prominent peak around zero on the x-axis would indicate that many predictions are very close to the correct timings (low error values). A wider spread with a less pronounced peak suggests that errors are more scattered across a larger range.

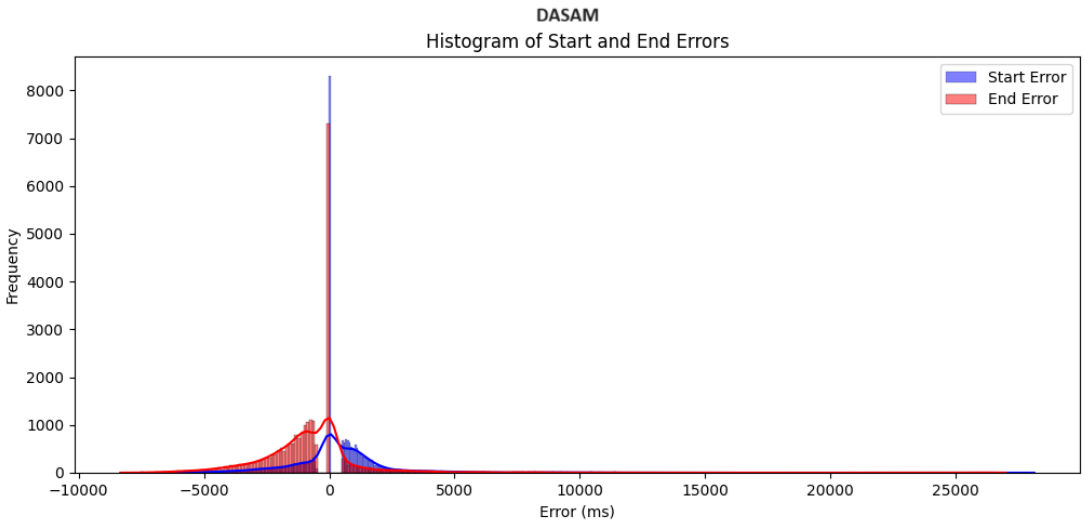


Figure 8.9: Histogram of DASAM's Alignment Errors: Peak concentration near zero error

Figure 8.9 illustrates a histogram of DASAM's errors, with a pronounced peak between 0ms and 8,000ms. This shows that most of DASAM's predictions are close to the actual timings. The histogram of Google STT (Figure 8.10) is more spread out with a peak from 0ms to 5,000ms. This suggests that errors are more dispersed across a range. DASAM demonstrates a higher accuracy, with a significant number of predictions near the correct timing, indicated by the concentration of data around 0ms error.

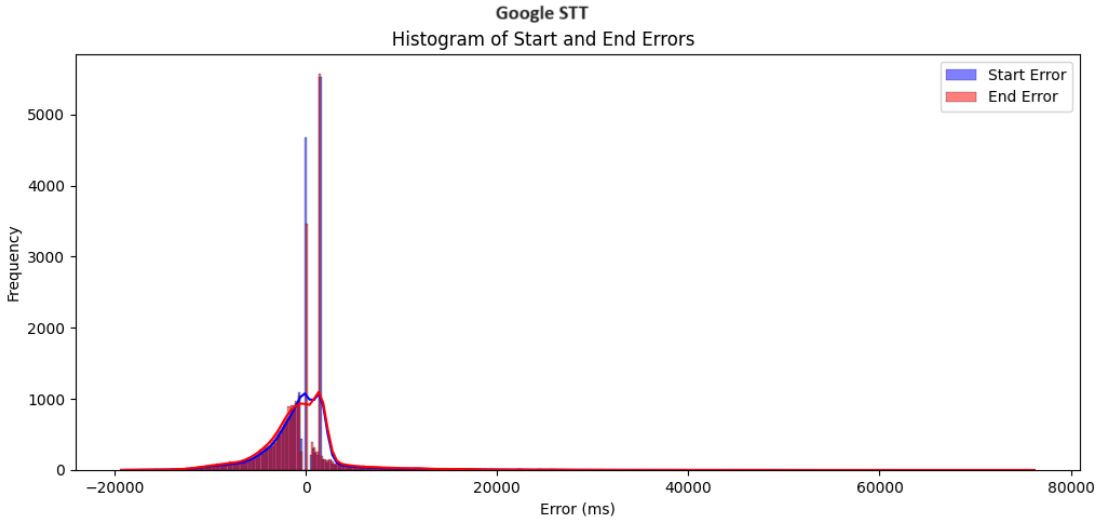


Figure 8.10: Histogram of Google SST's Alignment Errors: Less concentrated peaks indicating scattered errors

3. **Scatter Plot of Reference vs. Inferred Times:**

   A scatter plot visually represents the relationship between the actual reference word timings and the system's inferred (predicted) timings. Each data point on the plot corresponds to a single word, with its horizontal position representing the reference time and its vertical position representing the inferred time.

   As shown in Figure 8.11, DASAM's data points are tightly clustered around the diagonal line, indicating high accuracy in predictions. Figure 8.12 shows a wider spread of data points for Google STT, especially for end times, indicating less accuracy. DASAM's predictions closely match the reference times, showing its superior accuracy in word alignment.

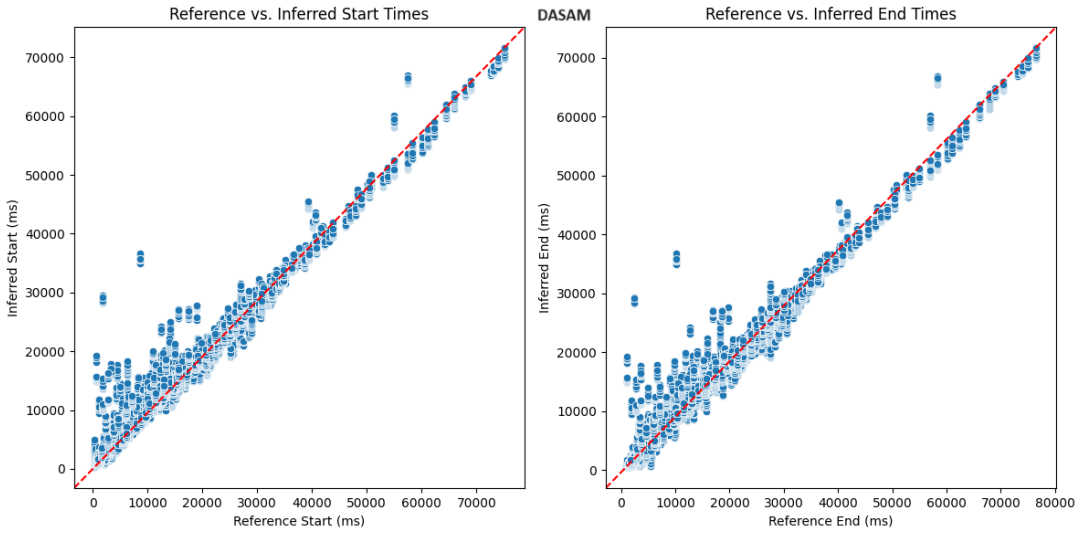


Figure 8.11: Scatter Plot of DASAM's Predictions: Scatter plot showing high alignment between predicted and reference timings.

Figure 8.12: Scatter Plot of Google STT's Predictions: More spread indicating varied prediction accuracy.

4. **Unveiling Systematic Bias with Residual Plots:** A residual plot depicts the difference between predicted and actual values (residuals) on the y-axis and the actual values (reference timings) on the x-axis. It helps identify any systematic bias in the errors.

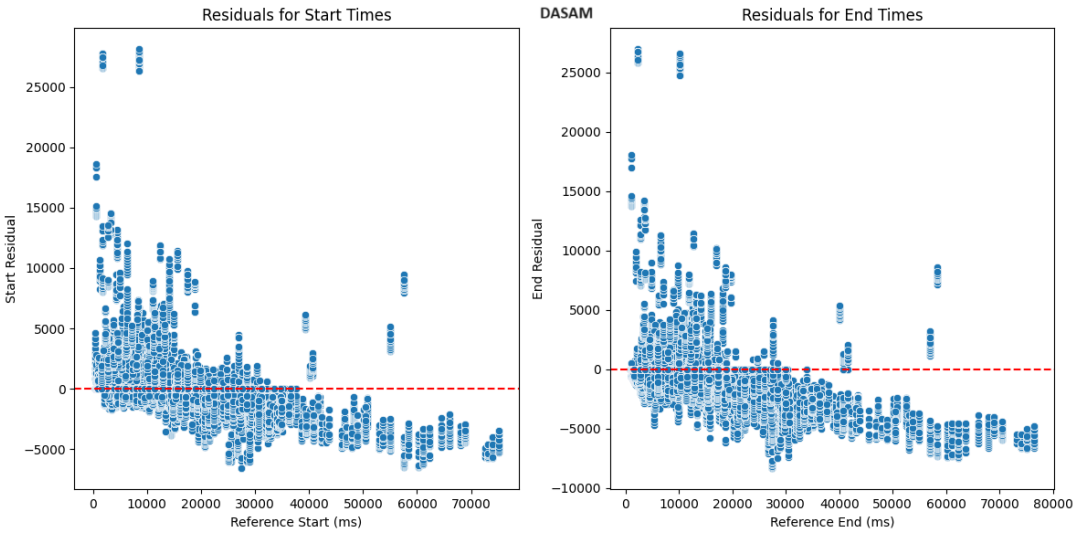


Figure 8.13: Residual Analysis for DASAM: Close clustering around zero indicating accurate predictions

In Figure 8.13, DASAM's residuals are clustered near the zero line, suggesting accurate predictions. Figure 8.14 illustrates that Google STT's residuals spread from -20,000ms to 80,000ms, indicating more significant deviations from true values. DASAM shows smaller and more consistent deviations from actual timings, demonstrating greater accuracy in word alignment.

The residual plots (Figure 8.15) also reveal an interesting observation:

- Anticipatory Behavior: The residuals consistently below zero indicate the algorithms tend to proactively predict word times before they actually occur. This might be due to their sensitivity to subtle sounds like rising pitch at word beginnings or breath before pronunciation. Both Google ASR and DASAM exhibit this behavior.

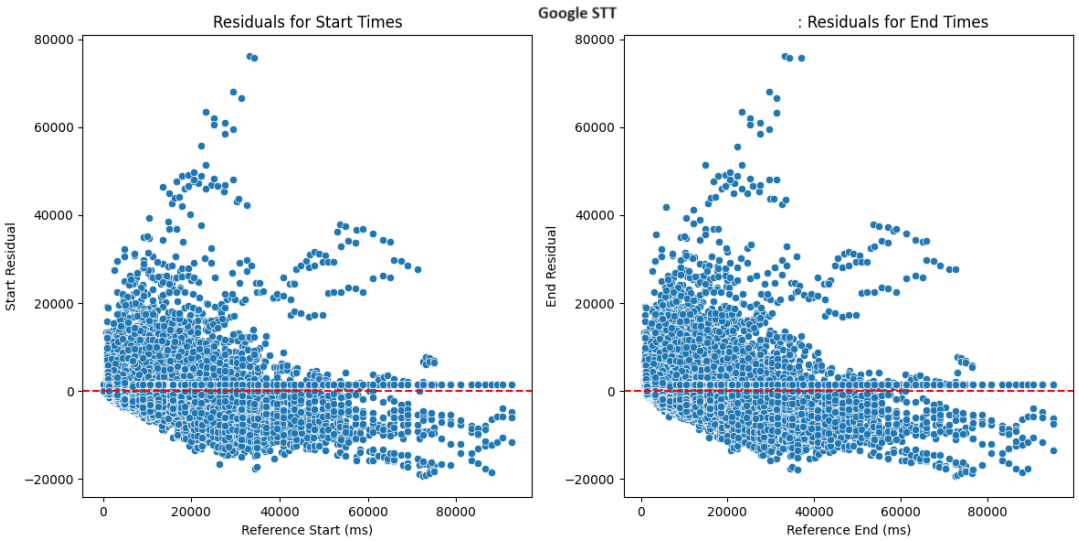


Figure 8.14: Residual Analysis for Google STT: Larger deviations from the zero line shown in residuals

- Normal Distribution at Shorter Lengths: In sentences under 40,000ms, recognition errors appear normally distributed, with both delayed and anticipatory predictions. Words at the beginning tend not to have a silent time before them.
- Long Sentences and Prediction: For sentences exceeding 40,000ms, anticipatory prediction errors persist, albeit less frequently, because there are fewer words present at this time, but delayed prediction errors decrease sharply.

This suggests that increased sentence length allows the algorithms to improve alignment accuracy and reduce late prediction, particularly for words towards the end. This could be due to the algorithms leveraging the additional information provided by longer sentences and the precise knowledge of sentence endings. Additionally, the distribution of more words over a longer timescale might facilitate their alignment.

5. **Trend of Scores:** These charts (Figure 8.16 & 8.17) depict the performance of both systems (DASAM and Google STT) across different audio variations. The X-axis typically represents the audio variations used for evaluation, while the Y-axis represents the corresponding performance score.

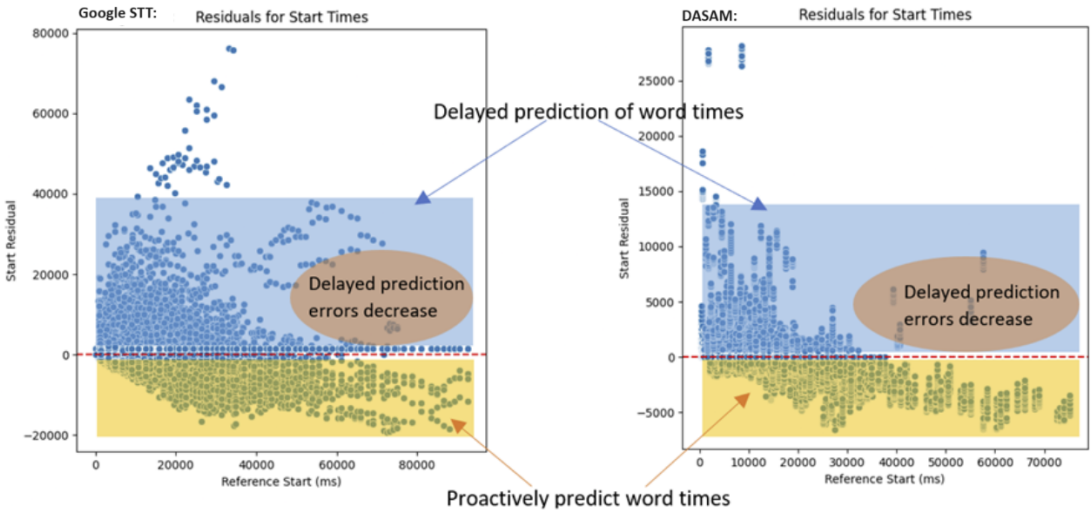


Figure 8.15: Interpreting Residual Plots: Unveiling patterns in prediction timing and systematic biases

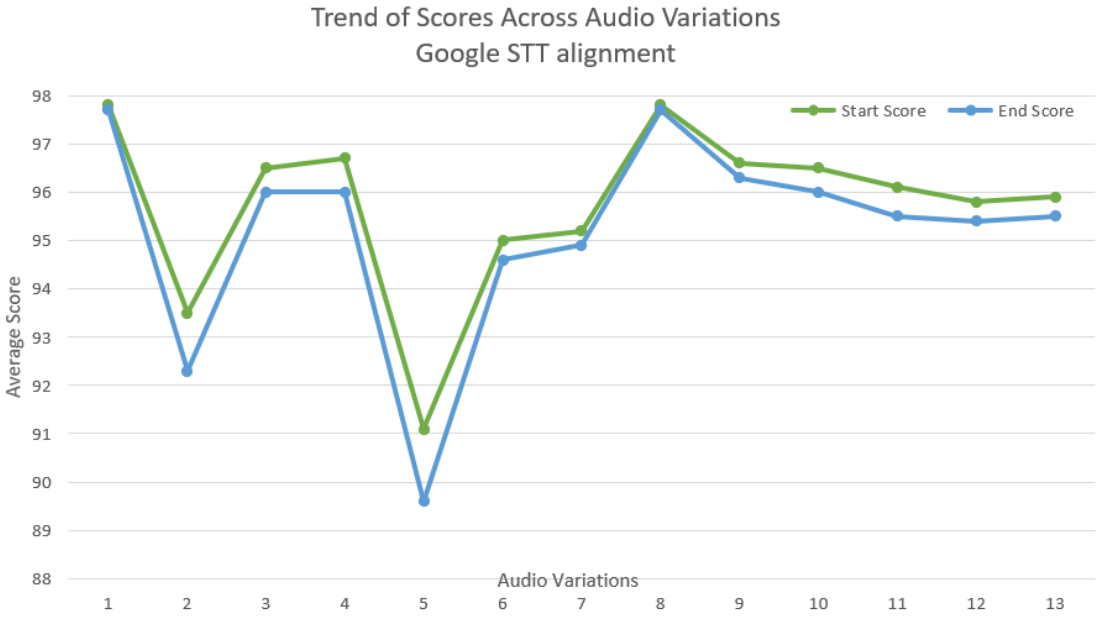


Figure 8.16: Google STT's Performance Trends.

Figure 8.16 depicts fluctuations in Google STT's performance across different audio variations. Notably, there are dips in scores for variations 2, 5, 6, and 7. This suggests that these specific audio variations might pose greater challenges for Google ASR's word-level alignment capabilities.

The chart might appear to show a slight decrease in DASAM's start score from a high of 96.6 to 95.1, while the end score seems to increase inversely from 94 to 96.7.

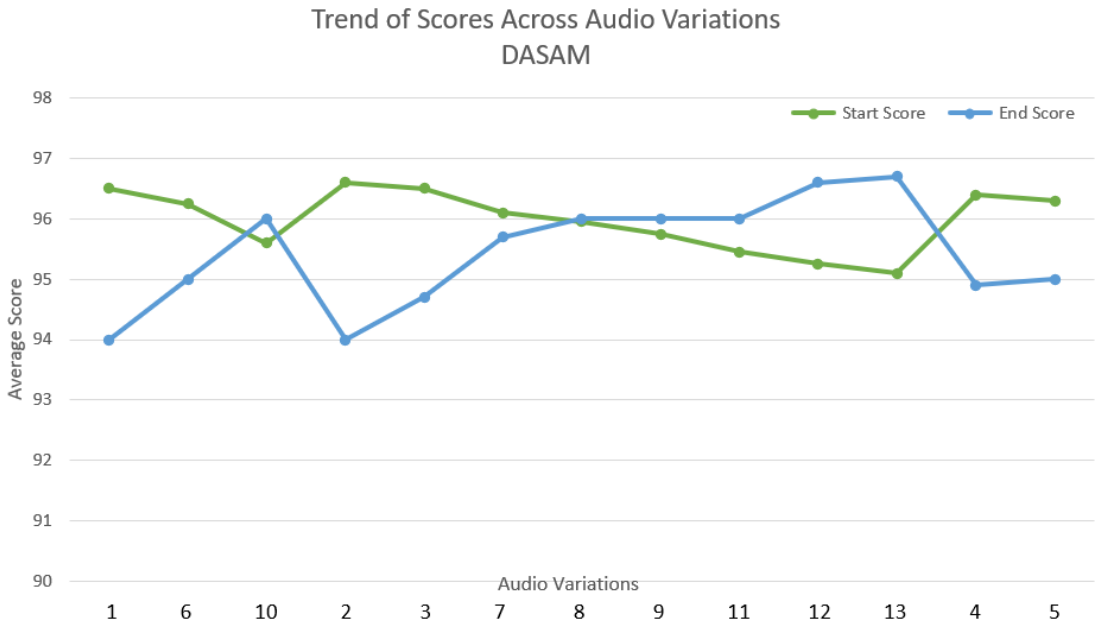


Figure 8.17: DASAM's Trend of Scores Across Audio Variations.

DASAM surpasses Google STT in word-level alignment accuracy based on the comprehensive evaluation metrics employed in this study. The analysis of statistical charts further strengthens this conclusion by revealing DASAM's consistently high performance and robust error distribution. Additionally, the paper explores interesting observations regarding how sentence length influences alignment accuracy. Overall, DASAM presents itself as a promising system for various applications that require precise word-level alignment in audio recordings for Arabic with diacritics.

8.6 Summary

This part of the dissertation has evaluated the performance of DASAM, a Word-Level Segmentation and Alignment Model for Arabic Diacritic-Based audio recordings. Our comparison with Google STT, a recognized benchmark in speech recognition, illustrates DASAM's superior accuracy. This superiority is supported by key metrics, most notably the R^2 values. DASAM's R^2 for start time is 0.959 compared to Google STT's 0.870, and for end time, DASAM scores 0.957 versus Google STT's 0.849. These values demonstrate that DASAM explains a greater proportion of variance in the actual word timings than Google STT, indicating higher accuracy and consistency in predictions.

The model also displays a higher concentration of predictions near the correct timings and exhibits less systematic bias compared to Google STT. Both systems demonstrated anticipatory behavior, likely reflective of their sensitivity to pre-speech cues, which is a common characteristic in advanced speech recognition technologies. Additionally, longer sentences enhanced alignment accuracy, particularly beneficial for DASAM in reducing late predictions. The consistent performance of DASAM across various audio variations underscores its robustness and reliability for practical applications.

Overall, this study establishes DASAM as a promising system for word-level alignment tasks, outperforming Google STT in accuracy and consistency, specifically for Arabic Diacritic-based audio recordings under the tested conditions.

Chapter 9

CONCLUSION AND FUTURE DIRECTIONS

This dissertation presents a **Multimodal Integration System** for diacritic-aware Classical Arabic language processing. The system integrates speech, text, and vision modalities to address the unique challenges of Arabic's rich linguistic features. We developed scalable databases and models such as QR-Vision for image processing, QRDiaRec for text diacritization, QRSR and DASAM for speech processing and alignment, and SemSim for semantic and numeric analysis. Our methodology includes techniques in data modeling, data quality validation, deep learning, computer vision, signal processing, and interactive data visualization. We used the Holy Quran as our case study due to its consistent accuracy and quality across multiple linguistic styles.

The proposed support system for Arabic Natural Language Processing (NLP) addresses the preservation and accessibility of Arabic heritage resources. The system focuses on the Quran and aims to create a centralized, scalable, cloud-based database. This database benefits researchers worldwide.

Specific features improve the system's effectiveness. The Text Digitizer and Analyzer Module standardizes Arabic data. The Computer Vision (CV) Text Localizer enhances text recognition accuracy. The Semantic Similarity Detection Module uncovers hidden semantic relationships among entities.

The Speech Localization, Segmentation, and Alignment Module addresses challenges in Arabic speech recognition. This leads to improved accuracy. The scalable, cloud-based database allows researchers to explore and analyze Arabic datasets. They can use direct SQL statements and APIs.

The ultimate goal is to create a comprehensive and advanced resource. This resource meets the diverse needs of Arabic and Quranic research. It also advances NLP capabilities in this domain.

Building on these achievements, future work will focus on expanding the system to cover more Arabic dialects and other widespread languages. This generalization will include both Modern Standard Arabic (MSA) and various dialects (DA).

- **Sentiment Analysis:** Implement a sentiment analysis module to determine the sentiment in Quran and Arabic texts. This will help researchers understand public opinion and emotions.
- **Topic Modeling:** Develop a topic modeling module to identify and extract underlying topics from large collections of Islamic and Arabic texts.
- **Text Summarization:** Create a text summarization module that generates concise summaries of long Arabic texts. This will facilitate quick understanding and information retrieval.
- **Language Translation:** Integrate a language translation feature to support translation between Arabic and other languages. This will promote cross-lingual communication and knowledge sharing.
- **The Partial Diacritization:** Our models recognize words that appear in multiple forms due to different linguistic styles. This capability might help accurately determine which words need diacritization and which are agreed upon across styles.Partial diacritization in Arabic facilitates reading, especially for readers with limited vision and language learners [63].
- **Language Understanding:** Use transformer-based models and other advanced techniques to improve the system's comprehension of complex Arabic language structures.

Building on these developments, we further advanced Arabic language understanding through automated diacritization and stylistic realization in Arabic textual data. Diacritics are essential in Arabic. They determine word meanings, sentence structures, and context. Our

system, QRDiaRec, uses three deep learning models. These models include bidirectional LSTM, bidirectional GRU, and transformer-based models. QRDiaRec achieves effective diacritization with accuracies between 91.52% and 92.15%.

By addressing diacritics and linguistic styles, we enhanced the interpretation of Arabic text. This improvement enables a better understanding of Arabic's complexities. Our case study on the Qur'an showed the importance of diacritics. Proper diacritization helps avoid misunderstandings and misinterpretations in automated translation systems.

This research made several contributions. We established a database of linguistic styles in the Arabic Qur'an text. We created the QRDiaRec auto-diacritical system. We recognized various Arabic dialects. We also advanced Arabic Natural Language Understanding (NLU) capabilities.

The developed system has significant potential benefits. It can enhance NLP, sentiment analysis, machine translation, speech recognition and synthesis, information retrieval, text-to-speech systems, and Arabic computational linguistics.

Although perfect accuracy may not be achieved due to multiple valid diacritization forms, our exploration of different models advances the field of Arabic text diacritization.

Ultimately, this research aims to improve language comprehension. It preserves the richness and depth of the Arabic language.

Future work for Arabic language understanding:

- **Expansion of Linguistic Styles:** Extend the system to include the 20 linguistic styles currently unavailable in the database.
- **Fine-tuning and Customization:** Implement techniques to fine-tune and customize the diacritization models for specific linguistic styles.
- **Linguistic Style Detection:** Develop a module to automatically identify and categorize the linguistic style of Arabic text.
- **Diachronic Analysis:** Conduct an analysis of Arabic texts to trace the historical

development of dialects from well-known linguistic styles.

- **Multilingual Approach:** Expand the system to handle multilingual data. Incorporate language transfer learning techniques. This will facilitate diacritization and stylistic realization for code-switched or mixed-language texts.
- **Integration with Other NLP Tasks:** Integrate the diacritization system with other NLP tasks. These tasks include sentiment analysis, named entity recognition, and part-of-speech tagging. This integration will support more comprehensive Arabic language processing.
- **Cross-Linguistic Studies:** Conduct studies to compare diacritization and stylistic realization challenges in Arabic with other languages. These studies may lead to valuable insights and improvements.

Building on these developments, we explored semantic and numeric similarities through the SemSim system.

Semantic similarity enhances our understanding of textual content. Numeric similarity provides quantitative representations for analysis. By combining these dimensions, we gain deeper insights into relationships between concepts. This enables more sophisticated and accurate analyses in NLP, recommendation systems, and machine learning tasks.

The SemSim system offers an approach to exploring similarities in multidimensional datasets. It provides tools to define entities, detect numeric similarities, and correlate them with semantic similarities in a knowledge graph. By leveraging both numeric and semantic similarities, the system uncovers hidden patterns. It enriches the understanding of complex datasets.

We confirmed a strong and significant association between numeric matches and semantic similarities in the Holy Qur'an dataset. However, the implications of this relationship in other languages and linguistic domains need further investigation. Future research can shed light on unique linguistic characteristics and their impact on natural language processing tasks across diverse languages.

Integrating semantic and numeric similarities benefits more than exploring linguistic rela-

tionships. It also extends to domains such as content clustering, word representations, recommendation systems, and machine learning tasks. The SemSim system provides a framework to harness both dimensions of similarity. It contributes to advancements in various fields and offers valuable insights for researchers and practitioners.

Looking ahead, future work should focus on expanding the SemSim system to handle large and heterogeneous datasets from different languages and domains. Further investigations can explore the generalizability of the relationship between numeric and semantic similarities across languages. Identifying unique linguistic patterns that impact similarity measures is also important. By refining and extending the SemSim system, we can unlock its potential for diverse applications. This will foster advancements in multidimensional analysis and understanding in NLP and beyond.

Future Work for the Numeric and Semantic Similarity:

- **Deep Learning Approaches:** Investigate the integration of deep learning techniques, such as transformer-based models, to enhance the detection and understanding of numeric and semantic relationships. Advanced models may improve SemSim's performance on complex and varied datasets.
- **Comparison with Existing Similarity Measures:** Conduct comprehensive comparisons of SemSim with existing similarity measures in NLP, both traditional and deep learning-based.
- **Applications in Cross-Language Information Retrieval:** Investigate how SemSim can be leveraged for cross-language information retrieval tasks. This will help users access relevant information in different languages based on numeric and semantic similarities.

We also developed systems to enhance Arabic text recognition. Our work includes two key systems: QR-Vision and AraDiaOCR. QR-Vision is designed to identify verse markers and chapter beginnings in Quranic texts. This system uses computer vision techniques to accurately detect these boundaries despite the challenges posed by handwriting styles and

diacritical marks. Proper boundary localization allows users to interact with specific verses, accessing related text, statistics, and audio features.

AraDiaOCR is an Optical Character Recognition system tailored for Arabic text with diacritics. Traditional OCR systems struggle with Arabic due to its complex script and diacritical marks. AraDiaOCR addresses these challenges by linking annotated Unicode text with handwritten images at the verse level. This approach improves recognition accuracy and handles the intricate connections between letters and words.

Together, QR-Vision and AraDiaOCR provide a comprehensive solution for digitizing and processing Quranic texts. QR-Vision localizes the text boundaries, while AraDiaOCR accurately recognizes the diacritics and characters within each verse. These systems work in tandem to enhance the accessibility and analysis of Arabic heritage texts.

Future Work for Automated Text Boundary Localization and our AraDiaOCR:

- **Applying OCR to Various Linguistic Methods:** Extend AraDiaOCR to support multiple linguistic methods beyond the current scope. This will enhance its versatility in processing different Arabic texts.
- **Expanding to General Arabic Texts:** Adapt AraDiaOCR to accept any Arabic texts, not just Quranic. This expansion will allow broader applications in diverse Arabic literary and scholarly works.
- **Improving Accuracy at Word and Letter Levels:** Enhance AraDiaOCR's accuracy by focusing on word-level and letter-level recognition. This involves refining the model to handle intricate connections and overlapping characters more effectively.
- **Handling Diverse Handwriting Styles:** Incorporate more diverse handwriting styles into the training dataset. This will improve the OCR system's ability to recognize and process various calligraphic forms.
- **Enhancing Data Augmentation Techniques:** Explore advanced data augmentation methods to increase the diversity and size of the training datasets. This will help the OCR model generalize better to different text variations.

- **Cross-Linguistic OCR Applications:** Investigate the application of AraDiaOCR's techniques to other languages with similar script complexities. This can lead to advancements in OCR technologies for a broader range of languages.

We also focused on enhancing Arabic speech recognition through the Quran Speech Recognition (QRSR) system.

The Quran Speech Recognition (QRSR) system aims to improve Arabic Automatic Speech Recognition (ASR). The system prepares a diacritic Arabic text-audio dataset. It aligns audio recordings with textual representations. This alignment creates a reliable audio-text dataset with accurate phonemes for pronunciation.

The QRSR system uses voice activity detection (VAD) techniques. VAD segments and localizes audio corresponding to Quranic verses. The system leverages diacritic-annotated text and audio recitations. Various Machine Learning (ML) algorithms are explored. These include Naïve Bayes, Support Vector Machine (SVM), and Random Forest. These algorithms enhance Arabic speech recognition in Quranic and Arabic studies.

This section's contributions include developing the QRSR system. It applies classification algorithms to align diacritic-annotated text with ASR transcripts. The FuzTPI algorithm is introduced to enhance ML models. By applying the FuzTPI-SVM approach, the QRSR system achieves a 92.2% accuracy rate. Additionally, the research expands a textual audio dataset. This expansion enables potential enhancements for Arabic speech recognition on a broader scale.

The FTARC approach is proposed to enhance audio segmentation and alignment. It achieves an accuracy rate of up to 91%. The research advances Arabic NLP and speech recognition systems. It also explores segmentation and localization techniques, particularly in Quranic studies.

Future Work for Segmenting and Localizing Diacritic-aware Arabic Audio

- **Expand the Diacritic Dataset:** Create a larger and more diverse dataset of diacritics and their associated phonemes. This dataset should cover various linguistic styles,

regional accents, and historical variations in Arabic speech. Expanding the dataset will improve the ASR system's accuracy and robustness.

- **Enhance Diacritic-Aware ASR Models:** Improve ASR models to be diacritic-aware. This will allow the models to differentiate between words with similar pronunciations but distinct diacritics, enhancing recognition accuracy.
- **Incorporate End-to-End ASR:** Explore integrating end-to-end ASR models. These models directly convert speech into diacritic-annotated text. This approach eliminates the need for intermediate steps like alignment.
- **Develop Diacritic-Aware Text-to-Speech (TTS):** Create a TTS system that correctly pronounces Arabic according to the diacritics in the text. This will ensure accurate and natural-sounding speech synthesis.
- **Explore Emotion-aware TTS:** Investigate emotion-aware TTS systems. These systems can modify speech delivery based on emotional cues in the text. This will provide more expressive and contextually appropriate speech synthesis.
- **Integrate with Existing Systems:** Combine QRSR with QR-Vision and Ara-DiaOCR for a comprehensive Arabic language processing suite. Integration will streamline workflows and enhance system capabilities.
- **Improve Real-Time Processing:** Develop real-time processing features for QRSR. This will enable immediate recognition and interaction with Arabic audio in digital formats.
- **Cross-Linguistic Applications:** Apply QRSR techniques to other languages with similar script complexities. This can lead to advancements in ASR technologies for a broader range of languages.
- **User Interface Enhancements:** Develop user-friendly interfaces for the QRSR system. Enhancing usability will make the system more accessible to researchers and practitioners.

Building on our work in sentence-level audio processing, then we focus on word-level segmentation and alignment. DASAM (Diacritic-Aware Segmentation and Alignment Model for Arabic) is a novel approach for word-level segmentation and alignment in Arabic audio. It combines linguistic analysis and Dynamic Time Warping to match reference audio words with unseen sentence audio. DASAM outperforms Google STT in predicting word timings, achieving higher accuracy in text-to-audio alignment. It's particularly valuable for applications involving diacritics in Arabic speech recognition and linguistic research. Tested on the Qur'an dataset, DASAM demonstrates significantly higher accuracy than Google Speech-to-Text (STT) for word boundary detection. DASAM's R^2 values are 0.959 for word start times and 0.957 for end times, outperforming Google STT's R^2 values of 0.870 and 0.849, respectively.

Future Work for Diacritic-Aware Segmentation and Alignment Model for Arabic.

- **Multi-dialect adaptation:** Extend DASAM to include Arabic dialects, enhancing versatility across regional speech varieties.
- **Integration with ASR:** Integrate DASAM with existing ASR systems to improve diacritic recognition and alignment accuracy.
- **Real-Time Alignment:** Adapt DASAM for real-time alignment in applications requiring immediate word segmentation.
- **Dataset Augmentation:** Develop a larger, varied Arabic dataset to test DASAM under more diverse speech conditions.
- **Error Correction Mechanisms:** Implement error correction to handle minor variations or misalignments in reference and target audio.
- **Multimodal Integration:** Integrate DASAM with visual cues from speakers for improved segmentation accuracy in video-based applications.
- **Emotion detection:** Extend DASAM to recognize emotional cues in Arabic speech.
- **Speaker diarization:** Develop DASAM's capabilities to identify and separate multi-

ple speakers.

- **Noise resilience:** Improve DASAM's performance in noisy environments.
- **Corpus expansion:** Use DASAM to build larger, more diverse Arabic speech corpora with diacritics.

These future directions aim to extend DASAM's utility and effectiveness across diverse applications in Arabic linguistic research and advanced NLP.

Future work will continue to expand these systems. We aim to cover more Arabic dialects, improve accuracy, and integrate with other NLP tasks. Our research paves the way for more sophisticated and culturally sensitive AI applications in Arabic-speaking contexts. By addressing the unique challenges of Arabic language processing, we contribute to the advancement of AI and NLP technologies in preserving and understanding the richness of Arabic heritage.

LIST OF PUBLICATIONS:

- Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. **"Auto-Diacritization and Stylistic Realization of Arabic Text using Deep Neural Networks."** Journal of Quranic Sciences and Research 5, no. 1 (2024): 12-29.
- Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. **"Arabic Diacritic-Aware Text Audio Segmentation and Alignment Model (DASAM)."** Elkawnie: Journal of Islamic Science and Technology 10, no. 1 (2024).
- Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. **"A Tapestry of Tongues: A Novel Provenance Approach for Arabic Linguistic Styles and Lineage Tracing."** Journal of Quranic Sciences and Research (JQSR) 5, no. 1 (2024).
- Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. **"Semantic Similarity Exploration in Heterogeneous Sparse Multidimensional Numeric Spaces: A Case Study of the Quran Text using SemSim."** The International Journal on Perceptive and Cognitive Computing (IJPCC) 9, no. 2 (2023): 25-32.
- Adel Sabour, Abdeltawab Hendawi, and Mohamed Ali. **"Diacritic-Aware Alignment and Classification in Arabic Speech: A Fusion of Fuzzy and ML Models."** JISTech (Journal of Islamic Science and Technology) 8, no. 2 (2023): 169-191.
- Abdulrahman Salama, Umar Siddiqui, Adel Sabour, and Mohamed Ali. **"Localization and Extraction of Qur'an Verses Using Computer Vision."** International Journal on Islamic Applications in Computer Science and Technology (IJASAT) 11, no. 3 (2023): 1 10.
- Umar Siddiqui, Habiba Youssef, Adel Sabour, and Mohamed Ali. **"Scalability, Availability, Reproducibility and Extensibility in Islamic Database Systems."** International Journal on Islamic Applications in Computer Science and Technology (IJASAT) 10, no. 1 (2022): 12-21.

BIBLIOGRAPHY

[1] 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.

[2] Samir Abdaljalil, Shaimaa Hassanein, Hamdy Mubarak, and Ahmed Abdelali. Towards generalization of machine learning models: A case study of arabic sentiment analysis. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 17, pages 971–980, 2023.

[3] Mohammed M Abdelgwad, Taysir Hassan A Soliman, Ahmed I Taloba, and Mohamed Fawzy Farghaly. Arabic aspect based sentiment analysis using bidirectional gru based models. *Journal of King Saud University-Computer and Information Sciences*, 34(9):6652–6662, 2022.

[4] Mohamed S Abdo and Ahmed H Kandil. Semi-automatic segmentation system for syllables extraction from continuous arabic audio signal. *International Journal of Advanced Computer Science and Applications*, 7(1), 2016.

[5] Mohamed S Abdo, Ahmed H Kandil, and Sahar Ali Fawzy. Mfc peak based segmentation for continuous arabic audio signal. In *2nd Middle East Conference on Biomedical Engineering*, pages 224–227. IEEE, 2014.

[6] Muhammad Abdul-Mageed, Abdelrahim Elmadany, Alcides Inciarte, Md Tawkat Islam Khondaker, et al. Jasmine: Arabic gpt models for few-shot learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 16721–16744, 2023.

[7] Sara A Aboalnaser. Machine learning algorithms in arabic text classification: A review. In *2019 12th International Conference on Developments in eSystems Engineering (DeSE)*, pages 290–295. IEEE, 2019.

[8] Ahmed Hamdi Abo Absa, Mohamed Deriche, Moustafa Elshafei-Ahmed, Yahya Mohamed Elhadj, and Biing-Hwang Juang. A hybrid unsupervised segmentation algorithm for arabic speech using feature fusion and a genetic algorithm (july 2018). *Ieee Access*, 6:43157–43169, 2018.

[9] Ahmad Abu Dalhoum and Ahmad Al-Taani. Data analytics of arabic language: A comprehensive study. *International Journal of Computer Science and Information Security (IJCSIS)*, 15(10):32–39, 2017.

[10] Shahriar Afandizadeh Zargari, Amirmasoud Memarnejad, and Hamid Mirzahossein. Novel hybrid method for travel pattern recognition based on comparison of origin-destination matrices in terms of structural similarity. *Promet-Traffic&Transportation*, 34(2):223–237, 2022.

[11] Sumayya Al-Fadhli, Hajar Al-Harbi, and Asma Cherif. Speech recognition models for holy quran recitation based on modern approaches and tajweed rules: A comprehensive overview. *International Journal of Advanced Computer Science & Applications*, 14(12), 2023.

[12] I.S. Al-Sheikh, M.A.S.N.I.Z.A.H. Mohd, and L. Warlina. A review of arabic text recognition dataset. *Asia-Pacific J. Inf. Technol. Multimedia*, 9(1):69–81, 2020.

[13] Firoj Alam, Abul Hasnat, Fatema Ahmed, Md Arid Hasan, and Maram Hasanain. Armeme: Propagandistic content in arabic memes. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024. arXiv preprint arXiv:2406.03916.

[14] Faris Alasmary, Orjuwan Zaafarani, and Ahmad Ghannam. Catt: Character-based arabic tashkeel transformer. *arXiv preprint arXiv:2407.03236*, 2024.

[15] Hanan Aldarmaki and Ahmad Ghannam. Diacritic recognition performance in arabic asr. *arXiv preprint arXiv:2302.14022*, 2023.

[16] Mohammed Aldawsari, Manjur Kolhar, and Omer Salih Dawood Omer. Within-document arabic event coreference: Challenges, datasets, approaches and future direction. *Applied Sciences*, 13(19):11004, 2023.

[17] Mansoor Alghamdi and William Teahan. Experimental evaluation of arabic ocr systems. *PSU Research Review*, 1(3):229–241, 2017.

[18] Salah Alghyaline. Arabic optical character recognition: A review. *CMES-Computer Modeling in Engineering & Sciences*, 135(3), 2023.

[19] Bashar Alhafni, Go Inoue, Christian Khairallah, and Nizar Habash. Advancements in arabic grammatical error detection and correction: An empirical investigation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv preprint arXiv:2305.14734.

[20] ZK Alkayyali, SAB Idris, and SAMY S Abu-Naser. A new algorithm for audio files augmentation. *Journal of Theoretical and Applied Information Technology*, 101(12), 2023.

[21] M. Almazrooie, A. Samsudin, A.A.A. Gutub, M.S. Salleh, M.A. Omar, and S.A. Hassan. Integrity verification for digital holy quran verses using cryptographic hash function and compression. *Journal of King Saud University-Computer and Information Sciences*, 32(1):24–34, 2020.

[22] Ammar Mohammed Ali Alqadasi, Rawad Abdulghafor, Mohd Shahrizal Sunar, and Md Sah Bin HJ Salam. Modern standard arabic speech corpora: a systematic review. *Ieee Access*, 11:55771–55796, 2023.

[23] Chouki Amazouz, Mohamed Khalidi Idrissi, Karim Afdel, and Mohammed Hafidi. Digital preservation of arabic heritage books: A review of the state of the art. *Journal of King Saud University-Computer and Information Sciences*, 32(9):1111–1119, 2020.

[24] Anthropic. Claude 3 opus, 2024. Accessed October 29, 2024.

[25] Muhammad Nabeel Asim, Muhammad Wasim, Muhammad Usman Ghani Khan, Waqar Mahmood, and Hafiza Mahnoor Abbasi. A survey of ontology learning techniques and applications. *Database*, 2018, 2018.

[26] Ahmad Mustaffa Azmi and Abdulrahman Alsaiari. A calligraphic based scheme to justify arabic text improving readability and comprehension. *Computers in Human Behavior*, 39:177–186, 2014.

[27] Tobias Baer and Vishnu Kamalnath. Controlling machine-learning algorithms and their biases. *McKinsey Insights*, 2017.

[28] Halit Bakır, Ayşe Nur Çayır, and Tuğba Selcen Navruz. A comprehensive experimental study for analyzing the effects of data augmentation techniques on voice classification. *Multimedia Tools and Applications*, 83(6):17601–17628, 2024.

[29] Nazik O'mar Balula, Mohsen Rashwan, and Shrief Abdou. Automatic speech recognition (asr) systems for learning arabic language and al-quran recitation: a review. *International Journal of Computer Science and Mobile Computing*, 10(7):91–100, 2021.

[30] Martijn Bartelds, Nay San, Bradley McDonnell, Dan Jurafsky, and Martijn Wieling. Making more of little data: Improving low-resource automatic speech recognition using data augmentation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. arXiv preprint arXiv:2305.10951.

[31] Muhammad Huzaifa Bashir, Aqil M Azmi, Haq Nawaz, Wajdi Zaghouani, Mona Diab, Ala Al-Fuqaha, and Junaid Qadir. Arabic natural language processing for qur'anic research: A systematic review. *Artificial Intelligence Review*, 56(7):6801–6854, 2023.

[32] Daniel Mesafint Belete and Manjaiah D Huchaiah. Grid search in hyperparameter optimization of machine learning models for prediction of hiv/aids test results. *International Journal of Computers and Applications*, 44(9):875–886, 2022.

[33] 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.

[34] Elisa Bertino, Chenyun Dai, and Murat Kantarcioglu. The challenge of assuring data trustworthiness. In *Database Systems for Advanced Applications: 14th International Conference, DASFAA 2009, Brisbane, Australia, April 21-23, 2009. Proceedings 14*, pages 22–33. Springer, 2009.

[35] Kaushal Bhogale, Abhigyan Raman, Tahir Javed, Sumanth Doddapaneni, Anoop Kunchukuttan, Pratyush Kumar, and Mitesh M Khapra. Effectiveness of mining audio and text pairs from public data for improving asr systems for low-resource languages. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[36] Amanda L Botticello, Lauren Murphy, Susan Charlifue, Yuying Chen, John Corrigan, Simon Driver, CB Eagye, Jereme Wilroy, and Anthony Lequerica. Development of composite measures of neighborhood economic factors for use in spinal cord injury outcomes studies: A spinal cord injury model systems database study. *Archives of physical medicine and rehabilitation*, 105(11):2118–2126, 2024.

[37] Naaima Boudad, Rdouan Faizi, Rachid Oulad Haj Thami, and Raddouane Chiheb. Sentiment classification of arabic tweets: a supervised approach. *Journal of Mobile Multimedia*, pages 233–243, 2017.

[38] GHIZLANE Bourahouat, MANAR Abourezq, and NAJIMA Daoudi. Systematic re-

view of the arabic natural language processing: challenges, techniques and new trends. *Journal of Theoretical and Applied Information Technology*, 101(3), 2023.

[39] Pengshan Cai, Wei Li, Yansong Feng, Yuanzhuo Wang, and Yantao Jia. Learning knowledge representation across knowledge graphs. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[40] Albert PC Chan, Daniel WM Chan, and John FY Yeung. Overview of the application of "fuzzy techniques" in construction management research. *Journal of construction engineering and management*, 135(11):1241–1252, 2009.

[41] Dhivya Chandrasekaran and Vijay Mago. Evolution of semantic similarity—a survey. *ACM Computing Surveys (CSUR)*, 54(2):1–37, 2021.

[42] Khansa Chemnad and Achraf Othman. Advancements in arabic text-to-speech systems: A 22-year literature review. *IEEE Access*, 2023.

[43] Farheen Chincholi and Harald Koestler. Detectron2 for lesion detection in diabetic retinopathy. *Algorithms*, 16(3):147, 2023.

[44] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[45] Google Cloud. Cloud speech-to-text api - language support, 2024. Accessed: 2024-10-29.

[46] Google Cloud. Supported languages - cloud vision api documentation, 2024. Accessed: 2024-10-29.

[47] King Fahd Complex. King fahd glorious qur'an printing complex. https://www.my.gov.sa/. Accessed: Jan 17, 2023.

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

[49] Guy Dar, Mor Geva, Ankit Gupta, and Jonathan Berant. Analyzing transformers in embedding space. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. arXiv preprint arXiv:2209.02535.

[50] Saad Mohamed Darwish and Khaled Osama Elzoghaly. An enhanced offline printed arabic ocr model based on bio-inspired fuzzy classifier. *IEEE Access*, 8:117770–117781, 2020.

[51] Ayan Das, Raghuveer Chanda, Smriti Agrawal, and Sourangshu Bhattacharya. Distributed weighted parameter averaging for svm training on big data. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[52] Marco De Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. Semantics-aware content-based recommender systems. *Recommender systems handbook*, pages 119–159, 2015.

[53] David Dean, Sridha Sridharan, Robert Vogt, and Michael Mason. The qut-noise-timit corpus for evaluation of voice activity detection algorithms. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, pages 3110–3113. International Speech Communication Association, 2010.

[54] 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.

[55] 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.

[56] Papa Senghane Diouf, Aliou Boly, and Samba Ndiaye. Variety of data in the etl processes in the cloud: State of the art. In *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, pages 1–5. IEEE, 2018.

[57] Tausif Diwan, G Anirudh, and Jitendra V Tembhurne. Object detection using yolo: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, 82(6):9243–9275, 2023.

[58] Phong Nguyen-Thuan Do, Son Quoc Tran, Phu Gia Hoang, Kiet Van Nguyen, and Ngan Luu-Thuy Nguyen. Vlue: A new benchmark and multi-task knowledge transfer learning for vietnamese natural language understanding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2024. arXiv preprint arXiv:2403.15882.

[59] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and Jakob Uszkoreit. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

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

[61] S.R. El-Beltagy and A. Rafea. Qdetect: An intelligent tool for detecting quranic verses in any text. *Procedia Computer Science*, 189:374–381, 2021.

[62] Yousif A El-Imam. Phonetization of arabic: rules and algorithms. *Computer Speech & Language*, 18(4):339–373, 2004.

[63] Muhammad ElNokrashy and Badr Alkhamissi. A context-contrastive inference approach to partial diacritization. In *Proceedings of The Second Arabic Natural Language Processing Conference*, pages 89–101, 2024.

[64] Khaled Nasser ElSayed. An arabic natural language interface system for a database of the holy quran. *International Journal of Advanced Research in Artificial Intelligence*, 4(7):9–14, 2015.

[65] 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.

[66] 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.

[67] Safiullah Faizullah, Muhammad Sohaib Ayub, Sajid Hussain, and Muhammad Asad Khan. A survey of ocr in arabic language: applications, techniques, and challenges. *Applied Sciences*, 13(7):4584, 2023.

[68] 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.

[69] Mohammad Fasha, Bassam Hammo, Nadim Obeid, and Jabir Widian. A hybrid deep learning model for arabic text recognition. *arXiv preprint arXiv:2009.01987*, 2020.

[70] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. Learning semantic hierarchies via word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1199–1209, 2014.

[71] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

[72] Shang Gao, Gabriel Antoniu, Ladislau Bölöni, and Florence Sèdes. A survey of data and process provenance. *ACM Computing Surveys (CSUR)*, 48(1):8, 2016.

[73] Yuan Gao, Feng Hou, and Ruili Wang. A novel two-step fine-tuning framework for transfer learning in low-resource neural machine translation. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3214–3224, 2024.

[74] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[75] Sunila Gollapudi and Sunila Gollapudi. Opencv with python. *Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs*, pages 31–50, 2019.

[76] Google DeepMind. Gemini ultra, 2024. Accessed October 29, 2024.

[77] Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[78] Jie Gu and Shan Lu. An effective intrusion detection approach using svm with naïve bayes feature embedding. *Computers & Security*, 103:102158, 2021.

[79] Imane Guellil, Houda Saâdane, 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.

[80] Nizar Y Habash. Introduction to arabic natural language processing. *Synthesis lectures on human language technologies*, 3(1):1–187, 2010.

[81] Nizar Y Habash. *Introduction to Arabic natural language processing*. Springer Nature, 2022.

[82] Zhen Hai, Gao Cong, Kuiyu Chang, Peng Cheng, and Chunyan Miao. Analyzing sentiments in one go: A supervised joint topic modeling approach. *IEEE Transactions on Knowledge and Data Engineering*, 29(6):1172–1185, 2017.

[83] S. Hakak, A. Kamsin, J. Veri, R. Ritonga, and T. Herawan. A framework for authentication of digital quran. In *Information Systems Design and Intelligent Applications: Proceedings of Fourth International Conference INDIA 2017*, pages 752–764. Springer Singapore, 2018.

[84] Salima Harrat, Karima Meftouh, and Kamel Smaili. Creating parallel arabic dialect corpus: pitfalls to avoid. In *18th International Conference on Computational Linguistics and Intelligent Text Processing (CICLING)*, 2017.

[85] Thomas C Hassett, Greta Stuhlsatz, and John E Snyder. A scoping review and assessment of the area-level composite measures that estimate social determinants of health across the united states. *Public Health Reports®*, page 00333549241252582, 2024.

[86] Thomas Hegghammer. Ocr with tesseract, amazon textract, and google document ai: a benchmarking experiment. *Journal of Computational Social Science*, 5(1):861–882, 2022.

[87] Ahmed M Hendawi, Dave Hazel, Jim Larson, Yan Li, Daniel Trummert, Mohammad Ali, and Ankur Teredesai. Amadeus: a system for monitoring water quality parameters and predicting contaminant paths. In *IEMSS*, 2014.

[88] Enrique Herrera-Viedma, Francisco Javier Cabrerizo, Janusz Kacprzyk, and Witold Pedrycz. A review of soft consensus models in a fuzzy environment. *Information Fusion*, 17:4–13, 2014.

[89] Gisli R Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems (TODS)*, 28(4):517–580, 2003.

[90] Hongren Huang, Chen Li, Xutan Peng, Lifang He, Shu Guo, Hao Peng, Lihong Wang, and Jianxin Li. Cross-knowledge-graph entity alignment via relation prediction. *Knowledge-Based Systems*, 240:107813, 2022.

[91] Huang Huang, Fei Yu, Jianqing Zhu, Xuening Sun, Hao Cheng, Dingjie Song, Zhihong Chen, Abdulmohsen Alharthi, Bang An, Juncai He, et al. Acegpt: Localizing large language models in arabic. In *Proceedings of the 2023 Conference of the North Amer-*

ican Chapter of the Association for Computational Linguistics, 2023. arXiv preprint arXiv:2309.12053.

[92] Yue Huang. Incorporating domain ontology information into clustering in heterogeneous networks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(4):e1413, 2021.

[93] TensorFlow Hub. Tensorflow. *Preuzeto s https://www. tensorflow. org/hub*, 2023.

[94] Mohammad Ali Humayun, Hayati Yassin, and Pg Emeroylariffion Abas. Dialect classification using acoustic and linguistic features in arabic speech. *IAES International Journal of Artificial Intelligence*, 12(2):739, 2023.

[95] Ihab F Ilyas, Xu Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

[96] Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. Language modeling with deep transformers. *arXiv preprint arXiv:1905.04226*, 2019.

[97] Aminul Islam and Diana Inkpen. Second order co-occurrence pmi for determining the semantic similarity of words. In *LREC*, pages 1033–1038, 2006.

[98] Md Saiful Islam, Fazla Elahi Md Jubayer, and Syed Ikhtiar Ahmed. A support vector machine mixed with tf-idf algorithm to categorize bengali document. In *2017 international conference on electrical, computer and communication engineering (ECCE)*, pages 191–196. IEEE, 2017.

[99] 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.

[100] M. Jannah and A.A. Nababan. Harfu jar detection system in al-quran using pierce similarity algorithm as a basic learning media of arabic language. In *2020 3rd Inter-*

national Conference on Mechanical, Electronics, Computer, and Industrial Technology (MECnIT), pages 349–354. IEEE, June 2020.

[101] Muhammad Javed, Mirza Muhammad Ali Baig, and Saad Ahmed Qazi. Unsupervised phonetic segmentation of classical arabic speech using forward and inverse characteristics of the vocal tract. *Arabian Journal for Science and Engineering*, 45:1581–1597, 2020.

[102] Mingyang Jiang, Yanchun Liang, Xiaoyue Feng, Xiaojing Fan, Zhili Pei, Yu Xue, and Renchu Guan. Text classification based on deep belief network and softmax regression. *Neural Computing and Applications*, 29:61–70, 2018.

[103] Chun-Xia Jin and Qiu-Chan Bai. Text clustering algorithm based on the graph structures of semantic word co-occurrence. In *2016 international conference on information system and artificial intelligence (ISAI)*, pages 497–502. IEEE, 2016.

[104] M.N. Kadir, R. Azmi, M.F.M. Saad, N.Z.N. Zainol, and M.A.M. Yunus. A review on the manuscript of turkish quran ywj68197msi. 4 in database management system (dbms). *Journal Of Quranic Sciences And Research*, 1(1):34–41, 2020.

[105] Hong Jin Kang, Fabrice Harel-Canada, Muhammad Ali Gulzar, Violet Peng, and Miryung Kim. Human-in-the-loop synthetic text data inspection with provenance tracking. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2024. arXiv preprint arXiv:2404.18881.

[106] Emdad Khan. Machine learning algorithms for natural language semantics and cognitive computing. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1146–1151. IEEE, 2016.

[107] VV Khodorovskii. On normalization of relations in relational databases. *Programming and Computer Software*, 28:41–52, 2002.

[108] Md Tawkat Islam Khondaker, Abdul Waheed, El Moatez Billah Nagoudi, and Muhammad Abdul-Mageed. Gptaraeval: A comprehensive evaluation of chatgpt on arabic nlp. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv preprint arXiv:2305.14976.

[109] Benjamin Kiessling, Gennady Kurin, Matthew Thomas Miller, Kader Smail, and Matthew Miller. Advances and limitations in open source arabic-script ocr: A case study. *Digital Studies/Le champ numérique*, 11(1), 2021.

[110] Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng. Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, 18(11):1457–1466, 2006.

[111] Armen Kostanyan. Fuzzy string matching with finite automat. In *2017 Computer Science and Information Technologies (CSIT)*, pages 9–11, 2017.

[112] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[113] F. Kurniawan, M.S. Khalil, M.K. Khan, and Y.M. Alginahi. Authentication and tamper detection of digital holy quran images. In *2013 International Symposium on Biometrics and Security Technologies*, pages 291–296. IEEE, July 2013.

[114] Ghania Larbi. Voice search in the holy quran. In *2013 Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences*, pages 413–418. IEEE, 2013.

[115] Bu-Sung Lee and Tai-Myoung Kim. Ensuring trustworthy data provenance: Challenges and solutions. *Journal of Network and Computer Applications*, 87:98–109, 2017.

[116] Minghui Liao, Baoguang Shi, Xiang Bai, Xinggang Wang, and Wenyu Liu. Textboxes: A fast text detector with a single deep neural network. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[117] Chee Sun Liew, Assad Abbas, Prem Prakash Jayaraman, Teh Ying Wah, Samee U Khan, et al. Big data reduction methods: a survey. *Data Science and Engineering*, 1(4):265–284, 2016.

[118] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[119] Bin Liu, Wencang Zhao, and Qiaoqiao Sun. Study of object detection based on faster r-cnn. In *2017 Chinese Automation Congress (CAC)*, pages 6233–6236. IEEE, 2017.

[120] Jingwei Liu, Zhe Wang, Jianmin Zhang, and Enhong Chen. A survey of knowledge graph embedding: approaches and applications. In *2021 IEEE 23rd International Conference on High Performance Computing and Communications; IEEE 19th International Conference on Smart City; IEEE 7th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1182–1187. IEEE, 2021.

[121] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Brian Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[122] Zheyuan Liu, Weixuan Sun, Yicong Hong, Damien Teney, and Stephen Gould. Bi-directional training for composed image retrieval via text prompt learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 5753–5762, 2024.

[123] Zhijie Liu, Xueqiang Lv, Kun Liu, and Shuicai Shi. Study on svm compared with the other text classification methods. In *2010 Second international workshop on education technology and computer science*, volume 1, pages 219–222. IEEE, 2010.

[124] Gong Lixia and Wang Junyi. Research on collaborative filtering recommendation algorithm for improving user similarity calculation. In *Proceedings of the 2021 1st International Conference on Control and Intelligent Robotics*, pages 331–336, 2021.

[125] Nitin N Lokhande, Navnath S Nehe, and Pratap S Vikhe. Voice activity detection algorithm for speech recognition applications. In *IJCA Proceedings on International Conference in Computational Intelligence (ICCIA2012), vol. iccia*, volume 6, pages 1–4, 2012.

[126] Shangbang Long, Jiaqiang Ruan, Wenjie Zhang, Xin He, Wenhao Wu, and Cong Yao. Textsnake: A flexible representation for detecting text of arbitrary shapes. In *Proceedings of the European conference on computer vision (ECCV)*, pages 20–36, 2018.

[127] 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.

[128] Fumin Ma, Ting Gong, Xintao Zhang, and Jie Lv. Rough k-prototypes clustering algorithm based on otc similarity and between-cluster information for mixed data. In *2022 34th Chinese Control and Decision Conference (CCDC)*, pages 5105–5111. IEEE, 2022.

[129] Amogh Mahapatra, Nisheeth Srivastava, and Jaideep Srivastava. Contextual anomaly detection in text data. *Algorithms*, 5(4):469–489, 2012.

[130] 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.

[131] S.L. Marie-Sainte, N. Alalyani, S. Alotaibi, S. Ghouzali, and I. Abunadi. Arabic natural language processing and machine learning-based systems. *IEEE Access*, 7:7011–7020, 2018.

[132] Meta. Llama 3, 2024. Accessed October 29, 2024.

[133] Microsoft. Language support - speech service - azure ai services, 2024. Accessed: 2024-10-29.

[134] Microsoft. Language support for computer vision - azure cognitive services, 2024. Accessed: 2024-10-29.

[135] 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.

[136] W M Muhammad, R Muhammad, A Muhammad, and AM Martinez-Enriquez. Voice content matching system for quran readers. In *2010 Ninth Mexican International Conference on Artificial Intelligence*, pages 148–153. IEEE, 2010.

[137] 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.

[138] Hillel Ofek. Why the arabic world turned away from science. *The New Atlantis*, pages 3–23, 2011.

[139] Devi Oktaviani, Moch Arif Bijaksana, and Ibnu Asror. Building a database of recurring text in the quran and its translation. *Procedia Computer Science*, 157:125–133, 2019.

[140] OpenAI. GPT-4 Technical Report. Technical report, OpenAI, Mar 2023.

[141] OpenAI. Gpt-4, 2024. Accessed October 29, 2024.

[142] Tim Pattison. Interactive visualization of formal concept lattices. In *ED/GViP@ Diagrams*, pages 78–89. Citeseer, 2014.

[143] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[144] Beatriz Pérez, Julio Rubio, and Carlos Sáenz-Adán. A systematic review of provenance systems. *Knowledge and Information Systems*, 57:495–543, 2018.

[145] Yurika Permanasari, Erwin H Harahap, and Erwin Prayoga Ali. Speech recognition using dynamic time warping (dtw). *Journal of physics: Conference series*, 1366(1):012091, 2019.

[146] Moritz Plenz, Juri Opitz, Philipp Heinisch, Philipp Cimiano, and Anette Frank. Similarity-weighted construction of contextualized commonsense knowledge graphs for knowledge-intense argumentation tasks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. arXiv preprint arXiv:2305.08495.

[147] Colin Puri, Doo Soon Kim, Peter Z Yeh, and Kunal Verma. Implementing a data lineage tracker. In *Data Warehousing and Knowledge Discovery: 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings 14*, pages 390–403. Springer, 2012.

[148] Aziz Qaroush, Abdalkarim Awad, Abualsoud Hanani, Khader Mohammad, Basam Jaber, and Ala Hasheesh. Learning-free, divide and conquer text-line extraction algorithm for printed arabic text with diacritics. *Journal of King Saud University-Computer and Information Sciences*, 34(9):7699–7709, 2022.

[149] Hiba Qasim and Huda Abdulaali Abdulbaqi. Arabic speech recognition using deep

learning methods: Literature review. In *AIP Conference Proceedings*, volume 2398, page 050029. AIP Publishing LLC, 2022.

[150] Xiaoye Qu, Yingjie Gu, Qingrong Xia, Zechang Li, Zhefeng Wang, and Baoxing Huai. A survey on arabic named entity recognition: Past, recent advances, and future trends. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[151] Siti Fairuz Mat Radzi, Muhammad Khalis Abdul Karim, M Iqbal Saripan, Mohd Amiruddin Abd Rahman, Iza Nurzawani Che Isa, and Mohammad Johari Ibahim. Hyperparameter tuning and pipeline optimization via grid search method and tree-based automl in breast cancer prediction. *Journal of Personalized Medicine*, 11(10):978, 2021.

[152] Ashifur Rahman, Md Mohsin Kabir, Muhammad Firoz Mridha, Mohammed Alatiyyah, Haifa F Alhasson, and Shuaa S Alharbi. Arabic speech recognition: Advancement and challenges. *IEEE Access*, 2024.

[153] Pradeep Rai and Shubha Singh. A survey of clustering techniques. *International Journal of Computer Applications*, 7(12):1–5, 2010.

[154] Joel Rajesh, Jan Sorensen, and Deborah A McNamara. Composite quality measures of abdominal surgery at a population level: systematic review. *BJS open*, 7(6):zrad082, 2023.

[155] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, volume 28, 2015.

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

[157] R. Rizal, F. Fadlisyah, M. Muhathir, and M. Muammar. Detection system tajwid al quran on image using bray curtis distance. *International Journal of Computing and Technology*, 2(8):32–38, 2015.

[158] Oscar Romero and Alberto Abelló. A survey of multidimensional modeling methodologies. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(2):1–23, 2009.

[159] Nick Roussopoulos, Stephen Kelley, and Frederic Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, 1995.

[160] Asma Salsabila Muhmad Rusli, Farida Ridzuan, Zulkifly Mohd Zaki, M Norazizi Sham Mohd Sayuti, and Rosalina Abdul Salam. A systematic review on semantic-based ontology for quranic knowledge. *International Journal of Engineering and Technology (UAE)*, 2018.

[161] 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.

[162] T. Sabbah and A. Selamat. A framework for quranic verses authenticity detection in online forum. In *2013 Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences*, pages 6–11. IEEE, December 2013.

[163] Adel Sabour and Mohamed Ali. Quran research, 2024. Accessed: November 10, 2024.

[164] Andreas Schreiber, Lynn von Kurnatowski, Annika Meinecke, and Claas de Boer. Visualization of software development provenance. In *International Conference on Human-Computer Interaction*, pages 121–139. Springer, 2024.

[165] Amazon Web Services. Amazon transcribe, 2024. Accessed: 2024-10-01.

[166] Amazon Web Services. Supported languages and language-specific features - amazon transcribe, 2024.

[167] Amazon Web Services. What is amazon textract?, 2024. Accessed: 2024-09-01.

[168] 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.

[169] Su-Jin Shin and Il-Chul Moon. Guided htm: Hierarchical topic model with dirichlet forest priors. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):330–343, 2016.

[170] Yogesh L Simmhan, Beth Plale, Dennis Gannon, et al. A survey of data provenance techniques. *Computer Science Department, Indiana University, Bloomington IN*, 47405:69, 2005.

[171] Pramod Singh. Deploy machine learning models to production. *Cham, Switzerland: Springer*, 2021.

[172] Adriana Stan, Yoshitaka Mamiya, Junichi Yamagishi, Peter Bell, Oliver Watts, Robert AJ Clark, and Simon King. Alisa: An automatic lightly supervised speech segmentation and alignment tool. *Computer Speech & Language*, 35:116–133, 2016.

[173] Pavel Stefanovič, Olga Kurasova, and Rokas Štrimaitis. The n-grams based text similarity detection approach using self-organizing maps and similarity measures. *Applied sciences*, 9(9):1870, 2019.

[174] Yanxiong Sun, Yeli Li, Qingtao Zeng, and Yuning Bian. Application research of text classification based on random forest algorithm. In *2020 3rd International Conference*

on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE), pages 370–374. IEEE, 2020.

[175] Katrina Sundus, Fatima Al-Haj, and Bassam Hammo. A deep learning approach for arabic text classification. In *2019 2nd International Conference on New Trends in Computing Sciences (ICTCS)*, pages 1–7. IEEE, 2019.

[176] Amril Syalim, Takashi Nishide, and Kouichi Sakurai. Preserving integrity and confidentiality of a directed acyclic graph model of provenance. In *Data and Applications Security and Privacy XXIV: 24th Annual IFIP WG 11.3 Working Conference, Rome, Italy, June 21-23, 2010. Proceedings 24*, pages 311–318. Springer, 2010.

[177] Bashar Talafha, Karima Kadaoui, Samar Mohamed Magdy, Mariem Habiboullah, Chafei Mohamed Chafei, Ahmed Oumar El-Shangiti, Hiba Zayed, Rahaf Alhamouri, Rwaa Assi, Aisha Alraeesi, et al. Casablanca: Data and models for multidialectal arabic speech recognition. *arXiv preprint arXiv:2410.04527*, 2024.

[178] Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv preprint arXiv:2310.07096.

[179] CVAT.ai Corporation team. Cvat - computer vision annotation tool. https://www.cvat.ai/.

[180] Marko Tkalcic, Matevz Kunaver, Jurij Tasic, and Andrej Košir. Personality based user similarity measure for a collaborative recommender system. In *Proceedings of the 5th Workshop on Emotion in Human-Computer Interaction-Real world challenges*, pages 30–37, 2009.

[181] Jingxuan Tu, Keer Xu, Liulu Yue, Bingyang Ye, Kyeongmin Rim, and James Pustejovsky. Linguistically conditioned semantic textual similarity. In *Proceedings of the*

Annual Meeting of the Association for Computational Linguistics (ACL), 2024. arXiv preprint arXiv:2406.03673.

[182] Nazanin Vafaei, Rita A Ribeiro, and Luis M Camarinha-Matos. Normalization techniques for multi-criteria decision making: analytical hierarchy process case study. In *doctoral conference on computing, electrical and industrial systems*, pages 261–269. Springer, 2016.

[183] 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.

[184] Ahlam Wahdan, Sendeyah Hantoobi, Said A Salloum, and Khaled Shaalan. A systematic review of text classification research based on deep learning models in arabic language. *Int. J. Electr. Comput. Eng*, 10(6):6629–6643, 2020.

[185] Tomasz Walkowiak and Mateusz Gniewkowski. Evaluation of vector embedding models in clustering of text documents. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1304–1311, 2019.

[186] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*, 2023.

[187] 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.

[188] Guangyong Wei, Zhikui Duan, Shiren Li, Xinmei Yu, and Guangguang Yang. Lfeformer: Local feature enhancement using sliding window with deformability for automatic speech recognition. *IEEE Signal Processing Letters*, 30:180–184, 2023.

[189] Di Wu, Ruixin Yang, and Chao Shen. Sentiment word co-occurrence and knowledge pair feature extraction based lda short text clustering algorithm. *Journal of Intelligent Information Systems*, 56:1–23, 2021.

[190] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. 2019.

[191] Shuo Xu, Yan Li, and Zheng Wang. Bayesian multinomial naïve bayes classifier to text classification. In *Advanced Multimedia and Ubiquitous Engineering: MUE/FutureTech 2017 11*, pages 347–352. Springer, 2017.

[192] Guangxu Xun, Yaliang Li, Wayne Xin Zhao, Jing Gao, and Aidong Zhang. A correlated topic model using word embeddings. In *IJCAI*, volume 17, pages 4207–4213, 2017.

[193] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Craft objects from images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6043–6051, 2016.

[194] 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.

[195] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, et al. Multilingual universal sentence encoder for semantic retrieval. *arXiv preprint arXiv:1907.04307*, 2019.

[196] Liang-Chih Yu, Jheng-Long Wu, Pei-Chann Chang, and Hsuan-Shou Chu. Using a contextual entropy model to expand emotion words and their intensity for the sentiment classification of stock market news. *Knowledge-Based Systems*, 41:89–97, 2013.

[197] Changtong Zan, Liang Ding, Li Shen, Yibing Zhen, Weifeng Liu, and Dacheng Tao. Building accurate translation-tailored llms with language aware instruction tuning. *arXiv preprint arXiv:2403.14399*, 2024.

[198] 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.

[199] Shengnan Zhang, Yan Hu, and Guangrong Bian. Research on string similarity algorithm based on levenshtein distance. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 2247–2251. IEEE, 2017.

[200] 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.

[201] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. Comparison of different implementations of mfcc. *Journal of Computer science and Technology*, 16:582–589, 2001.

[202] Jing Zheng, Horacio Franco, and Andreas Stolcke. Modeling word-level rate-of-speech variation in large vocabulary conversational speech recognition. *Speech Communication*, 41(2-3):273–285, 2003.

[203] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. East: an efficient and accurate scene text detector. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 5551–5560, 2017.

[204] Yueting Zhuang, Hanqi Wang, Jun Xiao, Fei Wu, Yi Yang, Weiming Lu, and Zhongfei Zhang. Bag-of-discriminative-words (bodw) representation via topic modeling. *IEEE transactions on knowledge and data engineering*, 29(5):977–990, 2017.