Cairo University

Faculty of Engineering

Credit Hours System – Senior Level

CMPS454 Natural Language Processing Fall 2025

CMPS454 : Natural Language Processing Project

Done By:

Belal Ashraf Sabbaq	1210048
Abdelrahman Ashraf	1210014
Mohamed Ayman	1210388
Yusuf Ayman	1210383

This detailed technical report outlines the development, implementation, and evaluation of a high-performance Arabic Diacritization system. By leveraging three distinct architectural paradigms—**BiRNN**, **BiLSTM**, and **Transformer-based (AraBERT)**—our team explored the optimal balance between linguistic depth and computational efficiency.

Technical Analysis of Multi-Modal Arabic Diacritization Pipelines

---

Abstract

Arabic diacritization is a critical task for downstream NLP applications, including Text-to-Speech (TTS) and Machine Translation. This project implements a comprehensive pipeline that integrates unsupervised semantic clustering (Word2Vec + K-Means) with deep learning architectures. We evaluate the impact of handcrafted features, such as word-boundary flags, against large-scale pre-trained contextual embeddings.

I. Data Preprocessing & Linguistic Normalization

The preprocessing phase addresses the high variance in Arabic script. We implemented a robust cleaning engine to ensure data consistency across all models.

1. Normalization and Orthographic Standardization

To combat vocabulary sparsity, we applied **Alif Normalization**, converting all decorated forms ($آ, أ, إ, ٱ$) to a plain Alif ($ا$). This prevents the model from treating different orthographic styles of the same semantic root as distinct entities. We utilized Unicode **NFKC normalization** to standardize character representations.

2. Target State Extraction

A specialized "Look-ahead" algorithm was developed to separate base characters from their diacritics. Unlike simpler languages, Arabic requires a complex mapping logic:

- **Compound Labels:** The logic recognizes "Shadda" combinations (e.g., Shadda + Fatha) as a single target state.
- **Null Labels:** Characters without diacritics are assigned a <NO_DIAC> token, ensuring a one-to-one alignment between input characters and output predictions.

II. Feature Engineering & Multi-Modal Fusion

The core strength of our statistical pipelines (BiRNN/BiLSTM) lies in the **Feature Fusion** layer. We define the input vector V for each character as a concatenation of four distinct information streams:

V = [E_{char} + E{word} + E{cluster} + F{last}]

1. Component Breakdown

- **Character Embeddings (E{char}):** 128-dimensional learned vectors capturing local morphology.
- **Word Embeddings (E{word}):** 256-dimensional vectors providing lexical memory for frequently occurring words.
- **Semantic Cluster Embeddings (E{cluster}):** 32-dimensional vectors derived from **K-Means clustering (K=50)** on Word2Vec representations. These act as "Pseudo-POS tags," helping the model generalize patterns to rare words.
- **Last Char Flag (F{last}):** A binary indicator of word boundaries, essential for distinguishing between internal morphology (*Bina'*) and syntactic case markings (*I'rab*).

III. Architectural Paradigms

III. Detailed Architectural Paradigms

The project focuses on two distinct technical philosophies: a **Feature-Engineering-First** approach using Recurrent Neural Networks (RNNs) and a **Context-First** approach using Pre-trained Transformers. Below is the granular technical breakdown of these systems.

---

1. The BiRNN and BiLSTM Pipelines

The recurrent pipelines were designed to evaluate how traditional sequence models handle the morphological density of the Arabic language when supplemented with handcrafted features.

A. The Necessity of Bidirectionality

Arabic is a highly contextual language where the grammatical case (I'rab) of a word—and thus its final diacritic—is determined by its syntactic role in a sentence. For instance, in the sentence *"أكل الولدُ التفاحةَ"* (The boy ate the apple), the diacritic on the last letter of "الولد" (the boy) is a *Damma* because it is the subject (*Fa'il*), while the last letter of "التفاحة" (the apple) is a *Fatha* because it is the object (*Maf'ul Bihi*).

By using a **Bidirectional** architecture, the model processes the sequence in two directions:

- **Forward Pass:** Captures the morphological history (prefixes and roots).
- **Backward Pass:** Captures the future syntactic context (governing agents or *Awamil* that appear later in the sentence).

The hidden states from both directions are concatenated at each time step t, providing the output layer with a comprehensive "view" of the character's environment.

B. Deep 3-Layer Architecture

We implemented a **3-layer stack** for both the BiRNN and BiLSTM. In deep learning for NLP, a single layer often only captures surface-level patterns. By stacking three layers, the model develops a hierarchy of features:

1. **Lower Layers:** Typically focus on local character-level transitions (e.g., the likelihood of a *Sukun* following a *Fatha* in specific patterns).
2. **Higher Layers:** Capture more abstract word-level and phrase-level structural information.

C. BiRNN vs. BiLSTM: The Memory Trade-off

- **BiRNN:** While faster and more computationally efficient, it traditionally struggles with the "Vanishing Gradient" problem. However, our use of 50K sentences and specific regularization made it a highly competitive and fast baseline.
- **BiLSTM:** Utilizes a gating mechanism (Input, Forget, and Output gates). This allows the model to "remember" syntactic dependencies over longer distances.

D. Justification for Hyperparameters

The stability of these models was heavily dependent on three core hyperparameters:

- **Dropout (0.5):** Because we fused character embeddings (128d), word embeddings (256d), and cluster embeddings (32d), our input feature vector was extremely high-dimensional. Without a high dropout rate of 0.5, the model may tend to "memorize" the training labels for specific words rather than learning the underlying linguistic rules. Dropout forces the network to find redundant paths for the same information, significantly improving generalization.
- **Gradient Clipping (1.0):** Deep recurrent networks are notorious for "Exploding Gradients," where backpropagation through many time steps results in massive updates that destabilize the weights. By clipping the norm of the gradients at 1.0, we ensured that the optimization process (Adam) remaines smooth and converged reliably even during the 25-epoch runs.
- **K-Means Clusters (K=50):** The choice of $K=50$ for our Word2Vec semantic clustering was a result of empirical testing.
  - If **K was too small** (e.g., 10), the clusters were too broad (e.g., mixing verbs and nouns).
  - If **K was too large** (e.g., 200), the model suffered from "Over-segmentation," where the cluster ID became too specific to the training data.
  - **At K=50**, the clusters effectively acted as a "Lightweight Part-of-Speech Tagger," identifying groups like "days of the week," "past-tense verbs," and "prepositions," which share identical diacritization logic.

---

2. The Transformer-Hybrid (AraBERT)

The second paradigm represents the state-of-the-art approach, utilizing the **aubmindlab/bert-base-arabertv2** model. This pipeline moves away from static word vectors toward **contextualized representations**.

A. The AraBERT Backbone

AraBERT is pre-trained on massive datasets (up to 70GB of Arabic text). Unlike Word2Vec, which gives the word "ذهب" (gold/went) a single vector, AraBERT generates a different embedding for "ذهب" depending on whether it appears in a sentence about jewelry or a sentence about movement. This ability to resolve **polysemy** is the primary driver of its performance.

B. Solving the Alignment Problem (Offset Mapping)

A significant technical hurdle in this project was the mismatch between **BERT tokens** and **Arabic characters**.

1. **Sub-word Tokenization:** AraBERT uses a WordPiece tokenizer. For example, a word with a prefix like "والمدارس" (and the schools) might be split into ['و', 'ال', 'مدارس'].
2. **Character-Level Labels:** Our target labels are character-by-character.
3. **The Solution:** We implemented an **Offset Mapping** logic. During data loading, we tracked the original start and end indices of every BERT sub-token. We then used a gathering operation in PyTorch to broadcast the embedding of a single sub-token back to all the characters it represents.

C. Feature Fusion: BERT + Char Embeddings

We discovered that while BERT understands the "meaning," it can sometimes lose the "surface details" of the characters. To bridge this gap, we concatenated the AraBERT hidden state with a Trainable Character Embedding:

V{final} = [E{AraBERT} + E{Char}]
This ensures the BiLSTM layer (which sits on top of BERT) sees both the high-level semantic "wisdom" of the Transformer and the granular "shape" of the individual letters.

D. Computational Complexity and Training Strategy

Training the Transformer pipeline was the most resource-intensive part of the project.

- **Sliding Window:** Because BERT has a limit of 512 tokens, we implemented a sliding window with a stride of 128 to process long Arabic documents without losing context at the boundaries.
- **Optimization:** We used **AdamW** with a very low learning rate (2E-5) to fine-tune the BERT weights.

---

3. Comparative Analysis of Architectures

Feature	BiRNN / BiLSTM Pipeline	AraBERT Hybrid Pipeline
Primary Input	Static character embedding optionally with any combination of the following : static word embedding , last character flag, cluster id embedding	Static character embedding with dynamic AraBERT word embeddings
Linguistic Depth	Handcrafted Word-Boundary Flags	Deep Self-Attention Layers
Accuracy	High (97.20%)	Extremely High (Context-Sensitive)

IV. Quantitative Results & Ablation Study

We conducted 8 distinct experiments on the BiRNN pipeline to determine the marginal utility of each feature.

BiRNN Ablation Results Table

Model Run	Configuration	Val Acc (%)
Experimet 1	Chars + Word + Cluster	97.20%
Experiment 2	Full Model (Chars + Word + Cluster + Flag)	97.18%
Experiment 2	Chars + Word + Flag	97.16%

Experiment 3	Chars + Word	97.16%
Experiment 4	Chars + Flag	96.66%
Experiment 5	Chars + Cluster + Flag	96.64%
Experiment 6	Chars Only (Baseline)	96.49%
Experiment 7	Chars + Cluster	96.48%

BiLSTM Ablation Results Table

Model Run	**Configuration**	**Val Acc (%)**
Experimet 1	Chars + Cluster	97.99%
Experiment 2	Full Model (Chars + Word + Cluster + Flag)	97.88%
Experiment 3	Chars + Word + Flag	97.86%
Experiment 4	Chars + Cluster + Flag	97.99%

Key Finding: The combination of **Word Embeddings and Semantic Clusters** were present in the most performing modles, outperforming even the Full Model in BiRNN. This suggests that the semantic information provided by K-Means clustering is highly effective for Arabic diacritization.

BiLSTM with AraBERT Ablation Results Table

Model Run	Configuration	Val Acc (%)
Experimet 1	Static character embedding with dynamic arabert word embedding	98.23%

V. Hardware Performance Analysis

The models were trained using **Kaggle P100 GPUs**. A stark contrast was observed in efficiency:

- **AraBERT:** Required around **6 hours** to complete just 10 epochs (training on around 86k sentences and validating on around 13k sentences).
- **BiRNN Pipeline:** Required **11 hours, 9 minutes** to run the **entire 8-run ablation study** (25 epochs each) on 50K sentences.
- **Inference Speed:** The BiRNN/BiLSTM pipelines are significantly faster, making them the preferred choice for real-time applications despite the Transformer's deep context.

VI. Project Management & Work Distribution

The project was executed through a modular division of labor, ensuring each architectural layer was optimized.

Role	Contribution
Belal Sabbaq	**BiRNN pipeline**
Abdelrahman Ashraf	**BiRNN pipeline**
Mohamed Ayman	**BiLSTM pipeline**

Yusuf Ayman	**BiLSTM pipeline**

All team members contributed to the pipeline.

VII. Conclusion

This study demonstrates that while large language models like AraBERT provide a powerful "black box" solution, **feature-engineered RNNs** are remarkably robust. By injecting semantic clusters and word-level information into a BiRNN, we achieved over **97% accuracy**, proving that traditional NLP techniques remain vital for resource-constrained, high-performance Arabic language processing.