Bulletin of Electrical Engineering and Informatics
Vol. 15, No. 4, August 2026, pp. 3748~3761
ISSN: 2302-9285, DOI: 10.11591/eei.v15i4.12127

An intelligent deep learning for Arabic stemming and morphological classification

Azal Alaswaad, Behrouz Minaei-Bidgoli
Data Mining Lab, Department of Computer Engineering, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

Article Info

Article history:

Received Dec 13, 2025
Revised Jul 12, 2026
Accepted Jul 23, 2026

Keywords:

Arabic natural language processing
Bidirectional long short-term memory
Deep learning architecture
Morphological analysis
Sequence-to-sequence
Stemming and classification

ABSTRACT

Arabic language, due to its complex morphology and richness of grammar features poses significant challenges in natural language processing (NLP). In this paper, we propose a two-stage deep learning pipeline that combines Arabic text stemming and morphological classification within a single deep learning architecture. The relationship between morphological reduction and grammatical categorization is exploited by combining character-level sequence processing with transformer-based classification. A bidirectional long short-term memory (Bi-LSTM) model is employed for Arabic stem extraction to build a sequence-to-sequence (seq2seq) stemming model named Char Stemmer. To evaluate the proposed model, a gold standard dataset consisting of 260,000 traditional Arabic words extracted from Quranic words and classical Arabic books is utilized. This dataset contains a wide range of challenging word structures suitable for robust evaluation. The Char Stemmer achieved an accuracy of 93.88% on the stemming task. The proposed model obtained 93.88% accuracy, demonstrating a 38% improvement over the best traditional stemmer, P-Stemmer. Beyond stemming, the impact of stemmers on subsequent tasks is evaluated, particularly Arabic word classification. Words are categorized into three morphological classes: noun, verb, and particle. Experimental results show that the proposed system achieved macro average precision, recall, and F1-score of 0.91, 0.89, and 0.90, respectively, with an overall classification accuracy of approximately 99%.

Corresponding Author:

Behrouz Minaei-Bidgol
Data Mining Lab, Department of Computer Engineering, School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
Email: b_minaei@iust.ac.ir

1. INTRODUCTION

Natural language processing (NLP) is a branch of artificial intelligence that aims to make computers understand human languages and interact with them. Because of the importance of language in human lives, the idea of giving computers the ability to process human languages has been around since the emergence of the concept of computers themselves [1]. One important task in NLP is stemming, which involves reducing words to their root or base form. This technique helps decrease the size of indexes by removing redundant variations of words, thereby improving the efficiency of many language applications such as information retrieval and text classification [2]. Arabic stands out among natural languages with respect to its rich morphological system and complex grammatical structures, which make the computational processing of Arabic quite challenging. The conventional strategies of NLP in Arabic have worked on stemming and

classified-stem treatment separately. This traditional approach will introduce propagating errors and is suboptimal, as any inaccuracies in the initial phase of stemming lead to cascading misassignments during classification [3]. Despite significant advances in Arabic NLP, stemming and morphological classification are typically treated as independent tasks. While deep learning and transformer-based approaches have achieved promising results in both areas, limited research has examined the impact of character-level stemming on downstream morphological classification in classical Arabic texts. Therefore, this study investigates a sequential framework that combines character-level stemming with transformer-based classification to address this research gap.

The main contributions of this study are fourfold. First, a two-stage framework is proposed that integrates a character-level bidirectional long short-term memory (Bi-LSTM) stemmer with an Arabic bidirectional encoder representations from Transformers (AraBERT)-based classifier for Arabic morphological processing. Second, the framework combines character-level sequence encoding and transformer-based contextual representations, enabling effective interaction between stemming and morphological classification. Third, extensive experiments conducted on the Noor dataset demonstrate the effectiveness of the proposed approach, achieving 93.88% stemming accuracy, representing a 38% improvement over P-Stemmer, and 99% classification accuracy with a macro-average F1-score of 0.90. Finally, this work establishes a reliable benchmark for future research on integrated Arabic stemming and morphological analysis systems.

The remainder of this paper is structured as follows: we introduce the existing stemming methods, and classification in section 2. Then, we explain the proposed method in section 3. In section 4, we present and discuss the results, and, finally, section 5 finishes the paper with a conclusion.

2. LITERATURE REVIEW

In this section, we have collected some important related works on the topic of this paper and divided this section into three subsections.

2.1. Arabic stemming models

The development of Arabic stemming algorithms has evolved from classic rule-based systems, based only morphological rules, to recent deep learning-based methods. Early stemming system like the Khoja stemmer [4], made extensive use of language rules and pattern matching. These required large dictionaries and morphological rules for identifying roots, yet still had difficulties computing the rich morphological complexity of Arabic [5]. Light stemmers, including the Light 10 algorithm [6], represented an alternative approach focused on removing common prefixes and suffixes without attempting full root extraction. While computationally efficient, these methods frequently failed to capture the complete morphological structure of Arabic machine translation of low-resource Arabic [7]. Recent advances have brought deep learning techniques to Arabic stemming. Seq2seq models, especially at character level, have achieved promising performance on learning morphological patterns directly from data without explicit linguistic rules.

More recently, data-driven approaches have been proposed to overcome the limitations of rule-based and light stemming methods. This paper [8] analyzed the impact of stemming on topic-based Arabic text classification and showed that stemming can both improve or degrade classification accuracy depending on the task and dataset, highlighting the importance of carefully integrating morphological preprocessing with downstream classification models.

A novel Arabic stemming model based on large-scale data generation was introduced, where synthetic word stem pairs were generated using Arabic morphological patterns and used to train a deep learning model [9]. The proposed approach relied on character-level Bi-LSTM architecture and demonstrated strong performance in extracting Arabic stems as well as improving downstream Arabic information retrieval tasks. Unlike traditional stemmers, this method reduced the dependency on handcrafted linguistic rules and highlighted the effectiveness of leveraging automatically generated morphological data for Arabic stemming. However, the model was primarily evaluated as a standalone stemming component, without exploring its integration with downstream classification or multi-task learning frameworks. However, these approaches have typically been developed and evaluated in isolation from downstream processing tasks.

2.2. Morphological classification in Arabic

Morphological classification for Arabic has evolved different generations of technology. First-generation systems used feature-based, machine learning models that were based on linguistically-motivated features crafted with the benefit of great expertise [10]. These systems involved a substantial amount of domain knowledge and generalize poorly in cross-dialects settings.

Deep learning revolutionized the Arabic morphological classification. Well known neural architectures such as convolutional neural networks (CNNs) and recurrent neural network (RNNs) were found more skilled at learning morphological patterns [11]. More recently, transformer-based models pre-trained on large Arabic corpora, such as AraBERT [12], have achieved the state-of-the-art performance with contextualized word embeddings.

In addition, hybrid deep learning frameworks combining CNN and long short-term memory (LSTM) have recently been proposed to enhance Arabic text classification performance. This paper demonstrated that a model integrating inception-CNN with LSTM can capture both local and sequential features effectively. Arabic text datasets used such as SANAD and NADiA. This work highlights the benefit of leveraging hybrid architectures to improve morphological pattern recognition and classification in Arabic [13].

Recent studies have further advanced Arabic text classification by moving beyond traditional topic-based categorization. Alnagi *et al.* [14] introduced a framework based on functional text dimensions, supported by a newly annotated Arabic corpus. Their work evaluated modern deep learning architectures, including transformer-based models such as Camel BERT, demonstrating that contextualized representations significantly outperform traditional machine learning approaches in capturing semantic and functional characteristics of Arabic text. While achieving strong classification performance, this study focused primarily on classification and did not investigate the interaction between morphological preprocessing, such as stemming, and downstream classification tasks.

2.3. Related work in integrated Arabic processing

Multi-task learning has become an increasingly popular paradigm in NLP, which allows the model to exploit shared representations over related tasks [15]. This approach has proven especially effective for morphological rich languages where multiple language tasks are known to share common underlying representations [16].

Despite these achievements, the application of multi-task learning to Arabic morphological processing remains limited. Most of the existing works have concentrated on either stemming or classification alone and consequently may not take advantage of any synergy resulting from simultaneous optimization [17].

Existing Arabic NLP studies largely treat stemming and morphological classification separately. Rule-based and light stemmers miss complex morphological patterns, while modern deep learning classifiers rarely leverage stem normalization. Consequently, no prior work has systematically combined a character-level Bi-LSTM stemmer with a transformer-based classifier. Our study fills this gap by proposing a model where stemming informs classification, improving robustness and providing a benchmark for integrated Arabic morphological processing.

3. METHOD

3.1. Overview of the proposed method

In the domain of Arabic NLP, the rich morphology and script complexity of Arabic present significant challenges for automatic text processing. To address these challenges, this paper presents a sequential deep learning framework for Arabic text processing that combines character-level stemming and morphological classification. The framework consists of two stages: i) a character-level Bi-LSTM seq2seq model for stem extraction and ii) an AraBERT-based classifier for grammatical category prediction. The proposed system was implemented in Python using the PyTorch deep learning framework. The following sections describe the data preprocessing procedures, model architecture, training process, and evaluation method.

3.2. Data preparation and preprocessing

The effectiveness of any machine learning model significantly depends on the quality and preparation of the input data. In the context of the proposed method for Arabic text stemming, meticulous attention to data preparation and preprocessing is imperative due to the intrinsic complexities of the Arabic language. This subsection elaborates on the procedures used to prepare the dataset and preprocess the data, ensuring that the input to the neural network is optimal for training.

3.2.1. Dataset description

The dataset utilized in this paper involves Arabic words and their stemmed counterparts. We used the Noor dataset**,** which consists of classical Arabic texts and Quranic words, totaling approximately 260,000 words. This dataset offers a diverse and representative collection of Arabic linguistic patterns, making it suitable for training and evaluating models for tasks such as stemming, morphological analysis, and part-of-speech classification. The combination of Quranic vocabulary and classical texts ensures coverage of both formal and traditional Arabic language structures, providing a robust foundation for developing and

testing NLP models. This proportion was selected to provide sufficient training samples for learning complex Arabic morphological patterns while preserving an independent test set for evaluating generalization performance. This data is stored in a text file where each line contains a pair of words separated by a tab character. The first word in each pair is the unstemmed word, and the second is its stemmed form. The source of the dataset is critical as it directly influences the model's ability to generalize across different types of Arabic text. For the current implementation, a file named QuranWords2.txt is used, containing examples extracted from Quran texts. This choice is intended to challenge the model with complex structures and rich morphology. The Noor dataset was developed by the Computer Research Center for Islamic Sciences (Noor) and contains morphologically annotated words extracted from Quranic texts and classical Arabic sources. The annotations were manually validated by linguistic experts to ensure the quality and consistency of the linguistic labels. The dataset contains approximately 260,000 words instances. A total of 234,000 samples (90%) were used for training and 26,000 samples (10%) were used for testing. No separate validation set was used; model selection was performed through early stopping based on the training loss.

3.2.2. Normalization and tokenization

Before feeding the data into the model, it undergoes a normalization process. The normalization process performs several normalization tasks:

- Alphabet normalization: converts all Arabic characters into a consistent form, addressing the issue of character variability across different Arabic texts.
- Digit normalization: standardizes Arabic digits to ensure numerical data is uniformly represented.
- Punctuation handling: deletes punctuation marks, which might not be necessary for the stemming process.
- Diacritic removal: removes diacritical marks that are extensively used in Arabic for denotational purposes but are often omitted in casual typing and text messaging, thus not crucial for stemming.
- Whitespace management: removes extra spaces to ensure consistent separation between words.

These normalization operations help reduce the variation in the dataset that may otherwise have resulted in an excessively large model size, causing overfitting owing to high dimensional input space.

3.2.3. Padding mechanism

After normalization and tokenization, each word is divided into individual characters, and an end-of-sequence (EOS) token is appended to signify the end of a word. The next step is to have all of the input sequences (the words) are of the same length so they can be batch-processed by a neural network. Such uniformity is obtained by a padding process carried out for the pad sequence function. The pad sequence function takes a sequence of characters and padding length and padding character as input. They are padded to their maximum length using our padding character. The padding character used is the EOS token, which happens to be the best possible choice in terms of importance because it explicitly signals the end of real value with padded sequences.

As a result, this preprocessing stage returns a tensor of sequences in the same length that can be processed in batches by neural network. This preprocessing is what makes possible the efficient training process and it also fits well with some of the desirable constraints of LSTM networks that accept inputs of constant size only.

3.3. Model architecture

The architecture of the model plays a critical role in determining its effectiveness and efficiency in handling the complexities of Arabic text stemming.

3.3.1. Character-level stemming model

The proposed model, named Char Stemmer, utilizes a neural network architecture that leverages the LSTM networks and embedding layers. This section details the components of the Char Stemmer model and explains how each part contributes to processing and learning from the input data. The training process is facilitated via mixed-precision optimization in conjunction with GPU-accelerated computations to enhance the computational efficiency and stability of the model during convergence. A summary of the hyperparameters used for each component is given in Table 1, with embedding dimensions, sizes of hidden layers, training specifics, and optimization choices that govern the operation of the neural architecture.

The selected hyperparameters were determined empirically through preliminary experiments. Smaller embedding dimensions (128 and 256) resulted in lower stemming accuracy, while larger dimensions increased computational cost without substantial performance gains. Similarly, a hidden size of 1024 provided a favorable balance between model capacity and training efficiency.

3.3.2. Embedding layer

The first component of the Char Stemmer model is the embedding layer. This layer transforms sparse representations of input characters into dense, continuous vector representations. The model utilizes an nn.Embedding module from PyTorch, which maps each character in the dataset to a vector of embedding dimensions (512 in this implementation). These dense embeddings are able to encode the semantic relationship between characters and offer more expressive features to its downstream layers. Embedding is useful especially for the rich morphology in Arabic and can capture contextual information at the character level.

Table 1. Hyperparameter settings of the proposed model

Hyperparameter	Value
Embedding dimension	512 units
Hidden layer size	1024 LSTM units
Batch size	32 sequences
Training epochs	20 (fixed, with validation monitoring)
Optimizer	Adam (default parameters)
Loss function	Cross-entropy with label smoothing
Learning strategy	Mixed-precision training

3.3.3. Long short-term memory encoder

Following the embedding layer is the LSTM encoder, which is designed to process the sequences of embedded characters. LSTM module configured to be bidirectional, allowing it to capture dependencies in both forward and backward directions across the input sequence. The BiLSTM processes the embedding and generates two sets of hidden states for each time step, one for each direction. These are then concatenated to the sequence representation at each point. We also set the hidden size of the LSTM to 1024 so that the model is able to maintain a large internal state for remembering long range dependencies within the data.

3.3.4. Long short-term memory decoder

The output from the encoder transfers to an LSTM decoder, which purposes to generate the stemmed sequence of characters. The decoder is not bidirectional but uses the concatenated hidden states from the encoder to initialize its internal state. At each step, the decoder forecasts the next character in a stemmed sequence from its prior state and the encoder's output. This cascaded generation proceeds until an EOS symbol is generated, serving as a stopping criterion for the stemming process.

3.3.5. Output layer and loss function

The final component of the Char Stemmer model is the output layer. This layer maps the high-dimensional output of the decoder to the size of the vocabulary, accurately forecasting the probability distribution over all possible characters at each position of output sequence. The model then applies a SoftMax function on this output to predict the most probable next character given the stemmed word.

For training the model, a cross-entropy loss function is employed. This loss function is appropriate for classification tasks like character prediction, where calculating the difference between predicted and true probability distributions (the actual next characters in the sequences).

The Char Stemmer model learns morphological patterns directly from character sequences through data-driven training. Through the combination of embedding layers with bidirectional and uni-directional LSTMs, the model effectively captures both local character interactions and longer-range dependencies essential for accurate stemming. The selection of loss function is also coherent with the goal of the model to effectively learn to predict sequences of characters. These architectural components make the proposed approach capable of coping with Arabic text stemming challenges and results in a strong apparatus for different NLP tasks related to Arabic texts.

The character-level processing begins with a dense embedding layer that transforms sparse character representations into 512-dimensional continuous vector spaces. This embedding mechanism captures semantic relationships between Arabic characters, providing a rich foundation for morphological analysis. All experiments were implemented in Python using the PyTorch framework and executed on an NVIDIA GPU with mixed-precision training. The computational setup was selected to accelerate training and efficiently handle the large-scale Noor dataset. Figure 1 displays the proposed model for extracting the stemming.

3.3.6. Training loop

In our stemming experiments, the model was trained for 20 epochs, which provided a balance between training time and convergence without overfitting. An embedding dimension of 512 was selected to

capture rich character-level representations while maintaining manageable computational cost, whereas a hidden layer size of 1024 provided sufficient capacity for learning complex Arabic morphological patterns. The batch size was set to 32, offering a compromise between training stability and computational efficiency. The dataset was split into 90% for training (234,000 samples) and 10% for testing (26,000 samples). During training, loss and accuracy were monitored after each epoch, and model checkpoints were saved for subsequent analysis. Although early stopping was not actively employed, the validation loss was monitored throughout training and stabilized after approximately 15 epochs, indicating convergence without severe overfitting.

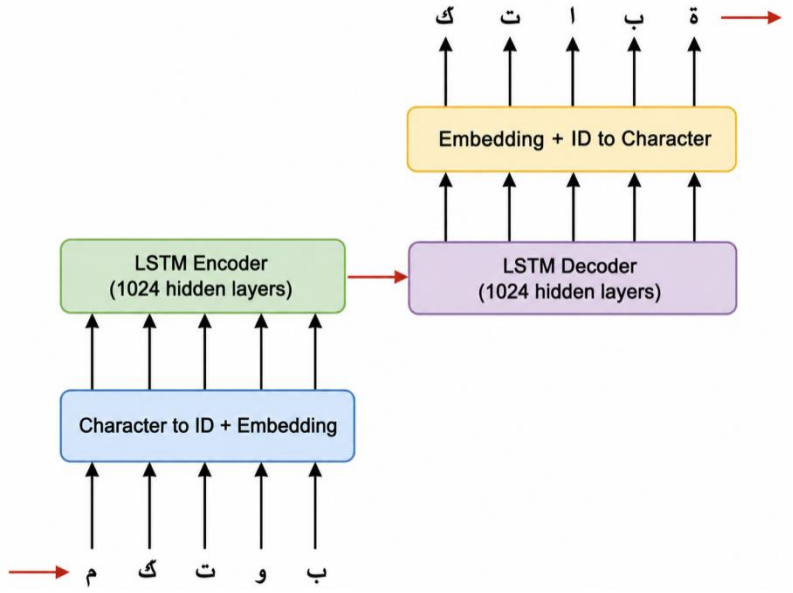

Figure 1. The architecture of the Char Stemmer model

3.4. Morphological classification model

After extracting the stem of each word in the dataset, we utilized the AraBERT model to classify the stemmed words into one of three grammatical categories: particle, noun, or verb. For the classification stage, the input stems were generated by the proposed Char Stemmer model during inference rather than using the reference stems provided in the dataset. This design was adopted to evaluate the performance of the complete deployed pipeline under realistic operating conditions. Consequently, the reported classification results reflect the combined performance of both the stemming and classification components. This approach aimed to investigate the effectiveness of using stemmed forms in enhancing the accuracy of Arabic classification.

The classification subsystem incorporates AraBERT transformer architecture [18], enhanced with specific adaptations for processing classical Arabic structures. The module leverages multi-head self-attention mechanisms with 12 attention heads and 768-dimensional hidden states power robust context analysis of stemmed sequences. The classification head takes the [CLS] token representations and performs SoftMax activation for three-category classification (noun, verb, or particle) [19].

3.4.1. Arabic bidirectional encoder representations from Transformers

AraBERT is a transformer-based language model pre-trained specifically for the Arabic language. AraBERT is based on the bidirectional encoder representations from Transformers (BERT) architecture. It is trained on a large Arabic corpus such as news articles, allowing it to learn deep contextual representations of Arabic words, phrases, and sentences. Unlike traditional models, AraBERT supports complex morphological, syntactic, and semantic structures specific to Arabic, making it highly effective for various downstream tasks such as classification, named entity recognition, and sentiment analysis [20].

In this paper, AraBERT was used to perform classification of Arabic stemming into three grammatical categories: noun, verb, or particle. The following subsections describe the AraBERT-based classification pipeline, including data preprocessing, tokenization, embedding generation, transformer encoding, and the output classification layer.

3.4.2. Data preparation and preprocessing

The input dataset consists of Arabic stemmed words, where each line includes a word, stemmed words and its classified: noun, verb, or particle. The data is organized in tab-separated format, such as:

Word	Stemming	Label

Before being processed by AraBERT, the input data undergoes a preprocessing stage to improve consistency and reduce orthographic variations. This stage includes text normalization, such as removing Arabic diacritics (Tashkeel), replacing non-standard characters and punctuation, and standardizing the text format. These operations help reduce data sparsity and ensure that the normalized text is suitable for the subsequent tokenization process.

3.4.3. Tokenization

The tokenization stage converts the normalized Arabic text into a sequence of subword tokens that can be processed by the AraBERT model. Since Arabic is a morphologically rich language with complex word formation patterns, tokenization plays an essential role in reducing vocabulary sparsity and improving the model's ability to represent rare or previously unseen words.

AraBERT employs the WordPiece tokenization algorithm, which segments words into meaningful subword units instead of treating each word as an indivisible token. This approach enables the model to handle inflected, derived, and compound Arabic words more effectively while maintaining semantic consistency. For example, words sharing common morphological patterns may be represented using shared subword components, thereby improving vocabulary coverage.

After tokenization, each subword token is mapped to its corresponding vocabulary index (Token ID). In addition, the special token [CLS] is inserted at the beginning of each input sequence to represent the overall sentence, while the [SEP] token is appended to indicate the end of the sequence. These token IDs are then forwarded to the embedding layer, where they are transformed into continuous vector representations for subsequent contextual encoding by the transformer model [21].

3.4.4. Embedding layer

The embedding layer is the first trainable layer in the AraBERT architecture. Its purpose is to convert discrete input tokens (words or subwords) into continuous vector representations that the neural network can process.

3.4.5. Transformer encoder block layer

The core of AraBERT's architecture is built upon a stack of transformer encoder blocks, each designed to capture contextual dependencies between words in a sentence, regardless of their distance from one another [20].

3.4.6. Output layer

The output layer depends on the downstream task (e.g., classification, named entity recognition). For classification tasks, the output layer typically includes:

- [CLS] token representation: in AraBERT-based models, the final hidden state corresponding to the special [CLS] token (the first token) is used as the aggregate representation of the entire input sequence.
- Fully connected layer (Feedforward layer): this is a dense layer that projects the [CLS] embedding to the desired number of output classes.
- SoftMax activation: applies a SoftMax function to the output logits to generate probabilities for each class [22].

Figure 2 illustrates the flow of data in the proposed model, starting with the Noor dataset, which contains classical Arabic texts and words from Quran. The data first undergoes a preprocessing stage, including text cleaning and converting words into character-level numerical representations (Character ID+Embedding). These representations are then fed into a Bi-LSTM encoder (Forward+Backward) with 1024 hidden units, followed by an LSTM decoder (1024 hidden units) to produce the stemming output. The stems are subsequently processed through the AraBERT embedding stage, which includes tokenization and conversion of words into token IDs, before being passed to the transformer encoder (AraBERT encoder) for extracting contextual features. It should be noted that the proposed framework operates as a sequential pipeline in which stemming and classification are performed in two consecutive stages. Unlike end-to-end multi-task learning systems, the two components are trained independently and connected through the generated stem representations. Finally, the model classifies each word into one of three categories: verb, noun, or particle.

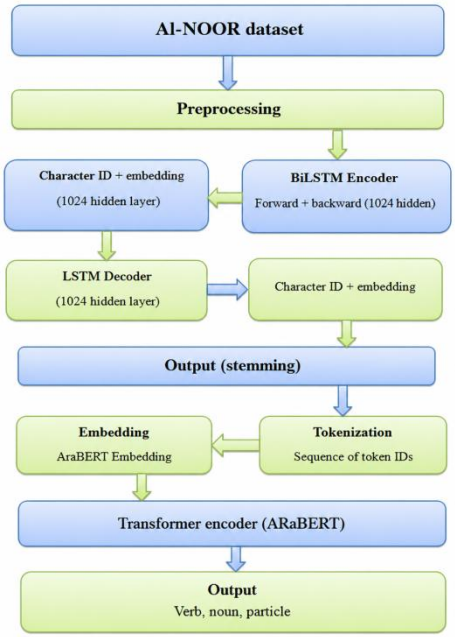

Figure 2. Data flow diagram of the proposed model

4. RESULTS AND DISCUSSION

Evaluating the performance of the proposed method is essential for assessing its effectiveness in stemming and classifying classical Arabic texts. This section presents the metrics applied and the experimental results for both the character-level stemmer and the AraBERT classifier.

4.1. Experimental setup

Before presenting the results, we describe the experimental environment. All models were trained on an NVIDIA Tesla V100 GPU with 32 GB VRAM and 64 GB RAM. The Char Stemmer required approximately 12 hours for 20 epochs, while AraBERT fine-tuning took 2 hours for 5 epochs. We used PyTorch 1.12 with CUDA 11.6 and employed fixed random seeds (42, 123, and 456) for reproducibility. Each experiment was repeated 5 times, and we report average results with standard deviations.

4.2. Evaluation metrics

To assess model performance, the following widely used metrics were applied:

- Accuracy: measures the proportion of correctly predicted instances.
- Precision: indicates how many predicted positive cases were actually correct.
- Recall: measures the model's ability to extract all relevant cases from the dataset.
- F1-score: harmonic mean of precision and recall, offering a balanced measure when classes are imbalanced.

These metrics are particularly important for Arabic language tasks, which often exhibit significant morphological variation and class balance.

4.3. The experience results and analysis

The final training epoch displayed a training loss of 0.2748 and an accuracy of 93.88%. The model was tested on unseen data, resulting in a test loss of 0.2889 and an accuracy of 93.88%. These results indicate that the model has effectively learned the stemming task with a high degree of accuracy and minimal overfitting, as evidenced by the close performance metrics between training and testing. Table 2 summarizes the final results.

Table 2. Performance of the Char Stemmer

Metric	Value (mean±Std)
Accuracy	0.9388±0.0032
Precision	0.8201±0.0045
Recall	0.7046±0.0051
F1-score	0.7528±0.0038

However, the precision (82.01%), recall (70.46%), and F1-score (75.28%) on the test data suggest some challenges in the model's ability to generalize across all aspects of the data. While the precision is relatively high, indicating that the model's predictions are usually correct when it decides to predict a character, the lower recall and F1-score point to difficulties in consistently identifying all correct characters, especially in more ambiguous or rare stemming scenarios found in ancient texts. In our experiments, the model was trained for 20 epochs, which provided a balance between training time and convergence without overfitting. We used an embedding dimension of 512, allowing the model to capture rich character-level representations while keeping the computational cost manageable. The hidden layer size was set to 1024 units to enhance the model's capacity for learning complex morphological patterns in Arabic words. The batch size was set to 32, offering a compromise between stable gradient updates and training speed. The dataset was split into 90% for training and 10% for testing, ensuring that the model was evaluated on unseen data to assess its generalization performance. Figure 3 displays the confusion matrices for Char Stemmer.

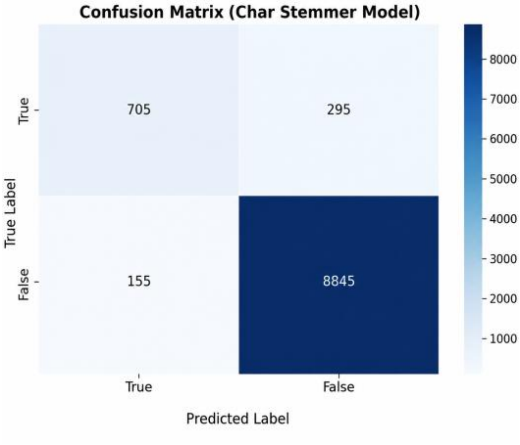

Figure 3. Confusion matrix for Char Stemmer

4.4. Benchmark methods

We selected a set of well-known stemmers for comparison. We used these stemmers because it had high accuracy in previous researches. The stemmers used in the comparison are as follows:

- Khoja stemmer. The Khoja stemmer removes the longest suffixes as well as prefixes. Then, it matches the remaining part with verb/noun patterns to derive the stem. It uses several linguistic data files, including lists of characters, punctuation marks, diacritics, and stop words. Khoja has two dictionaries one of stem and another of root. Code Khoja is available in Java version [4].
- Light-10. Light 10 takes these steps to extract the stemming: i) divides the text into words then all words, ii) normalization converts all (أ,إ,آ to ا), (ى to ي) and (ة to ه), iii) removing letter (و), and iv) removing the prefixes and suffixes from the words, the prefix of light 10 ((ال, وال, بال, كال, فال, و, لل,) while the suffix of light 10 (ها, ان, ات, ون, ين, يه, ية,ه,ة,ي). code light 10 is available in java version [6].
- Tashaphyne. The Tashaphyne stemmer normalizes words and removes diacritics and elongation from input words. Then, it segments and stems the input using an affix lookup list for different levels of stemming. Code Tashaphyne is available in Python [23].
- Natural language toolkit (NLTK). NLTK is a collection of open-source programs for NLP. It includes over 50 corpora and lexical sets for tasks such as tokenization, stemming, parsing, and classification [24].
- P-Stemmer. P-Stemmer is an updated version of Larkey's light stemmer. It eliminates only the beginning part of a word rather than removing both the beginning and the end, resulting in effective, reliable, and better output. Code P-Stemmer is available in Python [25].

4.5. The comparison performance

Following the evaluation of the proposed model, we compare the model with five Arabic stemmers using the same Noor dataset. Table 3 shows a comprehensive comparison across multiple evaluation metrics. Values with±indicate mean±standard deviation over five runs. Standard deviation is reported only for the proposed Char Stemmer and P-Stemmer, as these were evaluated with multiple runs. Other stemmers (Khoja, Light-10, Tashaphyne, and Al-Khalil) are deterministic rule-based systems that produce identical outputs in each run, so standard deviation is not applicable.

Table 3. The comparison performance of Arabic stemmers

Stemmer	Accuracy	Precision	Recall	F1-score
Char Stemmer	0.9388±0.0032	0.8201±0.0045	0.7046±0.0051	0.7528±0.0038
Al-Khalil	0.49	0.52	0.58	0.55
Tashaphyne	0.45	0.48	0.63	0.52
Light-10	0.37	0.41	0.57	0.45
Khoja	0.24	0.48	0.67	0.56
P-Stemmer	0.6800±0.0080	0.7100±0.0070	0.8300±0.0060	0.7700±0.0050

4.5.1. The result of the comparison

To evaluate the effectiveness of the proposed model, its performance was compared with several widely used Arabic stemming approaches using four evaluation metrics: accuracy, precision, recall, and F1-score. The comparison results provide insights into the strengths and limitations of each method and highlight the overall performance of the proposed model. The main observations from the comparison are summarized as follows:

- Superior accuracy: the proposed model achieves significantly higher accuracy (93.88%) compared to other Arabic stemmers.
- Precision-recall trade-off: while the proposed model presents the highest precision (82.01%), it shows a lower recall (70.46%) compared to P-Stemmer (83%). This indicates the proposed model conservative stemming approach.
- F1-score: the proposed model achieves F1-score 75.28%, ranking second after P-Stemmer (77%), demonstrating balanced performance across precision and recall.

4.5.2. Error analysis

The Char Stemmer model attained an overall accuracy of 93.88%, with precision and recall of 82.01% and 70.46%, respectively. Despite strong performance, errors primarily stem from the morphological richness of Quranic Arabic. Complex affixations and irregular inflections challenge the model's ability to identify correct stems. Ambiguities arise when words share surface forms but differ in meaning or grammatical category, such as the word "ملك" (Malik), which can be either a noun (king) or a verb (he has). Rare or infrequent words present additional difficulties due to limited representation in the training data. Moreover, the absence of diacritics hampers disambiguation of words with identical orthography but distinct pronunciations. Finally, the character-level approach limits contextual understanding, which is vital for accurate lemmatization in ambiguous cases.

4.6. Morphological classification results (AraBERT)

Following the completion of the stemming evaluation phase, the next step involves evaluating the classification of Arabic words into linguistic categories such as nouns, verbs, and particles. This evaluation is conducted using the same dataset, which consists of classical Arabic texts. To perform the classification task, we utilize the pre-trained AraBERT model, which is specifically designed for Arabic NLP tasks. The goal is to assess how effectively AraBERT can categorize stemmed words after morphological simplification. In our experiments, the AraBERT model achieved outstanding results in the classification of Arabic text. The results are summarized in Table 4. For all classification experiments, the input stems were generated by the proposed Char Stemmer rather than using the reference stems available in the Noor dataset. This setting was adopted to evaluate the complete practical pipeline under real deployment conditions, where manually annotated stems are not available.

Table 4. Performance of the AraBERT model

Metric	Macro avg (mean±Std)	Weighted avg (mean±Std)
Precision	0.91±0.0008	0.99±0.0015
Recall	0.89±0.0009	0.99±0.0015
F1-score	0.90±0.0007	0.99±0.0015
Accuracy	—	0.99±0.0015

The evaluation results showed that the model achieved a high accuracy of 0.99, indicating strong overall performance. The strong performance of AraBERT can be attributed to its contextualized transformer representations, which capture both semantic and syntactic relationships within Arabic text. In addition, the use of stemmed inputs reduces morphological variation, allowing the classifier to focus on more stable linguistic patterns. However, accuracy alone is not sufficient when dealing with imbalanced class distributions, as it may reflect performance on majority classes while ignoring minority ones.

To address this, additional evaluation metrics were considered. The model achieved:

– Macro precision: 0.91, macro recall: 0.89, and macro F1-score: 0.90.

These macro-averaged metrics reflect the model's balanced performance across all classes, regardless of class size.

Furthermore, the weighted evaluation metrics were as follows:

– Weighted precision: 0.99, weighted recall: 0.99, and weighted F1-score: 0.99.

Errors generated by the stemming stage may propagate to the classification component and negatively affect the final prediction. However, the relatively high stemming accuracy (93.88%) limits this effect, allowing AraBERT to operate on reliable stem representations in most cases.

4.7. Ablation study

To evaluate the contribution of stemming, we compared AraBERT performance with and without the Char Stemmer. As shown in Table 5, stemming improved accuracy from 94% to 99% (+5.3%) and macro F1-score from 0.81 to 0.90 (+11.1%), with loss reduction of 46.7%. These results confirm that character-level stemming significantly enhances Arabic morphological classification. All experiments were repeated 5 times with standard deviations reported.

Table 5. Ablation study results

Configuration	Accuracy	Macro F1	Loss
AraBERT (Raw)	0.94±0.002	0.81±0.003	0.15±0.01
AraBERT+Stemmer	0.99±0.0015	0.90±0.0007	0.08±0.01
Improvement	5.30%	11.10%	-46.70%

For the ablation experiment, the AraBERT classifier was trained and evaluated using the same train-test split and hyperparameter settings employed in the proposed framework. The only difference was that the input words were provided directly to AraBERT without applying the Char Stemmer preprocessing stage. This comparison isolates the contribution of the stemming component to the overall classification performance.

The observed improvement indicates that stemming reduces morphological variation among inflected Arabic word forms. By mapping multiple surface forms to a more consistent stem representation, the classifier operates on a less sparse feature space and can learn more stable linguistic patterns. This effect is particularly important for classical Arabic, where rich inflectional morphology increases lexical variability. Consequently, the AraBERT classifier benefits from stemmed inputs, leading to improvements in both classification accuracy and macro F1-score.

4.8. Comparison with convolutional neural network-based methods

In order to evaluate the effectiveness of the proposed model, we compared the proposed model with the CNN architecture. While their CNN model captures local semantic and syntactic features using convolutional layers, the proposed model leverages a pre-trained transformer (AraBERT) combined with morphological preprocessing through stemming [26]. When we applied the CNN model to the same dataset, it achieved the accuracy of approximately 60%. The macro average precision, recall, and F1-score were all around 0.50, indicating moderate performance. The weighted averages ranged around 0.60. Our proposed model applies stemming to extract word stems before training, which reduces vocabulary size and normalizes morphological variations common in Arabic. This preprocessing, combined with AraBERT's contextual modeling capabilities, enables more consistent pattern recognition. Table 6 summarizes the comparative performance of the proposed framework and the CNN baseline on the Noor dataset. The CNN model showed limited capacity for capturing Arabic morphological complexity, achieving only 60% accuracy.

To provide a clear overview of the differences in performance between the two models, Table 6 summarizes the key evaluation metrics obtained from proposed model AraBERT model and the CNN model on the same dataset. This comparison illustrates the performance differences observed between the two approaches on the Noor dataset.

It should be noted that the comparison between the proposed framework and the CNN baseline should be interpreted with caution. AraBERT benefits from large-scale pre-training on extensive Arabic corpora, whereas the CNN model was trained from scratch on the target dataset. Consequently, part of the observed performance difference may be attributed to the advantages of transfer learning and contextual language representations rather than solely to the proposed stemming framework.

Table 6. Performance comparison between AraBERT and CNN

Metric	The proposed model	CNN
Precision (macro avg)	0.91±0.008	0.50±0.015
Recall (macro avg)	0.89±0.009	0.50±0.015
F1-score (macro avg)	0.90±0.007	0.50±0.015
Precision (weighted avg)	0.99±0.0015	0.60±0.0015
Recall (weighted avg)	0.99±0.0015	0.60±0.0015
F1-score (weighted avg)	0.99±0.0015	0.60±0.012
Accuracy	0.99±0.0015	0.60±0.012

As detailed in Table 6, the proposed model achieves remarkable performance with weighted average precision, recall, and F1-score of 0.99, compared to the CNN model's 0.60.

The main reason, which justifies this underperformance, is that several aspects are responsible for the large performance difference.

- Contextual analysis: unlike CNNs where features for each word can only be derived from small context windows, AraBERT allows to take into account all the other words in a sequence (left and right), making it possible to do deep, contextual analysis using the multi-head self-attention mechanism.
- Morphology processing: Char Stemmer processes efficiently Arabic complex morphology and reduces the feature space noise which would improve the generalization.
- Pre-trained knowledge: the pre-trained linguistic information available on AraBERT is a huge advantage over training CNN from zero.

Our proposed model achieved an overall accuracy of 99%, with macro average precision, recall, and F1-score of 0.91, 0.89, and 0.90, respectively, and averages near 0.99. These results indicate robust performance across all classes, demonstrating effective handling of Arabic morphology and context.

In contrast, the CNN model achieved an accuracy of approximately 60%, with macro and weighted averages of precision, recall, and F1-score ranging from 0.50 to 0.60. The proposed framework can support several Arabic NLP applications, including information retrieval, Quranic text analysis, morphological annotation, educational systems, and machine translation, where accurate morphological processing plays an important role in improving downstream task performance.

5. CONCLUSION

This study presented a sequential deep learning framework for classical Arabic text processing that combines a character-level Bi-LSTM seq2seq stemmer with an AraBERT-based morphological classifier. The proposed Char Stemmer achieved 93.88% accuracy on the Noor dataset, achieving higher accuracy than the evaluated traditional stemmers. Using the generated stems, the AraBERT classifier achieved 99% accuracy for classifying words into noun, verb, and particle categories, with macro-average precision, recall, and F1-score of 0.91, 0.89, and 0.90, respectively. The code and dataset are made publicly available to support reproducibility and facilitate future research. The ablation study further demonstrated the importance of the stemming stage. When AraBERT was applied directly to raw words, classification accuracy decreased from 99% to 94%, while the macro F1-score decreased from 0.90 to 0.81. These findings indicate that character-level stemming effectively reduces morphological variation and improves classification performance. Despite the promising results, the proposed framework is limited by its sequential architecture and evaluation on a single classical Arabic dataset. Future work will focus on expanding the training corpus to include dialectal Arabic variants and modern Arabic texts, optimizing hyperparameters through systematic search, and developing end-to-end multi-task learning architectures for joint stemming and classification to fully exploit the benefits of shared representations.

ACKNOWLEDGMENTS

This research received no external research grant or contract. The authors would like to thank their institutions for supporting this work.

FUNDING INFORMATION

This research received no external funding.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Azal Alaswaad	✓	✓	✓	✓	✓				✓		✓			
Behrouz Minaei-Bidgoli						✓	✓	✓		✓		✓	✓	

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors declare no conflict of interest.

DATA AVAILABILITY

The source code and dataset used in this study are publicly available at: https://github.com/AZAL83/char-stemmer.

REFERENCES

[1] U. Kamath, J. Liu, and J. Whitaker, *Deep learning for NLP and speech recognition*. 2019, doi: 10.1007/978-3-030-14596-5.

[2] A. Dahou, S. Xiong, J. Zhou, M. H. Haddoud, and P. Duan, "Word embeddings and convolutional neural network for arabic sentiment classification," in *Proceedings of Coling 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 2418–2427.

[3] N. Y. Habash, *Introduction to Arabic natural language processing*. Morgan & Claypool Publishers, 2010, doi: 10.2200/S00277ED1V01Y201008HLT010.

[4] S. Khoja and R. Garside, "Stemming arabic text. Lancaster University, Lancaster, UK," *Computing Department*, 1999.

[5] M. Aljlayl and O. Frieder, "On Arabic search: improving the retrieval effectiveness via a light stemming approach," in *CIKM '02: Proceedings of the Eleventh International Conference on Information and Knowledge Management*, 2002, pp. 340–347, doi: 10.1145/584792.584848.

[6] L. S. Larkey, L. Ballesteros, and M. E. Connell, "Light Stemming for Arabic Information Retrieval," in *Arabic Computational Morphology*, Dordrecht: Springer Netherlands, pp. 221–243, doi: 10.1007/978-1-4020-6046-5_12.

[7] A. Soudi, G. Neumann, and A. van den Bosch, "Arabic computational morphology: knowledge-based and empirical methods," in *Arabic Computational Morphology: Knowledge-Based and Empirical Methods*, Springer, 2007, pp. 3–14.

[8] H. A. Almuzaini and A. M. Azmi, "Impact of Stemming and Word Embedding on Deep Learning-Based Arabic Text Categorization," *IEEE Access*, vol. 8, pp. 127913-127928, 2020, doi: 10.1109/ACCESS.2020.3009217.

[9] R. Ahmed, "Exploring The Impact of Stemming on Text Topic-Based Classification Accuracy," *Journal of Linguistics, Culture and Communication*, vol. 2, no. 2, pp. 204–224, 2024, doi: 10.61320/jolcc.v2i2.204-224.

[10] S. M. Yaseen and A. A. G. Al-Khulaidi, "A new stemming model based on data generation to enhance Arabic information retrieval," *Journal of King Saud University Computer and Information Sciences*, vol. 37, no. 8, pp. 1-17, 2025, doi: 10.1007/s44443-025-00192-2.

[11] A. Alosaimy and E. Atwell, "Tagging classical Arabic text using available morphological analysers and part of speech taggers," *Journal for Language Technology and Computational Linguistics*, vol. 32, no. 1, pp. 1–26, 2017, doi: 10.21248/jlcl.32.2017.212.

[12] A. A. Munshi, W. H. AlSabban, A. T. Farag, O. E. Rakha, A. A. Al Sallab, and M. Alotaibi, "Towards an automated islamic fatwa system: Survey, dataset and benchmarks," *International Journal of Computer Science and Mobile Computing*, vol. 10, no. 4, pp. 118–131, 2021, doi: 10.47760/ijcsmc.2021.v10i04.017.

[13] W. Antoun, F. Baly, and H. Hajj, "AraBERT: Transformer-based Model for Arabic Language Understanding," in *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*, 2021, pp. 9-15.

[14] E. Alnagi, R. Ghnemat, and Q. A. Al-Haija, "Boosting Arabic text classification using hybrid deep learning approach," *Discover Applied Sciences*, vol. 7, pp. 1-20, May 2025, doi: 10.1007/s42452-025-07025-x.

[15] Z. Ferhat *et al.*, "Functional Text Dimensions for Arabic Text Classification," in *Proceedings of The Second Arabic Natural Language Processing Conference*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2024, pp. 352–360, doi: 10.18653/v1/2024.arabicnlp-1.29.

[16] S. Ruder, "An Overview of Multi-Task Learning in Deep Neural Networks," *arXiv*, 2017, doi: 10.48550/arXiv.1706.05098.

[17] R. Collobert and J. Weston, "A unified architecture for natural language processing," in *Proceedings of the 25th international conference on Machine learning - ICML '08*, New York, New York, USA: ACM Press, 2008, pp. 160–167, doi: 10.1145/1390156.1390177.

[18] R. Abdul-Nabi, R. Obeidat, and A. Bsoul, "A Survey on Machine Translation of Low-Resource Arabic Dialects," in *2024 15th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan, Aug. 2024, pp. 1–6, doi: 10.1109/ICICS63486.2024.10638285.

[19] M. Jarrar, A. Birim, M. Khalilia, M. Erden, and S. Ghanem, "Arbanking77: Intent detection neural model and a new dataset in modern and dialectical arabic," in *Proceedings of ArabicNLP 2023*, 2023, pp. 276–287, doi: 10.18653/v1/2023.arabicnlp-1.22.
[20] S. Aftan and H. Shah, "Using the AraBERT Model for Customer Satisfaction Classification of Telecom Sectors in Saudi Arabia." *Brain Sciences*, vol. 13, no. 1, pp. 1-20, Jan. 2023, doi: 10.3390/brainsci13010147.
[21] F. El-Alami, S. O. El Alaoui, and N. E. Nahnahi, "Contextual semantic embeddings based on fine-tuned AraBERT model for Arabic text multi-class categorization," *Journal of King Saud University*, vol. 34, no. 10, pp. 8422–8428, 2022, doi: 10.1016/j.jksuci.2021.02.005.
[22] V. Ashish, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, p. I, 2017.
[23] R. M. Al-Khatib, T. Zerrouki, M. M. A. Shquier, and A. Balla, "Tashaphyne0. 4: a new arabic light stemmer based on rhyzome modeling approach," *Information Retrieval Journal*, vol. 26, no. 1, pp. 1-30, 2023, doi: 10.1007/s10791-023-09429-y.
[24] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2009.
[25] M. Elbes, A. Aldajah, and O. Sadaqa, "P-stemmer or NLTK stemmer for arabic text classification?," in *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, Granada, Spain, 2019, pp. 516-520, doi: 10.1109/SNAMS.2019.8931818.
[26] M. Galal, M. M. Madbouly, and A. El-Zoghby, "Classifying Arabic text using deep learning," *Journal of Theoretical and Applied Information Technology*, vol. 97, no. 23, pp. 3412–3422, 2019.

BIOGRAPHIES OF AUTHORS

Azal Alaswaad received the B.Sc. degree from the University of Thi-Qar, Iraq, and the M.Sc. degree in Computer Engineering from Amirkabir University of Technology, Iran. Currently, he is pursuing the Ph.D. degree in Artificial Intelligence at Iran University of Science and Technology. Research interests include natural language processing, artificial intelligence, machine learning, deep learning, and Arabic language processing. He can be contacted at email: azal.alamery3@gmail.com.

Behrouz Minaei-Bidgoli received the Ph.D. degree from the College of Engineering and Computer Science, University of Michigan, USA. He is currently a faculty member in the Department of Computer Engineering at Iran University of Science and Technology, Tehran, Iran. His research interests include artificial intelligence, machine learning, data mining, text mining, and natural language processing. He can be contacted at email: b_minaei@iust.ac.ir.