Arabic Auto-complete System Based On Long Short-Term Memory Network And Trie

Birzeit University

Faculty of Engineering & Technology

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Natural Language Processing and Information Retrieval Course

Ibraheem Alyan
1201180

Saleh Khatib
1200991

Feras Sandouka
1200779

***Abstract*—The project aims to develop an autonomous autocomplete system designed to predict Arabic keywords and phrases that users are likely to type. The system collects and uses aggregated queries frequencies and feeds them into an adaptive system, able to learn from correct and incorrect predictions to enhance its accuracy. The main objectives of the system is to be performant enough to keep up with user's typing speed. The system uses a long short-term memory recurrent neural network to predict phrases, and it uses a combination of the LSTM and a Trie data structure to predict the completion of words.**

I. INTRODUCTION

Autocomplete systems have become an essential part of our digital lives, helping us type faster and more accurately. However , traditional systems often rely on fixed dictionaries or rules, limiting their ability to adapt to how language changes and how we use it.

This project aims to create a smarter autocomplete system that learns and adapts over time. The system should be able to complete both the words themselves whole the user is still typing and also suggest next words and phrases based on the context.

It uses a special type of neural networks called a long short-term memory (LSTM) network, which is really good at understanding patterns in language and is very promising in sequential data prediction like in an auto complete system. In addition to the LTSM used for next-word prediction, a Trie data structure is used for same-word completion.

The system will continuously learn from what the user types, getting better at predicting the words and phrases we're likely to use.

The rest of this report is organized as follows. Section 2 provides an overview of existing autocomplete systems, highlighting relevant research in the field. Section 3 is about the methodology employed in this project, detailing the data collection, model architecture, and optimization techniques. Section 4 presents the results of the proof of concept of the system Finally, Section 5 concludes the report, summarizing the findings, and outlining potential directions for future research.

II. LITERATURE REVIEW

Auto completion systems are widely used in many industries, which is why many research and development teams came up with their own ideas and designs of auto completion systems, some of them are described as follows.

Dictionary-based methods, such as those described in [1], offer fast lookup times but are limited by the static nature of dictionaries. N-gram models, as explored in [2], capture common word sequences but struggle with rare terms or out-of-vocabulary words. Trie-based approaches, like those used in [3], provide efficient prefix-based search but can be memory-intensive for large dictionaries.

Recurrent Neural Networks (RNNs) and their variant, Long Short-Term Memory (LSTM) networks, have shown remarkable success in language modeling tasks [4]. These models can learn complex patterns in language and adapt to user-specific writing styles, making them suitable for personalized autocomplete systems.

III. METHODOLGY

There are multiple methods to implement an auto complete system. One of the most known machine learning methods is a neural network, there are many types of neural networks including convolutional, dense, recurrent, and many more. In this project the selected methodology was using a special type of RNNs called LSTM.

A. Recurrent Neural Networks

“A recurrent neural network (RNN) is a deep learning model that is trained to process and convert a sequential data input into a specific sequential data output.” [7], RNNs excel in tasks where the order and context of inputs matter, making them well-suited for language modeling, machine translation, and other natural language processing tasks including auto completion systems.

B. Long Short-Term Memory RNN

Long Short-Term Memory (LSTM) is a more advanced type of recurrent neural network (RNN) that captures long-term dependencies to handle sequence prediction tasks. Hochreiter and Schmidhuber developed LSTMs to address the limitations of traditional RNNs,

which struggle to learn long-term dependencies due to their single hidden state. LSTMs introduce a memory cell that can store information for long periods of time and is controlled by three gates: input, forget, and output. These gates control the addition, removal, and output of information from memory cells, allowing the network to selectively retain or discard data. This makes LSTMs especially useful for tasks like language translation, speech recognition, and time series forecasting.

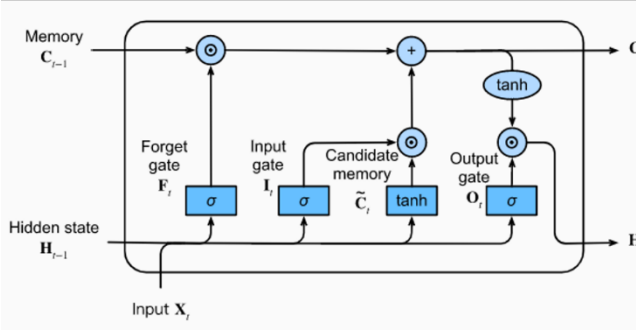


Figure 1 LSTM Architecture [6]

Bidirectional LSTMs (BiLSTMs) improve performance by processing data in both forward and backward directions, allowing for longer-range dependencies. Stacking LSTM layers allows networks to learn intricate patterns and hierarchies in sequential data. Despite being more complex and difficult to train than traditional RNNs, LSTMs have significant advantages in learning and retaining short-term dependencies, making them a valuable tool in an auto completion system.

C. *Trie Data Structure*

A Trie is a special type of tree that stores a collection of strings. It is useful for quickly locating and storing keys in a large dataset. It allows you to add, find, and delete keys, making it useful in computer science and information retrieval.

A Trie consists of nodes connected by edges. Each node represents a character or part of a string. The root node, where the Trie begins, represents an empty string. Each edge from a node exhibits a distinct character. The path from the root to any node displays the prefix of a string stored in the Trie.

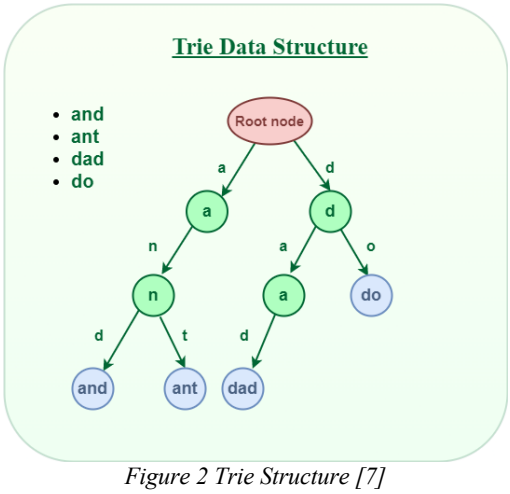


Figure 2 Trie Structure [7]

The Trie data structure stores strings using nodes and edges, each representing characters from a-z, with each node having up to 26 children corresponding to these letters. For example, to store the word "and," we start at the root node, mark 'a,' move to the 'a' node, mark 'n,' and then mark 'd' at the next level. Similarly, storing "ant" involves starting at the root node, noticing 'a' and 'n' are already marked, and then marking 't' at the next level. This structure efficiently handles the storage and retrieval of strings by their prefixes by leveraging character sequences.

Retrieving strings (words) by their prefixes is the core of word completion system.

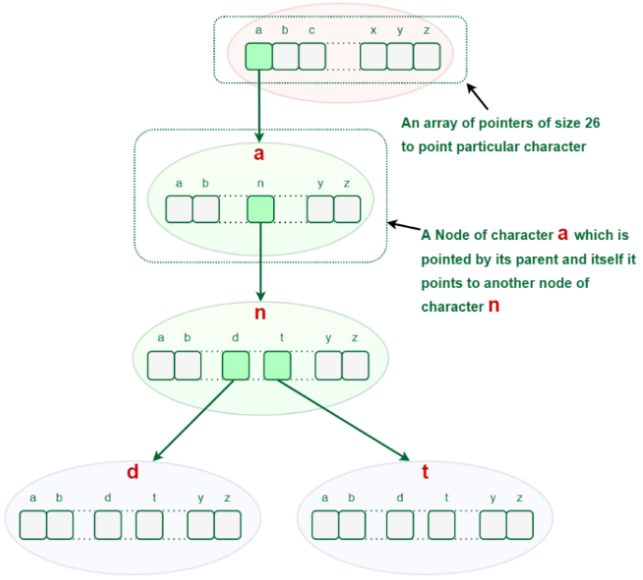


Figure 3 Insertion in Trie [7]

D. *Combining LSTM And Trie*

The system utilizes the LSTM model and tokenizer to generate predictions for the next word the user will type, and for word level completion the prefix of the word that the user is still typing is used to search the Trie, this will retrieve a large set of potential words. Then, it leverages the LSTM model to predict likely next words based on the preceding text. The intersection of these two sets of suggestions is returned, ensuring that the suggested words are both contextually relevant and valid completions for the current word.

E. *Adaptivity*

In order to make the system adaptive, after the user types a word, this word is taken with the previous text as feedback and are fed to the model in training mode, in a near-real-time manner, since real-time training would be quite expensive in terms of resources and might lag the system. In addition any new terms not found by the Trie are also added to it. This makes the system adaptive and can learn from user interactions.

IV. Proof Of Concept

Initially, the project aimed to leverage the "Large Arabic Twitter Data for Sentiment Analysis" [8] dataset from Kaggle, comprising over 4.51 GB of Arabic text. This was to be complemented with scraped Arabic Wikipedia pages. However, initial data processing, which involved converting the dataset into n-grams (up to 5-grams) using NLTK's `everygrams` function, resulted in an unmanageable data size exceeding 194 GB without completing quarter the dataset. Due to these limitations, the project shifted focus to developing a proof of concept (POC) to demonstrate the feasibility of the proposed methodology. This POC would use a smaller dataset, acknowledging the potential for lower accuracy and metrics.

A. Environment And Sample Data Set

To address this, an Amazon EC2 g5.8xlarge instance was procured, having 32 cores, an Nvidia A10G Tensor Core GPU with 24 GB of GPU memory, and 128 GB of RAM. Unfortunately, even this substantial computational power was insufficient for processing the entire dataset, or even a tenth of it.

B. Data Preparation

The sample dataset, comprising 270K entries, went through the pipeline shown in *Figure 4*, it was cleaned by removing symbols and emojis. Text was split based on newlines and dots. The **keras_nlp** tokenizer was employed for tokenization, and each word was assigned a unique ID. Everygrams up to 5-grams were generated for all combinations of subsequent words in each sentence.

Each resulting sequence was split, with the last word serving as the label for the sequence. These labeled sequences were then fed into the model.

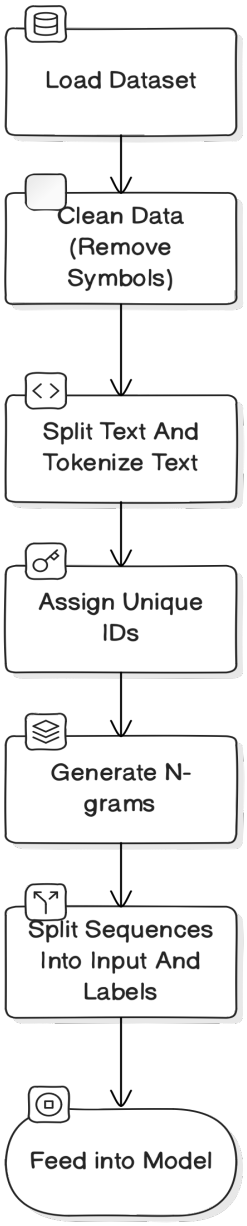


Figure 4 Data Preparation Pipeline

C. Model Architecture

The model consisted of the following layers:

1) ***Input Layer****: Words were converted into categorical representations using one-hot encoding (size: number of unique words).*

2) ***Embedding Layer****:* Words were transformed into dense vectors of fixed size (128), resulting in 1,616,384 parameters.

3) ***LSTM Layer****:* A bidirectional LSTM layer with 150 units (334,800 parameters) was used.

4) ***Dropout Layer****:* A dropout layer with a rate of 0.2 (no parameters) was added to mitigate overfitting.

5) ***Dense Layer****:* A dense layer with units equal to the number of unique words (3,801,028 parameters) produced the prediction as vector representing the predicted word.

D. Model Training

The model was fitted on the sample dataset with a categorical cross entropy loss function and an Adam optimizer, the training results were measured using metrics like accuracy in top 3 and top 5 in addition to perplexity and loss, all those metrics were plotted against epochs in the following figures.

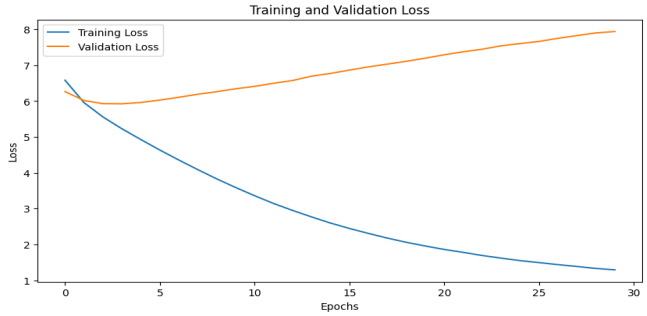


Figure 5 Loss Vs Epochs

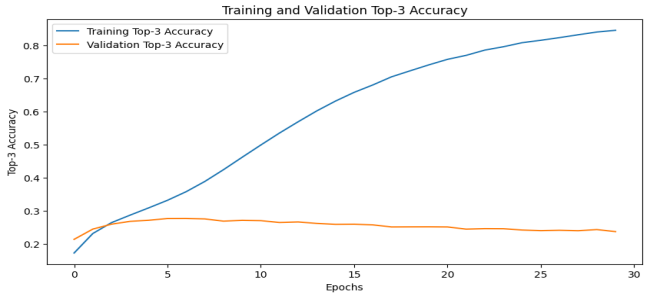


Figure 6 Top-3 Accuracy Vs Epochs

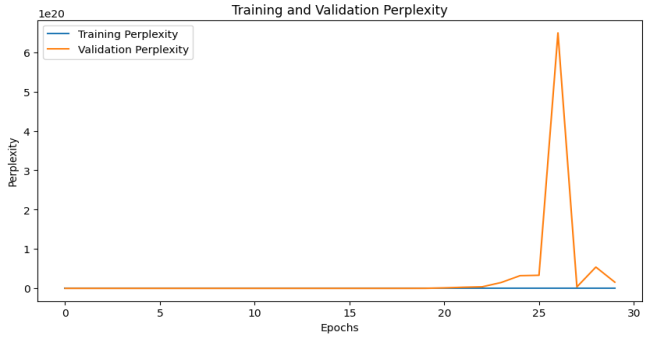


Figure 7 Perplexity Vs Epochs

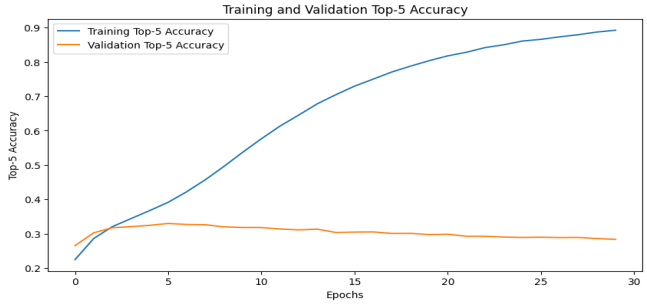


Figure 8 Top-5 Accuracy Vs Epochs

E. Backend And Model Integration

The backend of the autocomplete system is built using the FastAPI framework. The code establishes endpoints for predicting the text based on user input.

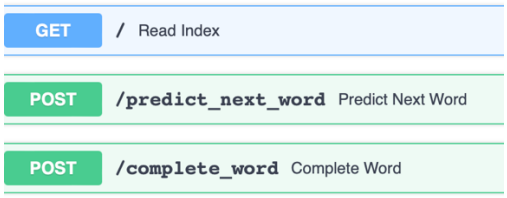


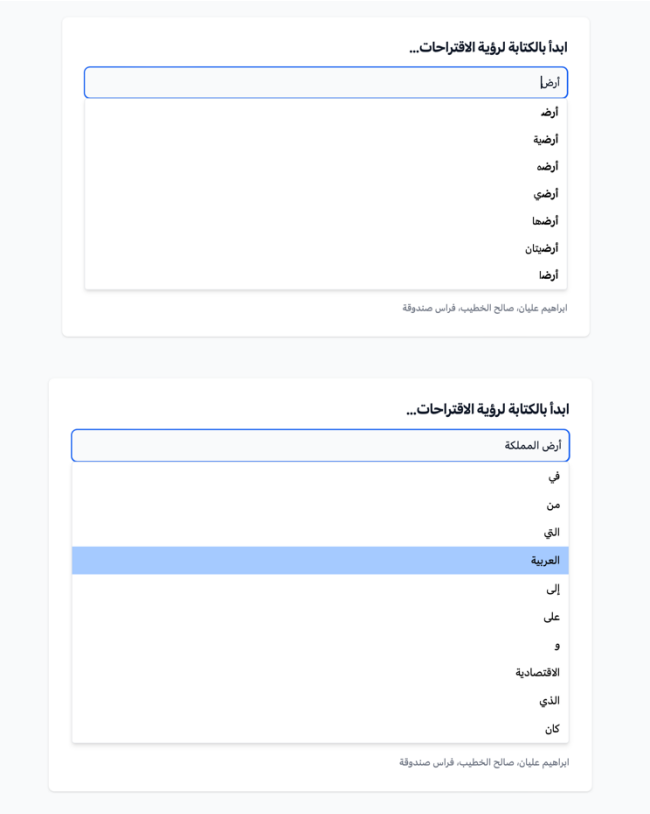
Figure 9 Backend Endpoints

1) ***predict_next_word****:* endpoint utilizes the loaded LSTM model and tokenizer to generate predictions based on the provided seed text. It returns a list of the top suggested words.

2) ***complete_word****:* endpoint combines the capabilities of the Trie and the LSTM model to suggest word completions. It first retrieves potential completions from the Trie based on the current word prefix. Then, it leverages the LSTM model to predict likely next words based on the preceding text. The intersection of these two sets of suggestions is returned, ensuring that the suggested words are both contextually relevant and valid completions for the current word.

F. User Interface And Demo

The user interface, built with HTML, Tailwind CSS, and vanilla JavaScript, features a minimalist design with a single text field. As the user types, the script dynamically determines whether to query the backend for next word suggestions or word completions based on the input. The fetched predictions are elegantly displayed in a dropdown menu beneath the text field, as showed in the following screenshots.



To enhance readability, the script highlights the typed prefix in the suggestions. Intuitive navigation is also present through arrow key controls, allowing users to easily scroll and select suggestions without requiring mouse interaction. Pressing enter either replaces the current word (for word completion) or appends the selected suggestion as the next word, and moves the cursor to the end.

The interface is designed for responsiveness, with the suggestion list updating in real-time as the user types, ensuring a smooth typing experience.

V. Conclusion

The results were not expected to be perfect since the models were trained on an extremely small subset of the dataset, the LSTM model which had over 6.4M parameter surely was unable to optimize its parameter using that small dataset, but it also showed the feasibility of such implementation and illustrated how recurrent networks can be extremely useful in sequential predictions.

Future work will focus on scaling up the model by incorporating larger datasets, optimizing computational resources, and exploring techniques to improve prediction accuracy.

VI. References

[1] J. Zobel and A. Moffat, "Inverted files for text search engines," ACM Computing Surveys (CSUR), vol. 38, no. 2, pp. 6-es, 2006.

[2] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," Computational linguistics, vol. 18, no. 4, pp. 467-479, 1992.

[3] R. de la Brianda, A. Navarro, and G. Pineda, "A practical implementation of compact directed acyclic word graphs," in International Symposium on String Processing and Information Retrieval. Springer, 2008, pp. 28–40.

[4] A. Graves, "Generating sequences with recurrent neural networks," arXiv preprint arXiv:1308.0850, 2013.

[5] "What is RNN? - Recurrent Neural Networks Explained - AWS," *Amazon Web Services, Inc.* https://aws.amazon.com/what-is/recurrent-neural-network

[6] G. P. Zhang, Y. Xia, and M. Xie, "Intermittent demand forecasting with transformer neural networks," Annals of Operation Research/Annals of Operations Research, Jun. 2023, doi: 10.1007/s10479-023-05447-7.

[7] GeeksforGeeks, "Trie Data Structure tutorial," *GeeksforGeeks*, Apr. 17, 2024. https://www.geeksforgeeks.org/introduction-to-trie-data-structure-and-algorithm-tutorials/

[8] "Large Arabic Twitter data for sentiment analysis," Kaggle, Jan. 31, 2020. https://www.kaggle.com/datasets/adelalharbi/rogue-content-and-arabic-spammers-on-twitter/data?select=TweetsStreamingTotal.csv

[9]