QTB

Quranic Toolbox

Graduation Project is submitted to Computer and Systems Engineering Department in Partial fulfillment of the requirements for bachelor's degree of Computer and Systems Engineering

BY

Omar Elsayed Abdelhadi Saeed

Sherif Emad Saad Zaghloul

Ali Ahmed Ali ElShimy

Mohamed Ali Mohamed ElBialy

Supervised By:

Dr.Mahmoud Zaki

2023

Abstract

QTB (Quranic toolbox) is Mobile application we developed using flutter utilizes speech-to-text technology and error detection features to enhance the user's ability to accurately recite the Quran and receive immediate feedback on their recitation. Also, it has a sematic search engine that searches through verses or Ahadith and returns semantically similar ones to the input query. It also includes a Tajweed correction feature that tells the user the percentage of right and wrong about how close the user pronounced the word correctly with the selected Tajweed rule. Our motivation behind this project is to address the challenges faced by individuals in memorizing the Quran, such as lack of personalized guidance and error detection. By integrating advanced technologies, the application aims to provide a user-friendly and interactive learning experience.

Table of Contents

Table of figures

Chapter 1: Introduction

1.1 Problem statement

The process of reciting and memorizing the Quran is a significant endeavor for Muslims worldwide, requiring dedication, focus, and repetition. However, traditional methods often lack personalized guidance and effective error detection mechanisms, making the memorization journey challenging and potentially discouraging for learners. In addition, limited opportunities for interactive engagement with the Quran further hinders the deep understanding and meaningful connection with its teachings.

To address these limitations, this graduation project aims to develop an innovative mobile application that leverages speech-to-text technology and error detection to facilitate Quran memorization and enhance user interaction. By utilizing advanced natural language processing techniques, the application will provide users with personalized guidance, immediate feedback, and tailored learning pathways, enabling efficient memorization and comprehensive understanding of the Quran.

Key Challenges to Overcome:

- Speech-to-Text Accuracy: Developing a robust speech recognition system capable of accurately converting user voice input into text, accounting for variations in pronunciation, intonation, and linguistic nuances specific to Quranic recitation.
- Error Detection and Feedback: Designing an intelligent error detection mechanism that analyzes user input against the correct Quranic verses, identifies mistakes, and provides timely and constructive feedback to facilitate error correction and improved recitation and memorization accuracy.
- Interactive Learning Experience: Creating an engaging and interactive user interface that encourages active participation with the Quran, offering a lot of features.
- Scalability and Performance: Ensuring the application can handle a vast database of Quranic verses, maintaining fast response times even with a large user base, and optimizing resource usage to accommodate various device specifications and network conditions.
- Usability: Developing a user-friendly interface with intuitive navigation, suitable for individuals of all ages and technological proficiency.

1.2 Objectives and Scope

We aimed to create a Quran application using the Flutter framework, a popular cross-platform development tool. The motivation behind this project is to provide users with an immersive and enriching experience, utilizing advanced technologies to enhance their engagement, and understanding of the Quran. By integrating features such as speech-to-text technology that can recognize Tashkeel marks, Tajweed correction, and a semantic search engine, the application seeks to revolutionize the way individuals interact with and learn from the Quran.

- **Chapter One** serves as an introduction and discusses the objectives and scope of the project, emphasizing the focus on utilizing Flutter to develop a robust and feature-rich application.

- **Chapter Two** Explains the system design, functional diagrams, non-functional diagrams also include sequence and Use-case diagrams, Also, talking about the software development workflow.

- **Chapter Three** explores the integration of speech-to-text technology, enabling users to recite Quranic verses and have their voice input transcribed accurately. This feature enhances the memorization process by eliminating the need for manual notetaking and providing users with an interactive and hands-on experience.

- **Chapter Four** delves into the Tajweed feature, the set of rules governing the proper pronunciation and intonation of Quranic verses, is crucial for understanding and reciting the Quran accurately.

- **Chapter Five** focuses on the development of a semantic search engine for the Quran. Leveraging advanced linguistic models and vector representations of data, users can search for specific Quranic verses based on meaning, allowing for a deeper exploration, and understanding of the sacred text.

- **Chapter Six** covers the deployment of the models to Hugging Face, a popular platform for hosting and sharing machine learning models. This

chapter provides insights into the process of preparing and deploying the speech-to-text and semantic search models.

- **Chapter Seven** delves into the Flutter workflow, providing an overview of project structure and organization. It explores the design of the user interface using Flutter's powerful UI toolkit and highlights different approaches to state management.

- **Chapter Eight** focuses on the integration of the developed features into the Quran application. It covers the implementation and seamless integration of the speech-to-text, and semantic search engine functionalities, ensuring a cohesive and user-friendly experience.

- **Chapter Nine** focuses on the used tools and technologies thorough out the application

- **Chapter Ten** concludes the project, reflecting on the achievements, challenges faced, and lessons learned throughout the development process. It also provides recommendations for future enhancements and opportunities to further improve the application.

The successful implementation of this graduation project will provide an innovative solution that empowers users to effectively memorize the Quran, with enhanced interactivity and error detection capabilities. By integrating technology with religious studies, this application will contribute to promoting a deeper understanding and connection to the Quranic teachings, fostering lifelong learning and spiritual growth.

1.3 Overview of Features

The Quran application we developed using Flutter encompasses a lot of features designed to enhance the user's experience and engagement with the Quran. By integrating advanced technologies,

Here are all application features that we implemented:

- Quran Recitation Feature
- Tashkeel Correction
- Tajweed Correction
- Memorization Feature
- Quran semantic search Engine
- Ahadith semantic search engine
- User-Friendly Interface
- Personalization Options
- Offline Access

Chapter 2: System Design:

2.1 System Requirements:

Application side:

To ensure optimal performance and functionality of our application developed using Flutter, the following system requirements are recommended:

- Operating System: The application is designed to run on various platforms, including Android and iOS. Therefore, it requires a compatible operating system, such as Android 5.0 (Lollipop) or higher for Android devices, and iOS 11 or higher for Apple devices.

- Device Memory: The application requires a device with sufficient memory to accommodate the application itself, as well as any downloaded Quranic content, recitations, and additional data. It is recommended to have at least 2GB of RAM for smooth performance.

- Storage Space: Adequate storage space is necessary to store the application, its data, and any downloaded content. Depending on the size of the application and the desired level of offline access, it is recommended to have at least 100MB of available storage space.

- Processor: The application is designed to run on devices with a minimum processor speed of 1.5GHz or higher. This ensures smooth navigation, search functionality, and audio playback.

- Connectivity: While the application supports offline access to Quran content, an internet connection is required to download updates, for recitation and memorization feature also for semantic search.

Models and Training side:

- Server: A server that has a least 2 CPU core and 8 GBs of memory to run the models and train the models on the new data

2.2 Stake holders

As a Quran memorization and interactive learning application, there are several stakeholders who have an interest or involvement in the project. Here are some key stakeholders:

- Users: The primary stakeholders are the individuals using the application for Quran memorization recitation and semantic search. They include students, learners, and enthusiasts seeking to deepen their understanding and connection with the Quran.

- Developers/Software Engineers: The team responsible for designing, developing, and maintaining the application. They play a critical role in implementing the required features, ensuring functionality, and addressing technical challenges.

- Quranic Scholars and Educators: Scholars and educators who provide guidance, insights, and expertise in Quranic studies, Tajweed rules, and understanding the Quranic text. Their input helps shape the content, accuracy, and educational aspects of the application.

- Designers and User Experience Specialists: Professionals responsible for creating an intuitive and visually appealing user interface, ensuring a seamless and engaging user experience. They focus on usability, accessibility, and incorporating user feedback.

- Islamic Institutions and Organizations: Organizations focused on promoting Quranic education and Islamic studies. They may provide support, guidance, and collaboration opportunities for the application, aligning it with their educational initiatives.

- Community and User Groups: Communities of Quran learners, students, and individuals interested in Quranic studies. They provide valuable feedback, suggestions, and support, helping shape the application's ongoing development and improvement.

2.3 Functional Requirements:

- User Registration and Login: Users should be able to create accounts, log in securely, and access personalized features and progress tracking.

- Quran Text Display: The application presents the Quran text in a user-friendly and visually appealing manner, with options for resizing and customizing the font.

- Speech-to-Text Conversion: Users can recite Quranic verses aloud, and the application converts their speech into text for further analysis and error detection.

- Error Detection and Feedback: The application detects errors in the user's recitation, such as mispronunciations or skipped words, and provides feedback to guide the user in making corrections.

- Memorization Tracking: The application allows users to mark and track their progress in memorizing specific verses or chapters.

- Semantic Search: Users should be able to search for Quranic verses based on keywords, meanings, or specific topics, with the application returning relevant results and displaying them in an organized manner.

- Verse Exploration and Contextual Information: The application should provide additional information about individual verses, including translations, explanations, historical context, and related scholarly interpretations.

2.4 Non-functional Requirements:

- Performance: The application should be responsive and provide seamless user experiences, with minimal latency and fast response times for speech-to-text conversion and error detection.

- Scalability: The application should be designed to handle a large user base and effectively manage the storage and retrieval of Quranic text, user data, and progress tracking.

- Security: User accounts, personal data, and progress should be securely stored and protected using industry-standard encryption and authentication mechanisms.

- Usability: our application has an intuitive and user-friendly interface, catering to users of different ages and technological proficiency.

- Cross-platform Compatibility: The application should be developed to be compatible with multiple platforms, such as iOS and Android, allowing users to access the application from their preferred devices.

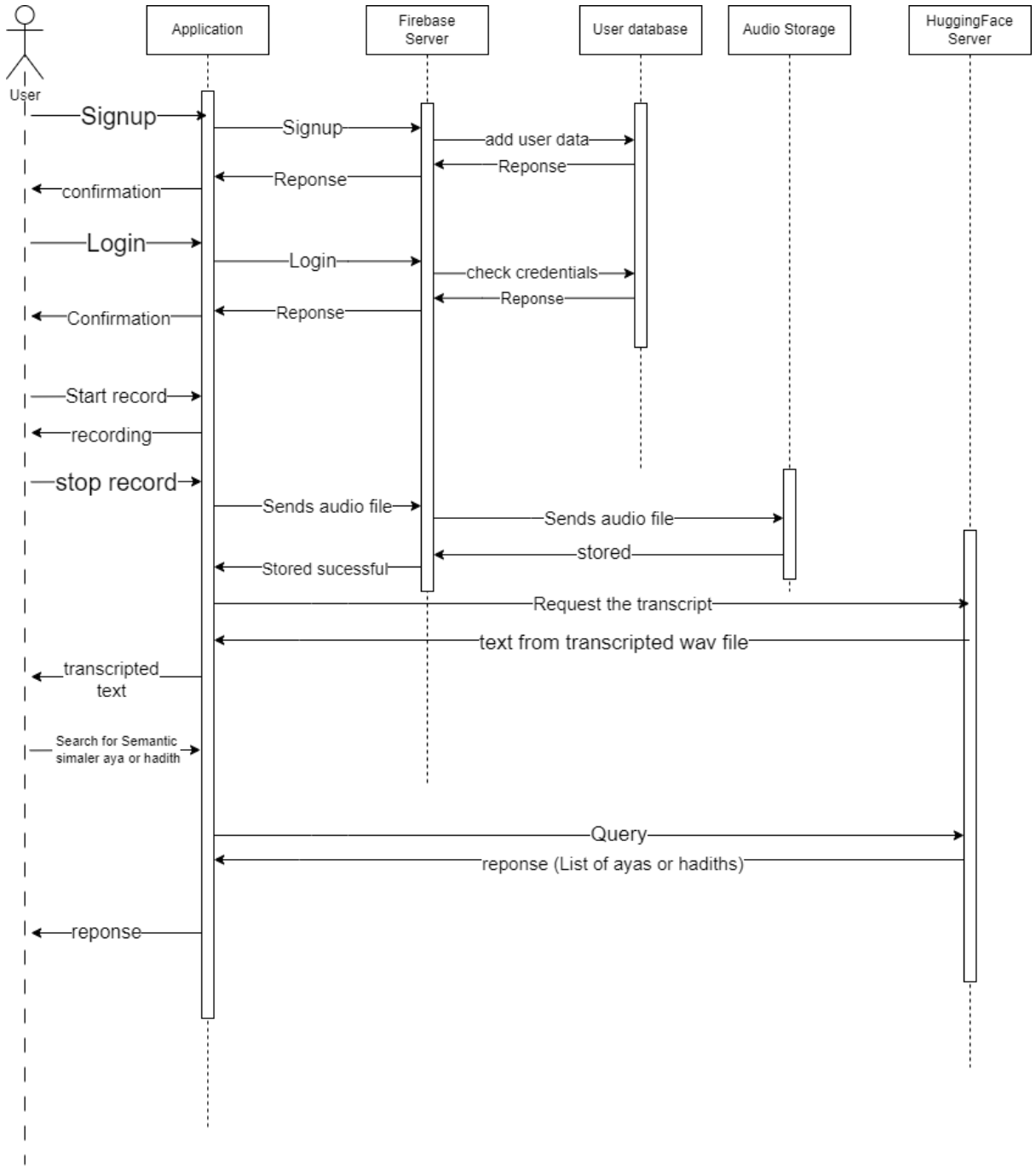
- Offline Functionality: The application should have the capability to allow users to access previously accessed content, including Quranic text, translations, and explanations, even when they are offline.

- Robustness and Reliability: The application is designed to handle unexpected errors or interruptions gracefully, minimizing data loss or disruption to the user experience.

2.5 Diagrams

2.5.1 Sequence diagram

An abstract sequence Diagram explain how the application works:



2.5.2 Use case diagram

use cases:

1. Quran Study and Research:

- Users can search for specific verses or topics in the Quran based on meaning, enabling them to conduct in-depth studies and research.

- The application provides access to Tashkeel (diacritical marks) for recitations, aiding users in understanding and analyzing the pronunciation and grammatical aspects of the Quranic text.

2. Recitation and Reflection:

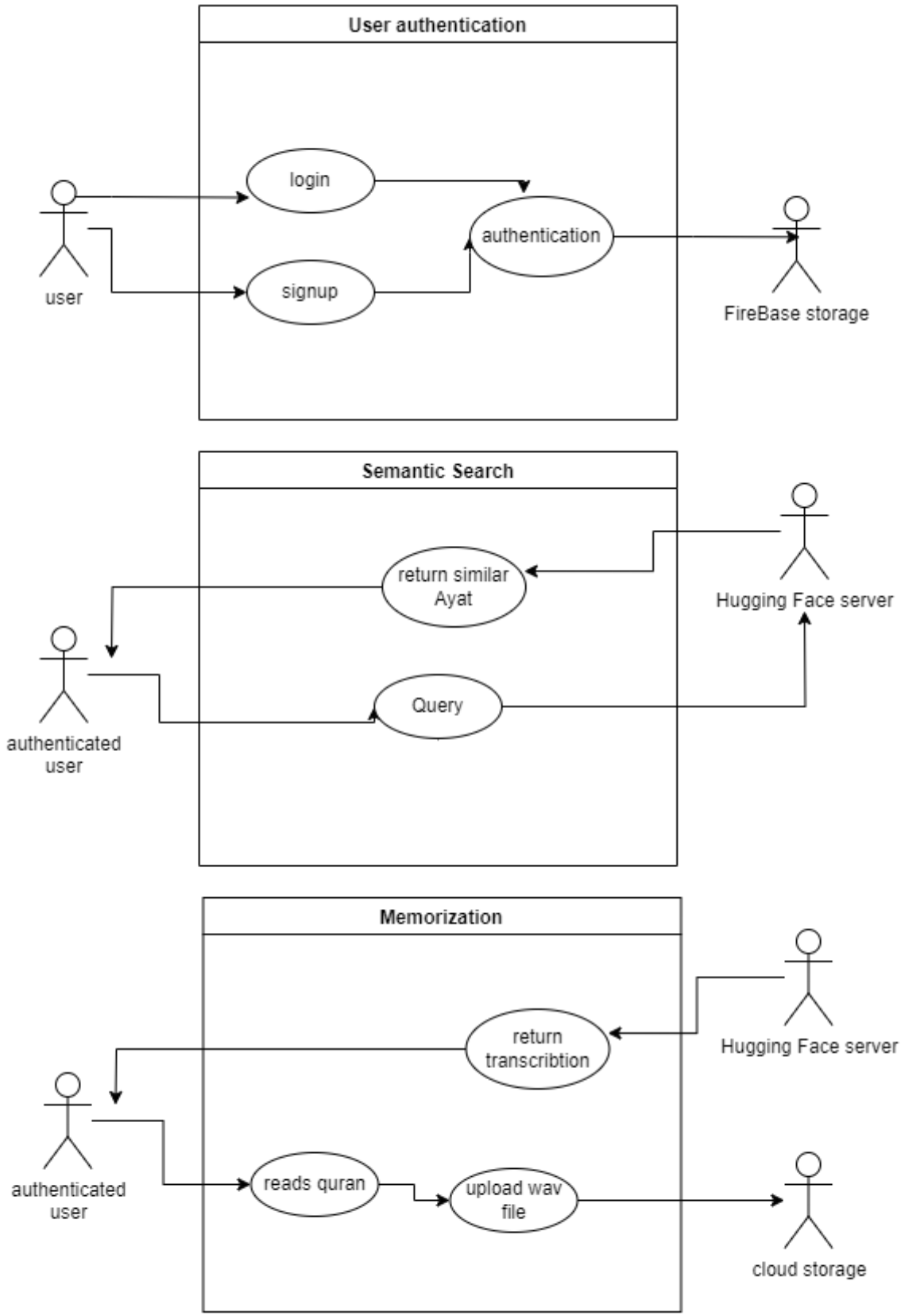
- The integration of Tashkeel enhances the recitation experience, enabling users to follow along with proper pronunciation and accentuation.

- The application encourages reflection and contemplation by providing a platform to engage with the Quranic text through recitation and introspection.

3. Quran Memorization:

- The application offers tools and features to support users in their journey of memorizing the Quran.

Diagram:


User authentication
login
signup
authentication
user
FireBase storage
Semantic Search
return similar Ayat
Query
Hugging Face server
authenticated user
Memorization
return transcribtion
Hugging Face server
reads quran
upload wav file
authenticated user
cloud storage

2.6 Software development methodology

While writing the application source code we focused on using a development methodology to help maximize the potential of all team members while making collaborating as easy as possible this is why we chose to work using Agile methodology, which is used in nearly all big software companies around the world. We split the features into sprints to make sure that we can track our progress on a two-week basis and that we're moving along with the project as planned.

In Addition to the Agile workflow, we used git as our version control system (VCS). Feature-branch workflow which was crucial to make contributing in our project easy and have less issues whilst integrating the project's different components. This workflow involves creating a new branch for each new feature and then after finishing it we use git pull request to revise the code and merge it to the master branch to ensure that everything is in sync.

After that comes the software releases versioning system which is a standard convention to keep track of the software version, backward compatibility as well the bug fixes. Currently we're in version 3.4.1.

The number 3 represents the major release number, 4 is the minor release and 1 represents the bug fixes in the application.

Chapter 3: Speech to Text:

3.1 Introduction

Deep Learning (DL) changes many Machine-Learning (ML) fields that heavily depend on domain knowledge. Decades of research in modeling this domain knowledge can be replaced with DL models with higher accuracy and less manual labor. In this section, we will focus on applying DL in speech recognition. But it is illusional to conclude ML will not play a major role in moving us forward. Many complex AI systems, including AlphaGo, apply many ML techniques. Speech recognition is no exception.

What is supervised deep learning?

Build a complex function estimator to make predictions.

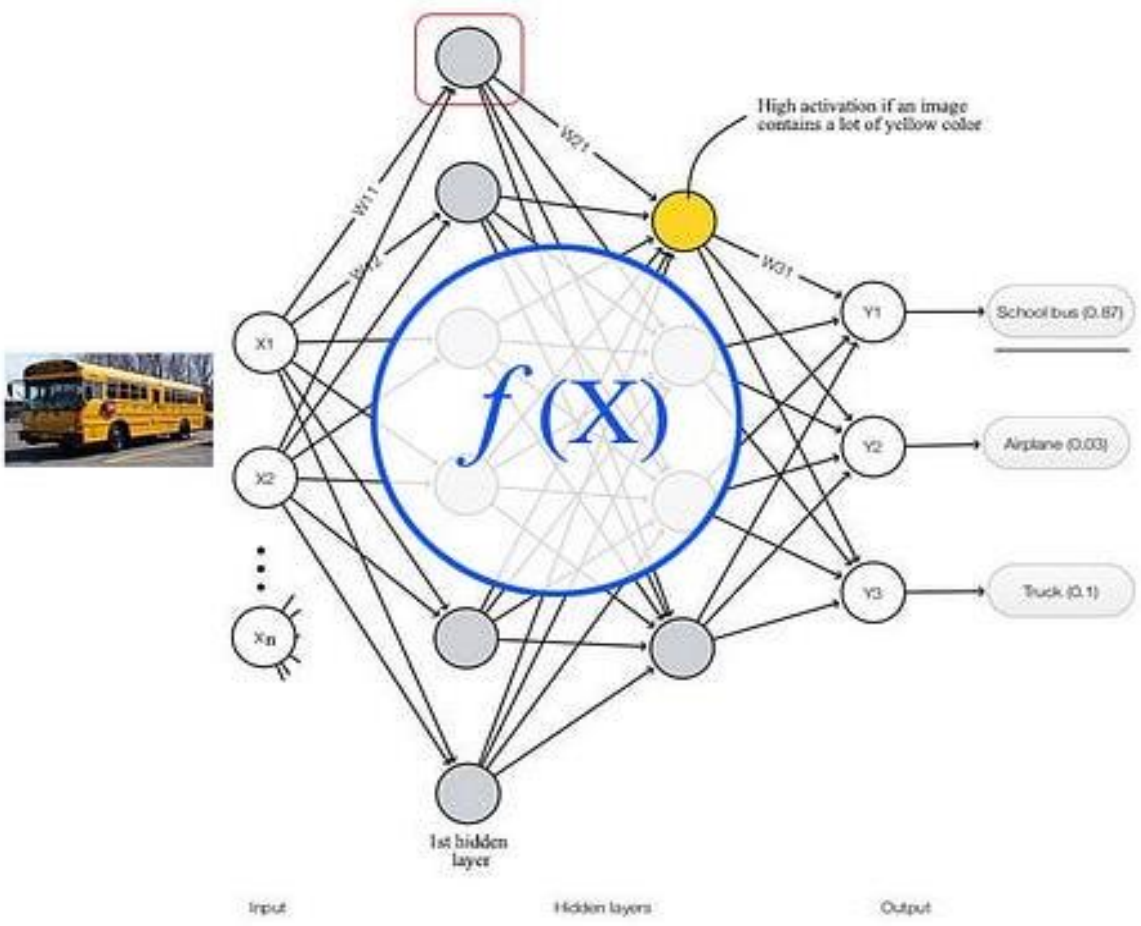


Figure 1 a typical neural network pipeline

X is the input features. From this perspective, whenever we need a complex mapping between an input and an output, we can apply DL. Let's recap the typical ASR approach (Automatic Speech Recognizer) in ML.

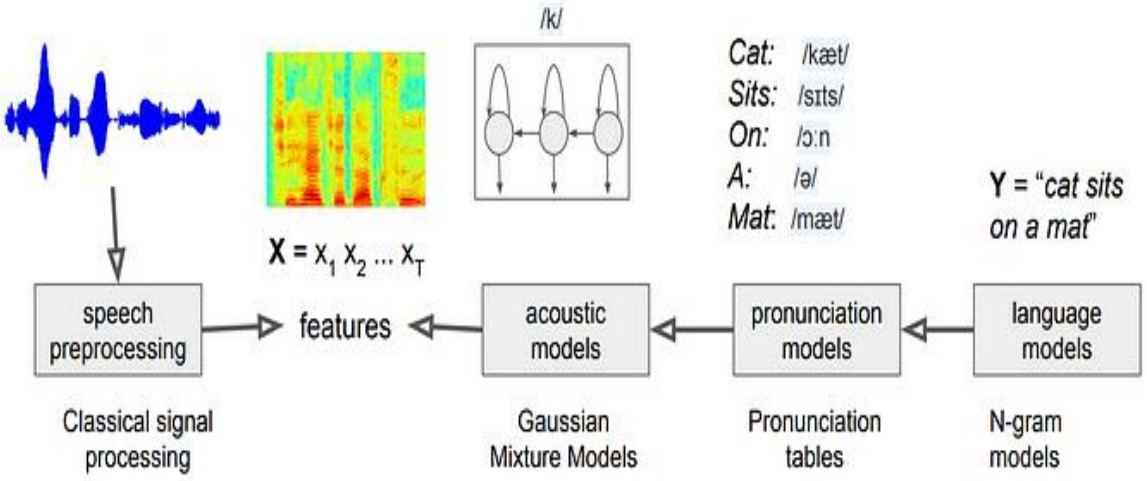


Figure 2 pipeline of speech2text

It involves a language model (say a bigram model)

$$P(W_n \mid W_1, W_2, \ldots, W_{n-1}) \approx P(W_n \mid W_{n-1})$$

, and an acoustic model like

$$b_j(\mathbf{x}) = p(\mathbf{x} \mid S = j) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$$

In the language model, the input is the previous word(s) and the output is the distribution for the next word. For the acoustic model, the input is the HMM state and the output is the audio feature distribution. We can model the distribution $p(y \mid x)$ as $f(x)$ using a deep network f.

But the acoustic model discussed before is a generative model $p(x|s)$. To adopt the same design, we compute $p(x|s)$ from $p(s|x)$ using the Bayes' Theorem. The evidence $p(x)$ is invariant of the word sequence. Therefore, it can be ignored in the optimization.

So given the features in an audio frame, we can use a deep network to predict $p(s|x)$ and apply Bayes' Theorem to estimate $p(x|s) \propto p(s|x) / p(s)$. In addition, we can include the neighbor frames as input. This creates a better phone context for better predictions.

Feature Extraction

In ML speech recognition, we extract MFCC or PLP features from the audio frames. It includes steps like an inverse discrete Fourier transform (IDFT) to make features less correlated with each other. Also, we only take the lower 12 coefficients. In general, ML is stingier in feature selection.

DL is good at untangling data and finding correlations. Therefore, it requires no or less pre-processing. It can handle a much larger number of input features and there is no need to un-correlate them first. We just need to design a more complex model and let's it learn from the data. In ASR with DL, we skipped the last few steps including IDFT. We simply take the Mel filter bank, apply the log and normalize it with the speaker or corpus' mean and variance.

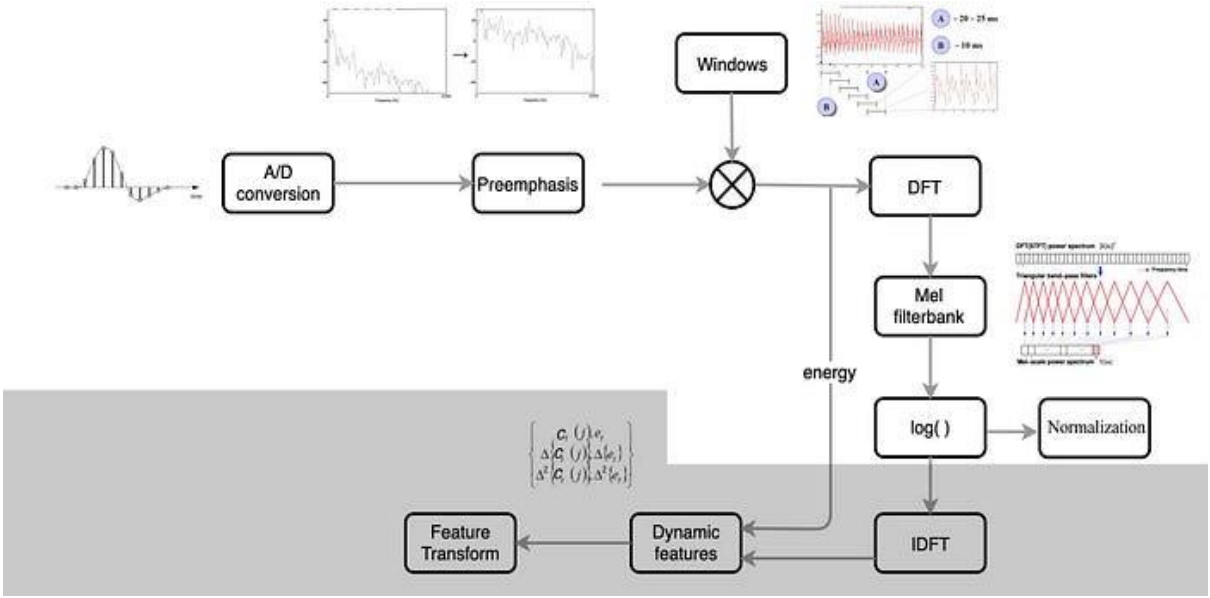


Figure 3 the inverse form

The next question we need to ask is what deep networks we should use. There are three major deep network categories.

Fully connected network (FC)

We can use the features from the Mel filter bank as input to an FC network. Some ASR FC models contain 3–8 hidden layers with 2048 hidden units in each layer. This model can predict the distribution of the context-dependent states (say 9304 CD triphones) from the audio frames. The following is an early comparison between a GMM-HMM ASR with a DNN-HMM (deep network). Lower values demonstrate better results.

CNN/TDNN

FC networks are computationally intense. It requires a lot of model parameters even for reasonable feature size. CNN takes advantage of locality and discovers local information hierarchically. CNN is more efficient if the information has a strong spatial relationship. It allows deeper layers and creates a more complex function.

Audio speech is time-sequence data. Instead of applying a 2D convolution filter, we use a 1-D filter to extract features across multiple frames in time. This is the Time-delay neural networks (TDNN). In the TDNN on the left below, the yellow box takes input from 5 nodes in the previous layers with each node containing 700 features.

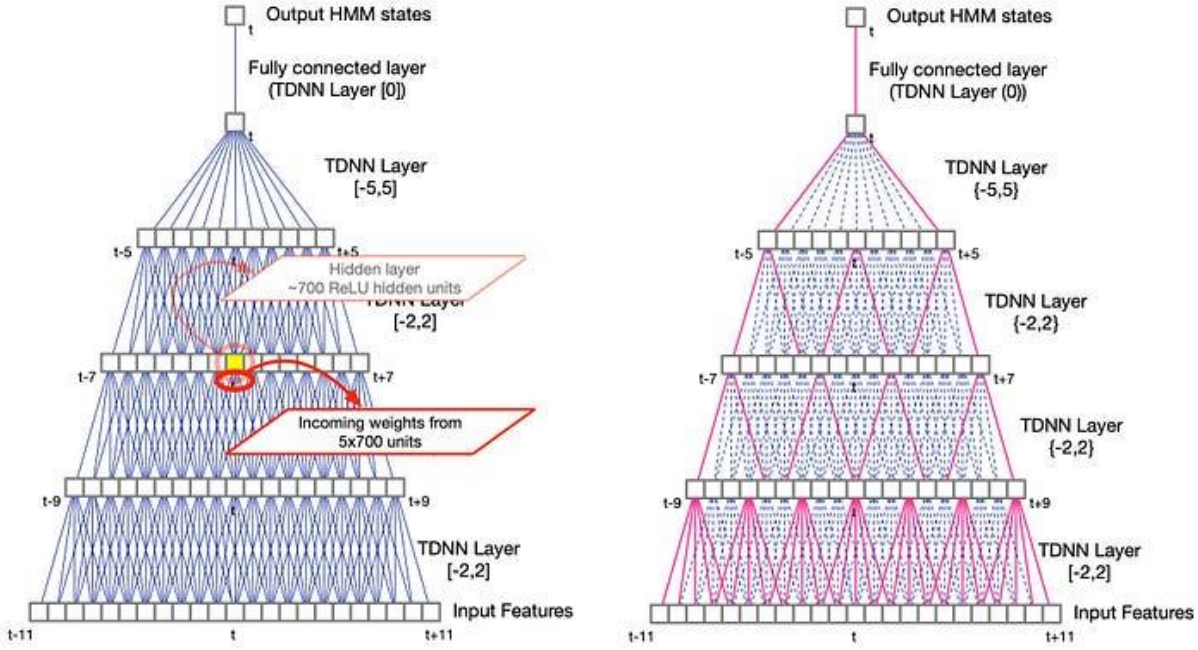


Figure 4 TDNN

here are some ASR performance comparisons between the fully connected network and TDNN.

Database	Size	WER		Rel.
		DNN	TDNN	Change
Res. Management	3h hrs	2.27	2.30	-1.3
Wall Street Journal	80 hrs	6.57	6.22	5.3
TedLIUM	118 hrs	19.3	17.9	7.2
Switchboard	300 hrs	15.5	14.0	9.6
Librispeech	960 hrs	5.19	4.83	6.9
Fisher English	1800 hrs	22.24	21.03	5.4

Figure 5 benchmark

RNN (LSTM & GRU)

Solving a temporal problem with a spatial concept may not be ideal. Alternatively, we can use a deep network designed for time-sequence data. The following is the RNN. Each cell contains a cell state c and outputs h. The cell state is determined by the current input and the previous states including the previous cell state and the previous output.

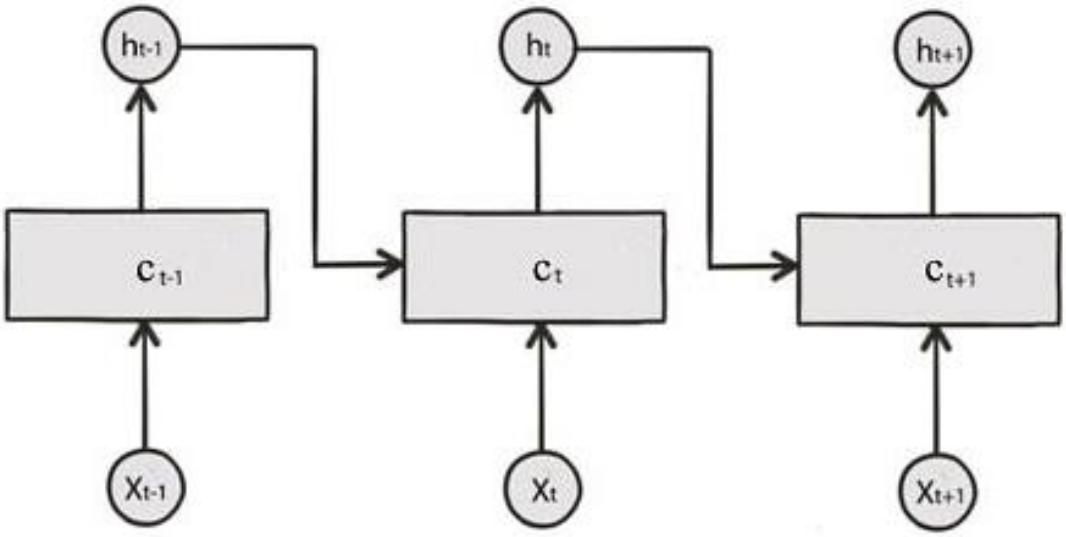
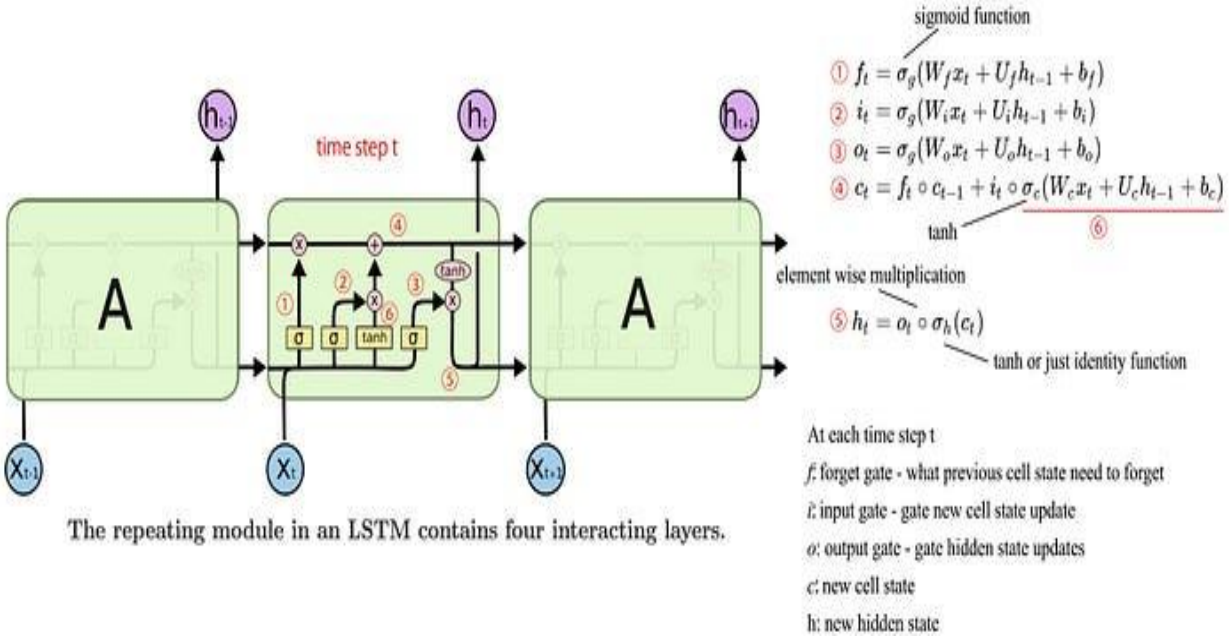


Figure 6 Rnn Unit

Below is the diagram for the LSTM (probably you would not understand the diagram until you are familiar with LSTM). It composes of the forget gate f, and the input gate i. f controls what information in the previous cell state will be forgotten. i controls what information in the current input and the previous output will be used to update the current cell state.



The repeating module in an LSTM contains four interacting layers.

Figure 7 LSTM

We can also stack the RNN cells to make the model more complex. Another possibility is bidirectional RNN. We have a forward pass and a backward pass with results combined later. For example, we add the result together followed by an FC layer. We can even combine both concepts together.

Here is an example of applying the concept above in classifying CD states.

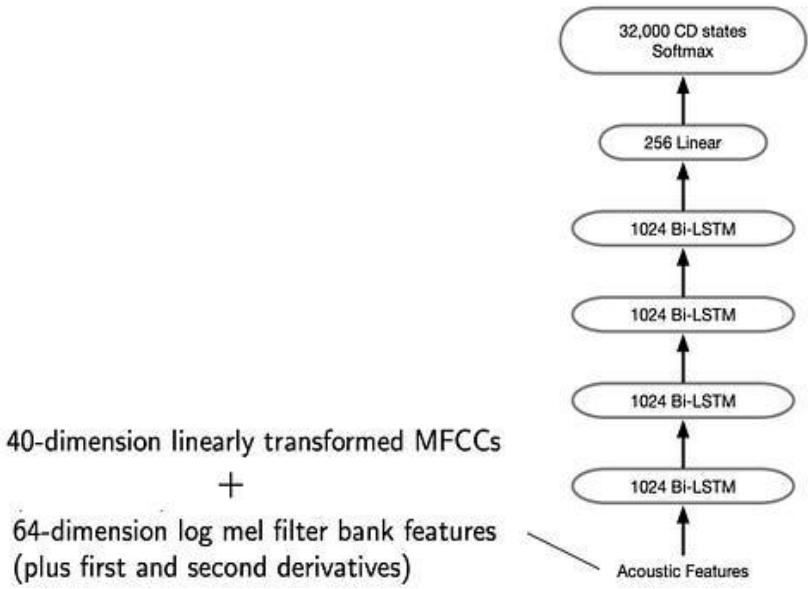


Figure 8 LSTM Classifier

The following is the testing result for different models.

Network Architecture	Test Set WER/% Switchboard	CallHome
GMM (ML)	21.2	36.4
GMM (BMMI)	18.6	33.0
DNN (7x2048) / CE	14.2	25.7
DNN (7x2048) / MMI	12.9	24.6
TDNN (6x1024) / CE	12.5	
TDNN (6x576) / LF-MMI	9.2	17.3
LSTM (4x1024)	8.0	14.3
LSTM (6x1024)	7.7	14.0
LSTM-6 + feat fusion	7.2	12.7

But so far, we have only taken incremental approaches, like DNN Hybrid Acoustic Models, to integrate DL into the HMM models. Can we design a new approach from scratch?

Connectionist temporal classification (CTC)

ASR decoding is mainly composed of two major steps: mapping and searching.

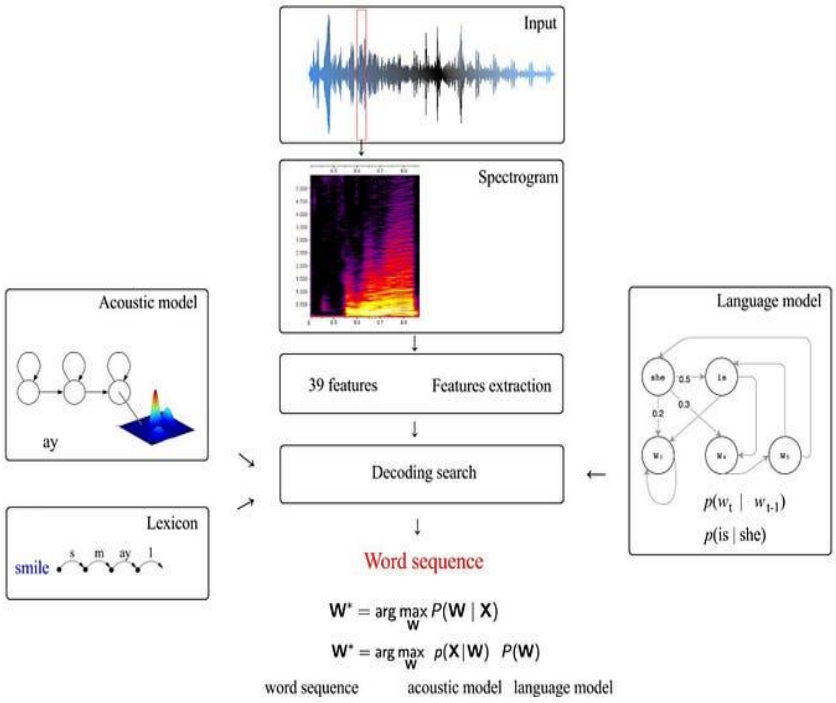


Figure 9 ASR Baselines

In the mapping, we map the acoustic information of an audio frame to a triphone state. This is the alignment process. This is a many-to-one mapping. It maps multiple audio frames to the same triphone state.

The alignment maps acoustic information to a phone. Then, we search for the phone sequence for the optimal word sequence. However, we need to account for multiple paths that produce the same results. This makes things complicated and requires us to use complex algorithms, like forward-backward (FB) or Viterbi algorithm.

DL learning is good at mapping. It opens the door for an alignment-free one-to-one mapping which maps an audio frame to a relatively high-level component. Then, we can search for it. This gives us a head start and bypasses the complicated alignment process. That is the core concept of CTC. While in early research, the deep network works with phonemes, now the deep network works with character sequences. This frees us further from the pronunciation lexicon and the phonetic decision tree.

The deep network generates a probability distribution for all characters. We don't greedily pick the most likely character in each time step. There are multiple paths that represent the same word (details later). We need to sum over them to find the optimal paths.

Since people pause in speech, within or between words, we introduce empty tokens <b> (or written as ε) to model such behavior. Other characters like ? and ! are also introduced. The following diagram indicates how we use bi-direction RNN to score characters for each observed audio frame. Then we apply softmax to create the probability distribution for characters in each time step.

CTC Compression Rule

Previously, we said multiple paths may produce the same word. Given the path "aaapɛppllɛe", what is the word that it represents? For that, we apply the CTC compression rule. Any repeated characters will be merged into one. Then, we remove ε for the final word.

As noted in this example, to output the word apple, the deep network needs to predict a ε between the two pp in a*pp*le. Therefore, ε also serves the purpose of separating repeated characters in a word. Here are the valid and invalid predictions for the word "cat".

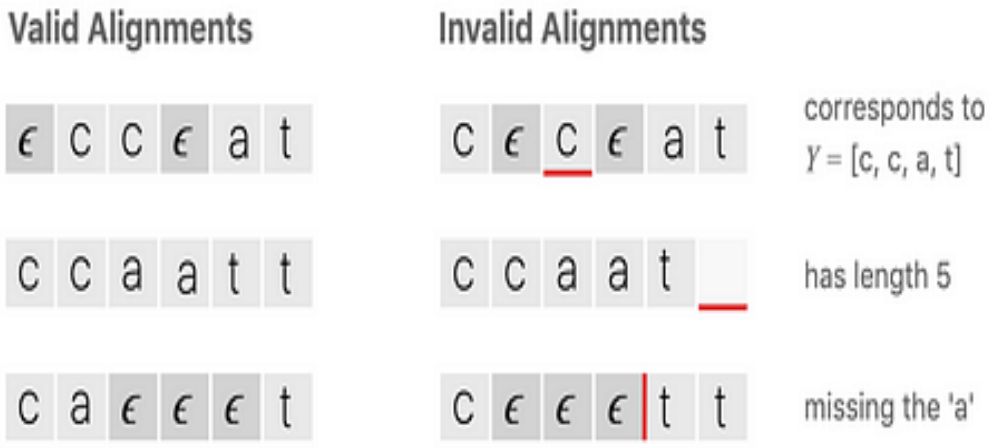


Figure 10 Aligmnent

The alignment-free concept in CTC pushes the complexity from mapping to searching. But by outputting character distributions and getting rid of lexicon and decision trees, the search may not be that bad.

CTC Loss

To find the most likely word sequence, we search for different path combinations. We need to sum over all paths that generate the same word sequence. For example, to find the probability for the word “hello”, we sum over all the corresponding paths like “heɛllɛlloo”, “hhellɛɛlɛo”, “ɛɛɛllɛɛloo” etc.

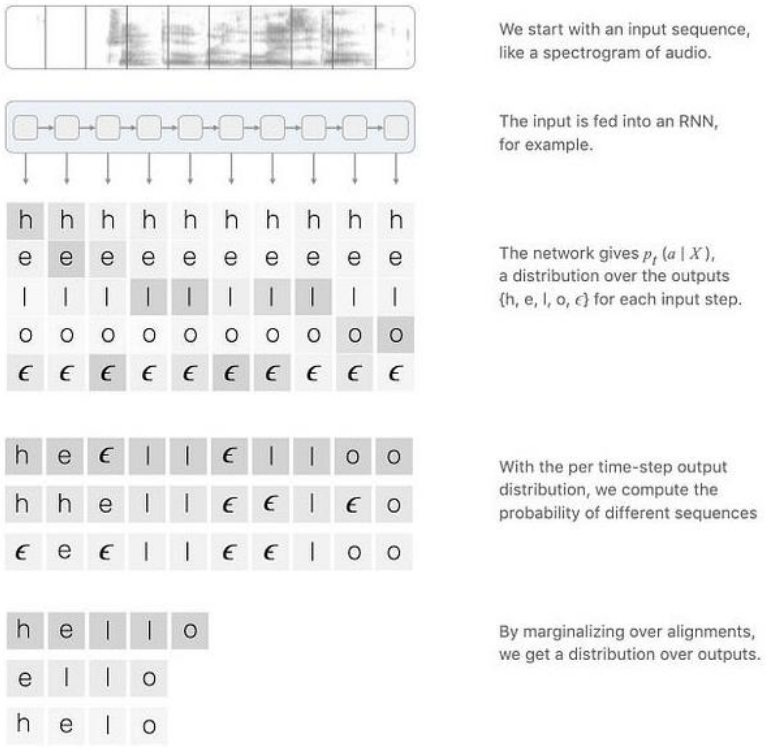
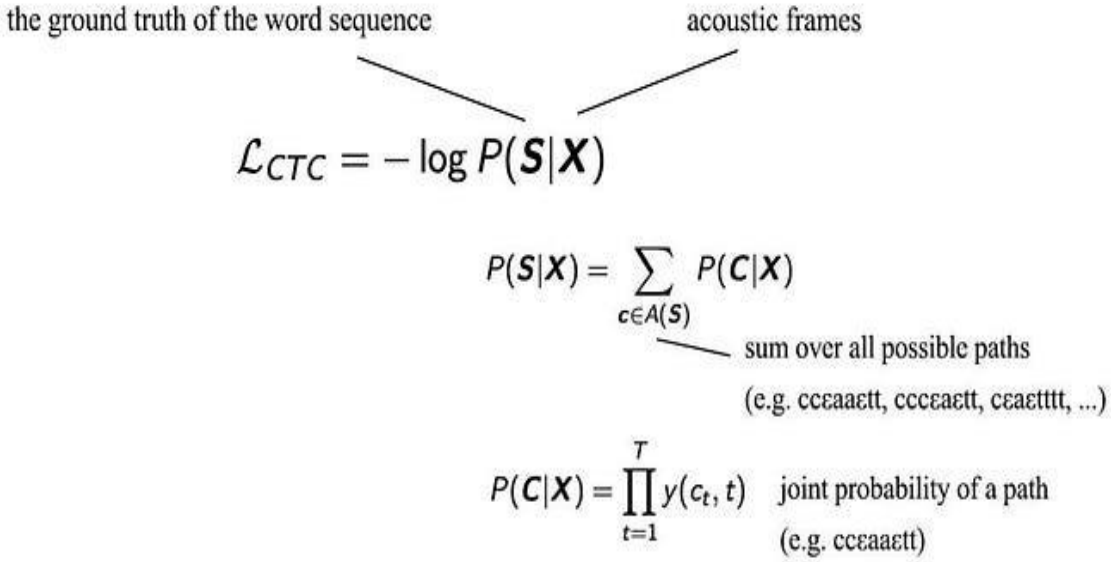


Figure 11 Decoding

Here is the corresponding CTC loss function:



Dynamic Programming

CTC delays the complexity of the algorithm from the mapping to the searching. In CTC, we sum over all paths that produce the same word sequence. Then, we find the sequence with the highest probability. To calculate the probability of a word sequence efficiently, we apply the same principle in FB or Viterbi algorithm by restating the problem as a recursive problem and solving it with dynamic programming. In FB, we compute the forward probability α. We will do something similar here. The following diagram demonstrates the possible paths that can be observed as the word "ab" during the length of the observed utterance (say the length is 6 timesteps). The starting state is either "a" or "ε" and we will either land on "b" or "ε" at timestep 6.

Let's expand the decoded word say "ab" to "εaεbε" with "ε" added. We will call this expanded sequence Z. This is the vertical axis in our transition diagram. Let's focus on the orange node below which recognizes "εa" in *Z* already. For the next time step, there are three possible transitions. We can stay with the same character for the next time step to the pink node, transit to the next character ε in Z — the green node or skip the next character ε and transit to the purple node.

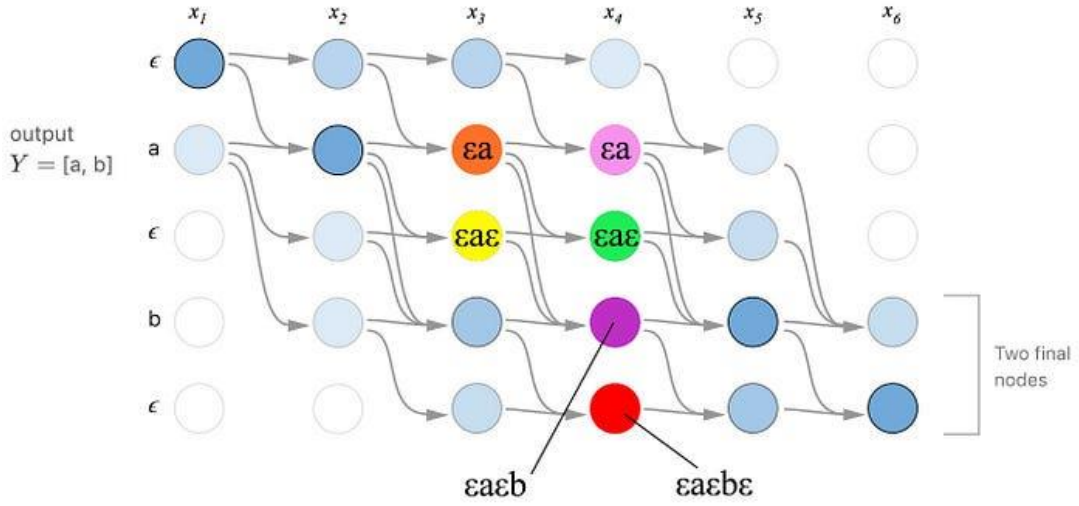


Figure 12 Decoding Nodes

But skipping the next character in *Z* is not always a valid choice for certain patterns in *Z*. For case 1, if we have a pattern of "a**ε**a" or "ε**a**ε" with "ε" or "a" be the next character respectively, we cannot skip it. We will reconstruct "a" and "" instead of "aa" and "a" after the CTC compression. Here is an example of "ε**b**ε" in demonstrating the issue. The cross-out transition below is invalid to produce "ab". Case 1

Let's consider another case, i.e., case 2. If we skip the middle character of "aɛb", we can still reconstruct "ab".

We can generalize this solution for any Z. Let's $\alpha(s, t)$ be the CTC probability where we have realized the character 1 to s in Z at time t. We can express $\alpha(s, t)$ in terms of the previous α recursively depending on the pattern in Z.

In short, we can consolidate paths that generate the same character sequence together. So, we don't need to handle them separately as we move forward in time. This improves efficiency significantly.

$$P(S|X) = \sum_{c \in A(S)} P(C|X)$$

sum over all possible paths

(e.g. ccɛaaɛtt, ccɛɛaɛtt, cɛaɛtttt, ...)

$$P(C|X) = \prod_{t=1}^{T} y(c_t, t)$$ joint probability of a path

(e.g. ccɛaaɛtt)

However, to reduce the searching space, beam-search is still required.

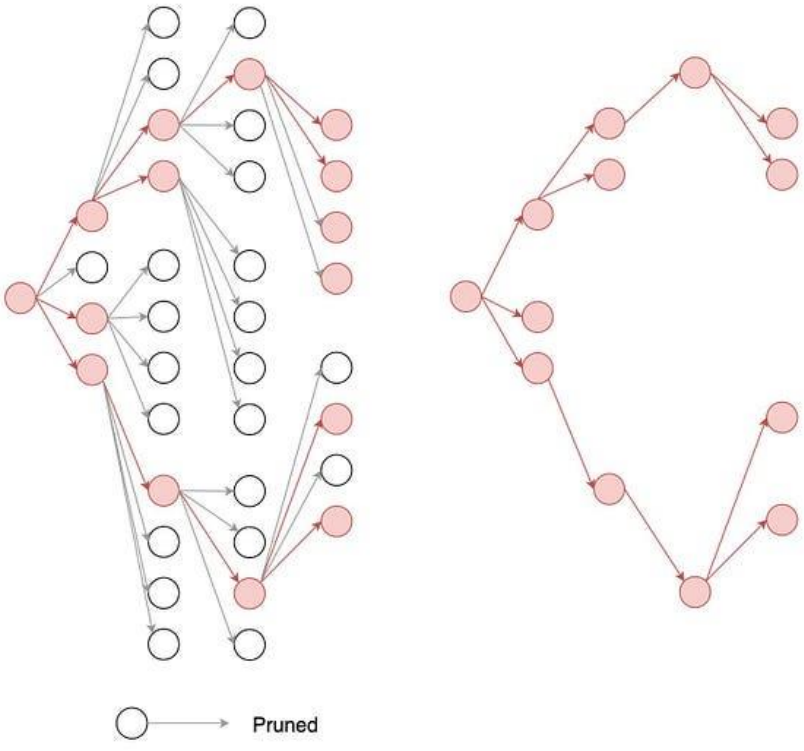


Figure 13 Pruning

Nevertheless, the CTC implementation remains complex handling edge cases. Fortunately, there are Baidu open-source implementation and the TensorFlow implementation on CTC loss and CTC beam-search that we can use.

Deep Speech

Next, let's look at building a deep network specifically from what has been learned. Below is a deep network called Deep Speech.

At each time step, the design above applies a sliding window to extract the power at different frequencies.

$x_{t,p}$ denotes the power of the p'th frequency bin in the audio frame at time t.

At each time step, Deep Speech applies three FC layers to extract features. Then, it is fed into a bi-directional RNN to explore the context of the speech. Deep Speech adds the results from the forward and backward direction together for each time step. Then, it applies another FC layer to transform the result. Finally, the probability for each character at each time step is computed with a softmax.

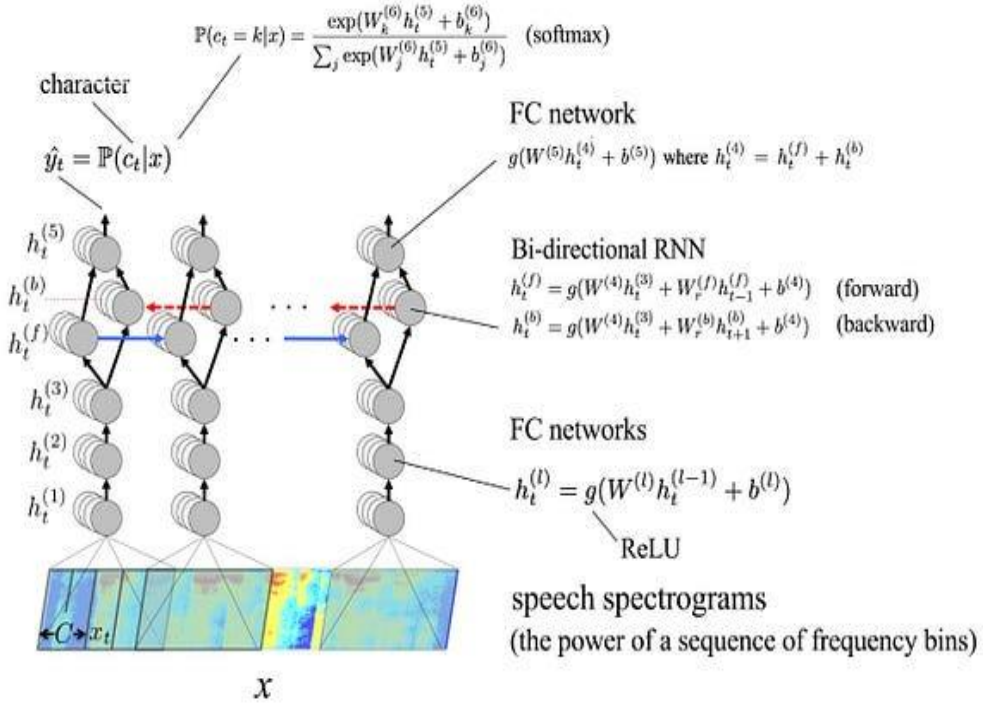


Figure 14 Full architecture

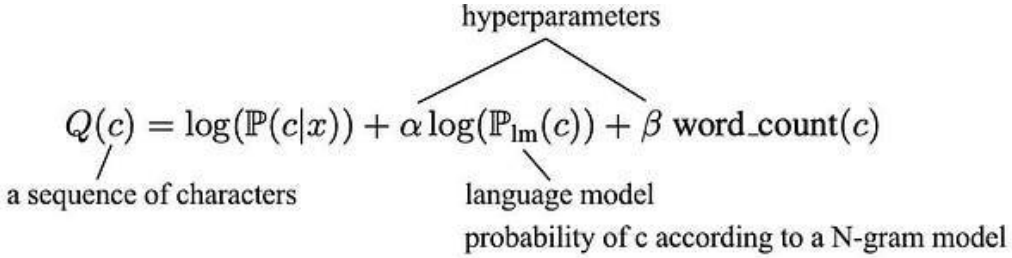
Here are the characters that are included for the probability distribution:

Output: character probabilities (a-z, <apostrophe>, <space>, <blank>)

Then, it applies CTC and beam-search to find the most likely word sequence for the utterances.

Deep speech uses the objective function below to find the optimal sequence of characters. This objective function is based on the distribution output of the deep network, an N-gram language model and the word count of the decoded

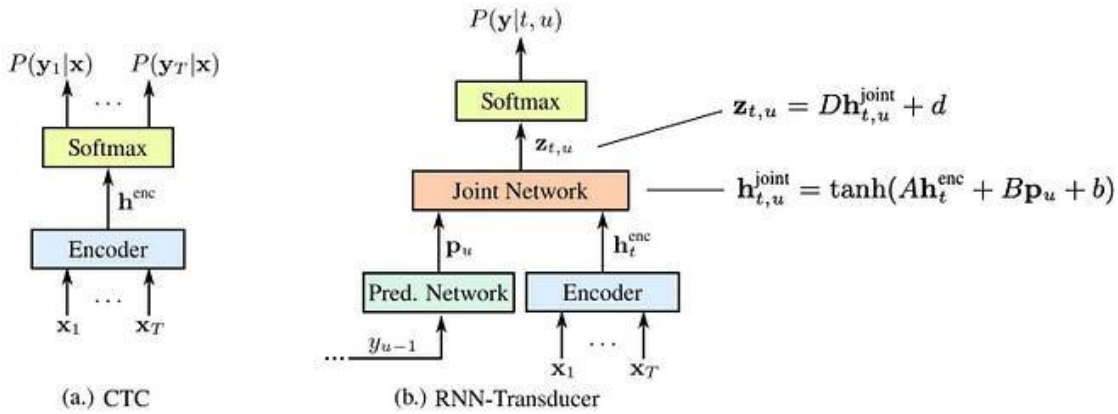
sentence. Both α and β are hyperparameters to be tuned. The language model allows us to produce grammatically sound sentences. But the deep network may be biased to create unnecessary more or fewer words than it should be. So, we added a word count objective to tune the selection process.



RNN Transducer

CTC predicts the distribution of characters for each audio frame. To find the optimal word sequence, we can incorporate a language model in the beam-search as shown before. But here we will focus on another approach in integrating the language model concept directly into the deep network.

Basically, a language model predicts what is next based on the previously decoded words. We can expand the concept further to create a deep network that makes predictions in the context of what it has already predicted.



The change requires two more components: the prediction network and the joint network. The prediction network receives the previous label prediction as input and computes an output vector pu. pu depends on the last prediction which also depends on the one before. Therefore, pu depends on the whole sequence of previous predictions. pu can be viewed as a guess on the coming prediction based on the previous predictions. On the other side, we have an encoder that creates a dense representation of the current input. We joint both information together using the equations above. Intuitively, we form a new deep network that makes predictions based on previous contexts and current observations. That is the same principle behind the language model. In fact, Deep Speech follows a similar principle. But Deep Speech looks at both the forward and the backward direction. This is the concept of RNN transducer which integrates a language model concept into the deep network.

Attention

Before discussing the last topic, we need to understand attention first.

When creating a context for our next prediction, we should not put equal weight on the information we have collected so far. This is a big waste of computing resources. We should create a context of what we are interested in. But this focus can shift in time. For example, if we mention the word ferry, we may focus on the ticket booth behind the second streetlamp instead. So how can we conceptualize this into equations and deep networks?

Before, in the RNN, we made predictions based on the new input x and the historical output context h.

For an attention-based system, we can look into the whole input x again for each step, but it will be replaced with the attention.

$$h_t = f(x, h_{t-1})$$

$$\downarrow \text{attention}$$

$$h_t = f(attention(x, h_{t-1}), h_{t-1})$$

We can visualize that the attention masks out input information that is not important for the current process.

For each input feature x_i, we train an FC layer with tanh output to score how important the feature i (or the pixel) is for the current time step under the previous output context. For instance, our last output maybe the word “ferry”.

$$s_i = \tanh(W_c h_{t-1} + W_x x_i)$$

Then, we normalize the score using a softmax function.

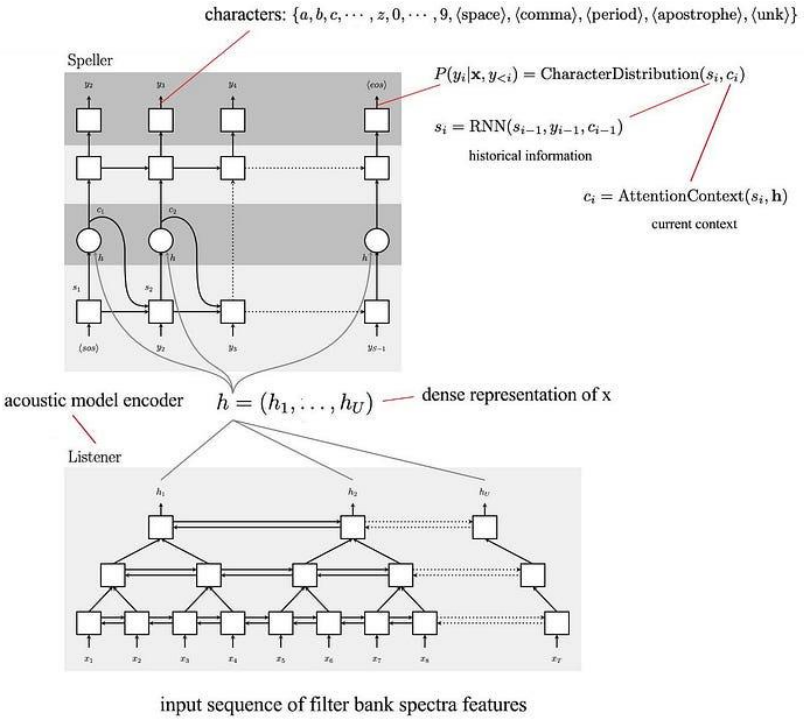
$$\alpha_i = softmax(s_1, s_2, \ldots, s_i, \ldots)$$

$$Z = \sum_i \alpha_i x_i$$

The attention Z will be the weighted output of the input features. In our example, the attention may fall around the ticket sign. The key point is we introduce an extra step to mask out information that we care less at the current step.

Listen, Attend and Spell (Encoder-Decoder)

Listen, Attend and Spell (LAS) is an encoder-decoder system. It composes of an encoder (the listener at the bottom) and a decoder (the speller on the top). The encoder takes audio frames and encodes them into a denser representation. The decoder decodes it into the distributions of characters in each decoding step.



The encoder is composed of pyramid layers of bidirectional LSTM. The output of the LSTM in each layer is fed into the upper LSTM layer with the time resolution reduced by a factor of two. This allows us to form a dense representation for hundreds or thousands of audio frames. It helps us to explore the context gradually and hierarchically.

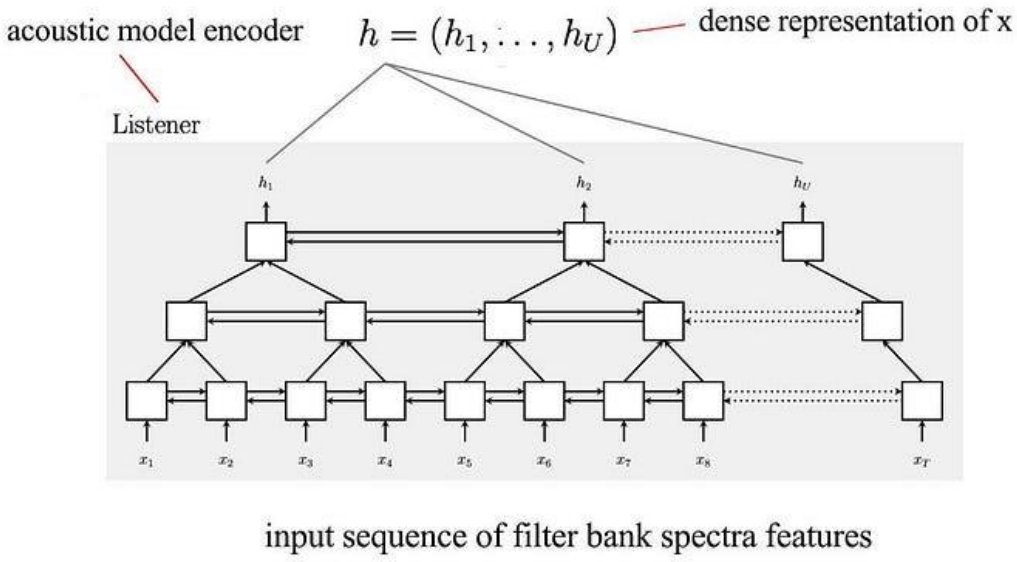


Figure 15 filterbanks

The speller is a decoder with attention. It is implemented with LSTM transducers.

In the first step, we compute the decoder state as:

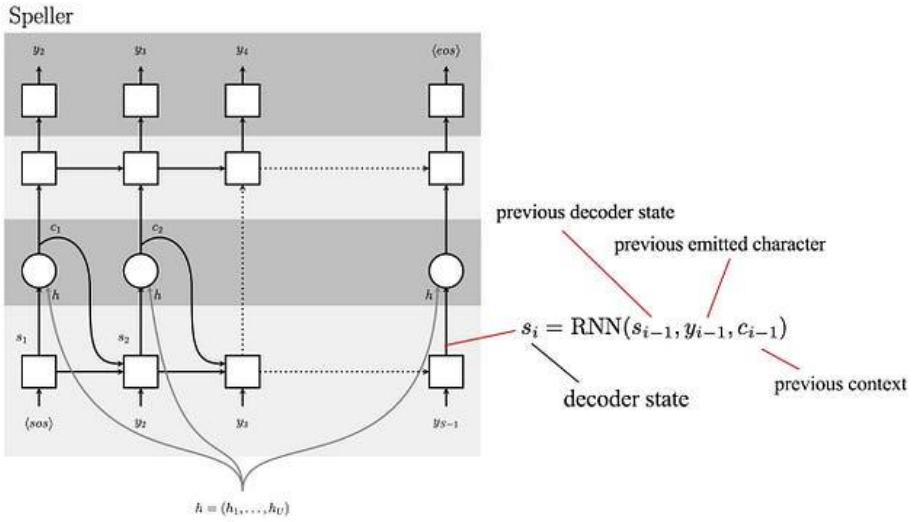


Figure 16 RNN

The decoder state will depend on the previous decoder state, previous emitted character (the character that the speller predicted) and the previous context (discussed later) computed in the last time step.

Next, we compute scores (attention) using the decoder state and the encoder state *h.* First, we use two MLP (Multilayer perceptron) to process the current decoder state *s* and *h* (the dense representation of the input *x*) further. Then we find their similarity using a dot product. This is the attention score for different components in *h*. We compute softmax and use that to create the current context *c*. This context is a weighted output of the encoder state *h* based on the attention score. In short, we find what components in *h* should we focus on.

Below is the heat map correlated the attention context and the utterance. As shown, for each character output, the attention focuses on a small number of audio frames only.

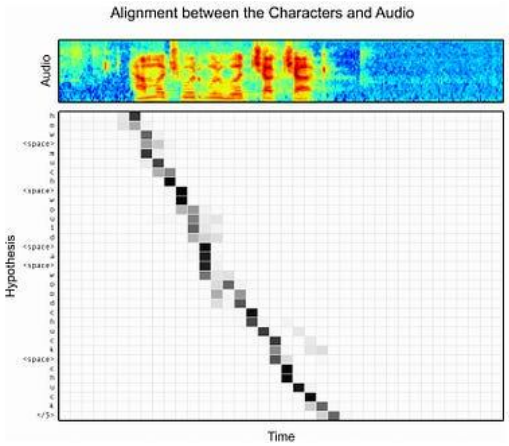


Figure 17 alignment

Finally, the character distribution is computed with the decoder state and the attention context using MLP.

In training, LAS creates a model that maximizes the probability of the ground truth. However, during inference, the ground truth is missing and the model can suffer if the previous emitted state we use to compute the decoder state turns to be a bad prediction. Therefore, during training, instead of feeding the ground truth for next step prediction all the time, LAS sometimes samples from the previous character distribution and use that as the emitted state in the next step prediction instead. So, the model is trained with the following objective instead.

$$\tilde{y}_i \sim \text{CharacterDistribution}(s_i, c_i) \text{ or ground truth}$$
$$\max_{\theta} \sum_{i} \log P(y_i|\mathbf{x}, \tilde{y}_{<i}; \theta) \qquad (90\% \text{ times})$$

In decoding, we want to find the most likely character sequence:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log P(\mathbf{y}|\mathbf{x})$$

LAS applies a beam-search from left to right. The search maintains a set of partial possibilities. The beam is expanded with the next possible characters and then later pruned to constrain the size of the search space. A word dictionary can be used to penalize non-word spelling. But it turns out that LAS with the context concept is already good at spelling and therefore not necessary. When the <eos> token is reached, we will remove the path from the beam and add one of the possible candidates for the later final selection. Once all the candidates are generated, we perform the last round of rescoring.

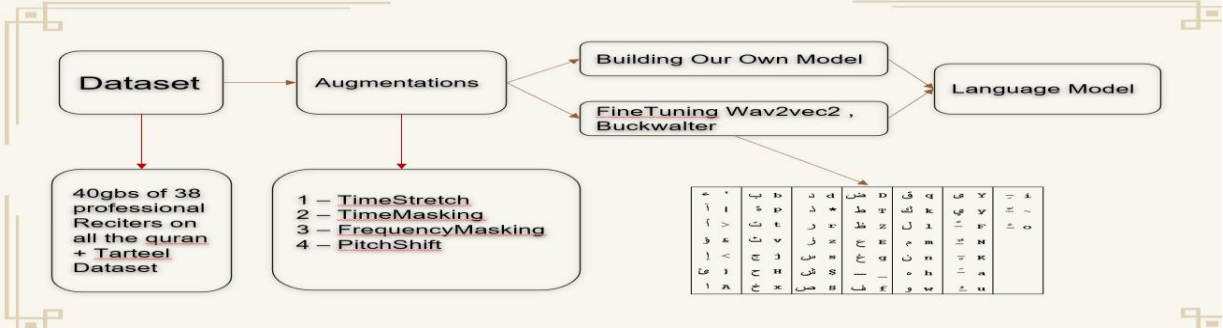
The amount of text corpus is much larger than transcript utterance. So, for the rescoring, we can put back the language model for final selection. In addition, LAS tends to create sentences with more characters than it needed, we will penalize our objective for the number of characters in the decoded text (|y|). Therefore, the objective below will be used to select the final candidate.

$$s(\mathbf{y}|\mathbf{x}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{|\mathbf{y}|_c} + \lambda \log P_{\text{LM}}(\mathbf{y})$$

3.2 Our Approach:

3.2.1 Enhanced DeepSpeech

Processing:



Dataset:

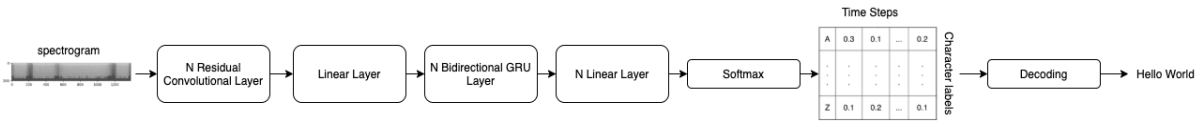
The dataset we used contained 40gbs of 38 reciters scraped from the internet, which is available on Kaggle, this dataset shows diversity in speakers so that the model will not be overfit to one speaker, but we had to augment the dataset to incorporate more regularization.

Also, we done some data augmentation to more generalize the model:

- Time Masking: the process of masking chunks of the speech so that the model would generalize better.
- Time Stretching: The process of making the audio faster or slower to incorporate people who speak slow and fast.
- Frequency Masking: the process of masking but in the frequency domain.
- Pitch Shift: changing the pitch to incorporate many users' frequencies.

```
train_audio_transforms = nn.Sequential(
    torchaudio.transforms.MelSpectrogram(sample_rate=16000, n_mels=128),
    torchaudio.transforms.FrequencyMasking(freq_mask_param=15),
    torchaudio.transforms.TimeMasking(time_mask_param=35)
)
```

The model is a variation of DeepSpeech.



Here is the network for the Model:

```
DeepSpeech(
  (cnn): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (rescnn_layers): Sequential(
    (0): ResidualCNN(
      (cnn1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cnn2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
      (layer_norm1): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
      (layer_norm2): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
    )
    (1): ResidualCNN(
      (cnn1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cnn2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
      (layer_norm1): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
      (layer_norm2): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (birnn_layers): Sequential(
    (0): BidirectionalGRU(
      (BiGRU): GRU(512, 512, batch_first=True, bidirectional=True)
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): BidirectionalGRU(
      (BiGRU): GRU(1024, 512, bidirectional=True)
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (2): BidirectionalGRU(
      (BiGRU): GRU(1024, 512, bidirectional=True)
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (classifier): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): GELU()
    (2): Dropout(p=0.1, inplace=False)
    (3): Linear(in_features=512, out_features=29, bias=True)
  )
)
Num Model Parameters 14233053
```

Let's discuss the differences in the variant of DeepSpeech compared to the original DeepSpeech model:

The variant of DeepSpeech includes a few architectural changes:

1. It introduces a convolutional layer (**cnn**) at the beginning of the model. This layer applies 32 filters with a kernel size of 3x3 to capture local patterns in the audio waveform. The stride of 2x2 and padding of 1x1 are used to downsample the input, helping the model learn more robust representations of the input at different scales.
2. The variant includes two **ResidualCNN** blocks within the **rescnn_layers**. Each block consists of two convolutional layers with 32 filters. Dropout and layer normalization are applied after each convolutional layer. The residual connections within these blocks allow the model to learn more effectively by propagating useful information and gradients through the network.
3. After the **ResCNN** layers, there is a fully connected layer (**fully_connected**) that reduces the dimensionality of the features from 2048 to 512. This layer helps capture more abstract and higher-level information from the input.
4. The **birnn_layers** module consists of three **BidirectionalGRU** blocks. Each block contains a bidirectional GRU layer with 512 hidden units. Layer normalization and dropout are applied to the output of each block. This architecture modification enables the model to capture both forward and backward temporal dependencies in the audio sequence, improving its understanding of the input data.
5. The **classifier** module further processes the output of the BiRNN layers. It includes a linear layer that maps the input from 1024 to 512 dimensions, followed by the GELU activation function, dropout, and another linear layer that maps to the final output size of 29. These modifications provide non-linearity, regularization, and dimensionality reduction, respectively, to the model's final output layer.

In summary, the variant of DeepSpeech incorporates additional convolutional layers, residual connections, a fully connected layer, and multiple bidirectional GRU layers. These modifications aim to capture local patterns, propagate gradients effectively, learn high-level representations, and capture long-range temporal dependencies. The variant is designed to improve the overall performance and accuracy of speech recognition compared to the original DeepSpeech model.

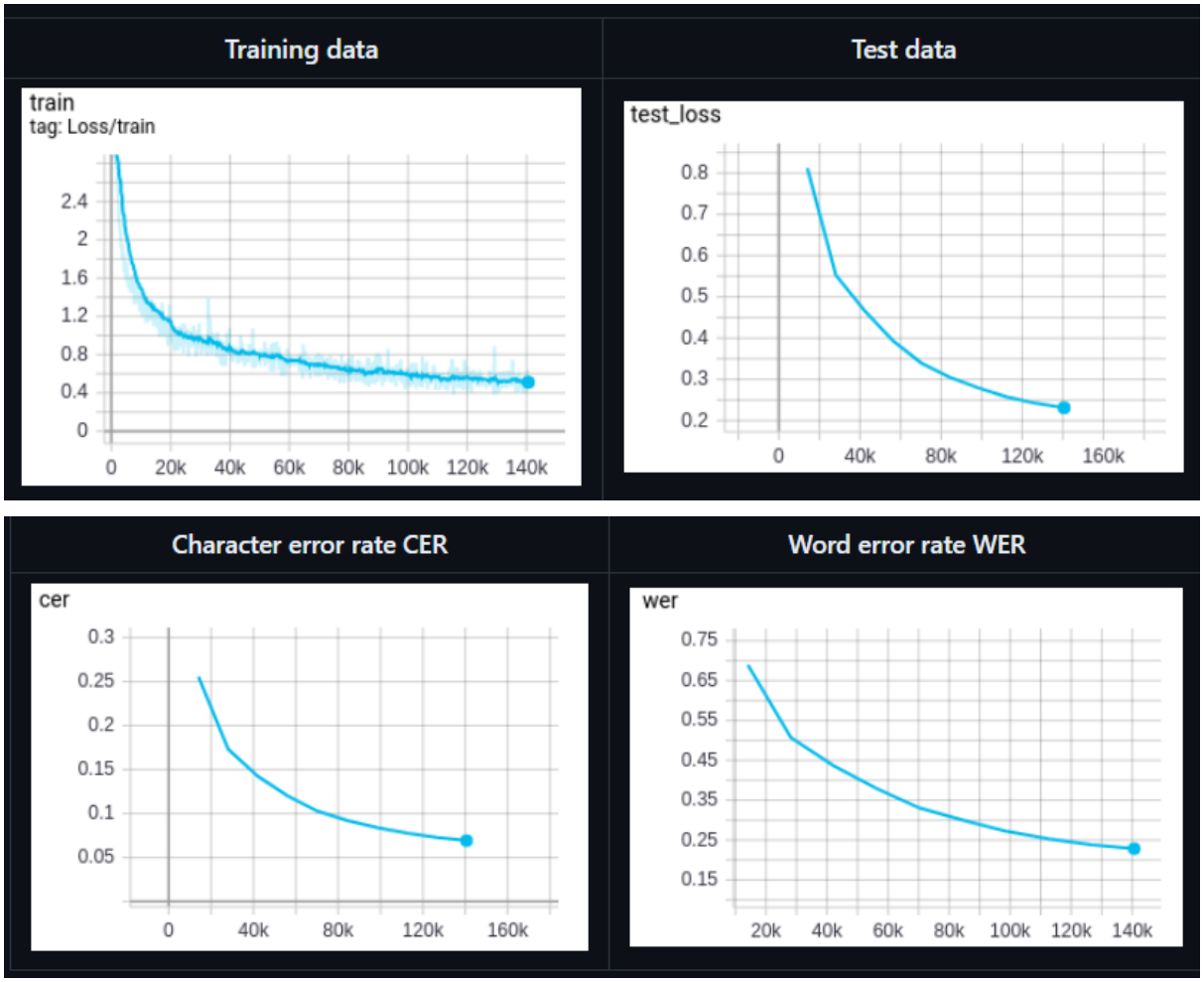


Figure 18 Metrics

This model got **33% WER** in our dataset, which we split into 10% - 90% test – train.

3.2.2 Our Second Model (Finetuned Wav2Vec)

Our second model is a finetuned wav2vec2 model, which achieved better accuracy but needed much more computing as it was orders of magnitude bigger than DeepSpeech model.

Transformer-based neural networks have been revolutionizing the natural language processing field but are only starting to become popular in the speech processing community. **Wav2vec 2.0** is set to change that. Its architecture is based on the Transformer's encoder, with a training objective that's similar to **BERT's masked language modeling** objective but adapted for speech.

This new method allows for efficient semi-supervised training: first, pre-train the model on a large quantity of unlabeled speech, then fine-tune on a smaller labeled dataset. In wav2vec 2.0's original paper, the authors demonstrated that fine-tuning the model on only one hour of labeled speech data could produce better results than the previous state-of-the-art systems trained on 100 times more labeled data.

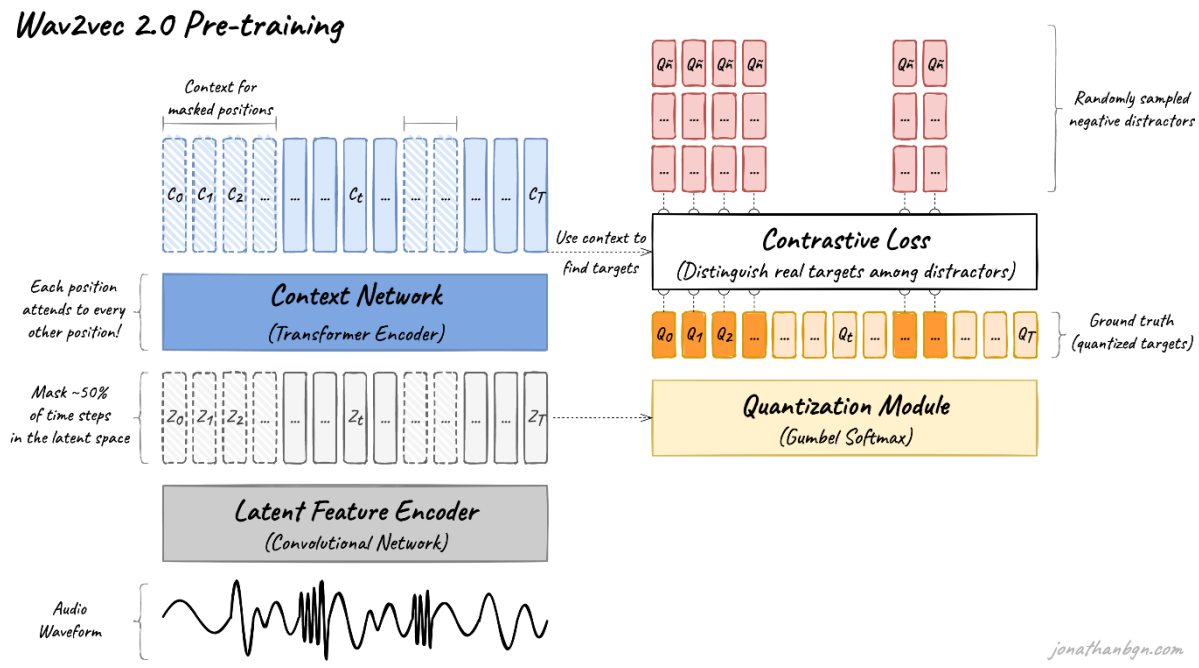


Figure 19 Wav2vec2 process

Above is an overview of the wav2vec 2.0 architecture and its pre-training process. There are four important elements in this diagram: the **feature encoder**, **context network**, **quantization module**, and the **contrastive loss** (pre-training objective). We will open the hood and look in detail at each one.

Feature encoder

The feature encoder's job is to reduce the dimensionality of the audio data, converting the raw waveform into a sequence of feature vectors Z_0, Z_1, Z_2, …, Z_T each 20 milliseconds. Its architecture is simple: a 7-layer convolutional neural network (single-dimensional) with 512 channels at each layer.

The waveform is normalized before being sent to the network, and the kernel width and strides of the convolutional layers decrease as we get higher in the network. The feature encoder has a total receptive field of 400 samples or 25 ms of audio (audio data is encoded at a sample rate of 16 kHz).

Quantization module

One of the main obstacles of using Transformers for speech processing is the **continuous nature of speech**. Written language can be naturally discretized into words or sub-words, therefore creating a finite vocabulary of discrete units. Speech doesn't have such natural sub-units. We could use phones as a discrete system, but then we would need humans to first label the entire dataset beforehand, so we wouldn't be able to pre-train on unlabeled data.

Wav2vec 2.0 proposes to **automatically learn discrete speech units**, by sampling from the Gumbel-Softmax distribution. Possible units are made of *codewords* sampled from *codebooks* (groups). *Codewords* are then concatenated to form the final speech unit. Wav2vec uses 2 groups with 320 possible words in each group, hence a theoretical maximum of 320 x 320 = 102,400 speech units.

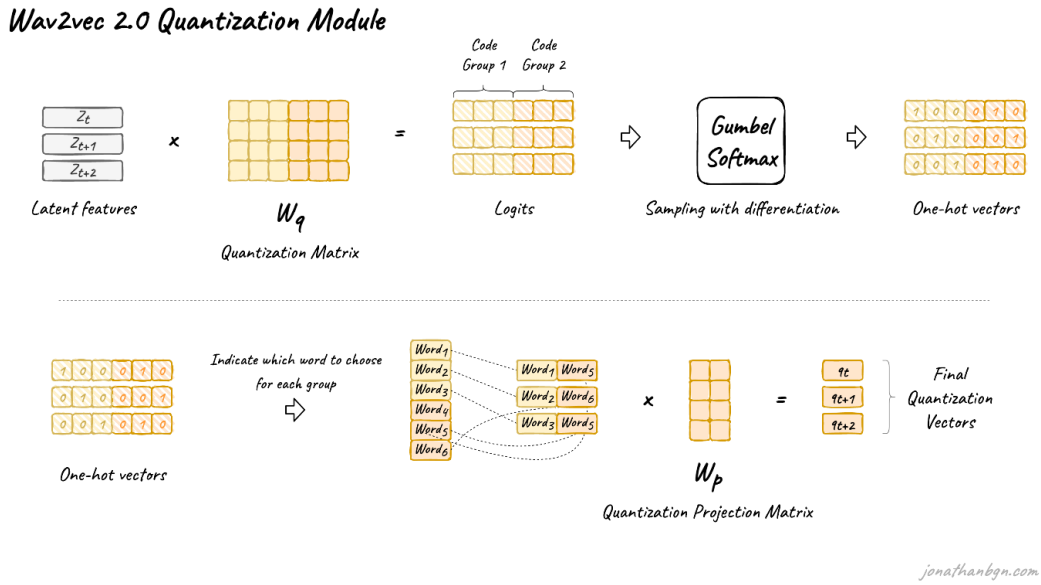


Figure 20 Quantization

The latent features are multiplied by the *quantization matrix* to give the logits: one score for each of the possible *codewords* in each *codebook*. The Gumbel-Softmax trick allows sampling a single codeword from each codebook, after converting these logits into probabilities. It is similar to taking the argmax except that the operation is fully differentiable. Moreover, a small randomness effect, whose effect is controlled by a temperature argument, is introduced to the sampling process to facilitate training and codewords utilization.

Context network

The core of wav2vec 2.0 is its Transformer encoder, which takes as input the latent feature vectors and processes it through 12 Transformer blocks for the *BASE* version of the model, or 24 blocks for the *LARGE* version. To match the inner dimension of the Transformer encoder, the input sequence first needs to go through a feature projection layer to increase the dimension from 512 (output of the CNN) to 768 for *BASE* or 1,024 for *LARGE*. I will not describe further the Transformer architecture.

One difference from the original Transformer architecture is how positional information is added to the input. Since the self-attention operation of the Transformer doesn't preserve the order of the input sequence, fixed pre-generated positional embeddings were added to the input vectors in the original implementation. The wav2vec model instead uses a new grouped convolution layer to learn relative positional embeddings by itself.

Pre-training & contrastive loss

The pre-training process uses a contrastive task to train on unlabeled speech data. A mask is first randomly applied in the latent space, where ~50% of the projected latent feature vectors. Masked positions are then replaced by the same trained vector Z'_M before being fed to the Transformer network.

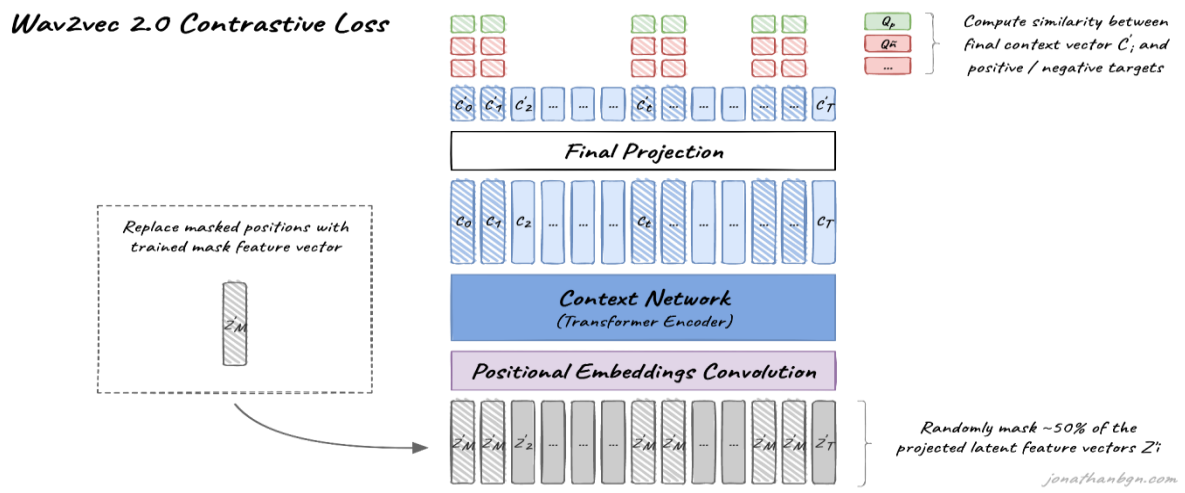


Figure 21 Positional Embeddings

The final context vectors then go through the last projection layer to match the dimension of the quantized speech units Q_t. For each masked position, **100 negative distractors are uniformly sampled from other positions in the same sentence**. The model then compares the similarity (cosine similarity) between the projected context vector C'_t and the true positive target Q_p along with all negative distractors $Q_{\tilde{n}}$. The contrastive loss then encourages high similarity with the true positive target and penalizes high similarity scores with negative distractors.

Diversity loss

During pre-training, another loss is added to the contrastive loss to encourage the model to use all codewords equally often. This works by maximizing the entropy of the Gumbel-Softmax distribution, preventing the model to always choose from a small sub-group of all available codebook entries. You can find more details in the original paper.

Conclusion

This concludes our tour of wav2vec 2.0 and its pre-training process. The resulting pre-trained model can be used for a variety of speech downstream tasks: automatic speech recognition, emotion detection, speaker recognition, language detection… In the original paper, the authors directly fine-tuned the model for speech recognition with a CTC loss, adding a linear projection on top of the context network to predict a word token at each timestep.

Wav2vec2 Result:

Wav2vec2 was trained for 72 Hours on our dataset which is 40gb, we took only the less than 5 seconds audio so we can fit in memory, and finetuned a pretrained Arabic wav2vec2 which achieved higher results than the DeepSpeech model, it achieved **27% WER** on the same data test set.

3.3 Recitation with Tashkeel

Our approach for incorporating Tashkeel into the models involved making certain modifications. First, we added Tashkeel characters to the vocabulary of both models to ensure that they could recognize and generate the appropriate diacritical marks. This step was crucial in accurately transcribing and recognizing Arabic words with Tashkeel.

- DeepSpeech, we trained the model from scratch, including the feature extractor. This allowed us to fine-tune the model specifically for the task of Arabic recitation with Tashkeel. The training process on a T4 16GB GPU took a significant amount of time, spanning over two weeks. However, the effort was worth it, as the model demonstrated excellent performance in recognizing Tashkeel and producing accurate transcriptions.
- Wav2Vec2, we decided to freeze the feature extractor and only fine-tune the language model. This approach saved considerable training time, as we trained Wav2Vec2 for just three days on a subset of the dataset. Although the training duration was shorter compared to DeepSpeech, the model still showed promising results.

To detect Tashkeel, we implemented a decoding technique called beam search. Beam search allowed us to extract the characters directly from the model without relying on a language model. By removing the language model, we focused solely on obtaining the characters and preserving the integrity of Tashkeel. This approach proved effective in both models, further enhancing their ability to accurately recognize and generate Tashkeel during the recitation process.

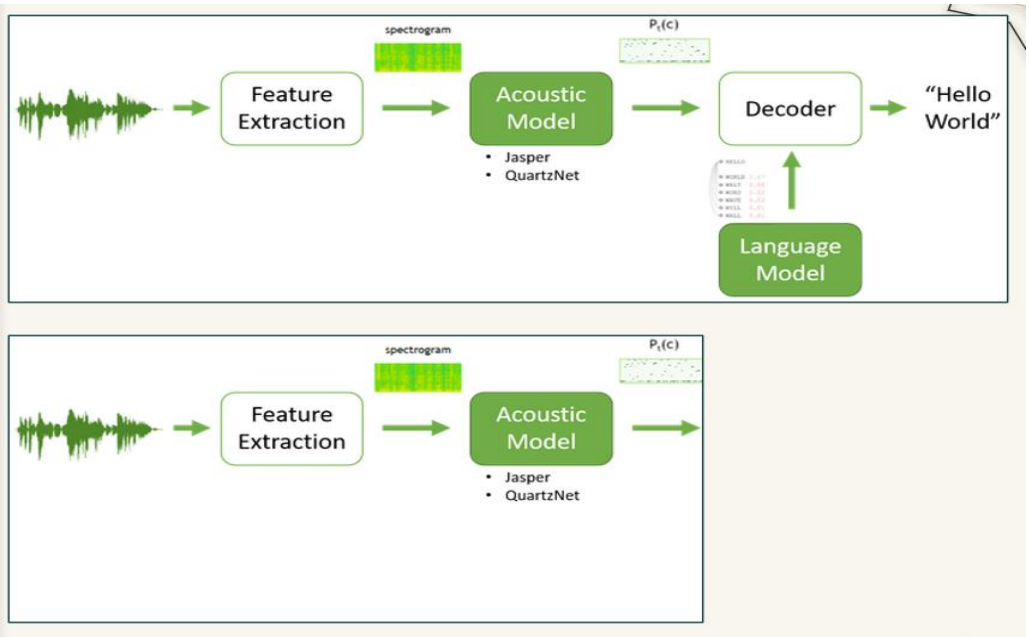
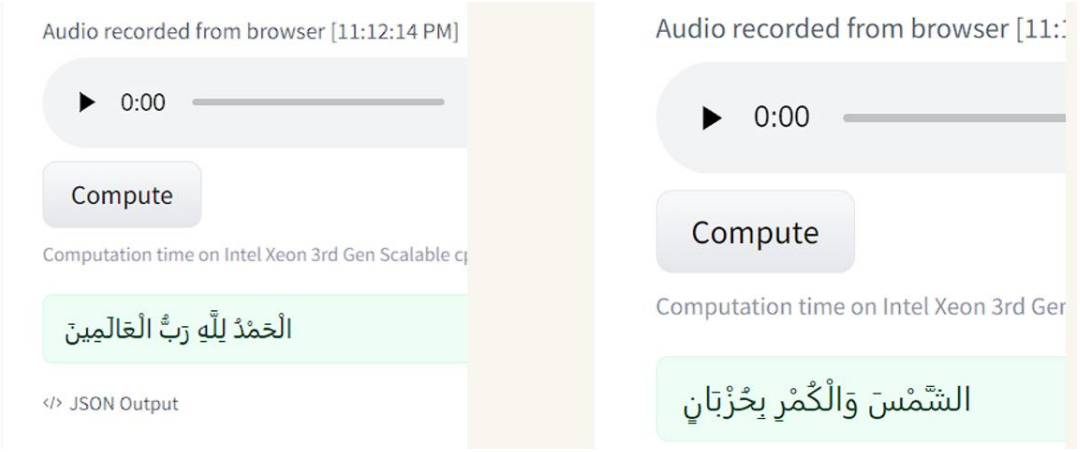
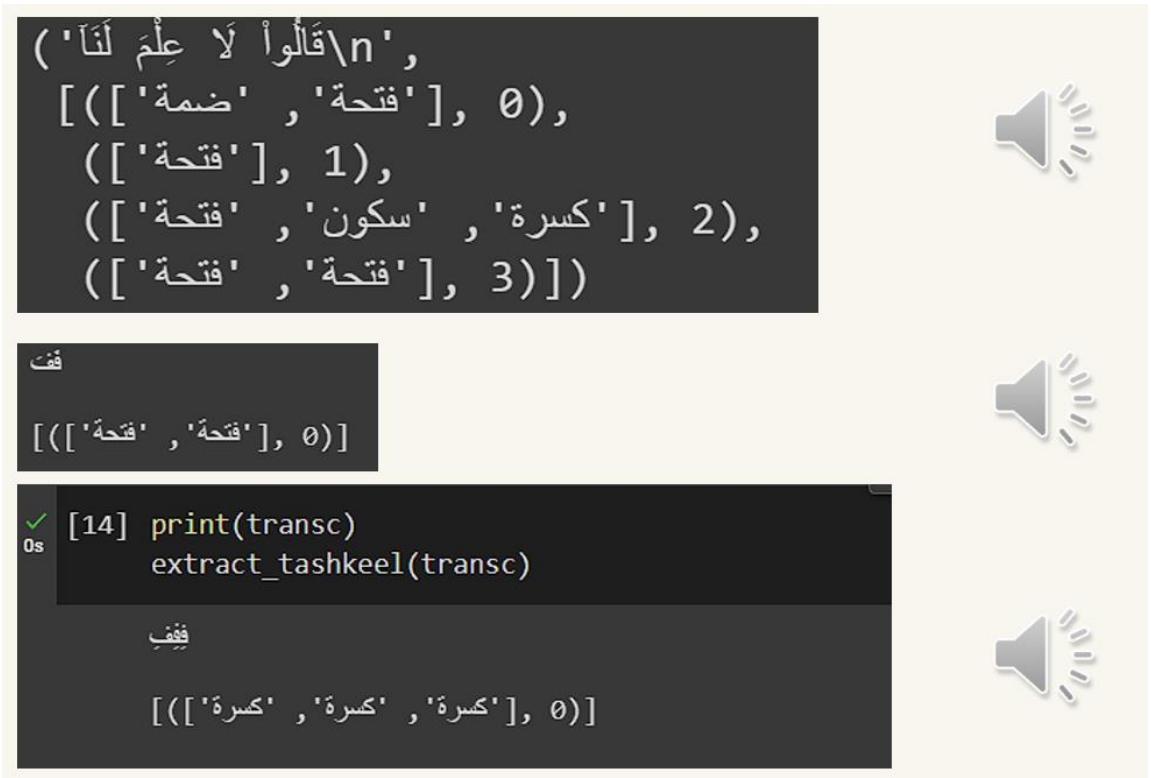


Figure 22 Removing the LM

The incorporation of Tashkeel in speech recognition models presents a unique challenge, but our approach showcased the potential for effectively recognizing and generating Tashkeel in Arabic recitation. Further research and experimentation in this domain may lead to even more refined models that can excel in recognizing and handling Tashkeel across a wider range of Arabic texts and recitation styles.





In conclusion, our experiment demonstrated that DeepSpeech, with its feature extractor trained from scratch and extended training duration, outperformed Wav2Vec2 in terms of accuracy and Tashkeel recognition. However, it is important to note that the modified Wav2Vec2 model, with a frozen feature extractor and trained language model, still showcased promising results despite the shorter training period.3.4 Sample from tests on the model.

Chapter 4: Tajweed Correction

4.1 Our Approach

Tajweed is an integral part of the recitation of the Quran, It refers to the set of rules and principles that govern the proper pronunciation of the Arabic letters and words found in the Quran. The word "Tajweed" itself means to improve, beautify, or enhance something, and in the context of Quranic recitation, it aims to enhance the recitation by ensuring the correct pronunciation, rhythm, and melody.

So, we choose 5 rules of Tajweed which are:

- ادغام النون الساكنة والتنوين بالغنة
- ادغام النون الساكنة والتنوين بدون غنة
- إخفاء النون الساكنة والتنوين
- اقلاب النون الساكنة والتنوين
- النون والميم المشددة

We choose them as they are acoustically visible and classifiable in our opinion.

We made a model for each rule than could classify if it’s spoken right or wrong, this model gives probability of being right or wrong that will be shown to the user.

4.2 Dataset Collection

To train the model for each Tajweed rule, we required a specific dataset containing records with correct and incorrect pronunciations. Each record's duration was limited to five seconds.

To gather the necessary dataset, we obtained recordings from Quran readers and educational videos since we couldn't find an existing dataset specifically tailored to Tajweed rules.

By collecting these recordings, we aimed to create a comprehensive dataset that would enable the model to learn and differentiate between correct and incorrect pronunciations according to Tajweed principles. This dataset acquisition process allowed us to build a robust and accurate model for Tajweed rule recognition and evaluation.

4.3 Speech Classification Model:

The Quran, the holy book of Islam, is revered for its profound spiritual and linguistic beauty. It is recited by millions of Muslims around the world, and proper recitation is of utmost importance. Tajweed, the set of rules governing the pronunciation and intonation of Quranic recitation, ensures the accurate and

melodious delivery of its verses. With advancements in natural language processing (NLP) and machine learning, the wav2vec2 pretrained model emerges as a valuable tool for automating the classification of Tajweed rules. This essay explores the application of the wav2vec2 model in this domain, discussing its significance, benefits, and potential challenges.

1. Understanding the wav2vec2 Pretrained Model: The wav2vec2 model, developed by Facebook AI Research (FAIR), is a state-of-the-art unsupervised representation learning technique for speech data. It leverages self-supervised learning and transfer learning to extract rich representations from raw audio, which can be further utilized for various downstream tasks. By training on large-scale audio datasets, the model learns to map speech signals to meaningful representations, making it suitable for tasks like speech recognition, speaker identification, and now, Tajweed rule classification.
2. Significance of Tajweed Rule Classification: Tajweed plays a vital role in preserving the integrity of the Quran's recitation. It ensures the correct pronunciation of Arabic letters, rules for elongation, proper voice modulation, and rhythmic flow. Human experts traditionally performed the role of Tajweed instructors, relying on their expertise and experience. However, the wav2vec2 model offers a unique opportunity to automate Tajweed rule classification, potentially providing feedback and assistance to learners and individuals reciting the Quran.
3. Benefits of Using the wav2vec2 Model for Tajweed Rule Classification: a. Increased Accessibility: By leveraging the wav2vec2 model, Tajweed instruction and feedback can be made more accessible to a wider audience. Individuals can utilize speech recognition systems powered by the model to receive immediate feedback on their recitation, aiding in self-improvement and adherence to Tajweed rules.

b. Consistency and Accuracy: The model's objective nature ensures consistent and objective evaluation of Tajweed rules. It reduces the reliance on subjective human judgment, minimizing potential discrepancies and errors. This can be particularly helpful for learners who may not have access to knowledgeable Tajweed instructors.

c. Scalability: As the wav2vec2 model is pretrained on vast amounts of data, it possesses the ability to generalize well across various speakers and contexts. This scalability allows for widespread adoption of Tajweed rule classification, benefiting learners in different regions and linguistic backgrounds.

4. Potential Challenges and Considerations: a. Linguistic Complexity: The Quranic Arabic used in the recitation of the Quran can be linguistically complex, with variations in pronunciation and intonation. Adapting the wav2vec2 model to capture these nuances and accurately classify

Tajweed rules can pose a challenge, requiring further research and fine-tuning.

b. Dataset Limitations: Training a Tajweed rule classification model would necessitate a sufficiently large and diverse dataset containing labeled examples of various rules. The creation of such a dataset may require the expertise of Quranic scholars and recitation experts to ensure accuracy and coverage.

c. Interpretability and Explain ability: Deep learning models like wav2vec2 are often regarded as black boxes due to their complexity. Ensuring the transparency and interpretability of the model's decisions regarding Tajweed rule classification is essential to gain user trust and acceptance.

Converting Wav2vec2 from an ASR model to a classification model is very simple:

1. Prepare the Classification Dataset: Gather or create a labeled dataset specific to the classification task you wish to perform. Ensure that the dataset contains speech samples along with their corresponding labels or categories. The dataset should be diverse and representative of the classes you want to classify.
2. Fine-tune the Wav2Vec2 Model: Take the pretrained Wav2Vec2 model, which is trained for ASR, and fine-tune it using the classification dataset. During fine-tuning, the model's weights are updated to adapt its speech representation capabilities for the classification task. This involves training the model on the classification dataset, adjusting the learning rate, and determining the optimal hyperparameters.
3. Modify the Output Layer: Since the original Wav2Vec2 model was designed for ASR, the final layers of the model are typically tailored for transcription. To convert it to a classification model, you need to modify the output layer to match the number of classes or labels in your classification dataset. Adjusting the architecture of the output layer allows the model to make predictions based on the classification task.
4. Training and Evaluation: Once the modifications are made, train the adapted Wav2Vec2 model using the classification dataset. During training, the model learns to classify speech samples based on the provided labels. Monitor the model's performance using appropriate evaluation metrics, such as accuracy, precision, recall, or F1-score, to assess its effectiveness in classification.
5. Fine-tuning and Hyperparameter Tuning: Depending on the specific classification task and dataset, additional fine-tuning and hyperparameter tuning may be necessary. This process involves adjusting the model's

architecture, learning rate, batch size, or other parameters to optimize its performance on the classification task. Experimentation and iterative refinement are often required to achieve the best results.

6. Post-processing and Inference: Once the model is trained and performs well on the validation or test data, it can be used for inference on unseen speech samples. Feed the speech input into the adapted Wav2Vec2 model and use the modified output layer to obtain classification predictions. Post-processing techniques, such as thresholding, can be applied to refine the model's predictions if necessary.

It's important to note that converting Wav2Vec2 to classification requires a well-annotated dataset, careful fine-tuning, and consideration of task-specific modifications. The success of the conversion relies on the quality of the dataset, appropriate model adjustments, and rigorous training and evaluation procedures.

As we took our pretrained model which already could transcript Arabic speech to text with it's Tashkeel, it was fairly easy to achieve good results on the Tajweed dataset that we created.

[35/35 00:32, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.601397	1.000000
2	No log	0.480999	1.000000
3	No log	0.384799	1.000000
4	No log	0.331117	1.000000
5	No log	0.306653	1.000000

As we can see from the results the validation loss was fairly descent from the start of the training.

Here are the training arguments:

```
training_args = TrainingArguments(
    output_dir = "./rule3" ,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=3e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    metric_for_best_model="accuracy",
    warmup_ratio=0.1,
)
```

Chapter 5: Semantic Similarity Models Tested on Quran

5.1 Introduction

Semantic similarity determination between texts, particularly in the context of the Arabic language, poses a significant challenge for Natural Language Processing (NLP). This challenge becomes even more complex when dealing with the Holy Qur'an, a sacred Arabic text consisting of 6,236 verses across 114 chapters. The Qur'an's intricate interweaving of concepts and details throughout its chapters makes it difficult to identify related passages, requiring a holistic approach to understand its topics. This presents a hurdle for tasks such as document collection, subject identification, question answering, machine translation, and text summarization, which heavily rely on semantic similarity.

To address this challenge, we developed a Word2Vec model, trained on the Qur'an and other datasets. The Word2Vec model predicts target words based on their context, enabling the identification of words with similar meanings by analyzing their co-occurrence patterns. By utilizing the model's output embeddings, we can compare Qur'anic verses for semantic similarity using cosine similarity as a measure of closeness. This approach allows users to input a word or phrase related to a specific topic and retrieve verses from the Qur'an that are semantically connected. This model is particularly valuable for uncovering themes and concepts within the Qur'an, which can be challenging due to the dispersed nature of certain topics in the text.

For example, if the input prompt is "family," our model can retrieve a related verse that may not explicitly mention the word "family." However, the verse might recount the story of Prophet Ibrahim and his son Ismael, showcasing their dedication to sacrifice for the sake of Allah. Although the word "family" may not appear, the verse's underlying message highlights the significance of family and obedience to Allah's command, making it relevant to the concept of family in Islam. Thus, our objective was to achieve relatively accurate results even for short inputs that are not exact matches.

In addition to the Word2Vec model, we explored the finetuning of various models, such as AraGPT2 and AraBert, on our collected dataset. However, these models presented certain disadvantages that we considered. They required substantial computational resources due to their large sizes, and the resulting embeddings were often quite large, posing challenges for storage and memory limitations. Moreover, these models could be influenced by training data bias, leading to embeddings that may not encompass all language use cases or may exhibit biases towards certain language types or contexts. Fine-tuning these models also demanded significant time and labeled training data. Furthermore, the complex architectures of models like BERT and GPT made it difficult to

interpret their predictions or embeddings, hindering performance improvement in specific cases.

To address these challenges, we finetuned AraVec, a word embedding model trained using the spacy framework on a Twitter dataset. This model played a pivotal role in understanding general Arabic language use. We further trained AraVec on our dataset, which included the Qur'an, Q&A datasets from Tarteel, Hadith books, and Tafsir books. By incorporating this diverse range of texts related to the Qur'an, the model gained a broader understanding of contextual word usage and their relationships.

Additionally, we developed a Doc2Vec model specifically for determining semantic similarity between verses in the Holy Qur'an. Doc2Vec represents documents as fixed-length vectors, capturing their semantic meaning. By training the model on Quranic verses and related texts, we obtained embeddings for each document. To determine similarity, we employed cosine similarity to compare these embeddings, enabling the identification of shared themes and concepts across the Qur'an. Doc2Vec offers advantages such as capturing contextual information, accommodating longer texts, reducing computational requirements, and providing interpretability.

5.2 Data collection

Our data is what gave our model the edge in this specific task. AraVec baseline was trained on Arabic tweets of approximately 66.9m Docs, 1.47m vocab size. Also, we collected 160k rows of text that contained the following:

- The Quran itself was sampled 3 times.
- Hadith books scraped from the internet. (Shown in table 1).
- All the Taseer books scraped from the Internet are structured as verse and explanation.

  The following books are the ones we used:

  - Baghawy,
  - Qurtubi,
  - Saddi,
  - Tanweer,
  - Waseet
- Q&A Dataset that Tarteel AI published on hugging face.

Book Name	Number of Ahadtih
Sahih Bukhari	**7008**
Sahih Muslim	**5362**
Sunan al Tirmidhi	**3891**
Sunan al-Nasai	**5662**
Sunan Abu Dawud	**4590**
Sunan Ibn Maja	**4332**
Musnad Ahmad ibn Hanbal	**26363**
Maliks Muwatta	**1594**
Sunan al Daraimi	**3367**

5.3 Data Preprocessing

The preprocessing steps included:

- removing non-Arabic characters and punctuation
- removing stop words and applying stemming and lemmatization to reduce words to their base form.
- removing Tashkeel from dataset so it's easier for the model to perform.

We kept the dataset before and after preprocessing (without Taksheel) In the training.

Through this approach we achieved better results. And obtain better accurate and meaningful results all models, which helped in advance of the study of semantic similarity in Quranic text.

5.4 Models

5.4.1 Word2Vec

Word2Vec is a neural network-based algorithm that is used to create vector representations (or embeddings) of words in a corpus of text. These embeddings can then be used to perform a wide range of natural language processing (NLP) tasks, such as text classification, sentiment analysis, and named entity recognition. It is trained using a neural network that has an input layer, a hidden layer, and an output layer. The input layer represents the words in the corpus, and the output layer represents the context in which those words appear. The hidden layer is where the vector representations (or embeddings) of the words are generated.

The training process involves feeding the neural network pairs of words and their contexts. A "context" can be defined as the words that appear in a fixed window around the target word. For example, if the window size is set to 5, then the context for the word "apple" might be "the red juicy fruit is an". The neural network is then trained to predict the context of a given word based on its vector representation.

The vector representations are initialized randomly, and the neural network is trained using a technique called **stochastic gradient descent**. This involves minimizing a cost function that measures the difference between the predicted context and the actual context. The cost function used in Word2Vec is typically the negative log likelihood of the context words given the target word.

As the neural network is trained, the vector representations of the words in the corpus are updated to better reflect their semantic relationships. Words that appear in similar contexts will have similar vector representations, while words

that appear in dissimilar contexts will have dissimilar vector representations. This means that words with similar meanings will be clustered together in the vector space.

There are two main architectures used in Word2Vec: continuous bag-of-words (CBOW) and skip-gram. In the CBOW architecture, the neural network is trained to predict the target word based on its context. In the skip-gram architecture, the neural network is trained to predict the context words based on the target word. Both architectures can be used to generate vector representations of words but skip-gram tends to perform better on larger datasets.

Once the training process is complete, the vector representations can be used for a wide range of NLP tasks. For example, they can be used as input to a machine learning model for sentiment analysis, or to calculate the similarity between two words in a text corpus.

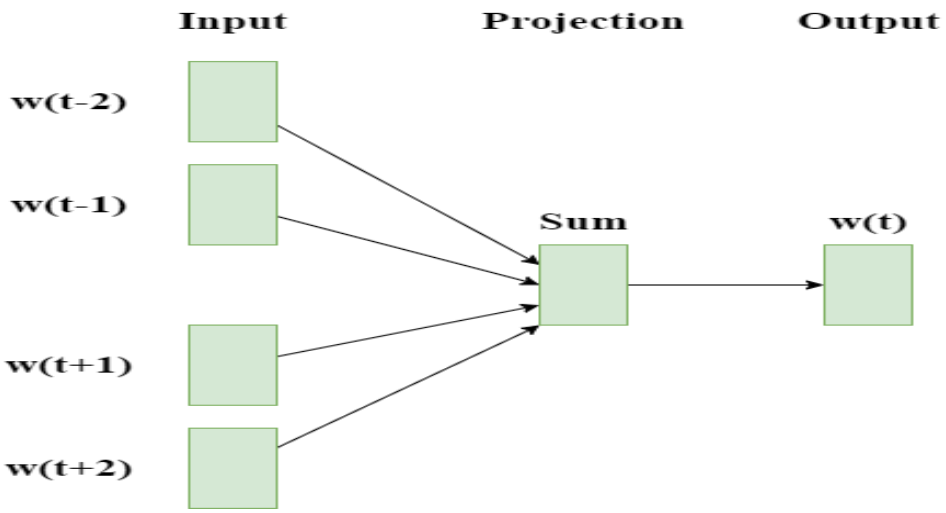


Figure 23 Word2Vec Unit

AraVec is a pre-trained distributed word representation (word embedding) open-source project which aims to provide the Arabic NLP research community with free to use and powerful word embedding models. The first version of AraVec provides six different word embedding models built on top of three different Arabic content domains; Tweets and Wikipedia This paper describes the resources used for building the models, the employed data cleaning techniques, the carried out pre-processing step, as well as the details of the employed word embedding creation techniques.

The third version of AraVec provides 16 different word embedding models built on top of two different Arabic content domains: Tweets and Wikipedia Arabic articles. The major difference between this version and the previous ones, is that we produced two different types of models, unigrams and n-grams models. We

utilized a set of statistical techniques to generate the most used n-grams of each data domain.

Twitter tweets, Wikipedia Arabic articles, AraVec comes in N-gram and unigrams, 100 and 300 vector sizes.

Libraries used for training the model:

Gensim:

- Vector Space Modeling: Gensim provides a Corpus concept for representing collections of text documents. It supports vector space models like Bag-of-Words (BoW) and TF-IDF representations.
- Topic Modeling: Gensim offers efficient implementations of topic modeling algorithms, including LSA, LDA, and HDP.
- Word Embeddings: Gensim supports training and using word embeddings, including the Word2Vec model.
- Document Similarity: Gensim allows computing document similarity using measures like cosine similarity.
- Text Preprocessing: It includes utilities for text preprocessing tasks like tokenization, stop word removal, stemming, and lemmatization.
- Scalability: Gensim is designed for scalability and can handle large text corpora efficiently. It employs memory-friendly algorithms, streaming, and incremental training for big data scenarios.

spaCy:

- Tokenization: spaCy offers advanced tokenization capabilities that consider linguistic rules and patterns.
- Part-of-Speech Tagging: It performs part-of-speech (POS) tagging, assigning grammatical labels to each token.
- Named Entity Recognition (NER): spaCy includes pre-trained models for named entity recognition (NER), identifying entities like names, organizations, and dates.
- Dependency Parsing: The library provides dependency parsing to analyze the grammatical structure and relationships between words in a sentence.
- Text Classification: spaCy supports text classification tasks, allowing you to train models to categorize text into predefined classes.
- Integration with Deep Learning: It seamlessly integrates with deep learning frameworks like TensorFlow and PyTorch, enabling the incorporation of spaCy's linguistic features into deep learning pipelines.

5.4.2 Doc2Vec

Doc2vec, also known as Paragraph Vector, is an unsupervised deep learning algorithm for generating fixed-length numerical representations (vectors) for variable-length pieces of texts, such as sentences, paragraphs, or entire documents. The algorithm was developed as an extension of the popular Word2Vec algorithm for word embeddings.

Doc2vec uses a neural network architecture to learn dense vector representations of text documents in an unsupervised way. It generates vector representations for documents by training the neural network to predict a word's identity based on the surrounding context of words in the document, as well as the document's unique identity, represented as a document-specific vector. During the training process, the document-specific vector is updated along with the word embeddings to optimize the model's ability to predict the words in the document.

There are two main variants of the Doc2vec algorithm: Distributed Memory version of Paragraph Vectors (PV-DM) and Distributed Bag of Words version of Paragraph Vectors (PV-DBOW).

In the PV-DM variant, the model learns to predict a target word from the context words and a document-specific vector, while in the PV-DBOW variant, the model learns to predict a target word from a document-specific vector alone, without considering any context information.

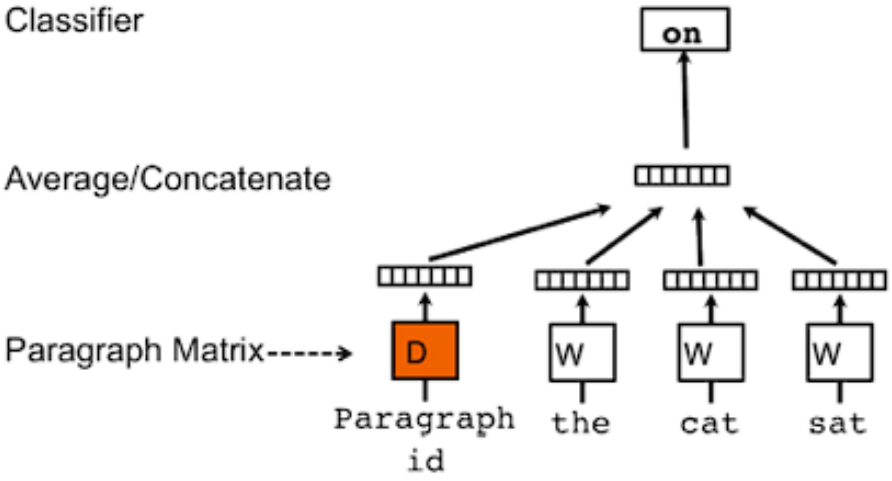


Figure 24 Doc2Vec

This approach was fairly close to Word2Vec, but we couldn’t use AraVec as a baseline as it didn’t have a document vector, so we trained it from scratch, but the results were not good.

5.4.3 AraBert

AraBert is based on the transformer architecture, like BERT and other popular language models. The transformer is a neural network architecture that uses attention mechanisms to model the relationships between words in a sentence.

Pre-training: AraBert was pre-trained on a large corpus of Arabic text using an unsupervised learning technique called Masked Language Modeling (MLM). In MLM, some of the tokens in a sentence are randomly masked and the model is trained to predict the masked tokens based on the context.

We finetuned AraBert using sentence transformers which applies a task of Semantic Textual Similarity, which gives similarity score to 2 sentences, it extracts the embeddings of the 2 sentences then calculates the cosine similarity on them and the loss is the difference between this score and the actual score.

As of the limitation of this training approach we only trained it on the QurSim dataset, which was the only annotated verse pair dataset available, we split the dataset as a 90% train and 10% test and achieved 81% accuracy on the test set but the resulting results was not as good as the word embeddings as we couldn't utilize our dataset and AraBert was computationally more expensive than the Word2Vec model.

As you can see from the results table the model was performing poorly on verses that are not similar.

5.4.4. AraGPT2

AraGPT2 comes from the people who created AraBert, trained on the same dataset that AraBert was trained on. GPT models are trained differently than Bert based models, rather than predicting the masked word, GPT is trained to predict the next word in a sentence or sequence of words, given the preceding words or context. This is done using a self-supervised learning approach where the model is its own training data, and the objective is to maximize the likelihood of predicting the next word. The GPT model uses a variant of the Transformer architecture called the decoder-only Transformer, which is designed to generate text output from the input sequence.

The training of the GPT model involves massive amounts of compute resources, such as GPUs and TPUs, and takes several days or weeks to complete. The resulting model can generate high-quality text output for a wide range of natural language processing tasks.

To finetune AraGPT we use AITextGen library, only on the Tafseer books where we structured the dataset as verse: {Tafseer}, the model can also generate Tafseer for verses but it's not the safest to use as this topic is very sensitive.

The resulting model's precision was 90% but recall was very bad and nearly 0 as it predicted that all Quran verses are similar, and the quality of generation was very poor.

5.4.5 Extracting Embeddings using chatGPT.

This approach achieved the highest accuracy of **88**% but had the same problem as the AraGPT2 finetuning as it related all verses of Quran as similar with all scores very close (+80%) and had another problem which is the cost of the extraction of a topic, So we decided that is not the best to use it.

5.6 Experimental Results

In this section, we present the results of our experiments comparing multiple word embedding models. We evaluate the models on cosine similarity metric to assess their performance and analyze their strengths and weaknesses.

1- Sentence Embeddings:

As the embeddings is per word, to get a full embedding vector of an input sentence we tried 2 techniques, averaging the embeddings and concatenating them but concatenation will suffer from space & length problems, so we went along with averaging the word embeddings.

2- Similarity Metrics:

The cosine distance measures the angle between two vectors in high-dimensional space, and it is computed by taking the dot product of the two vectors and dividing it by the product of their magnitudes. The resulting value ranges from -1 to 1, with 1 indicating that the two vectors are identical and 0 indicating that they are completely dissimilar. The cosine distance is often used to compare the similarity of word embeddings, where each word is represented by a vector. For example, two words that are similar in meaning, such as "car" and "automobile", are likely to have similar vector representations, and their cosine distance would be close to 1.

$$cosine\_similarity(A, B) = \frac{(A \cdot B)}{(\|A\| \cdot \|B\|)}$$

3. Model Evaluation:

We Evaluated The Models on QurSim dataset, which consists of 7680 records of pairs of similar verses and their similarity score , we capped the labels to 0 , 1 , 2 as 0 was unrelated , 1 as being average relation , 2 as closely related , and we then combined the 1 , 2 classes to calculate an overall True Positive where the

threshold was 0.6 , if it's higher than or equal to 0.6 that counts as a true positive if the original label was 1 or 2 .

	premise	hypothesis	label
0	بسم الله الرحمن الرحيم	الحمد لله رب العالمين	2
1	بسم الله الرحمن الرحيم	الرحمن الرحيم	1
2	بسم الله الرحمن الرحيم	مالك يوم الدين	0
3	بسم الله الرحمن الرحيم	الم	0
4	بسم الله الرحمن الرحيم	الله لا إله إلا هو الحي القيوم	0
...	...	...	...
7674	لم يلد ولم يولد	وجعلوا بينه وبين الجنة نسبًا ۚ ولقد علمت الجنة...	2
7675	لم يلد ولم يولد	سبحان الله عما يصفون	1
7676	ولم يكن له كفؤا أحد	بديع السماوات والأرض ۖ أنى يكون له ولد ولم تكن...	2
7677	قل أعوذ برب الفلق	فالق الإصباح وجعل الليل سكنًا والشمس والقمر حس...	2
7678	من الجنة والناس	وكذلك جعلنا لكل نبي عدوا شياطين الإنس والجن يو...	2

Snippet of QurSim data

4. Model Comparison

	Accuracy	***Recall***	***Precision***	***F1 Score***
Word2Vec	***0.8263***	***0.898***	***0.898***	***0.87***
Doc2Vec	*0.76*	*0.93*	*0.4*	*0.56*
AraBert	*0.81*	*0.65*	*0.89*	*0.75*

Highest F1Score was AraVec embeddings finetuned on our dataset which is the reason why we get a surprisingly good output results as the Tafseer books included many explanations of each word which helped the context embeddings be representative,

Our word embeddings are hosted on hugging face as open source for everybody to try and use and the dataset is also available on hugging face as a combined Quran related corpus for training language models and word embeddings.

Sample Output for Word2Vec:

Input Query: "الزواج"

Output: [الأحزاب] يا أيها الذين آمنوا إذا نكحتم المؤمنات ثم طلقتموهن من قبل أن تمسوهن فما لكم عليهن من عدةٍ تعتدونها ۖ فمتعوهن وسرحوهن سراحًا جميلًا (Score 0.65)

Input Query: "التعب"

Output: [فاطر] الذي أحلنا دار المقامة من فضله لا يمسنا فيها نصب ولا يمسنا فيها لغوب (0.66 score)
(where the model found meaningful matches not just literal)

Input Query: "الصبر"

Output Query: [البقرة] واستعينوا بالصبر والصلاة ۚ وإنها لكبيرة إلا على الخاشعين (0.86 score)

Input Query: "الخمر":

Output Query: [المائدة] يا أيها الذين آمنوا إنما الخمر والميسر والأنصاب والأزلام رجس من عمل الشيطان فاجتنبوه لعلكم تفلحون (0.70 score)

5.7 Conclusion

The results of our study indicate that Word2Vec outperformed larger and stronger models like BERT or GPT in producing meaningfully better top 10 results. This suggests that context-specific models like Word2Vec is more effective for measuring semantic similarity in certain types of text, such as the Quran. Moreover, the failure of larger models in identifying unrelated verses from the dataset indicates that they are not suitable for context-specific tasks. Our findings have important implications for the field of natural language processing, as they demonstrate that context-specific models such as Word2Vec are more appropriate for certain types of text than general-purpose models such as BERT or GPT.

Code and Data Snippets:

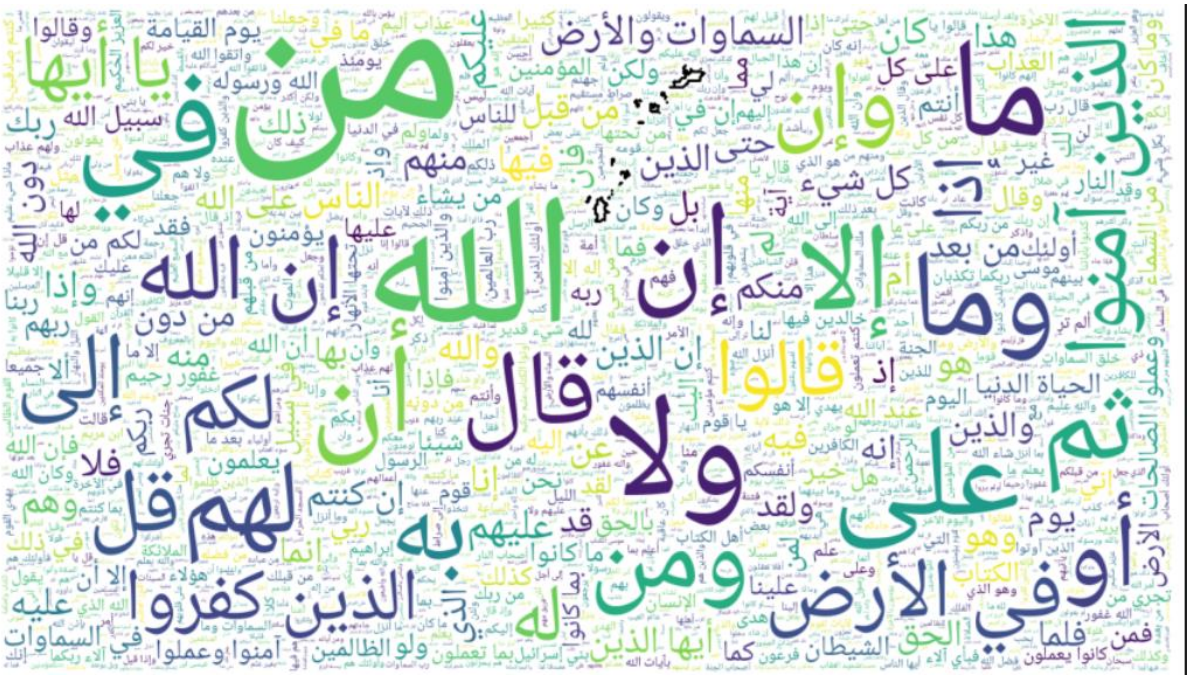
```
similarity_dataset
```

	ID	Source_Sura	premise	target_sura	hypothesis	common_roots	label
0	1	1	بسم الله الرحمن الرحيم	1	الحمد لله رب العالمين	0	2
1	2	1	بسم الله الرحمن الرحيم	1	الرحمن الرحيم	2	1
2	3	1	بسم الله الرحمن الرحيم	1	مالك يوم الدين	0	0
3	4	1	بسم الله الرحمن الرحيم	3	الم	0	0
4	5	1	بسم الله الرحمن الرحيم	3	الله لا إله إلا هو الحي القيوم	0	0
...	...	...	...	...	...	...	...
7674	7675	112	لم يلد ولم يولد	37	وجعلوا بينه وبين الجنة نسبًا ولقد علمت الجنة...	0	2
7675	7676	112	لم يلد ولم يولد	37	سبحان الله عما يصفون	0	1
7676	7677	112	ولم يكن له كفوًا أحد	6	بديع السماوات والأرض أنى يكون له ولد ولم تكن...	1	2
7677	7678	113	قل أعوذ برب الفلق	6	فالق الإصباح وجعل الليل سكنًا والشمس والقمر حس...	1	2
7678	7679	114	من الجنة والناس	6	وكذلك جعلنا لكل نبي عدوا شياطين الإنس والجن يو...	2	2

7679 rows × 7 columns

Tafseer Scraping:

```python
import requests
from bs4 import BeautifulSoup
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

url = 'https://quran.ksu.edu.sa/tafseer/saadi/'
data = []

print("starting scrapping ... ")
while url:
    # Get the HTML content of the current page
    response = requests.get(url, verify=False)
    soup = BeautifulSoup(response.text, 'html.parser')
    div_saadi = soup.find('div', {'class': 'nass', 'id': 'div_saadi'})
    text_content = div_saadi.get_text()
    a_element = soup.find_all('a', {'class': 'btn no-print'})[3]['href']

    if a_element:
        url = a_element
    else:
        url = None
    data.append((a_element , text_content))
```




```
dataset.rename({'0':'text'},axis=1,inplace=True)
dataset['text'] = dataset['text'].str.replace('[^\w\s]',' ') #remove punct
dataset['text'] = dataset['text'].str.strip()
# remove tashkeel
dataset['text'] = dataset['text'].str.replace('[\u064B-\u0652]','')
# remove tatweel
dataset['text'] = dataset['text'].str.replace('\u0640',' ')
# remove harakat
dataset['text'] = dataset['text'].str.replace('[\u064E-\u0651]','')
# remove alef maksura
dataset['text'] = dataset['text'].str.replace('\u0649','ا')
# remove alef with madda
dataset['text'] = dataset['text'].str.replace('\u0622','ا')
# remove alef with hamza above
dataset['text'] = dataset['text'].str.replace('\u0623','ا')
# remove alef with hamza below
dataset['text'] = dataset['text'].str.replace('\u0625','ا')



# remove numbers from dataset
dataset['text'] = dataset['text'].str.replace('\d+', ' ')
#remove multiple spaces
dataset['text'] = dataset['text'].str.replace(' +',' ')

model.build_vocab(corpus, update=True , keep_raw_vocab=True , min_count=None ,
 trim_rule=None)
model.train(corpus, total_examples=model.corpus_count, epochs=10)
```


Enter a Quran Verse
مثلهم كمثل الذي استوقد نارًا فلما أضاءت ما حوله ذهب الله بنورهم وتركهم في ظلماتٍ لا يبصرون
Clear
Submit
output
مثلهم كمثل الذي استوقد نارًا فلما أضاءت ما حوله ذهب الله بنورهم وتركهم في ظلماتٍ لا يبصرون [البقرة]
99%
مثلهم كمثل الذي استوقد نارًا فلما أضاءت ما حوله ذهب الله بنورهم وتركهم في ظلماتٍ لا يبصرون [البقرة]
91%
أومن كان ميتًا فأحييناه وجعلنا له نورًا يمشي به في الناس كمن مثله في الظلمات ليس بخارجٍ منها كذلك زين للكافرين ما كانوا يعملون [الأنعام]
90%
أو كظلماتٍ في بحرٍ لجي يغشاه موج من فوقه موج من فوقه سحاب ظلمات بعضها فوق بعضٍ إذا أخرج يده لم يكد يراها ومن لم يجعل الله له نورًا فما له من نورٍ [النور]
89%
أنزل من السماء ماءً فسالت أودية بقدرها فاحتمل السيل زبدًا رابيًا ومما يوقدون عليه في النار ابتغاء حليةٍ أو متاعٍ زبد مثله كذلك يضرب الله الحق والباطل فأما الزبد فيذهب جفاءً وأما ما ينفع الناس فيمكث في الأرض كذلك يضرب الله الأمثال [الرعد]
89%
فلما قضى موسى الأجل وسار بأهله آنس من جانب الطور نارًا قال لأهله امكثوا إني آنست نارًا لعلي آتيكم منها بخبرٍ أو جذوةٍ من النار لعلكم تصطلون [القصص]
89%
إن في خلق السماوات والأرض واختلاف الليل والنهار والفلك التي تجري في البحر بما ينفع الناس وما أنزل الله من السماء من ماءٍ فأحيا به الأرض بعد موتها وبث فيها من كل دابةٍ وتصريف الرياح والسحاب المسخر بين السماء والأرض لآياتٍ لقومٍ يعقلون [البقرة]
89%
أو كصيبٍ من السماء فيه ظلمات ورعد وبرق يجعلون أصابعهم في آذانهم من الصواعق حذر الموت والله محيط بالكافرين [البقرة]
89%
وهو الذي جعل لكم النجوم لتهتدوا بها في ظلمات البر والبحرٍ قد فصلنا الآيات لقومٍ يعلمون [الأنعام]
88%
هو الذي يسيركم في البر والبحرٍ حتى إذا كنتم في الفلك وجرين بهم بريحٍ طيبةٍ وفرحوا بها جاءتها ريح عاصف وجاءهم الموج من كل مكانٍ وظنوا أنهم أحيط بهم دعوا الله مخلصين له الدين لئن أنجيتنا من هذه لنكونن من الشاكرين [يونس]
88%
مثل ما ينفقون في هذه الحياة الدنيا كمثل ريحٍ فيها صر أصابت حرث قوم ظلموا أنفسهم فأهلكته وما ظلمهم الله ولكن أنفسهم يظلمون [آل عمران]

Chapter 6: Deployment on Hugging face

All our models are currently hosted on hugging face, which is a hub of models that incorporate all kinds of tasks, classification, regression, summarization, etc.

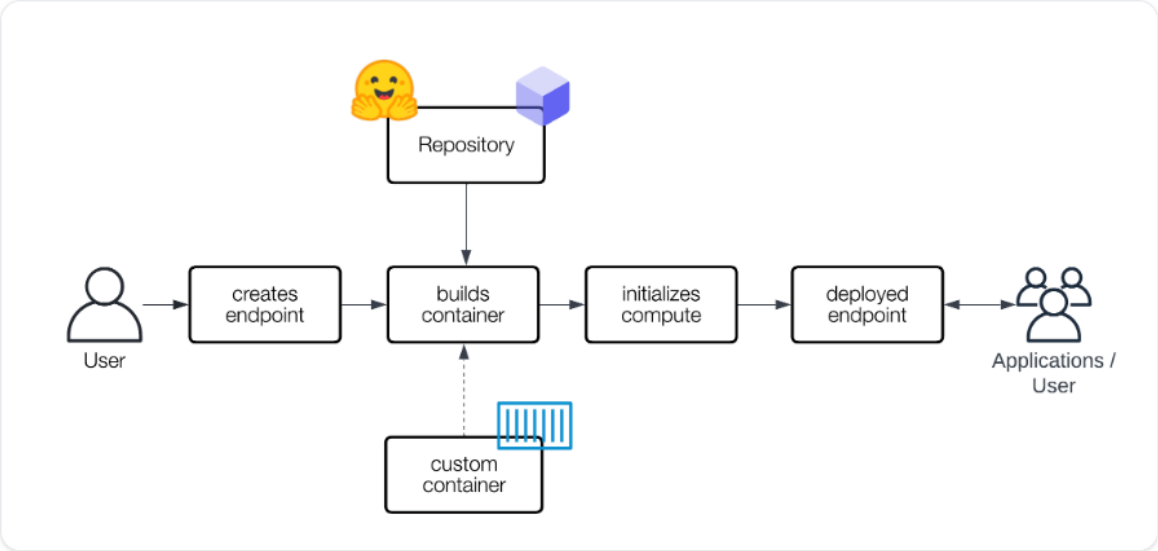
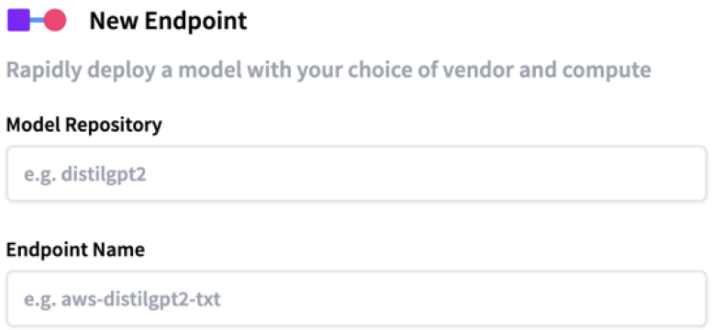


Figure 25 Deployment Pipeline

Deploy models for production in a few simple steps.

1. Select your model

Select the model you want to deploy. You can deploy a custom model or any of the 60,000+ Transformers, Diffusers or Sentence Transformers models available on the 🤗 Hub for NLP, computer vision, or speech tasks.



2. Choose your cloud

Pick your cloud and select a region close to your data in compliance with your requirements (e.g. Europe, North America or Asia Pacific).

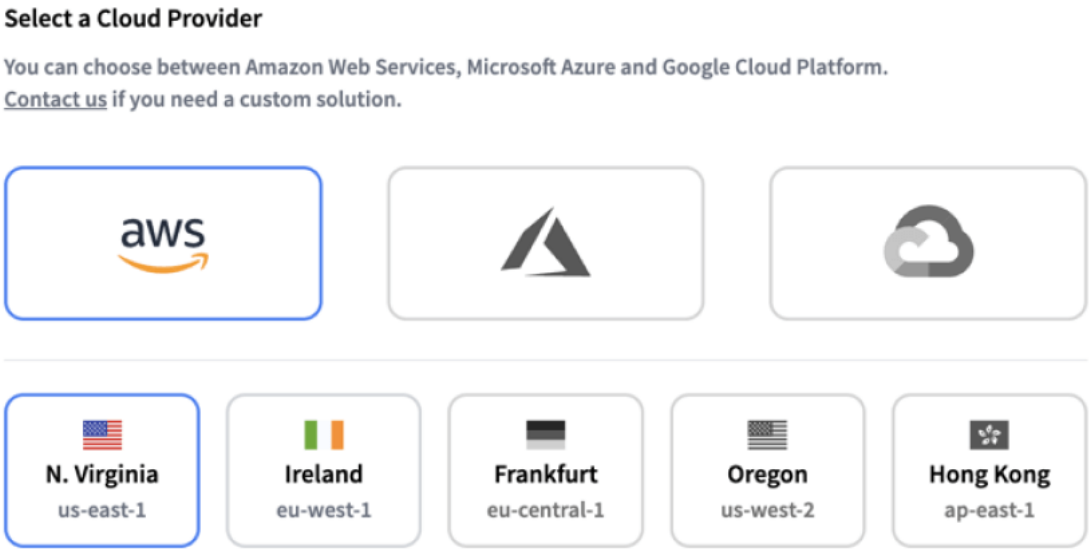
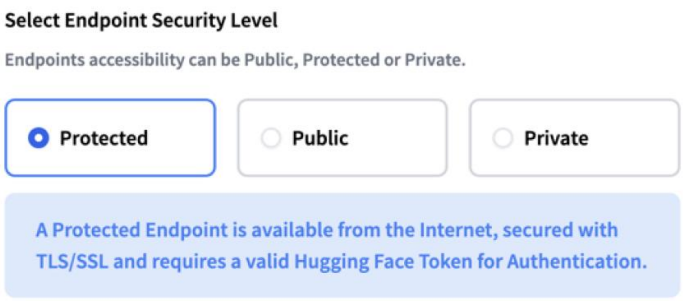


Figure 26 available providers

3. Select your security level

- **Protected Endpoints** are accessible from the Internet and require valid authentication.
- **Public Endpoints** are accessible from the Internet and do not require authentication.
- **Private Endpoints** are only available through an intra-region secured AWS or Azure PrivateLink direct connection to a VPC and are not accessible from the Internet.



4. Create and manage your endpoint

Click create and your new endpoint is ready in a couple of minutes. Define autoscaling, access logs and monitoring, set custom metrics routes, manage endpoints programmatically with API/CLI, and rollback models - all super easy.

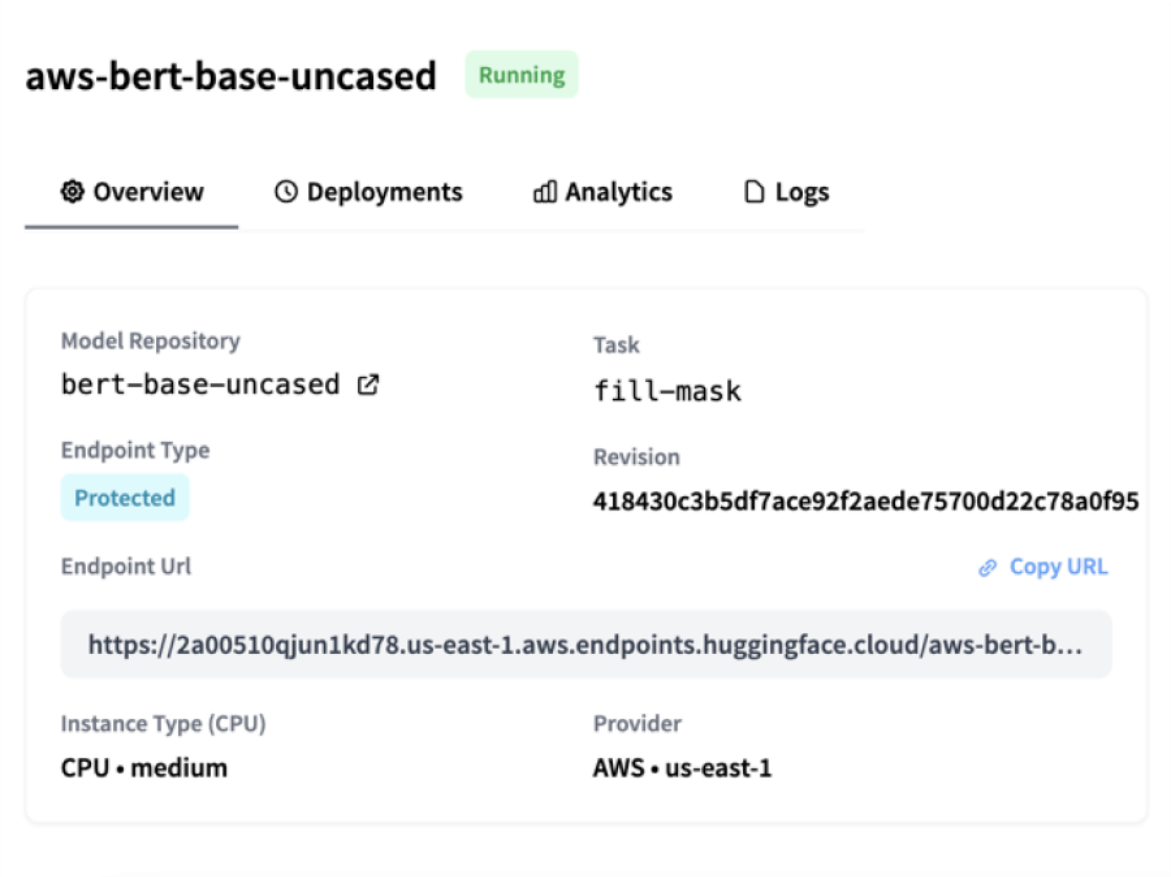


Figure 27 example of a running service

Chapter 7: Flutter Workflow

7.1 Introduction

Developing a mobile application is a complex and challenging task. There are many frameworks available to develop a mobile application. Android provides a native framework based on Java language and iOS provides a native framework based on Objective-C / Swift language. However, to develop an application supporting both the OSs, we need to code in two different languages using two different frameworks. To help overcome this complexity, there exists mobile frameworks supporting both OS.

Flutter is a powerful and versatile framework for building beautiful and high-performance applications across multiple platforms. Developed by Google, Flutter has gained immense popularity among developers for its ability to create stunning user interfaces and deliver a seamless experience on both iOS and Android devices. With a single codebase.

The notable feature of Flutter that helped us a lot during development is its **hot-reload** capability, which allows us to see the changes they make to the code reflected instantly in the app without the need for a full restart.

7.2 Design Pattern and Project Structure

We used MVVM design pattern, MVVM is useful to move business logic from view to ViewModel and Model. ViewModel is the mediator between View and Model which carries all user events and returns the result.

The key benefit is allowing true separation between the View and Model and the efficiency that you gain from having that. What it means in real terms is that when your model needs to change, it can be changed easily without the view needing to and vice-versa.

There are three key things that flow out of applying MVVM −

- Maintainability: - The presentation layer and the logic are loosely coupled, due to this code is easily maintainable and reusable. As the code base will increase over the course of time, this will help you to distinguish between them.

- Testability: - The ViewModel is easier to unit test than code-behind or event-driven code.
- Extensibility: - This architecture gives you assurance which enables code to get extensible over the period of time.
- Reuseability: this architecture enables us to reuse some components in other projects.

7.2.1 Design pattern explanation

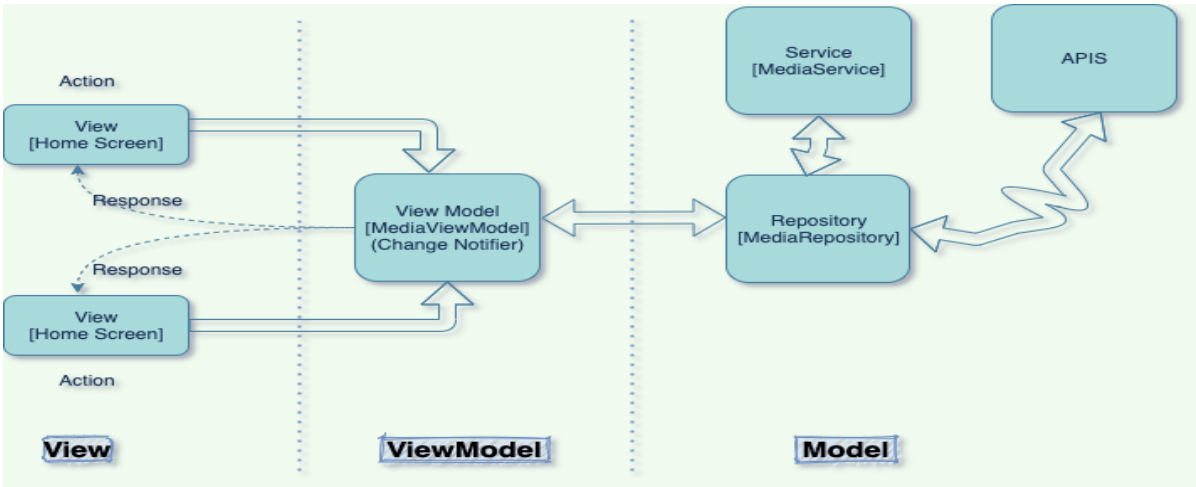


Figure 28 Design pattern overview

Model

The model represents a single source of truth that carries the real-time fetch data or database-related queries.

This layer can contain business logic, code validation, etc. This layer interacts with ViewModel for local data or for real-time. Data are given in response to ViewModel.

ViewModel

ViewModel is the mediator between View and Model, which accepts all the user events and requests that Model for data. Once the Model has data then it returns to ViewModel and then ViewModel notify that data to View.

ViewModel can be used by multiple views, which means a single ViewModel can provide data to more than one View.

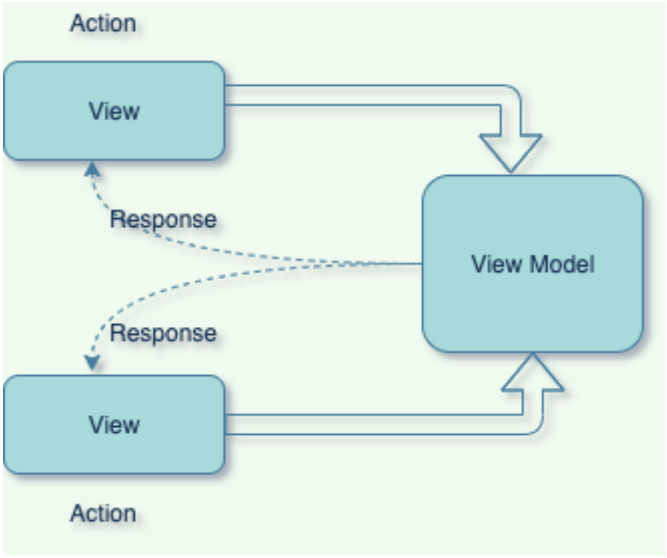


Figure 29 model

View

The view is where the user is interacting with Widgets that are shown on the screen. These user events request some actions which navigate to ViewModel, and the rest of ViewModel does the job. Once ViewModel has the required data then it updates View.

7.2.2 folder structure

- The **common** directory is where you can place files that are shared across the entire application. This includes helper classes, constants, that are used throughout the project.

- The **config** directory where it holds the configuration parameters, environment-specific settings, or other application-specific configurations.

- The **data** directory is used for storing data-related files, such as JSON files, local databases, or assets used by your application. It provides a dedicated location for managing and accessing data resources. For example, you can store sample data or configuration files in this directory, and access them when needed during development or testing.

- The **models** directory is where you define classes that represent the data structures used in your application. These models are typically used to parse and serialize JSON responses from API calls. By mapping the

JSON data to model objects, so we can easily work with the data in a type-safe manner throughout your app.

- The `**providers**` or `**viewmodels**` directory houses the classes responsible for managing the state and business logic of your application. These classes, often referred to as providers or view models, interact with the models, handle data processing, and expose the necessary data and methods to the UI. They act as intermediaries between the UI and the underlying data, ensuring separation of concerns and promoting a clean architecture.

- The **repository** directory is commonly used in the context of implementing a repository pattern for data management. It contains an abstract class or interface that defines the contract for accessing data from various sources, such as APIs or local databases. By separating the data access logic into a repository, you can decouple the UI components from the specific data sources, promoting modularity and testability.

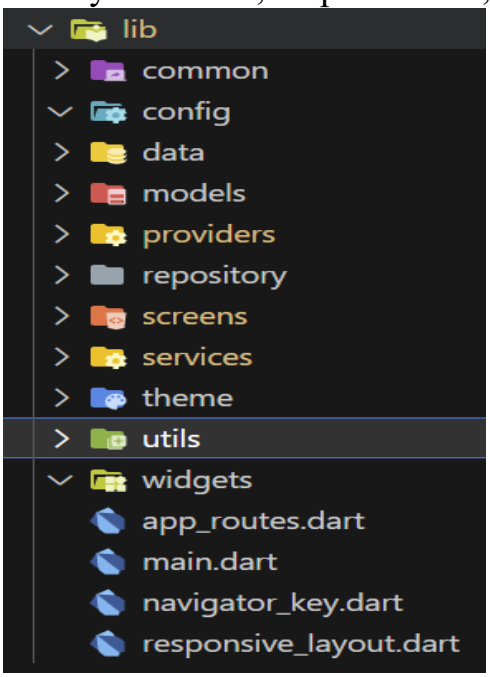
- The **screens** directory contains the UI views or screens of the application. Each screen represents a specific page or view that the user interacts with. Screens are responsible for displaying the relevant UI components, handling user input, and coordinating with other parts of your application.

- The **services** folder is where you can store classes that handle various tasks such as making API calls, managing local storage, handling network connectivity, or interacting with other external services.

- The **theme** directory to your Flutter project can be useful for managing the application's themes and styles. it can house files that define the various themes, color palettes, typography, and other visual styles used throughout your app.

- The **widgets** directory is used for storing reusable UI components that are shared across multiple screens or views. Widgets encapsulate specific parts of the UI and can be composed together to create complex user interfaces. By breaking down your UI into smaller, reusable widgets, you can improve code maintainability and facilitate easier UI modifications.

- The **utils** folder can be utilized to store utility functions, helper classes, or constants that are commonly used throughout your project. This folder is particularly handy for organizing code snippets that provide commonly needed functionalities, such as date formatting, string manipulation, or conversion utilities.



- main.dart is the entry point of the app.
- app_routes.dart contains all app routes and navigation
- navigator_key.dart contains a global key navigator
- responsive_layout.dart contains the code for responsiveness

7.3 UI Design and Styling in Flutter

When it comes to UI design and styling in our application, there are several aspects that we considered some of them are:

- **Material Design:** Material Design is a design system developed by Google, offering ready-made UI components and guidelines for creating visually appealing and intuitive interfaces. We leveraged Flutter's built-in Material Design widgets to implement a consistent and familiar look and feel.
- **Widgets and Layouts**: Widgets are the building blocks of your application's UI, and they control the appearance and behavior of the user interface elements.
- **Themes and Styles:** Flutter allows us to define themes and styles to ensure a consistent and cohesive visual appearance across your app. By creating a `ThemeData` object, we set global styles for text, colors, typography, buttons, and other UI elements.
- **Responsive Design:** We used both `LayoutBuilder` and `MediaQuery` widgets, so we can dynamically adjust the UI layout and component sizes based on the available screen real estate.
- **Typography and Fonts:** we included custom font files in our project and specified them in our 'ThemeData' object.

7.4 State Management Approaches

In Flutter, managing and updating your application's state (data) is known as state management. It entails monitoring different data values, processing data changes, and reflecting such changes in the user interface.

Due to the reactive programming style used to create the UI, state management is essential in Flutter. The UI must be updated to reflect changes in your application's state as soon as they occur. For managing state, Flutter offers a variety of strategies and packages, each with unique advantages and applications.

State management approaches that we used:

- **Local State**: For simple applications with limited state requirements, you can manage state directly within the widget that uses it. Stateful widgets in Flutter have a mutable state that can be updated and trigger UI updates using the `setState()` method.

- **InheritedWidget**: Flutter provides the `InheritedWidget` class, which allows you to share state down the widget tree to all its descendants. It is suitable for cases where multiple widgets need access to the same state.

- **Provider**: The `provider` package, as mentioned before, is a popular state management solution. It simplifies sharing and updating state across the widget tree using the concept of providers and consumers. Providers hold the state and notify listeners when it changes, while consumers rebuild specific parts of the UI that depend on the state.

Chapter 8: Integration of features in application and Testing

As we discussed in the introduction our two main features are Recitation with Tashkeel, memorization and Semantic search for both Verses and Ahadith. We will go through each of these features explaining how they are implemented.

- First, we explained how the model is deployed on Hugging face that is a very important part as it provides us with an API (Application Programming Interface) as the models are too big in size to be local on the phone and requires a large computational power that phone doesn't have in this moment.
- Second, we discussed how the application structure is done and the design pattern we followed (MVVM) that helped us a lot in separating different parts of the code for easier way to test and debug the code and make changes without going through my parts of the code.

The above steps created a strong base so we can move on to creating and integrating a lot of features into the Application without any issues.

After implementing the main features, we started to expand the application with complimentary features such as the ability to sign in into the application and save the progress for memorization. The user-authentication feature is implemented using a Firebase database to create different users and to authenticate and save progress so it's easily accessible from any device.

8.1 Integrating Recitation, Memorization, and Semantic Search Features

8.1.1 Recitation and Memorization feature

Overview of the implementation architecture of this feature:

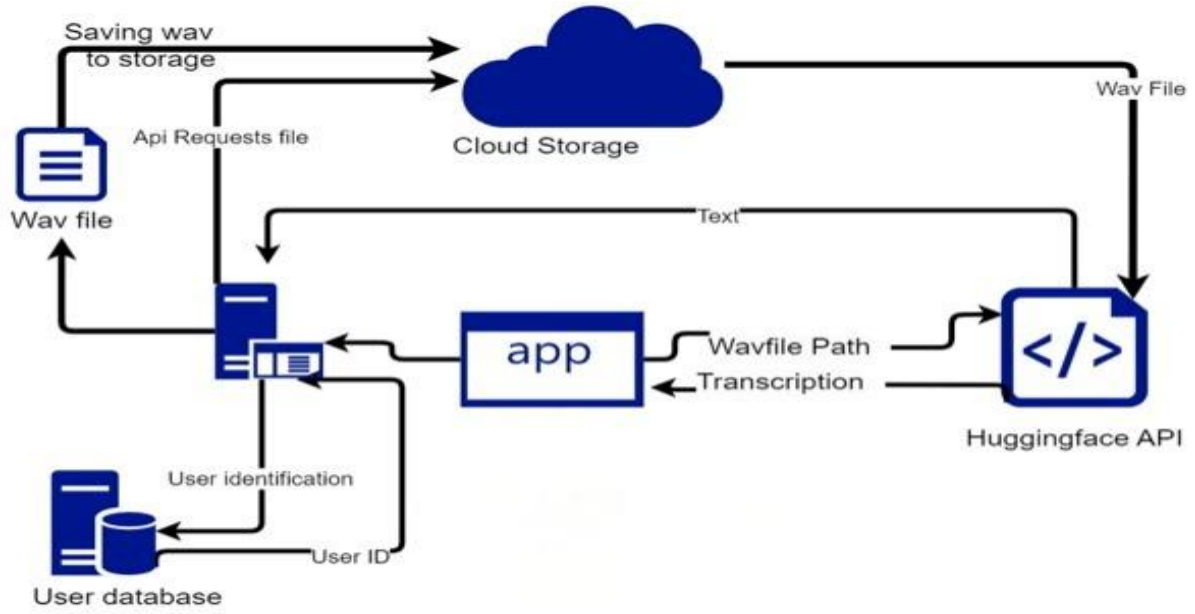


Figure 30 Features infrastructure

The application begins by retrieving user data from the Firebase server and storing it locally for efficient access. When the user interacts with the record button on the screen, the application initiates the audio recording process. Once the user finishes recording by pressing the stop button, the audio file is uploaded to the server. Each audio file is appropriately labeled with the corresponding Surah number and verse number, ensuring easy referencing for future model training.

Next, the application makes an API call to model we discussed above wav2vec, a popular NLP platform, to transcribe the uploaded audio file. The response received from Hugging Face contains the transcription of the audio file in the form of a string.

Following the transcription, the application determines whether the user is in recitation mode or memorization mode. This distinction enables the application to employ different string comparison algorithms based on the user's intended usage. By using appropriate algorithms, the application can effectively compare

and analyze the transcriptions to provide relevant feedback or support based on the user's specific mode.

Why do we use different string comparison algorithms?

In the memorization mode, string comparison is performed without taking Tashkeel (diacritical marks) into account. Therefore, a more straightforward algorithm (**Levenshtein algorithm**) is employed to check for differences in string length and missed words. If any discrepancies are found, the application prompts the user to repeat the verse.

On the other hand, in the recitation or Tarteel mode where Tashkeel is relevant, a different string comparison algorithm called **Needleman-Wunsch** is utilized. This algorithm offers greater flexibility and highlights all the missed Tashkeel or words by replacing them with a visual indicator such as a dash. This visual representation clearly illustrates the incorrect portions for the user to review and correct accordingly.

We will go through each of them explaining how they work and their implementation of code. Then we will provide some screenshots of the application demonstrating each algorithm.

String comparison algorithm used for Recitation. (Needleman Wunsch)

Needleman-Wunsch algorithm, which is a dynamic programming algorithm used to determine the optimal alignment between two sequences in our case the original Arabic verse text(reftext) and the transcript text(recitedtext) that comes from the response. The algorithm assigns scores to different alignment choices and finds the alignment with the highest score.

Here's a step-by-step explanation of the Algorithm:

1. Initialize the dimensions of the score matrix based on the lengths of `refText` and `recitedText`.

2. Initialize the score matrix with gap penalties for the first row and column.

3. Define the match score and mismatch penalty.

4. Iterate over the rows and columns of the score matrix, calculating the scores for different alignment choices.

   - Calculate the score for a diagonal alignment by comparing the characters at the corresponding positions in `refText` and `recitedText`.

   - Calculate the score for an upward alignment by adding a gap penalty to the score in the cell above.

   - Calculate the score for a leftward alignment by adding a gap penalty to the score in the cell to the left.

   - Store the maximum score among these choices in the current cell of the score matrix.

5. Trace back the optimal alignment by following the path with the highest scores in the score matrix.

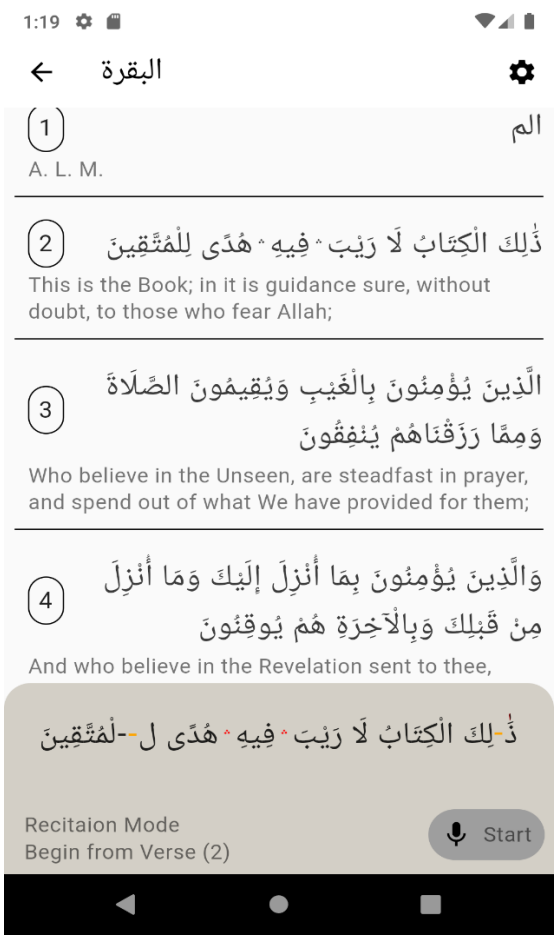
   - Starting from the bottom-right cell, move diagonally if the characters match, upward if there's a gap in `recitedText`, and leftward if there's a gap in `refText`.

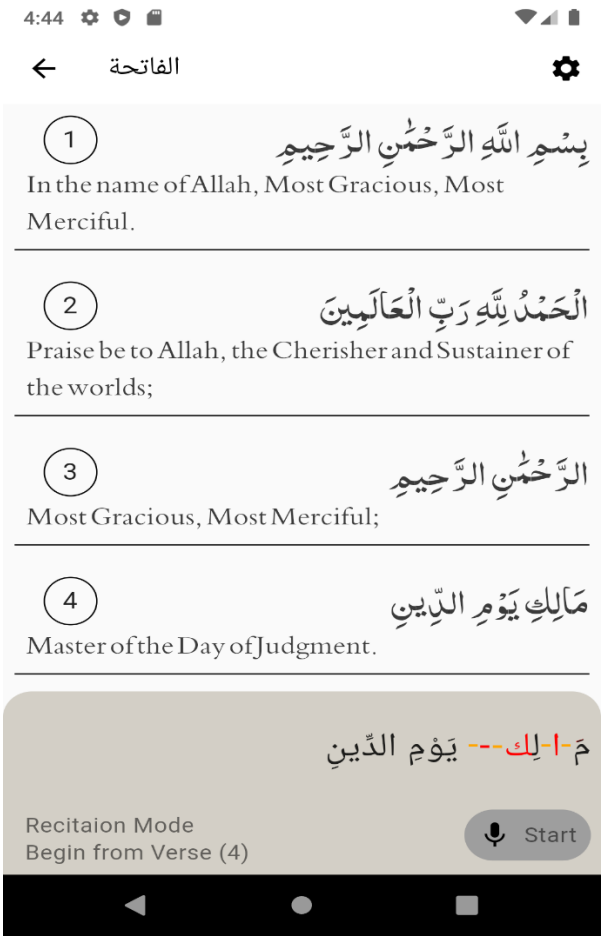
   - Append the aligned characters to `alignedRefText` and `alignedRecitedText`.

The function is useful for aligning and highlighting the differences between two texts, such as a reference text and a recited text.

```
static String needlemanWunsch(String refText, String recitedText) {
    int rows = refText.length + 1;
    int cols = recitedText.length + 1;
    List<List<int>> scoreMatrix =
        List.generate(rows, (_) => List.filled(cols, 0));

    int gapPenalty = -1;

    for (int i = 1; i < rows; i++) {
      scoreMatrix[i][0] = gapPenalty * i;
    }

    for (int j = 1; j < cols; j++) {
      scoreMatrix[0][j] = gapPenalty * j;
    }

    int matchScore = 1;
    int mismatchPenalty = -1;

    for (int i = 1; i < rows; i++) {
      for (int j = 1; j < cols; j++) {
        int diagonalScore;
        if (refText[i - 1] == recitedText[j - 1]) {
          diagonalScore = scoreMatrix[i - 1][j - 1] + matchScore;
        } else {
          diagonalScore = scoreMatrix[i - 1][j - 1] + mismatchPenalty;
        }

        int upScore = scoreMatrix[i - 1][j] + gapPenalty;
        int leftScore = scoreMatrix[i][j - 1] + gapPenalty;
        scoreMatrix[i][j] =
            [diagonalScore, upScore, leftScore].reduce((a, b) => a > b ? a : b);
      }
    }

    String alignedRefText = "";
    String alignedRecitedText = "";

    int i = rows - 1;
    int j = cols - 1;

    while (i > 0 || j > 0) {
      if (i > 0 && j > 0 && refText[i - 1] == recitedText[j - 1]) {
        alignedRefText = refText[i - 1] + alignedRefText;
        alignedRecitedText = recitedText[j - 1] + alignedRecitedText;
        i--;
        j--;
      } else if (i > 0 &&
          scoreMatrix[i][j] == scoreMatrix[i - 1][j] + gapPenalty) {
        alignedRefText = refText[i - 1] + alignedRefText;
        alignedRecitedText = "-$alignedRecitedText";
        i--;
      } else {
        alignedRefText = "-$alignedRefText";
        alignedRecitedText = recitedText[j - 1] + alignedRecitedText;
        j--;
      }
    }
}
```

Screenshots from the Application:





From the above Screenshots we notice that Needleman Wunsch detected the difference between the transcript text and the original Arabic verse text

The red means that the user misspelled some letters in a word while the yellow represents the miss alignment of characters.

String comparison algorithm used for Memorization. (Levenshtein algorithm)

The Levenshtein distance, also known as the Edit Distance, is a measure of the similarity between two strings. It quantifies the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another.

The algorithm to calculate the Levenshtein distance between two strings involves constructing a matrix with dimensions (m+1) x (n+1), where m and n are the lengths of the two strings. Each cell in the matrix represents the minimum edit distance required to transform a substring of the first string into a substring of the second string.

Here's a step-by-step explanation of the Levenshtein algorithm:

1. Initialize the matrix with values from 0 to m in the first row and from 0 to n in the first column. These values represent the number of deletions required to transform an empty string into the corresponding prefix of the other string.

2. Iterate through each cell of the matrix, starting from the second row and second column.

3. For each cell (i, j), calculate the minimum edit distance by comparing the current characters at positions (i-1) and (j-1) in the two strings:

- If the characters are equal, the cost is 0, and the value in the current cell is equal to the value in the previous diagonal cell (i.e., matrix[i-1][j-1]).
- If the characters are different, the cost is 1, and the value in the current cell is the minimum of the following three values:
  - The value in the previous diagonal cell + 1 (substitution)
  - The value in the cell above + 1 (deletion)
  - The value in the cell to the left + 1 (insertion)

4. After iterating through all cells, the final value in the bottom-right cell of the matrix represents the Levenshtein distance between the two strings.

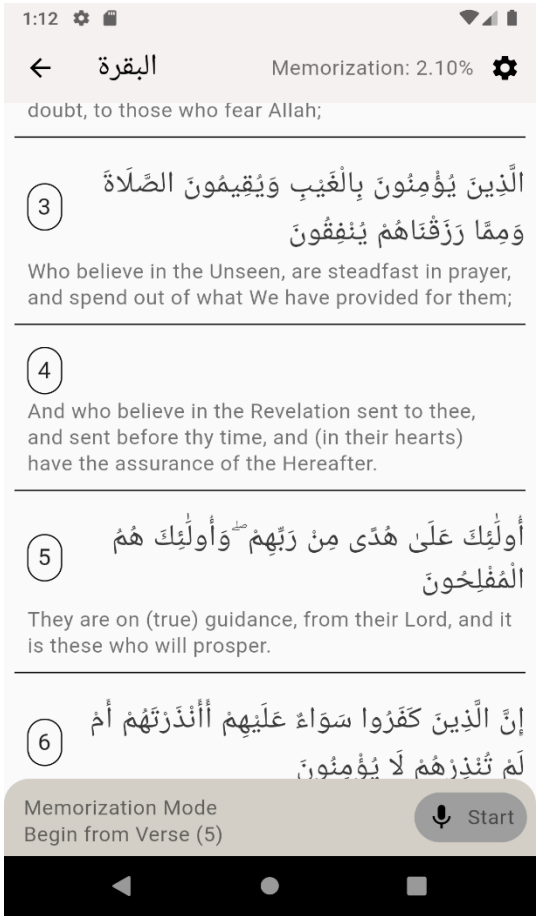
Code for Levenshtein algorithm that we implemented.

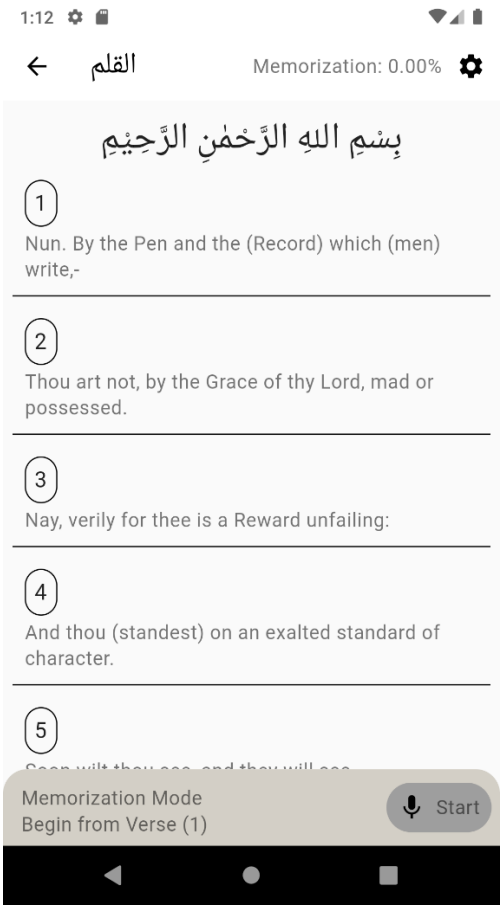
```
static int _levenshtein(String a, String b) {
  // Convert the strings to uppercase for case-insensitive comparison
  a = a.toUpperCase();
  b = b.toUpperCase();

  // Get the lengths of the two strings
  int sa = a.length;
  int sb = b.length;

  int i, j, cost, min1, min2, min3;
  int levenshtein;

  // Create a matrix to store the intermediate values
  List<List<int>> d = List.generate(sa + 1, (int i) => List.filled(sb + 1, 0));

  // Handle empty strings
  if (a.isEmpty) {
    levenshtein = b.length;
    return (levenshtein);
  }
  if (b.isEmpty) {
    levenshtein = a.length;
    return (levenshtein);
  }

  // Initialize the first row and column of the matrix
  for (i = 0; i <= sa; i++) {
    d[i][0] = i;
  }
  for (j = 0; j <= sb; j++) {
    d[0][j] = j;
  }

  // Calculate the Levenshtein distance
  for (i = 1; i <= a.length; i++) {
    for (j = 1; j <= b.length; j++) {
      if (a[i - 1] == b[j - 1]) {
        cost = 0;
      } else {
        cost = 1;
      }

      min1 = (d[i - 1][j] + 1);
      min2 = (d[i][j - 1] + 1);
      min3 = (d[i - 1][j - 1] + cost);

      d[i][j] = _min(min1, _min(min2, min3));
    }
  }

  // The final value in the bottom-right cell represents the Levenshtein distance
  levenshtein = d[a.length][b.length];
  return (levenshtein);
}
```

To calculate the Levenshtein similarity between two strings, you can use the formula: similarity = (1 - (distance / maxLen)), where maxLen is the maximum length of the two strings being compared.

```
static double similarity(String a, String b) {
    double similarity;
    a = a.toUpperCase();
    b = b.toUpperCase();
    similarity = 1 - _levenshtein(a, b) / (_max(a.length, b.length));
    return (similarity);
  }
```

UI screenshot:





In this Screen we demonstrate that the memorized verse after the user said it and passed the allowed threshold in similarity and the length of words are the same then the verse will be visible indicating that the user memorized it correctly

8.1.2 Semantic search feature

The first step in processing the entered query is to determine if it is an Arabic string. If it is indeed an Arabic string, we proceed to make an API call to retrieve the response. The response consists of a list of verses or Ahadiths that share semantic similarities with the entered query. The matching is done based on the meaning rather than exact textual match. This allows us to provide relevant results that align with the intended context and understanding of the query.

Function for checking if the query is Arabic.

```
static bool isProbablyArabic(String str) {
    for (int i = 0; i < str.length; i++) {
      int c = str.codeUnitAt(i);
      if (c >= 0x0600 && c <= 0x06E0) {
        return true;
      }
    }
    return false;
  }
```

The Function that returns the list of Sentence similarity Model from the input and API response

```dart
Future<List<VersesSimilarityModel>> similarVerseModelhApi(text) async {
  // Define the API endpoint URL
  var uri = Uri.parse('https://omarelsayeed-test.hf.space/api/predict');

  // Make a POST request to the API
  var post = await http.post(
    uri,
    headers: {
      HttpHeaders.contentTypeHeader: 'application/json; charset=UTF-8',
      HttpHeaders.acceptHeader: 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, List<String>>{
      'data': [text]
    }),
  );

  try {
    // Check the response status code
    if (post.statusCode == 200) {
      // Parse the response and convert the data into a list of SenstenceSimilarityModel
objects
      var x = (((json.decode(utf8.decode(post.bodyBytes))['data']) as List)
              .elementAt(0) as List)
          .map((element) =>
              VersesSimilarityModel.fromJson(element as Map<String, dynamic>));

      return x.toList();
    } else if (post.statusCode >= 400 && post.statusCode <= 499) {
      // Handle client errors (4xx)
      debugPrint(post.statusCode.toString());
    }
  } on Exception catch (e) {
    // Handle and print any exceptions that occur during the API call
    debugPrint("Error: $e");
  }

  // Return an empty list if there was an error or the response is not successful
  return [];
}
```

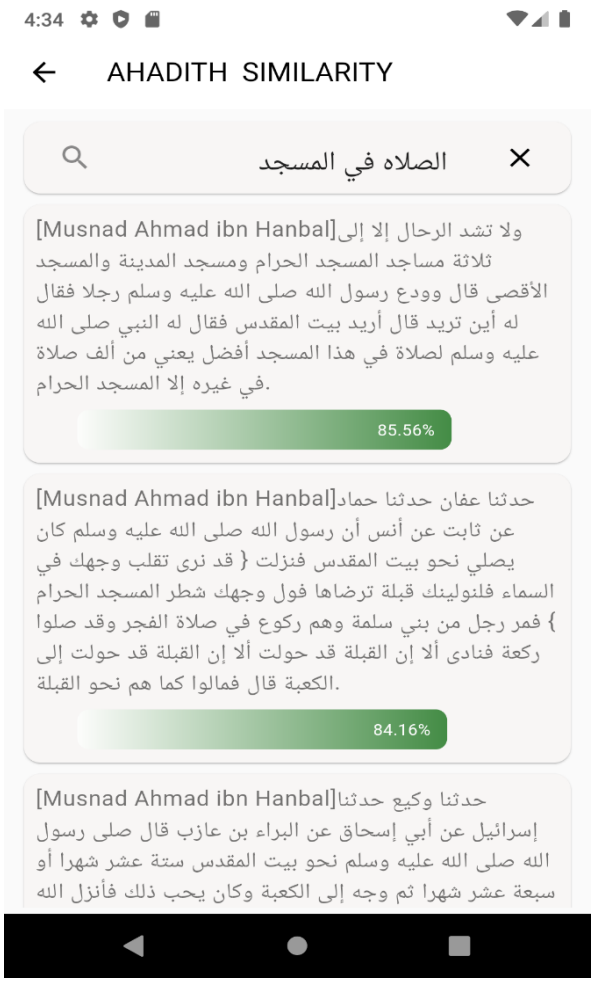
Class that represents Verse similarity model code below

```dart
List<VersesSimilarityModel> versesSimilarityModelFromJson(String str) =>
    List<VersesSimilarityModel>.from(
        json.decode(str).map((x) => VersesSimilarityModel.fromJson(x)));

String versesSimilarityModelToJson(List<VersesSimilarityModel> data) =>
    json.encode(List<dynamic>.from(data.map((x) => x.toJson())));

class VersesSimilarityModel {
  VersesSimilarityModel({
    required this.similarSentence,
    required this.similarityScore,
    required this.surahName,
    required this.ayahNo,
    required this.surahNumber,
  });

  final String similarSentence;
  final double similarityScore;
  final String surahName;
  final int ayahNo;
  final int surahNumber;

  factory VersesSimilarityModel.fromJson(Map<String, dynamic> json) =>
      VersesSimilarityModel(
        similarSentence: json["similar_sentence"],
        similarityScore: json["similarity_score"]?.toDouble(),
        surahName: json["surahName"],
        ayahNo: json["AyahNo"],
        surahNumber: json["SurahNumber"],
      );

  Map<String, dynamic> toJson() => {
        "similar_sentence": similarSentence,
        "similarity_score": similarityScore,
        "surahName": surahName,
        "AyahNo": ayahNo,
        "SurahNumber": surahNumber,
      };
}
```

Class that represents Ahadith similarity model code below.

```dart
class AhadithModel {
  AhadithModel({
    required this.similarSentence,
    required this.similarityScore,
  });

  final String similarSentence;
  final double similarityScore;

  factory AhadithModel.fromJson(Map<String, dynamic> json) => AhadithModel(
        similarSentence: json["similar_sentence"],
        similarityScore: json["similarity_score"]?.toDouble(),
      );

  Map<String, dynamic> toJson() => {
        "similar_sentence": similarSentence,
        "similarity_score": similarityScore,
      };
}
```

UI Design for both Verse similarity and Ahadith similarity



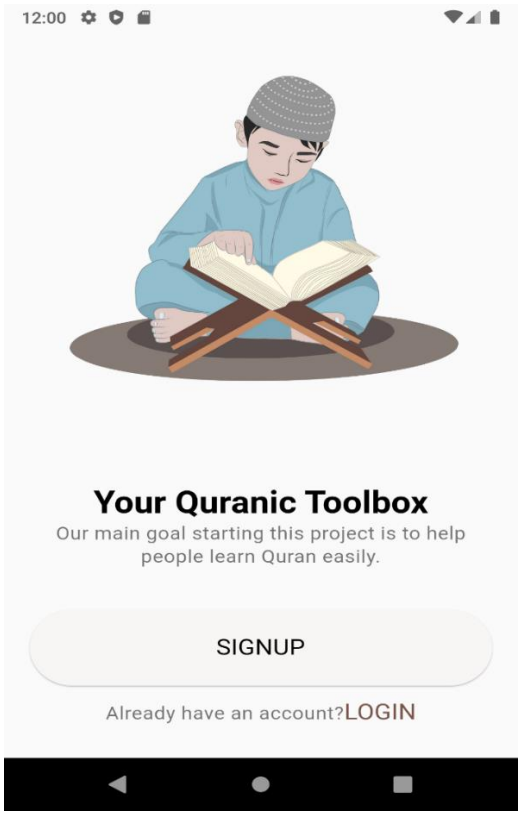
8.1.3 Other features

User database and authentication:

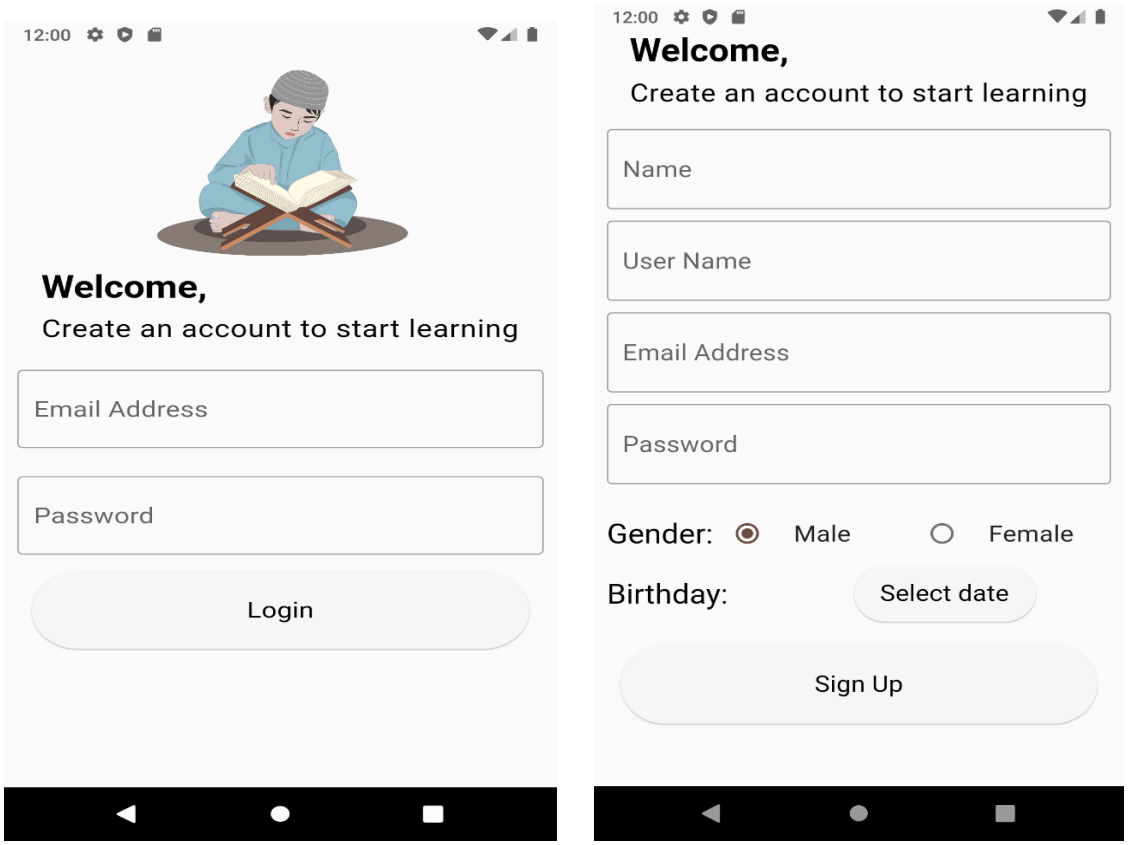
In our application, an essential feature is to determine the age and gender of each user. This information is crucial as it allows us to assign the metadata to the recorded audio files. By associating age and gender information with the audio recordings, we aim to enhance the model's generalization across different voices.

By incorporating age and gender metadata during model training, we can account for the potential variations in pronunciation and speech patterns based on these factors. This improves the model's ability to accurately transcribe and understand audio inputs from users of different ages and genders. Also plays a significant role in enhancing the overall performance and adaptability of the models used in our application. It enables better fine-tuning and customization, leading to improved accuracy and user experience across diverse user demographics.

UI design for welcome, sign and login screens



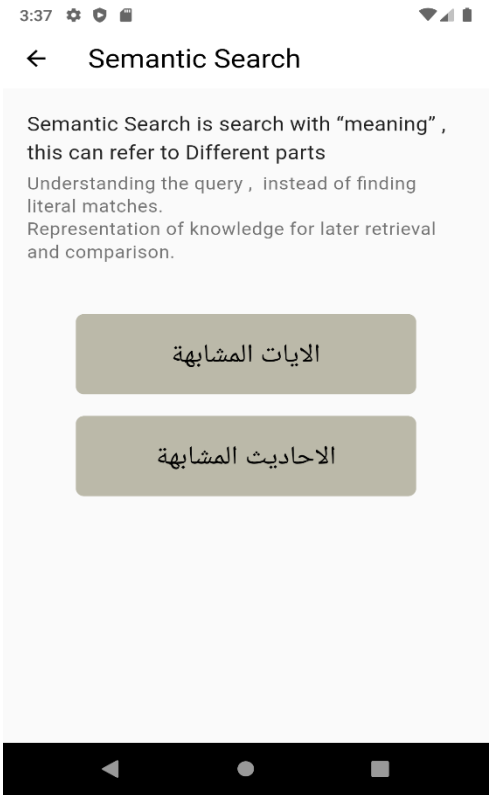
Welcome screen



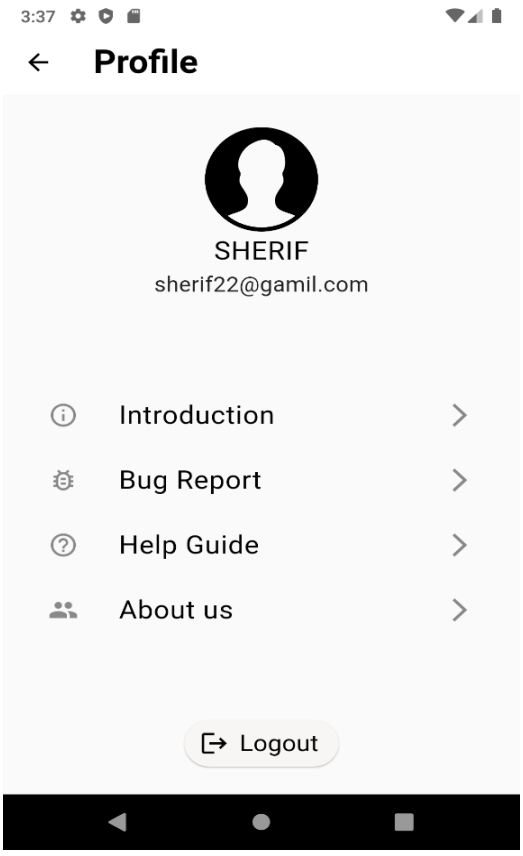
Login and Signup screens

User-Friendly Interface:

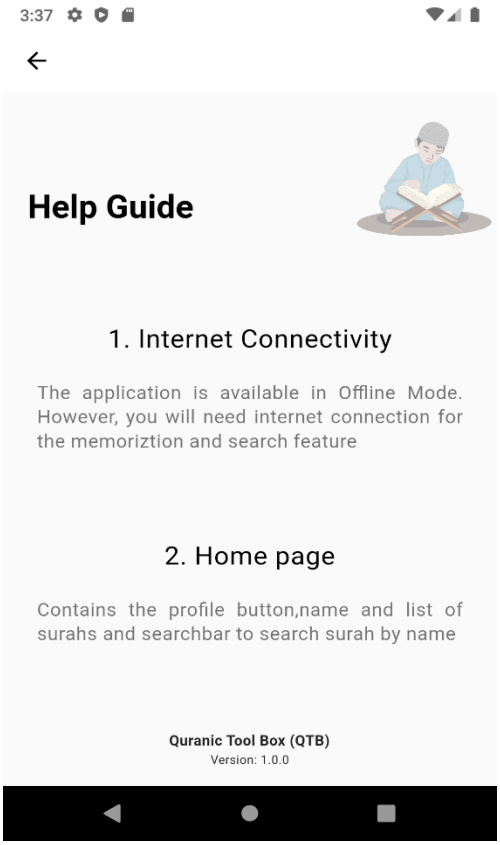
Home page



Semantic search page feature

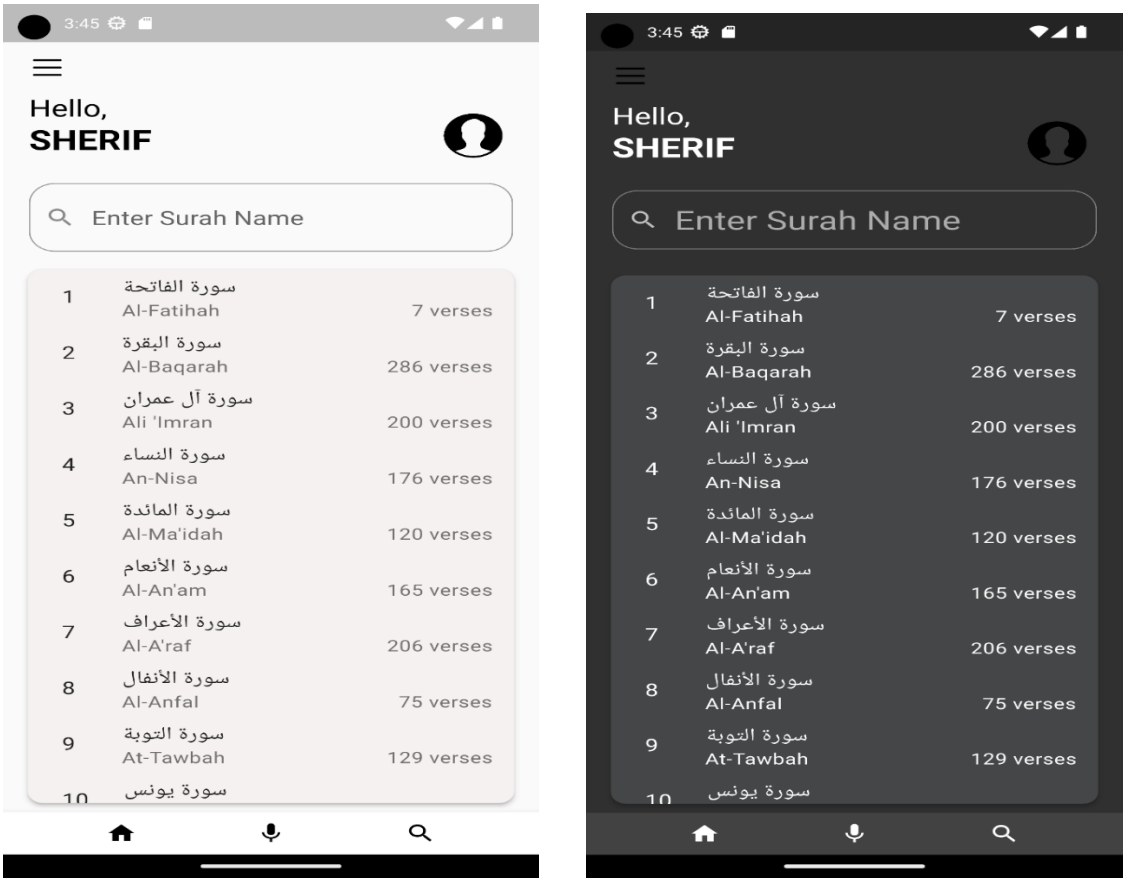


Profile page

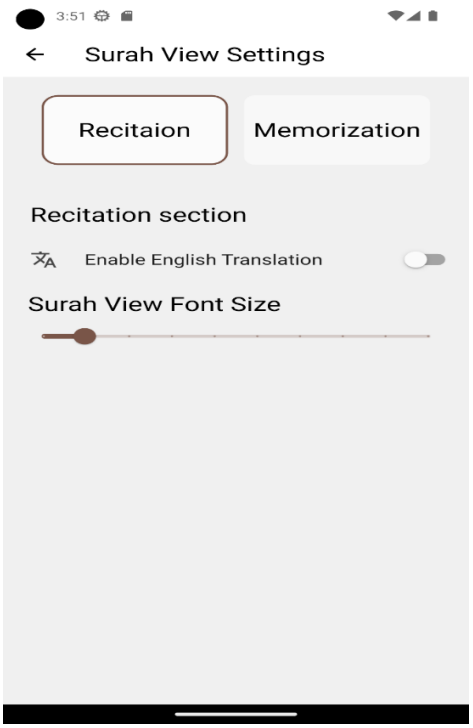


Help guide page.

Personalization Options



Light and dark modes



Surah View Settings screen

Customize font size and switch button to enable or disable English translation.

8.2 Testing the Application

First testing a flutter application is an iterative process some tests are done then add new code and test again so on, so will did unit test on the crucial part of the application like sign and login credentials.

Also, to ensure a seamless user experience, we conducted thorough testing of the user interface design on multiple devices. This allowed us to identify and resolve any pixel overflow bugs that may occur across different screen sizes and layouts. By testing on diverse devices, we ensured the UI elements were appropriately scaled, layout remained intact, and readability was maintained. Through this process, we addressed any visual glitches and improved the overall consistency and responsiveness of the application's design.

Login functionally test cases:

Test Scenario ID	Login-(Application side constrains)	**Test Case ID**	Login-1B
Test Case Description	Login – Negative test case	**Test Priority**	High
Pre-Requisite	NA	**Post-Requisite**	NA

S. No	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comments	
1	Launch application go to login screen	Email: ahmed@gmail.com (valid) Password:gkdkgg123 (valid)	Login successfully (go to home screen)	Login successfully (go to home screen)	Pass	[Priya 12/17/2016 11:44 AM]: Launch successful	
2	Launch application go to login screen	Email: ahmed@gmail.com (valid) Password: gkdkgg (wrong password)	Error (incorrect password)	Error (incorrect password)	Pass	[Priya 12/17/2016 11:46 AM]: Invalid login attempt stopped	

Test Scenario ID	Login- (server side constrains)	**Test Case ID**	Login-1B
Test Case Description	Login – Negative test case	**Test Priority**	High
Pre-Requisite	NA	**Post-Requisite**	NA

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**	**Test Comments**
1	Launch application go to login screen	Email: ahmed@gmail.com (non-exist email) Password:gkdkgg123 (valid)	Error (user doesn't exist)	Error (user doesn't exist)	IE -11	Pass	[Priya 12/17/2016 11:44 AM]: Launch successful
2	Launch application go to login screen	Email: ahmed@gmail.com (non-exist email) Password:gkdkgg123 (valid)	Error (user doesn't exist)	Error (user doesn't exist)	IE -11	Pass	[Priya 12/17/2016 11:45 AM]: Invalid login attempt stopped
3	Launch application go to login screen	Email: ahmed@gmail.com (valid) Password: gkdkgg (wrong password)	Error (incorrect password)	Error (incorrect password)	IE -11	Pass	[Priya 12/17/2016 11:46 AM]: Invalid login attempt stopped

Sign up functionality test cases:

Test Scenario ID	Sign up – (application side constrains)	**Test Case ID**	Sign up -1A
Test Case Description	Sign up – negative test case	**Test Priority**	High
Pre-Requisite	A valid user account	**Post-Requisite**	NA

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**	**Test Comments**	
1	Launch application go to sign up screen	Name: ah (Less than 4) Username: ahmed (valid) Password: gkdkgg123 (valid) Email: ahmed@gmail.com (valid) Select gender & birthday	Error message (enter correct name)	Error message (enter correct name)	Pass	[Priya 12/17/2016 11:44 AM]: Launch successful	
2	Launch application go to sign up screen.	Name: ahmed (valid) Username: Ah (less than 4) Password: gkdkgg123 (valid) Email: ahmed@gmail.com (valid) Select gender & birthday	Error message (enter correct name)	Error message (enter correct name)	Pass	[Priya 12/17/2016 11:45 AM]: Login successful	
3	Launch application go to sign up screen	Name: ahmed (valid) Username: Ahmed (valid) Password: gkdkgg123 (valid) Email: ahmedgmail.com (doesn't contain "@") Select gender & birthday	Error message (enter correct email)	Error message (enter correct email)	Pass		

4	Launch application go to sign up screen	Name: ahmed (valid) Username: Ahmed (valid) Password:gkd (less than 4) Email: ahmed@gmail.com (valid) Select gender & birthday	Error message (enter correct password)	Error message (enter correct password)	Pass		
5	Launch application go to sign up screen	Name: ahmed (valid) Username: Ahmed123 (valid) Password: gkdkgg123 (valid) Email: ahmed@gmail.com (valid) Select gender &birthday	Sign up successfully. (go to home screen)	Sign up successfully. (go to home screen)	pass		

Test Scenario ID	Sign up (server side constrains)	**Test Case ID**	Sign up -1A
Test Case Description	Sign up – negative test case	**Test Priority**	High
Pre-Requisite	A valid user account	**Post-Requisite**	NA

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**
1	Launch application go to sign up screen	Name: ahmed (valid) Username: Ahmed123 (valid) Password: gkdkgg123 (valid) Email: ahmed@gmail.com (used before) Select gender & birthday	Error message (user already exists)	Error message (user already exists)	IE -11	Pass

8.3 Users Feedback

User feedback after testing the application.

We sent the application to a few users and a feedback form to get their responses as well as any future features they may want in the application to get a better understanding of what to add next and what the users actually need.

Rate the idea of application	Rate how did "Quran learner" help you in memorization of Quran.	Rate how much "Quran learner" help you in Recitation of Quran?	Rate Semantic search feature?	Rate your overall experience in application	Features that I Want in the application
9	10	9	10	9	
9	7	10	8	9	Quran duas and social sharing verses
10	9	9	9	8	
10	10	7	9	9	
9	10	9	7	9	
10	9	7	9	10	
10	10	8	9	9	
9	9	9	9	9	
10	8	8	9	9	Reminders
8	10	8	8	9	
9	9	8	9	9	
9	9	8	9	9	
10	9	8	9	9	

Chapter 9: Tools and technologies we used.

In this project, various tools and technologies have been utilized to develop a application with Flutter, train models, and integrate with Hugging Face. Here are the key tools and technologies employed:

- Flutter

- Firebase is a comprehensive suite of cloud-based services provided by google.

Firebase

- Hugging Face

- Wav2vec2 is a state-of-the-art automatic speech recognition (ASR) model.
- Dart is the programming language used in Flutter development.

- GitHub, a widely used version control platform, has likely been employed for source code management and collaboration among the development team.
- spaCy

- Tensorflow
- Python
- PyTorch, a popular deep learning framework, was likely used for training and fine-tuning the Wav2vec model.
- Wav2Vec2 Tools: Specific tools and scripts designed for Wav2vec models were likely used for data preprocessing, feature extraction, training, and evaluation.
- Gensim is a Python library used for topic modeling and natural language processing.
- Word2Vec: The Word2Vec algorithm,
- AraVec model. These tools handle tasks such as tokenization, text normalization, and filtering of unwanted elements

Chapter 10: Conclusion and Future Enhancements

10.1 Conclusion

In conclusion, the development of the application using Flutter has resulted in a transformative platform for Quranic learning and engagement. By leveraging advanced technologies and user-centric design, the application offers a range of features that enhance understanding, recitation, and memorization of the Quran. The integration of Word2Vec, AraVec, and Wav2Vec models empowers users to search for verses based on meaning, listen to accurate recitations with Tashkeel, and embark on a systematic memorization journey. The user-friendly interface and personalization options cater to individual preferences, making the application accessible and adaptable. Overall, this project contributes to the preservation of Islamic heritage and enables individuals from diverse backgrounds to connect with the Quran in a meaningful and profound way.

10.2 Challenges and learned Lessons.

While building an application with Flutter and training models we faced some challenges that lead us to learn some lessons for future work.

Here are some of the challenges and lessons learned.

Models and Training side:

- Data Collection and Preparation: Gathering the necessary data for training the models can be a complex task. It requires us to locate reliable and relevant sources, extract and preprocess the data, and ensure the data is labeled correctly. Proper data preparation is essential for training accurate and effective models.
- Model Training and Fine-Tuning: Training and fine-tuning models, such as Wav2vec2, requires careful consideration of the available hardware resources, training time, and hyperparameter tuning. Experimentation and iteration are crucial to achieving optimal results. Fine-tuning pre-trained models can expedite the training process and improve performance.
- Handling Arabic Language Challenges: Building Quran applications involves specific challenges related to the Arabic language, such as

complex text rendering, right-to-left layout, and proper handling of diacritical marks (Tashkeel).

Application side:

- User Experience and Interface Design: Designing a user-friendly interface that aligns with the needs and expectations of Quranic app users is crucial. Incorporating intuitive navigation, clear instructions, and seamless audio recording and playback features significantly enhanced the overall user experience.
- Testing and Validation: Rigorous testing is essential to identify and address any issues, such as inaccuracies in transcriptions or incorrect string comparisons. Validation through user feedback and continuous improvement helps refine the application's performance and reliability.
- Community Engagement and Support: Engaging with the Quran application user community and actively seeking feedback and suggestions can provide valuable insights. Incorporating user feedback and addressing their needs fosters a stronger and more supportive user community.

10.3 Future Enhancements and Recommendations

We plan to train the finetuned model more on our data that is being stored on the server to improve the model also add some features in application like Quranic duas and social sharing verses.

Also, the Tajweed model has produced good results so far, but it can still be better with more data, and this is why we plan to use the same approach to reuse the data from the application to the training pipelines to ensure that the model is performing as expected.

The last and most crucial enhancement is real time audio processing. While the Application is currently working fine it would be better if we could process the audio from the user in a real time audio processing dedicated server. The only downside would be that renting or building this server would cost a great amount of resources, financial as well as computational.

References and Resources

M. Alshammeri, E. Atwell, and M. A. Alsalka, “Detecting Semantic-based Similarity Between Verses of The Quran with Doc2vec,” ResearchGate, Jan. 01, 2021.

S. Saeed, S. Haider, and Q. Rajput, “On Finding Similar Verses from the Holy Quran using Word Embeddings,” 2020 International Conference on Emerging Trends in Smart Technologies (ICETST), Mar. 2020, Smith, “An approach to graphs of linear forms (Unpublished work style),” unpublished.

H. A. A. A. H. Aliwy, “Automatic Extraction of Semantic Relation among Verses of Quran’s,” 2022.

D. Jatnika, M. A. Bijaksana, and A. A. Suryani, “Word2Vec Model Analysis for Semantic Similarities in English Words,” Procedia Computer Science, vol. 157, Elsevier BV, pp. 160–167, Jan. 01, 2019.

A. -Baquee, M. Sharaf, and E. Atwell, “QurSim: A corpus for evaluation of relatedness in short texts,” ResearchGate, Jan. 2012,

M. Alqahtani and E. Atwell, “Arabic Quranic Search Tool Based on Ontology,” Springer eBooks, pp. 478–485, Jun. 2016.

K. Shaalan, S. Siddiqui, M. Alkhatib, and A.Abdel Monem, “Challenges in Arabic Natural Language Processing,” in Computational Linguistics, Speech and Image Processing for Arabic Language, no. November, WORLD SCIENTIFIC, 2018, pp. 59–83

Y. Goldberg and O. Levy, “Word2Vec Explained: deriving Mikolov et al.’s negativesampling word embedding method,” no. 2, pp. 1–5, Feb. 2014,

A. B. M. Sharaf and E. S. Atwell, “QurSim: A corpus for evaluation of relatedness in short texts,” in Proceedings of the 8th International Conference on Language Resources and Evaluation, LREC 2012, 2012, pp. 2295–2302. doi: 10.13140/2.1.4007.0088.

E. M. B. Nagoudi and D. Schwab, “Semantic Similarity of Arabic Sentences with Word Embeddings,” in Proceedings of the Third Arabic Natural Language Processing Workshop, 2017, pp. 18–24

A. Basharat, D. Yazdansepas, and K.Rasheed, “Comparative study of verse similarity for multi-lingual representations of the Qur’an,” in Proceedings of the 2015 International Conference on Artificial Intelligence, ICAI 2015 - WORLDCOMP 2015,2019, pp. 336–342. [Online]. Available: http://tanzil.net/

E. Atwell and M. Sawalha, “An Artificial Intelligence Approach to Arabic and Islamic Content on the Internet Designing a Model for Automatic Classical

Arabic Annotation Based On Machine Learning Techniques View project Text Mining and Similarity Measures of Quran and Bible View pro," 2011.

WissamAntoun , FadyBaly , Hazem Hajj , "AraBert: Transformer Based Model For Arabic Language Understanding" arxiv:2003.00104

W. Antoun, F. Baly, and H. Hajj, "Aragpt2: Pre-trained transformer for Arabic language generation," ACL Anthology. [Online]. Available: https://aclanthology.org/2021.wanlp-1.21/.

Abu Bakr Soliman, Kareem Eisa, and Samhaa R. El-Beltagy, "AraVec: A set of Arabic Word Embedding Models for use in Arabic NLP", in proceedings of the 3rd International Conference on Arabic Computational Linguistics (ACLing 2017), Dubai, UAE, 2017.

A. Baevski, M. Auli, and A. Mohamed. Effectiveness of self-supervised pre-training for speech recognition. arXiv, abs/1911.03912, 2019.

A. Baevski, S. Schneider, and M. Auli. vq-wav2vec: Self-supervised learning of discrete speech representations. In Proc. of ICLR, 2020. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. arXiv, abs/2002.05709, 2020.

J. Chorowski, R. J. Weiss, S. Bengio, and A. van den Oord. Unsupervised speech representation learning using wavenet autoencoders. arXiv, abs/1901.08810, 2019.

Y. Chung, W. Hsu, H. Tang, and J. R. Glass. An unsupervised autoregressive model for speech representation learning. arXiv, abs/1904.03240, 2019.

C. Cieri, D. Miller, and K. Walker. The Fisher corpus: a resource for the next generations of speech-to-text. In LREC, volume 4, pages 69–71, 2004.

D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In International Joint Conference on Artificial Intelligence, pages 1237–1242, 2011.

D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In Computer Vision and Pattern Recognition, pages 3642–3649, 2012.

A. Coates, B. Carpenter, C. Case, S. Satheesh, B. Suresh, T. Wang, D. J. Wu, and A. Y. Ng. Text detection and character recognition in scene images with unsupervised feature learning. In International Conference on Document Analysis and Recognition, 2011.

A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, and B. Catanzaro. Deep learning with COTS HPC. In International Conference on Machine Learning, 2013.

https://docs.flutter.dev/platform-integration/android/splash-screen

Flutter's fonts and typography

https://docs.flutter.dev/ui/advanced/typography

Layouts in Flutter

https://docs.flutter.dev/ui/layout

Packages

https://api.flutter.dev/javadoc/

Assets, images, and icon widgets

https://docs.flutter.dev/ui/widgets/assets

Flutter YouTube Channel:

https://www.youtube.com/c/flutterdev

Samples

https://flutter.github.io/samples/#