MOVIE RECOMMENDATION VOICE BOT

A MINI PROJECT REPORT

Submitted by:

AHANA SADH
202100483

in partial fulfillment in the award of the degree

of

BACHELOR IN TECHNOLOGY
in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SMIT SIKKIM MANIPAL UNIVERSITY
SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY

SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY, SIKKIM MANIPAL UNIVERSITY - 737136

APRIL, 2024

BONAFIDE CERTIFICATE

Certified that this project report "**MOVIE RECOMMENDATION VOICE BOT**." is the bonafide work of "**AHANA SADH**" who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Mr. Sital Sharma
SUPERVISOR
Assistant Professor
B.Tech AI&DS
SMIT, SMU

Prof. (Dr.) Om Prakash Singh
HEAD OF DEPARTMENT
B.Tech AI&DS
SMIT, SMU

Submitted for Semester Mini-Project examination held on 27 April 2024

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

ABSTRACT

The proposed work explores the expanding significance of voice assistance chat interfaces for answering user inquiries, especially in light of their growing popularity for general knowledge and recommendation systems. It emphasizes the use of cutting-edge deep neural network methods like NCF, classification models like ScANN, and language models like BERT (LaBSE).

The goal of using these state-of-the-art models is to improve user experience by offering precise and pertinent suggestions that are customized for their queries. For example, BERT (LaBSE) makes it possible to comprehend the semantic meaning of user queries in a variety of languages, and ScANN makes it possible to efficiently find the approximate nearest neighbor when making similarity-based suggestions. Furthermore, individualized recommendations based on user preferences and previous interactions are facilitated by NCF model. In practice, the suggested model is implemented within a Google Text to Speech (gTTS) and WhisperAI voice assistant framework. The integrated model interprets user queries, produces recommendations, and provides them to users through synthesized speech output from gTTS when they engage with the voice assistant.

The study does, however, also recognize some of these models' drawbacks, including their computational complexity, scaling problems, and potential for bias in recommendation results. The paper suggests methods for improving the recommendation model's hyperparameters, adding user feedback channels, and reducing biases in the data and algorithms in order to address these issues.

In general, the proposed system seeks to apply cutting-edge AI methods to give consumers real-time, personalized, and contextually relevant recommendations, thereby enhancing their overall experience with voice assistance chat interfaces.

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Voice-activated chabots have proven to be more successful in the last few years and have emerged as the primary devices for improving human-computer interaction. It allows the users to communicate with digital systems quickly and efficiently through guidance. The purpose of this paper is to assess for what role voice-assisted dialogue is acceptable for users when providing feedback. In particular, whether it is prudent to rely on such facilities in network systems with a large number of recommendation processes and dialogues with minimal significance.

Due to the development of NLP technology and devices with voice support, such an interface has become more popular and is now actively used in a variety of industries, including Information retrieval, during formal work, and entertainment, such as because of its ability to simplify and signal free speech.

This project aims to explore and evaluate voice help chat interfaces used to interact with advanced AI models in recommendation systems. It focuses on the following:

- Voice chat interface, data preprocessing and deep learning model training
- Development of AI model creation and its implementation in a voice assistant system to establish the basis of real-time interaction with the model via text-to-speech.

The completion of this project creates possibilities for implementing further research directions and potential enhancements in voice-operated recommendation systems.

In summary, this project aims to provide insights into the design, implementation, and evaluation of voice assistance chat interfaces integrated with advanced AI models for recommendation systems. Through empirical analysis and user-centric evaluation, the project seeks to contribute to the advancement of voice-enabled technologies and their applications in enhancing user experience.

2. LITERATURE SURVEY

S.No	Author(s)	Paper and Publication Details	Findings	Relevance to the Project
1.	Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhaga, Wei Wang	Language-agnostic BERT Sentence Embedding	This paper explores using pretraining language models in combination with the best of existing methods for learning cross-lingual sentence embeddings.	Importance of sentence embedding and the implementation of the BERT based LaBSE algorithm.
2.	Amine Abdaoui, Camille Pradel, Gregoire Sigel	Load What You Need: Smaller Versions of Multilingual BERT	This paper proposes to generate smaller models that handle fewer number of languages according to the targeted corpora. It presents an evaluation of smaller versions of multilingual BERT on the XNLI data set, which may be applied to other multilingual transformers	Reduce parameter size in LaBSE since it is a very computationally expensive task
3.	Everlyn Asiko Chimoto, Bruce A. Bassett	Very Low Resource Sentence Alignment: Luhya and Swahili	This work explores the test results of using LASER and LABSE to investigate how they would perform on an unseen low-resource language: Luhya, Marama dialect, and Swahili	This paper lays down a very systematic difference between LaBSE and LASER to help choose the right algorithm.
4.	Jiang Zhang, Yufeng Wang, Zhiyuan Yuan, and Qun Jin	Personalized Real-Time Movie Recommendation System: Practical Prototype and Evaluation	It explores the traditional CF methods, the time complexity problem and proposes a solution which exploits users' profile attributes to partition them into several clusters	Collaborative Filtering, it's implementation and limitations

LITERATURE SURVEY

S.No	Author(s)	Paper and Publication Details	Findings	Relevance to the Project
5.	Marius Muja, David G. Lowe	Scalable Nearest Neighbor Algorithms for High Dimensional Data	This paper evaluates the most promising nearest neighbor search algorithms in the literature, propose new algorithms and improvements to existing ones, present a method for performing automatic algorithm selection and parameter optimization.	Selecting the right algorithm to find the nearest neighbours for recommendation system
6.	Xiangnan He, Lizi Liao, Hanwang Zhang, LiqiangNie, Xia Hu, Tat-Seng Chua	Neural Collaborative Filtering	This paper explores the use of deep neural networks for learning the interaction function from data, rather than a handcraft that has been done by many previous work	Limitations of using traditional CF and it's DNN alternative
7.	Holger Schwenk, Matthijs Douze	Learning Joint Multilingual Sentence Representations with Neural Machine Translation	In this paper it has been shown that the framework of NMT with multiple encoders/decoders can be used to learn joint fixed-size sentence representations which exhibit interesting linguistic characteristics.	-
8.	Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, Ilya Sutskever	Robust Speech Recognition via Large-Scale Weak Supervision	This paper studies the capabilities of speech processing systems trained simply to predict large amounts of transcripts of audio on the internet.	Usage of WhisperAI for Speech to Text processing

3. PROBLEM DEFINITION

Movie recommendations systems nowadays can respectively take users to the part where they can learn more through analyzing their reviews, likings, and watching history. In the face of the fact that the current recommendations on systems are less precise and can be called personalized collection of tastes, they still perform their best. Hence, the standardized recommendations which are not personalized enough to blend with the users' style of preference may get provided.

To address this challenge, this project proposes the development of a novel solution: I propose that a chatbot is based on conversational AI, voice-enabled, and backed up with advanced algorithm and analysis of data. The solution, for the first time, is not totally based on the previous ratings and reviews of users, but it utilizes a linguistic capability of a chatbot to get context-rich information by means of natural language that deals with the personal choices, interests and viewing habits of the audience.

This project is going to be directed on how to reach out a goal to develop a real-time recommendation system capable of analyzing and interpreting user preferences. The system will rely on innovative algorithms including machine learning models & data analytics tools such as cooperative filtering techniques to acquire these subtle and personalized user preferences and then use the information gathered to personalize recommendations that will match up perfectly to a consumer's wants.

4. SOLUTION STRATEGY

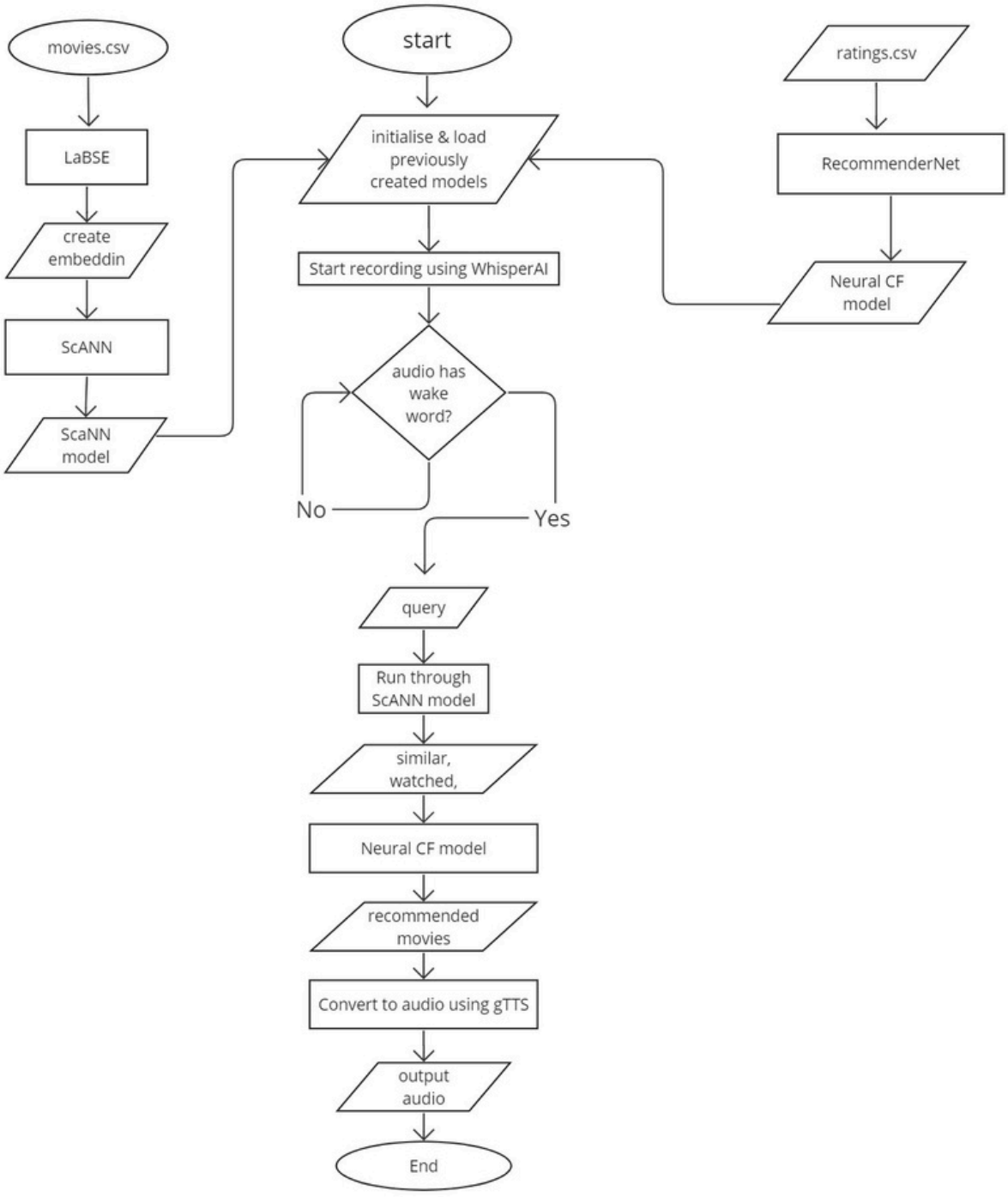

Fig 1: Complete overview flowchart for solution strategy for movie recommendation system. This includes the algorithms used, and the flow of the model.

SOLUTION STRATEGY

The solution strategy can be broken down into four major stages. A detailed explanation for each stage, the algorithms used and model implementation will be presented later in the report. These stages are as follows:

Stage 1: Speech to Text Conversion

Input: User audio recorded on command through microphone.
Audio starts being recorded for 7 seconds when wake word is detected. The input audio is passed through the Whisper library by OpenAI which then transcribes the speech to text
Output: String of text resulting from user audio
WhisperAI: Whisper is an automatic speech recognition (ASR) system trained on 680,000 hours of multilingual and multitask supervised data collected from the web

Stage 2: Create embeddings using LaBSE
LaBSE is a Language-Agnostic BERT Sentence Embedding model, developed by Google AI. It learns to encode the semantic meaning of sentences into fixed-size vectors, known as sentence embeddings. BERT's bidirectional context modeling and self-attention mechanism enable LaBSE to encode the semantic meaning of sentences by considering the context of each word within the sentence and its interactions with other words.

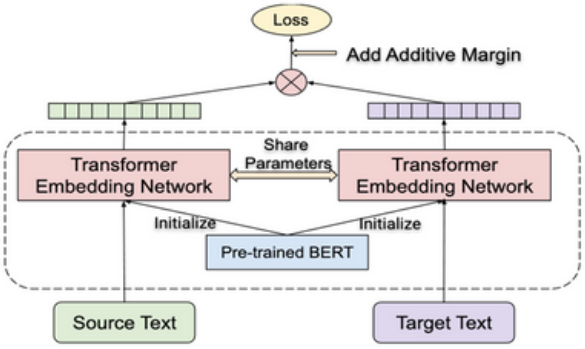

Figure 2: Dual encoder model with BERT based encoding modules.

Stage 3: Approximate Nearest Neighbor Candidate Generation
The SCANN (Scalable Nearest Neighbors) algorithm is an approximate nearest neighbor search algorithm designed to efficiently find the nearest neighbors of a query point in high-dimensional spaces. Overview of the working of the ScANN algorithm:

- Hashing: The SCANN algorithm partitions the data points (represented as embeddings) into multiple hash buckets using locality-sensitive hashing (LSH) techniques.
- Indexing: Once the data points are hashed into buckets, an index structure is built to organize and store the hashed data.
- Search: When a query point is provided, SCANN first hashes the query into the same hash buckets as the data points. It then retrieves candidate neighbors from the hash buckets based on their proximity to the query. Finally, it performs a more refined search among the candidate neighbors to find the nearest neighbors of the query point.
- ANN Search: SCANN provides an approximation of the nearest neighbors rather than exact neighbors. The algorithm trades off accuracy for efficiency, enabling fast retrieval of approximate nearest neighbors in high-dimensional spaces.

SOLUTION STRATEGY

Figure 3: Working steps of ScANN algorithm

Parameters specified during instantiation are leaves (partitions) and the value of K (no. of nearest neighbours). The index item is created by partitioning the embedding space and storing the embeddings of the input items in the corresponding partitions. To perform a search, the algorithm identifies the partitions (leaves) that are most likely to contain the nearest neighbors of the query embedding based on their similarity to the query.

Stage 4: Neural Collaborative Filtering
In collaborative filtering, we find similar users and recommend what similar users like. Generally, CF uses the concept of Matrix Factorization to recognise similarity but due to the simple fact that basic inner product is not sufficient to capture the relationships between the data. Proposed by Xiangnan He et. al in the paper Neural Collaborative Filtering, it is discussed how implementing deep neural networks in place of matrix factorization due to the high non-linearity it brings by stacking several non-linear layers.

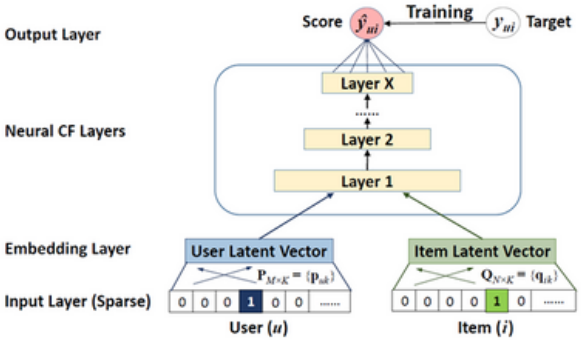

Figure 4: Neural Collaborative Filtering framework

Stage 5: Text-to-Speech using gTTS
The output received from NCF will be in the form of a list of movies deemed best for the user. This output is then relayed back to the user using gTTS library present in Python. It is a CLI tool to interface with Google Translate's text-to-speech API.

5. MODEL IMPLEMENTATION

This project is divided into to main sections - a movie recommendation system and a voice assistant. The voice assistant consists of a Speech to Text model made using WhisperAI by Google, and a Text to Speech model made using Google TTS. The Movie Recommendation system is uses the following algorithms - LaBSE (BERT), ScANN & NCF. The comparison between the algorithms used with their best fit alternatives will also be discussed further in this report.

5.1 WhisperAI

Whisper is a general-purpose speech recognition model. It is trained on a large dataset of diverse audio and is also a multitasking model that can perform multilingual speech recognition, speech translation, and language identification.

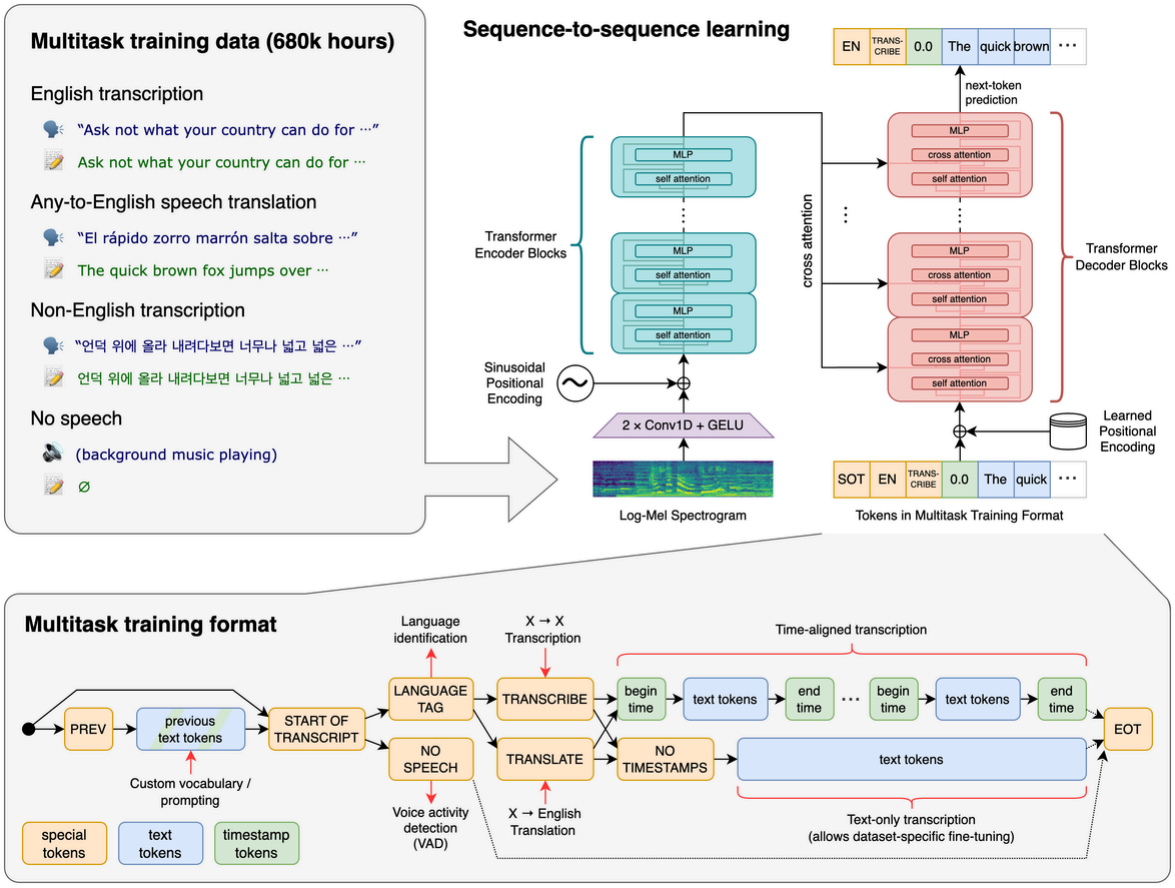

Figure 5: Overview of the working of WhisperAI approach

A Transformer sequence-to-sequence model is trained on various speech processing tasks, including multilingual speech recognition, speech translation, spoken language identification, and voice activity detection. These tasks are jointly represented as a sequence of tokens to be predicted by the decoder, allowing a single model to replace many stages of a traditional speech-processing pipeline. The multitask training format uses a set of special tokens that serve as task specifiers or classification targets.

The source code for the implementation of WhisperAI in this project is given in Appendix 1

5.2 Google TTS

TTS serves as one of the primary engines behind gTTS, which refers to a platform that utilizes a complex conversion process to deliver a natural-sounding spoken language output. First, gTTS deals with the text processing, fetching and normalizing it (tokenize and normalize) and then the phonetic features extraction is performed. By implementing a modern and phonetic analysis algorithms, the API will translate words into their phonetic equivalent, and the resulting words will be represented according to the transcription. Following, gTTS looks for the right phonemes from the enormous inventory of pre-recorded speech units that are in the database, biasing factors, like prosody and coarticulation, also in account. Concatenative synthesis takes the selected phonemes and concatenates (string together end-to-end) them together until the ultimate words and the sentence is produced, which makes it sound natural. Such process makes the synthetic voice as close to a real human speech as possible, leading to a final audible outcome of a high quality and intelligible sound for the end-users or applications. Through the partial implementation of deep learning technologies in Natural Language Processing, phonetics, and speech synthesis, gTTS is able to deliver a high-level, user-friendly and reliable solution for text-to-speech conversation.

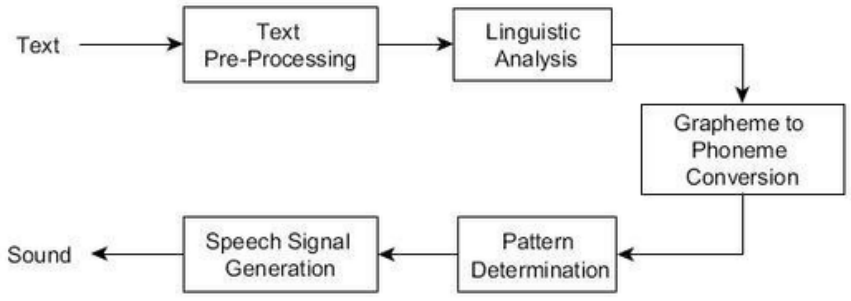

Figure 6: Working overview of gTTS

5.3: Language Agnostic BERT Sentence Embeddings

While BERT is an effective method for learning monolingual sentence embeddings for semantic similarity and embedding based transfer learning, LaBSE makes use of dual encoder models, which have been demonstrated as an effective approach for learning bilingual sentence embeddings. The dual encoder approach represents a powerful technique. Bilingual embeddings techniques must be handled to solve the problem. Such models consist of a two-headed sequence model which takes into consideration one of the side score function. The tags as well as the start and end characters are omitted. Sentence embeddings are extracted from each encoder. Cross-lingual embeddings are calculated with the inbatch constraint by using a ranking framework for translation task training:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{\phi(x_i,y_i)}}{e^{\phi(x_i,y_i)} + \sum_{n=1,n\neq i}^{N} e^{\phi(x_i,y_n)}}$$

Additive margin softmax extends the scoring function φ by introducing margin m around positive pairs:

$$\phi'(x_i, y_j) = \begin{cases} \phi(x_i, y_j) - m & \text{if } i = j \\ \phi(x_i, y_j) & \text{if } i \neq j \end{cases}$$

The margin m improves the separation between translations and nearby non-translations. Using φ(xi, yj) with the bidirectional loss Ls, we obtain the additive margin loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \frac{e^{\phi(x_i,y_i)-m}}{e^{\phi(x_i,y_i)-m} + \sum_{n=1,n\neq i}^{N} e^{\phi(x_i,y_n)}}$$

5.3.1 Comparison between LaBSE and LASER

A study conducted to test extracting bitext for two related low-resource African languages: Luhya and Swahili. The performance of LaBSE showed that there are great gains achieved by utilising a pre-trained model in the sentence embedding model. LaBSE model utilises BERT in its training offering crosslingual benefit that results in up to 22% accurate alignment on a language it has not seen before. LASER on the other hand was trained from scratch and does not provide great results in aligning Luhya, an unseen language.

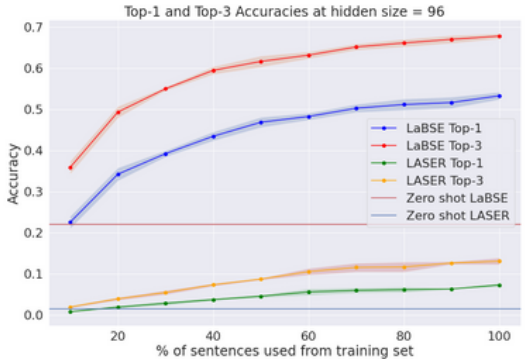

Figure 7: LaBSE outperforms LASER in both the Top- 1 and Top-3 results from fine-tuning the respective embeddings (with a hidden layer dimension of 96) for Luhya.

5.4: Scalable Artificial Nearest Neighbours

Google's ScaNN (Scalable Nearest Neighbors) was proposed and released by Google Research in a paper titled **ScaNN: Fast and Effective Mapping with AI on Large-Scale Storage**. Since the introduction of ScaNN, it has become the most widely used library on Google for scalable nearest neighbor search customization. ScaNN is being used in various Google services and products such as Google Photos and YouTube.

Here is an overview of how ScaNN works:

- Data preparation: The first step is to polish the dataset by converting each data item into high-dimensional vector. ScaNN for Various Vector representation, is presented as bag-of-words, embeddings, and feature vectors.
- Indexing: ScaNN uses LSH technique of the index construction by analysing the dataset and by contrast it has high possibility to group the similar vectors into the same bucket. The LSH index provides a fast way to identify the buckets which mostly have the nearest neighbor point locations.
- Refinement: ScaNN then locates the best buckets with the LSH and with the help of the transportation topic it improves the initial results. Optimal transport is an overall winner method recently developed which measures the distance between two probability distributions effectively and thus makes ScaNN accurately & efficiently distance the query vector from all the vectors in the bucket.
- Ranking: ScaNN comes next and uses the distance as the criteria on which to rank the candidate vectors, by which it returns the top k nearest neighbors.

ScaNN also has the ability to support different kind of distances, among these are L2 (Euclidean) distance, cosine distance, and inner product similarity.

5.4.1 Comparison between ScaNN and FAISS

Parameter	ScANN	FAISS
Architecture	two-level hierarchical approach where a smaller index is built on top of a larger index	inverted index with coarse quantizers and product quantization.
Query Speed	Faster in single-query search	Faster in multi-query search
Ease of Use	Easier to use and requires fewer hyperparameters to be tuned	Steeper learning curve and requires more manual tuning.

Parameter	ScANN	FAISS
Language support	ScaNN is written in C++ and has bindings for Python and TensorFlow.	FAISS is written in C++ with Python bindings
Algorithm support	ScaNN supports only approximate nearest neighbor (ANN) search.	FAISS supports a wider range of algorithms, including hierarchical Navigable Small World (HNSW) graphs
Memory usage	Uses more memory than FAISS	FAISS uses less memory than ScaNN

Table 1: Comparison between Google's ScaNN and Facebook's FAISS

5.5 Neural Collaborative Filtering

Two fundamental issues in real movie recommendation systems are often neglected: scalability and practical usage feedback/verification based on real implementation. Collaborative Filtering (CF) is one of the most widely-used algorithms for making rating predictions within an RS. It is based on the core assumption that users who have expressed similar interests in the past will share common interests in the future. Therefore, the idea of collaborative filtering is to identify users in a community that share appreciation for similar things. As the number of users and items grows, CF-based recommendation systems need more resources to process information and form recommendations. The majority of these resources are consumed in determining users with similar tastes and items with similar descriptions. Therefore, CF algorithms face a scalability problem, which can become an important factor for a recommendation system.

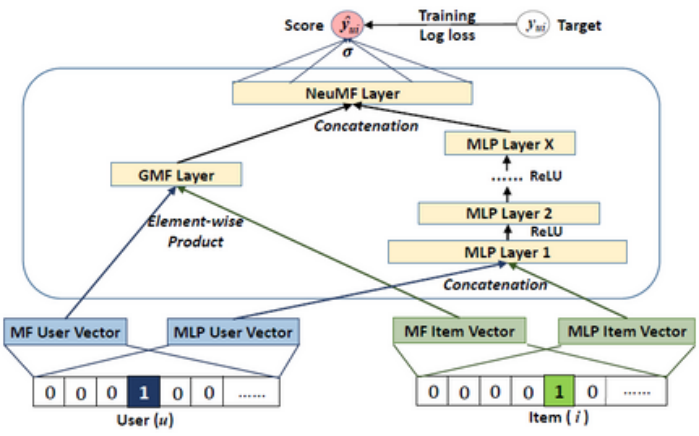

Figure 8: Neural Matrix Factorization Model

Neural Matrix Factorization is an extension of traditional matrix factorization methods like CF, incorporating neural networks to model complex user-item interactions. NMF enhances recommendation performance by learning latent representations of users and items through neural networks, capturing intricate patterns in user-item interactions for more accurate recommendations.

6. OUTPUT & RESULTS

The output of the project represents a culmination of efforts to create a comprehensive and highly functional voice-enabled chatbot solution tailored specifically for personalized movie recommendations. After the integration platform is put within the perfection of the engineering process, involve of the most state of the art algorithms and up-to-date data analysis tools, chatbot is able to give an easy-to-use and enjoyable customer experience. The system has a state-of-the-art ensemble for machine learning that uses BERT-LaBSE for the accurate understanding of the query semantics and the NCF algorithm for the recommendation of items. The chatbot uses advanced intelligent models that help to understand movie choices and viewing habit of each individual at very highest level of accuracy that enable great recommendation delivery thus developing a friendly and bonding relation between users and the chatbot.

Moreover, the bot does not just generate suggestions but has a vehicle for contextual analysis and user feedback incorporated in its decision-making. This follows that the system will be able to incorporate and perfect its recommendations whereas different users in different cases are using it over time and hence will find them appropriate and interesting in every scenario. In conclusion, this project outcome operates as a pioneer work in the sphere of personalized recommendation systems by offering the users with an interactive and enriching experience of the movie discovery going in line with the convenience and high accuracy requirements.

7. DATASET DESCRIPTION

This project uses the MovieLens 25M dataset. It contains 25000095 ratings and 1093360 tag applications across 62423 movies. These data were created by 162541 users selected at random for inclusion. All selected users had rated at least 20 movies

ratings.csv : has the following headers – userID, movieID, rating, timestamp **.**Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

tags.csv: Tags are user-generated metadata about movies. Each tag is typically a single word or short phrase. The meaning, value, and purpose of a particular tag is determined by each user.

movies.csv: has the following headers- movieID, title, genre. Movie titles are entered manually or imported from https://www.themoviedb.org/, and include the year of release in parentheses. Genres are a pipe-separated list, and are selected from the following: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance. Sci-Fi, Thriller, War, Western.

genome-scores.csv: The tag genome is a data structure that contains tag relevance scores for movies. The structure is a dense matrix: each movie in the genome has a value for *every* tag in the genome. The tag genome encodes how strongly movies exhibit particular properties represented by tags

8. LIMITATIONS AND FUTURE SCOPE

8.1 Limitations

- **Computational resources:** Takes large amounts of computational power and a considerable amount of time to train and implement.
- **Dataset**: Lack of an updated and extensive dataset for movie recommendations and incase of speech recognition, the lack of enough data to train a speech model from scratch
- **Cold Start:** When using NCF, the new users face a cold start due to lack of any previous information of their account to base future predictions on.

8.2 Future Scope

- **Exploring IMDB Datasets:** IMDB has a large collection of non-commercial datasets that are regularly updated. However, these are segregated based on different criteria and must be combined to be used.

- **Integration with RaspberryPi:** This model can be integrated with microprocessors to make a voice assistance device

- **Conversational Abilities:** Enabling the voice-bot to carry out a conversation with the user using Deep-Q networks and Transformers Models.

9. GANTT CHART

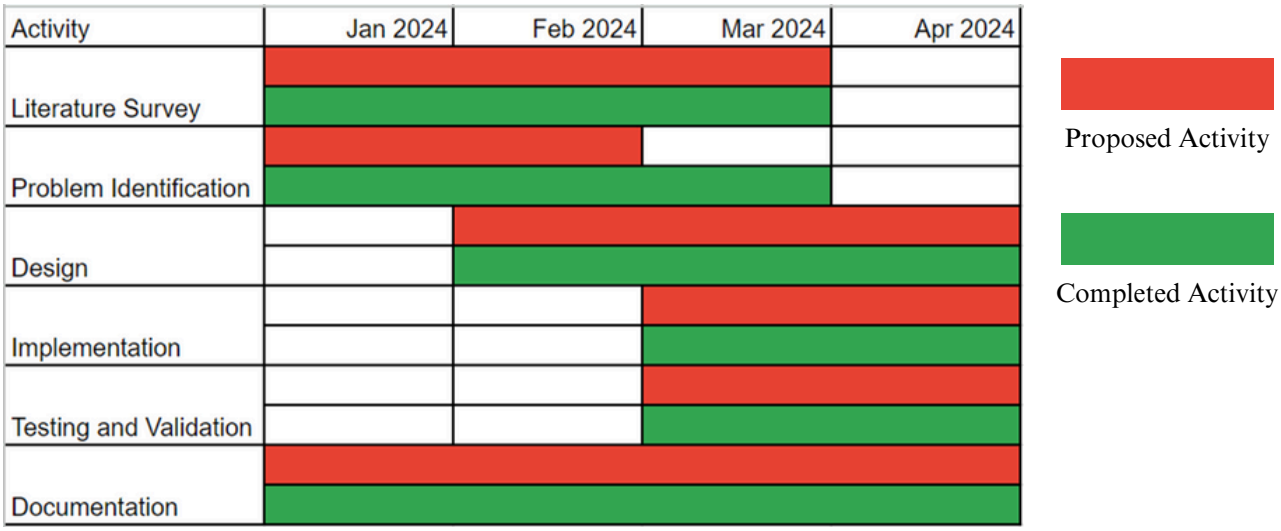

Figure 9: Gantt Chart

APPENDIX - 1

```python
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers

EMBEDDING_SIZE = 128

@keras.utils.register_keras_serializable(package="MyLayers")
class RecommenderNet(keras.Model):
    def __init__(self, **kwargs):
        super(RecommenderNet, self).__init__(**kwargs)
        num_users=200000
        num_movies=200000
        embedding_size=128
        self.num_users = num_users
        self.num_movies = num_movies
        self.embedding_size = embedding_size
        self.user_embedding = layers.Embedding(
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.movie_embedding = layers.Embedding(
            num_movies,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.movie_bias = layers.Embedding(num_movies, 1)

    def call(self, inputs):
        user_vector = self.user_embedding(inputs[:, 0])
        user_bias = self.user_bias(inputs[:, 0])
        movie_vector = self.movie_embedding(inputs[:, 1])
        movie_bias = self.movie_bias(inputs[:, 1])
        dot_user_movie = tf.tensordot(user_vector, movie_vector, 2)
        # Add all the components (including bias)
        x = dot_user_movie + user_bias + movie_bias
        # The sigmoid activation forces the rating to between 0 and 1
        return tf.nn.sigmoid(x)
```

Figure 10: Source code for RecommenderNet used in NCF

```
import pandas as pd
import matplotlib.pyplot as plt
from transformers import AutoTokenizer, AutoModel
import torch
import numpy as np
import tensorflow_recommenders as tfrs
import tensorflow as tf
from tensorflow import keras

class PersonalisedSearcher():
    def __init__(self):
        self.recommender = RecommenderNet()
        self.recommender.load_weights('/content/Ahana/CF_Final.keras')
        self.movies = pd.read_csv("/content/ml-25m/ml-25m/ml-25m/movies.csv")
        self.ratings = pd.read_csv("/content/ml-25m/ml-25m/ml-25m/ratings.csv")
        self.embeddings = pd.read_csv("/content/Ahana/data.csv", index_col=0)
        self.item_tensor = tf.convert_to_tensor(self.embeddings, dtype=tf.float32)
        self.scann = tfrs.layers.factorized_top_k.ScaNN(num_leaves=1000, num_leaves_to_search = 100,
                                                        k=round(np.sqrt(len(self.item_tensor))))
        self.scann.index(self.item_tensor)
        self.model = AutoModel.from_pretrained("./Labse")
        self.tokenizer = AutoTokenizer.from_pretrained("./Labse")

    def get_user_encodings(self):
        user_ids = self.ratings["userId"].unique().tolist()
        user2user_encoded = {x: i for i, x in enumerate(user_ids)}
        userencoded2user = {i: x for i, x in enumerate(user_ids)}

    def get_movie_encodings(self):
        movie_ids = self.ratings["movieId"].unique().tolist()
        movie2movie_encoded = {x: i for i, x in enumerate(movie_ids)}
        movie_encoded2movie = {i: x for i, x in enumerate(movie_ids)}

        return movie2movie_encoded, movie_encoded2movie

    def update_ratings(self):
        user2user_encoded, _ = self.get_user_encodings()
        movie2movie_encoded, _ = self.get_movie_encodings()
        self.ratings["user"] = self.ratings["userId"].map(user2user_encoded)
        self.ratings["movie"] = self.ratings["movieId"].map(movie2movie_encoded)

        return self.ratings

    def get_user_history(self, user_id):
        df = self.update_ratings()
        watched_movies = df[df.userId == user_id]

    def get_candidate_movies(self, query):
        encoded_input = self.tokenizer(query,
                                       padding=True,
                                       truncation=True,
                                       max_length=64,
                                       return_tensors='pt')
        with torch.no_grad():
            model_output = self.model(**encoded_input)
        query_embeddings = model_output.pooler_output
        query_embeddings = torch.nn.functional.normalize(query_embeddings)
        test_case = self.scann(np.array(query_embeddings))
        return self.movies.iloc[test_case[1].numpy()[0]][0:11]
```

Figure 11: Source code for PersonalisedSearcher( ) which is the function performing the movie recommendation. This is inclusive of all algorithms previously discussed

```
def filter_candidates(self, user_id, query):
    movies_watched_by_user = self.ratings[self.ratings.userId == user_id]
    candidates = self.get_candidate_movies(query)
    movies_not_watched = candidates[
        ~candidates["movieId"].isin(movies_watched_by_user.movieId.values)
    ]["movieId"]
    movie2movie_encoded, _ = self.get_movie_encodings()
    movies_not_watched = list(set(movies_not_watched).
                              intersection(set(movie2movie_encoded.keys())))
    movies_not_watched = [[movie2movie_encoded.get(x)] for x in movies_not_watched]
    user2user_encoded, _ = self.get_user_encodings()
    user_encoder = user2user_encoded.get(user_id)
    movie_array = np.hstack(([[user_encoder]] * len(movies_not_watched), movies_not_watched))

    return movie_array, movies_not_watched, movies_watched_by_user

def personalised_search(self, user_id, query):
    movie_array, movies_not_watched, movies_watched_by_user = self.filter_candidates(user_id, query)
    scored_items = self.recommender.predict(movie_array).flatten()
    top_rated = scored_items.argsort()[-10:][::-1]
    _, movie_encoded2movie = self.get_movie_encodings()
    recommended_movie_ids = [movie_encoded2movie.get(movies_not_watched[x][0]) for x in top_rated]

    return recommended_movie_ids, movies_watched_by_user

def print_recs(self, user_id, query):
    recommendations, movies_watched_by_user = self.personalised_search(user_id, query)

    print("Showing Top movie recommendations for user:")
    print("====" * 9)
    # print("Movies with high ratings from user")
    # print("----" * 8)
    # top_movies_user = (
    #     movies_watched_by_user.sort_values(by="rating", ascending=False)
    #     .head(5)
    #     .movieId.values
    # )
    # movie_df_rows = self.movies[self.movies["movieId"].isin(top_movies_user)]
    # for row in movie_df_rows.itertuples():
    #     print(row.title, ":", row.genres)
    # print("----" * 8)
    # print("Top movie recommendations")
    # print("----" * 8)
    recommended_movies = self.movies[self.movies["movieId"].isin(recommendations)]
    for row in recommended_movies.itertuples():
        print(row.title, "\t:\t", row.genres)
    return recommended_movies.itertuples()
```

Figure 12: Source code for PersonalisedSearcher( ) which is the function performing the movie recommendation. This is inclusive of all algorithms previously discussed

```
RECORD = """
const sleep  = time => new Promise(resolve => setTimeout(resolve, time))
const b2text = blob => new Promise(resolve => {
  const reader = new FileReader()
  reader.onloadend = e => resolve(e.srcElement.result)
  reader.readAsDataURL(blob)
})
var record = time => new Promise(async resolve => {
  stream = await navigator.mediaDevices.getUserMedia({ audio: true })
  recorder = new MediaRecorder(stream)
  chunks = []
  recorder.ondataavailable = e => chunks.push(e.data)
  recorder.start()
  await sleep(time)
  recorder.onstop = async ()=>{
    blob = new Blob(chunks)
    text = await b2text(blob)
    resolve(text)
  }
  recorder.stop()
})
"""
```

Figure 13: JavaScript program to record audio in Google Colab platform

```
def record(sec=7):
  display(Javascript(RECORD))
  s = output.eval_js('record(%d)' % (sec*1000))
  b = b64decode(s.split(',')[1])
  with open('audio.wav','wb') as f:
    f.write(b)
  return 'audio.wav'

def transcribe(audio,model):
  result = model.transcribe(audio,language="English")
  return result['text'].lower()

def preprocess_query(query,wakeword):
    query = query.lower()
    query=query.replace(wakeword,"")
    query = query.translate(str.maketrans("", "", string.punctuation))
    tokens = nltk.word_tokenize(query)
    stop_words = set(stopwords.words("english"))
    tokens = [token for token in tokens if token not in stop_words]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    query = " ".join(tokens)
    return query

def textToSpeech(text):
    tts = gtts.gTTS(text=text, lang='en')
    name=str(uuid.uuid1())+'.wav'
    tts.save(f"./audios/{name}")
    display(Audio(f"./audios/{name}", autoplay=True))
    time.sleep(librosa.get_duration(filename=f"./audios/{name}"))
```

Figure 14: Python program for speech recognition

```
while True:
  print(f"\n<<Listening for wake word 'hello'>>\n")
  time.sleep(1)
  audio=record(record_time)
  text=transcribe(audio,model)
  if text.count(wakeword) > 0:
      display(Audio("./Ahana/hello.wav", autoplay=True))
      time.sleep(4)
      print(f"<<Now Listening for Query for {query_record_time} seconds>>")
      audio=record(query_record_time)
      print("Now Transcribing")
      text=transcribe(audio,model)
      print("\n"+text+"\n")
      print("Now Preprocessing")
      query=preprocess_query(text,"")
      textToSpeech("Finding your movies. Please be patient")
      recommendations=searcher.print_recs(random.randrange(20, 50, 3),query)
      rec_text="Here are some movies you might like."
      for movie in recommendations:
        rec_text+=" ".join(movie.title.split(" ")[:-1])+". "
      print("\n\n")
      textToSpeech(rec_text)
  else:
      print("Wakeword Not Detected")
```

Figure 15: Python program for speech recording and transcribing

These are the codes to run the main application. All additional codes for training the NLP models are attached in the folder **Final_Mini_Project.ipynb**

REFERENCES

[1] Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, Wei Wang (2020) 'Language-agnostic BERT Sentence Embedding' https://arxiv.org/abs/2007.01852v2

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova (2018) 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding' 'https://arxiv.org/abs/1810.04805'

[3] Everlyn Asiko Chimoto (2022) 'Very Low Resource Sentence Alignment: Luhya and Swahili' https://aclanthology.org/2022.loresmt-1.1.pdf

[4] Amine Abdaoui, Camille Pradel, Gregoire Sigel (2020) 'Load What You Need: Smaller Versions of Multilingual BERT' https://arxiv.org/abs/2010.05609

[5] Jiang Zhang, Yufeng Wang, Zhiyuan Yuan, and Qun Jin (2023) 'Personalized Real-Time Movie Recommendation System: Practical Prototype and Evaluation' https://arxiv.org/abs/2010.05609

[6] M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 11, pp. 2227-2240, 1 Nov. 2014

[7] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, Tat-Seng Chua, "Neural Collaborative Filtering" https://arxiv.org/abs/1708.05031

[8] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, Ilya Sutskever "Robust Speech Recognition via Large-Scale Weak Supervision" https://arxiv.org/abs/2212.04356