Design of AI-enabled Chatbot

A Major Qualifying Project Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In Partial Fulfillment of the Requirements for the

Degree in Bachelor of Science

By:

Rolando Salamea-Lopez

QiHan He

Federico Perez

Collin Mettler

Quincy Payne

Xiao Xiao

Hrishikesh Nair

Guillermo Morel Mendez

Date: *April* 16th 2024

WPI Advisor: Professor Bo Tang

Professor Lin Cheng

Abstract

This project is about the design and implementation of a robotic autochat system which will enhance human-robot interaction through natural language understanding and response capabilities. The system integrates several audio and video sensors into a comprehensive framework, making use of cutting-edge AI-enabled chatbot systems, especially ChatGPT. The embedded system processes the audio and video signals, enabling the robot to understand human speech and engage in conversations. And the monitor displays the responses of the system, creating a seamless interaction experience.

One of the key advantages of the system is that the integration of language models enables the robot to understand and respond effectively to human language, resulting in a more natural and engaging interaction. The project includes multiple domains, such as embedded system design, text-speech translation, speech-text translation, and human-robot interaction. This project provides an in-depth study of cutting-edge technology, and the robotic autochat system presented as a novel idea for improving communication between people and robots, which indicates that the robotic autochat system is an innovative solution for fostering the communication between humans and robots.

Acknowledgements

We want to express our thanks to all the people who contributed to this project. First of all, we want to express our thanks to Professor Bo Tang. He is giving us support and sharing invaluable experience and knowledge in the course of project implementation all the time. His insights and mentorship are important in shaping our approach and guiding us towards achieving our goals.

We also want to express our thanks for the resources, facilities, and support required for us to carry out it to the WPI. This opportunity is also taken to put on record the appreciation with due respect to the contribution made by fellow colleagues, peers, for their valuable suggestions, and encouragement added value towards the development of the AI-enabled system.

We would also like to thank the contributors of open-source tools, libraries, and datasets, without which it would be hard to imagine the completion of this project. Their works in robotics and artificial intelligence have been very useful in making our work possible. This project would not be possible without the collective efforts and support of all those mentioned above.

Table of Contents

1 Introduction

In recent years, robotics has made significant advances, especially in the area of human-robot interaction. Human speech understanding and effective communication; one of the points is to work on enhancing the user experience and engaging the use of robotics in the possible domains. The latter includes customer service and healthcare. This project is devoted to developing intelligent conversational agents for sensible dialogs between a robot and a human.

This report introduces a novel approach to designing a robotic autochat system that integrates cutting-edge technologies to facilitate natural language understanding and response. The use of current AI-enabled chatbot systems is the foundation for the conversational ability of the robot — ChatGPT. By using the power of this advanced language model, the system can bridge the gap between human language and robotic interaction, enabling seamless communication between the two entities.

The key components of the robotic auto chat system for this project include multiple audio and video sensors, an embedded system, and a monitor for displaying results. These elements work together to capture and interpret human speech and gestures, facilitating real-time interaction between the robot and the user.

1.1 Overview of Human-Robot Interaction (HRI)

Human-Robot Interaction (HRI) belongs to an interdisciplinary area aimed at studying the interactions that might appear between human beings and robots; it has gained popularity in very versatile research and design activity. Over time, HRI has gained more attention than before due to the growing numbers of robots in all kinds of domains, from industrial and professional settings to everyday environments at homes, in hospitals, and public spaces. Understanding the ways in which people and robots can communicate and work together will help develop the integration of robotic technologies into society. "From 2006 on, the symposium on human–robotic interaction has been sponsored by the IEEE (IEEE Robotics and Automation Society, 2015). Goodrich and Schultz (2007) offer a useful, if slightly dated, survey of the literature". (Sheridan, 2016, p.525)

The HRI was born in the early days of industrial robotics, where industrial robots are primarily employed for boring and unsafe handling manufacturing activities in a manufacturing environment. The first interactions under these systems were usually confined to routines pre-programmed with little room for human contribution or the capability to adapt to a change in condition.

However, the HRI development greatly improved with the progress of Robotics, AI, and sensing technologies. Today, robots are capable of accomplishing quite complex behaviors, such as perception, learning, and adaptation, which allows a robot to serve for interaction with a human in a quite humane way.

1.2 Challenges in Human-Robot Interaction

In human-human interactions, social cues are used to convey context-bound behaviors. However, in human-robot interactions, it is required that signals from both directions be explicit and unambiguous. It includes verbal and non-verbal delivery of cues, for example, people communicate with each other by gesture, facial expression, or body language.

Moreover, getting a robust natural language understanding language poses a big problem to HRI. Sometimes, people usually use complex language, expressions, and phrases amongst themselves in communications, leaving the task of the robot to interpret and respond accurately.

And with their physical interaction and safety, strong sensing and control mechanisms will be needed to avoid accidents and any risk during the robots' closed interactions with humans.

1.3 Purpose and Significance of the Project

This project aims to design and implement a robotic autochat system capable of understanding human speech and engaging in natural conversations with users. The existing large-scale AI-enabled chatbot systems like ChatGPT will be implemented, with the advanced audio and video sensors. It is an innovative idea to facilitate seamless communication between humans and robots across various contexts.

2 System Design

2.1 Introduction

At Worcester Polytechnic Institute (WPI), our team has developed the first AI-enabled chat robot designed to pioneer advanced human-robot interaction. This section outlines the architecture and key components, emphasizing the system's modularity, scalability, and the robust integration of both hardware and software to accommodate future technological expansions and increased user interaction demands.

2.2 System Architecture Overview

The architecture integrates cutting-edge hardware with sophisticated software modules, supporting extensive data processing and real-time interaction capabilities. It lays the foundation for handling complex computations and managing large datasets efficiently, while maintaining flexibility for future enhancements. Multithreading is used to keep track of interactions as well as run the facial recognition software in the background.

2.2.1 Modular Design

Our team prioritized modularity in the robot's design, allowing for easy updates as technologies evolve and preventing the robot from becoming obsolete whenever a new model comes out. Utilizing an LLM framework, our system enhances the flexibility of the language learning model (LLM) pipeline, allowing components such as the GPT API integration and RAG (Retrieval-Augmented Generation) module to be updated independently.

LLM Framework: LangChain simplifies the process of swapping out modules and altering the models used at each stage of the pipeline, reducing our dependence on specific companies such as OpenAI. This allows for the integration of other models as alternatives to the GPT API, even running a model locally on our hardware.

Modular PC Build: The computer that controls the robot was also designed with modularity in mind and for future teams to have an easy time replacing any parts that might be obsolete.

2.2.2 Scalability

To ensure the system can handle increasing amounts of data and growing user bases without degradation in performance, we chose to implement several scalable solutions:

- **Vector Memory with ChromaDB**: We use ChromaDB for efficient and scalable storage and retrieval of vector embeddings, which store interaction histories and user information. This allows our system to scale as the number of interactions and the amount of stored data increases.
- **OpenAI API**: By leveraging the OpenAI API for heavy LLM tasks, we offload computational demands to a scalable cloud service, allowing for dynamic resource allocation based on system load.
- **Local Processing**: Critical real-time applications such as STT, TTS, and facial recognition are processed locally. This not only enhances performance and response times but also addresses potential data privacy concerns by minimizing external data transmission.

2.3 Hardware Integration

High-Performance Computing Hardware: Features a high-end NVIDIA A5000 GPU, robust consumer CPU, RAM, and fast NVMe SSD storage, ensuring efficient real-time operations. The modular nature of the PC build allows future teams to upgrade or repurpose components without needing to replace the entire system.

Externalized Computing Hardware: Initially designed to be housed within a compact MicroATX chassis, space and cooling constraints necessitated externalizing the computing hardware. This is an area that we left open to other groups, there are benefits of having the robot be self-contained but we chose to have the hardware externalized.

2.4 Sensory and Interaction Hardware

Audio and Visual Sensors: The robot is equipped with a torso and a head featuring two high-definition cameras mounted in stereo on its head, which can rotate to track people during interactions. This setup not only supports advanced facial recognition but also potentially enables depth sensing, significantly enhancing the robot’s environmental understanding and interaction capabilities.

Mechanical Design: The robot’s design is focused on functionality and user engagement, with a friendly, non-humanoid form that is approachable yet distinctly robotic. Instead of trying to make the robot humanoid and running into the issue of looking uncanny we wanted to go for a charismatic robot approach.

2.3 Key Functionalities

2.3.1 User System and Data Handling

Each user interaction is linked to a unique user profile managed by our system, allowing the chat robot to maintain context over multiple sessions. Facial recognition technology is employed to identify returning users, enhancing the personalization of interactions. Data related to user interactions and biometrics is handled with strict adherence to privacy regulations, ensuring user data is secure and confidential.

2.3.2 Speech and Visual Interaction

The integration of speech-to-text (STT) and text-to-speech (TTS) technologies enables our chat robot to communicate with users through natural spoken language, making the system accessible and user-friendly. The local processing of these functionalities ensures quick response times and reduces latency, which is crucial for maintaining a natural conversational flow.

2.3.3 Decision Making and Response Generation

Our system uses a combination of direct LLM responses and RAG for generating replies. Decisions on which approach to use are made based on the context of the query and the specific needs of the interaction, ensuring that responses are both relevant and timely. This decision-making process is supported by NLP techniques that analyze the input and determine the most suitable response strategy.

2.3.4 Voice Activity Detection and Dialogue Management

The system includes voice activity detection (VAD) to accurately identify when a user starts speaking and manage pauses in speech. This ensures the robot can maintain natural dialogues by detecting speech endings and managing conversational turns effectively.

2.3.5 Long Term Memory

Enhances the chatbot's ability to recall and reference previous conversations, employing a combination of RAG for retrieving relevant past interactions and a session-based memory buffer. This dual approach ensures comprehensive memory management, with important conversation details summarized and stored for long-term use, facilitating continuity across multiple interactions.

2.4 Future Directions and Recommendations

This inaugural project at WPI establishes a solid foundation for further innovation in AI-enabled chat systems. We encourage future projects to build upon this base, we had a lot of ideas that we didn’t get to explore but encourage future students to look into:

Gesture Recognition and Multimodal Interaction: Extend the system's capabilities to include gesture recognition, enriching user interaction by integrating multiple modes of communication.

Depth Perception: Advance the robot’s sensory hardware to include depth perception for better environmental awareness and improved navigation, particularly useful in complex interaction scenarios and multi-user environments.

Speech Diarization: Develop speech diarization technologies to distinguish between multiple speakers effectively, enhancing the robot's utility in group settings by attributing speech to the correct user.

Enhanced Security Measures: Improve facial recognition algorithms to support more robust authentication processes, increasing both the security and personalization of user interactions.

Local LLM Implementation: Investigate deploying a fully local LLM to minimize reliance on cloud services, enhancing the system's operational reliability and data privacy, especially in scenarios with limited internet access.

Server-Based Processing: Consider a server-based approach where the robot's "brain", which currently is the PC attached to the robot, are instead managed by a dedicated server completely. This would allow the robot to be a smaller or more approachable device.

2.5 Conclusion

The architecture of the system underscores a forward-thinking approach. And this emphasizes the modularity, scalability, and robust integration of hardware and software components. By adopting a modular design philosophy, we make sure that the system can be used for future technological advancements, preventing obsolescence and facilitating seamless updates. Furthermore, we are focusing on the scalability that should enable the system to handle growing user bases and data volumes without compromising performance. And this lays a foundation for sustained growth and expansion.

Combined with high-performance computing hardware and sophisticated sensory and interaction capabilities, the chat robot we made can engage users in natural, meaningful interactions. Each component is from speech and visual interaction to decision-making and response generation. This is finely tuned to create a seamless and lifelike user experience.

And this project gives more avenues for further innovation and exploration in AI-enabled chatbot. We aim to build on this foundation by exploring capabilities such as gesture recognition, depth perception, and enhanced security measures to increase system capability and user engagement.

All in all, this project represents a significant step forward in the application of AI technologies at WPI, establishing a scalable, modular, and innovative platform. The design choices made reflect a forward-thinking approach to educational and technological advancements, ensuring that the system remains adaptable and relevant for future explorations in AI-enabled interactions.

3 Hardware Design

3.1 Mechanical Engineering Aspect

This section reviews what others are doing regarding AI-powered robots and robotics, hardware, materials, and manufacturing processes.

One of the first AI-powered robots found during research was a GPT-powered robot built using LEGO. *Creative Mindstorms* on YouTube posted a video including the robot's creation process, the hardware used, and how the robot interacts.

When thinking about the design and manufacturing process, 3D printing, CNC machining, and woodworking were possible options for creating the robot. The mechanical team believes that 3D printing will fit this project best. Printing allows for rapid iterative design at a low cost, meaning the hardware engineering team can create many versions and manufacture them quickly. This fast-paced, iterative design process allowed for quick changes and implementation of feedback.

The strength of 3D-printed parts is also sufficient for a robot. While wood and metal will be stronger and able to resist more force before breaking, the robot will not undergo large external forces. Therefore, 3D-printed parts will be strong enough for this project.

3.1.1 Constraints

This section discusses the constraints regarding the mechanical engineering of AI-powered robots.

One of the most important constraints identified in the beginning of the process was approachability. People are supposed to interact with this robot in a conversational environment. Because of this, the design should feel approachable and friendly. Accommodating for this constraint included the addition of a bowtie and hat in the attempt to resemble a butler. Making the likeness of the robot closer to a butler also gives the impression that they are able to provide a service, in this case, integration with an AI model.

Other constraints included features that must be included in the final design. These include, microphones, cameras, and motors for the head and body.

Because of these constraints, 3D printing was used as the manufacturing method. It would allow for rapid prototyping and turn around for many design versions. With the assumption that there will be many iterations of the robot, 3D printing proved to be affordable.

3.1.2 Motivation

The difference with our robot when compared to other GPT-based robots is that it is specialized for WPI. We have integrated WPI data so that it can cater to the needs of users on the WPI campus. It can help answer questions, aid in campus tours, or even give directions to a different building.

3.1.3 Methodology and Design

We decided that 3D printing would fit our design constraints the best. For the design process, 3D CAD was used. Onshape was picked because it allows easy collaboration and sharing as well as simple version control. CAD will also incorporate all parts needed, allowing the design to work around existing hardware. It also allows dimensional analysis, meaning parts such as motors can be calculated to see if they work correctly.

One aspect of the robot that was needed in the design was approachability. This robot will be used to interact with community members; the design must incorporate this aspect of its use.

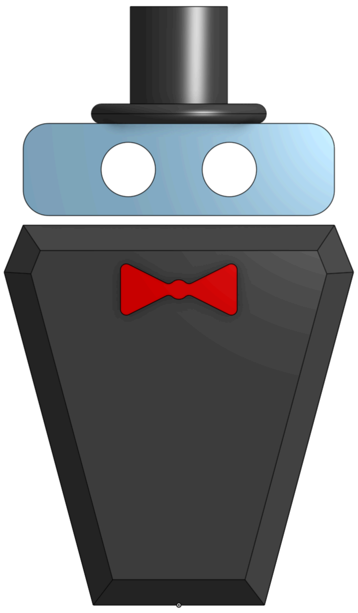


Figure 1 : Version 3 of the robot

3.14 Dimensional Analysis

The required torque was calculated such that the robot could rotate 180 degrees in 1. 5 seconds. This allowed us to buy a small motor that could move the robot. The dimensional analysis helped the design process because we could predict whether the motor would be powerful enough. MATLAB was used for calculations, and the initial data was gathered from the 3D CAD program.

Like the table shown below, Lxx, Lyy, and Lzz are the moments of inertia about the principle axes x,y, and z. This represents the resistance to rotational motion about each axis.

As for Lxy, Lyx,Lyz,Lzy,Lxz, and Lzx are the products of inertia, which represents the correlation of how the mass is distributed in two dimensions between any two axes. For example, Lyx is the product of inertia that is related to the y-axis and x-axis.

mass moment of inertia	$g * in^2$	**mass moment of inertia**	$g * in^2$	**mass moment of inertia**	$g * in^2$
Lxx	23776.63000	Lxy	0.032	Lxz	0.000528
Lyx	0.03173	Lyy	30870.85082	Lyz	72.156624
Lyzx	0.00053	Lzy	72.156624	Lzz	11116.43353

	x	y	z
angular acceleration (rad/s)	0	0	2

Mass moment in (kgm^2)	0.00001	1.33E-11	2.20E-13
	0.00000	1.28E-05	3.00E-08
	0.00000	3.00E-08	4.63E-06

Torque in (Kgm^2*rad/s) Same as Nm	0.00000000000043954	x
	0.00000006006800000	y
	0.00000925400000000	z
Torque of Motor (Nm)	0.16	

3.2 Electrical and Computer Engineering Aspect

The electrical team was tasked with figuring out how to pair all the input and output devices so it would be compatible with the chatbot. In the following sections, it will be expanded upon as to how we decided on what components we used and how they were setup.

3.2.1 Arduino Uno

To ensure compatibility with the code created for the chatbot robot we had to make sure we chose a microcontroller that would take real time inputs from the code and control the stepper motor chosen by the mechanical team. We ultimately settled with the Arduino Uno microcontroller for its compact design and versatility. It has the power requirements to operate the stepper motor and can be directly connected to the computer to receive real time inputs to move the stepper motor from the chatbot code.

3.2.2 Stepper Motor and Driver

The mechanical team determined that we should use a stepper motor to control the robot’s head. Having determined that we are using the Uno, we looked for a stepper motor that would fulfill the mechanical team’s requirements. this led us to use NEMA17HS3401 stepper motor which does fulfill the requirements and is compact to fit within the robot. In order to operate the stepper motor we had to choose a compatible stepper motor driver, the TMC2208 v3. The driver is compatible with the Uno and also compact enough to fit within the robot chassis.

3.2.3 Assembly of Motor and Microcontoller

We connected the TMC2208 stepper motor driver to the Arduino Uno. This involved setting up the power supply by linking the VMOT pin of the TMC2208 to a 12V power source, which was also connected to the Arduino Uno’s VIN pin for shared power. Control connections were made by assigning digital pins 7, 8, and 9 on the Arduino for the enable function, and for controlling the stepping and direction of the motor, respectively. The stepper motor itself was carefully connected to the driver, ensuring that each of the motor’s coil wires was correctly matched to the corresponding pins on the TMC2208, following the manufacturer's wiring configuration.

Once we verified that the stepper motor moved properly, we redid the connections from the Uno to the driver to be permanent connections by soldering the wires together, putting heat shrink on them to enclose connections and replacing wires from the original testing wires with shorter, sturdier wires that snag on the pins so connections aren’t undone while within the chassis.

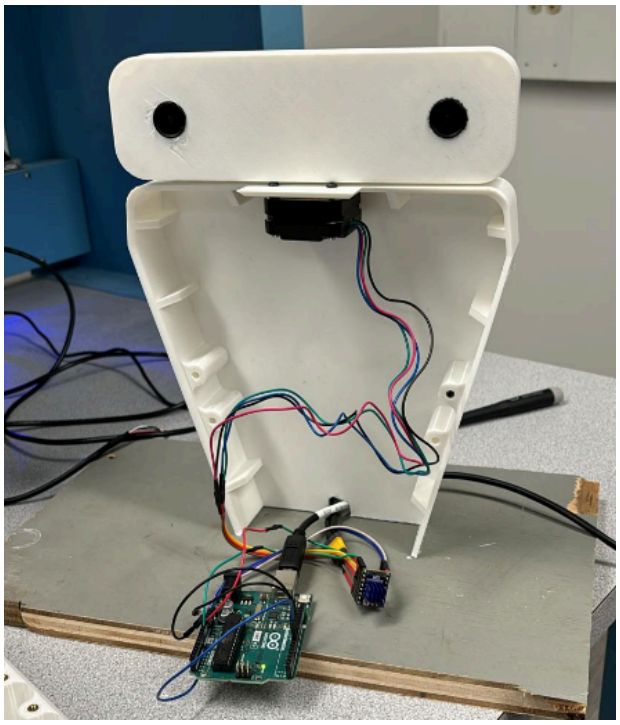


Figure 2:Robot assembled with the refined Arduino Uno microcontroller and motor driver setup

4 Software Design

The software design is composed of five main parts: Speech to text, text to speech, face detection and verification, LLM pipeline, and databases. The details for the process and theory will be shown in each part.

4.1 Speech-to-Text

The Speech-To-Text (STT) architecture for this project was designed so that it could work within the flow of a conversation. In this case, the conversation starts when a user's face has been either identified, or a user has opted out of being identified but a face is still visible in frame. Once this occurs, it means a user is actively standing in front of the robot. And this makes the microphone activate, and the output is listened to until a voice is recognized. Once the program recognizes that someone is talking, it records the speech until there is a long enough pause that the robot is making sure the user finishes talking.

The model used is Silero Models for the project, and it can fully rely on the leading Speech-to-Text technology known to have broad multi-language support and the best-in-class transcriptions, even for CPU-heavy tasks. Simply built and with little dependencies, we integrate smoothly to perform our core functionality and not battle with trying to set up a convoluted build. The advantages of Silero Model is that it includes one-step set-up with no need for add-on tools, like Kaldi; swift deployment, since no compilation is needed; and a ready pre-trained model, hence not needing a lot of instructions. Moreover, Silero models ensure the sound of natural results and offer an extended library of voices, including an infinite number of languages to work with. All these benefits materialize without GPU acceleration or heavy training, meaning the efficiency of these models is incredible with limited requirements.

From this figure of chart, the model versions V3, V4, and V5 WER are represented. The bar plots are the indication of the WER of the model over several test sets, which go from clean audio in "librispeech_test_clean" to even more adverse conditions from "multi_ted_val. The test sets are composed of the scenarios that have a reflection of acoustic conditions—those that take place in reality and pertain to lectures, discussions on financial data, and dialect variations. The lower WERs observed for newer versions of the models indeed point to improvements and better transcription accuracies, which are very key in the performance of our speech recognition application.

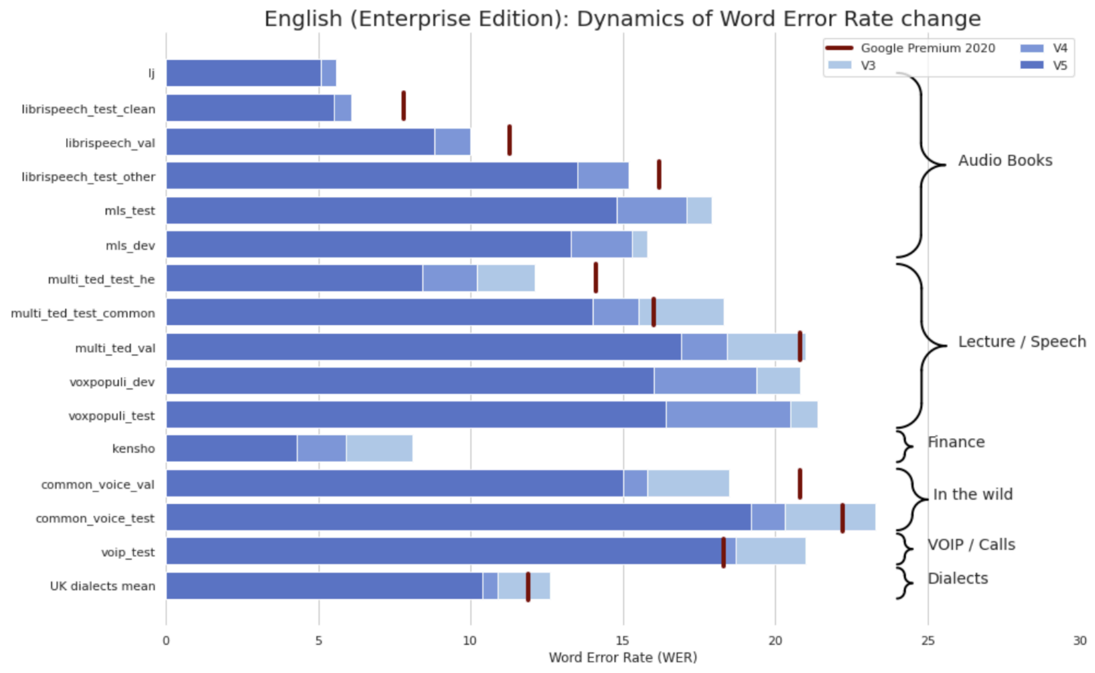
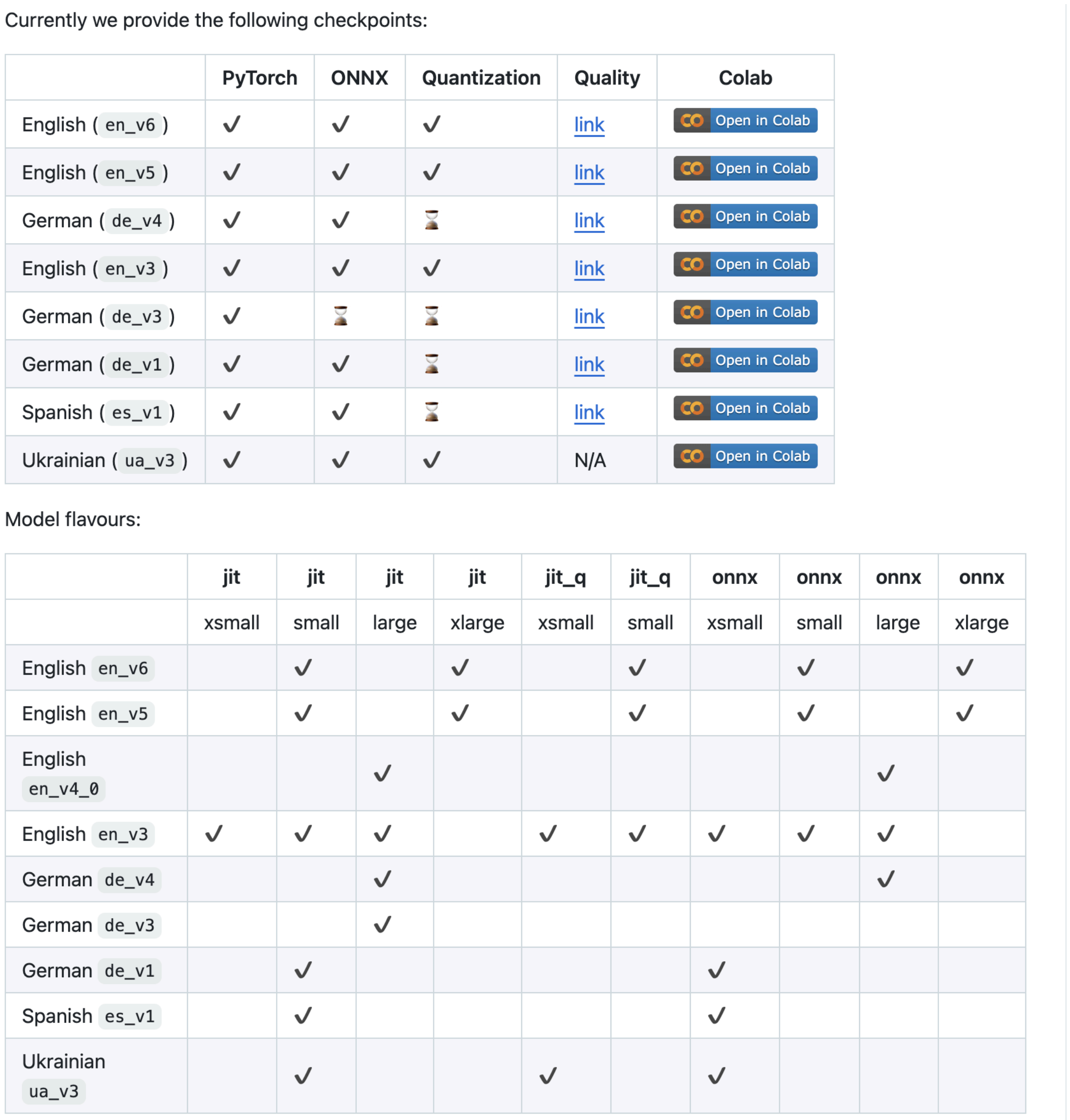


Figure 3 : Dynamics of World Error Rate change


Currently we provide the following checkpoints:

	PyTorch	ONNX	Quantization	Quality	Colab
English (`en_v6`)	✓	✓	✓	link	Open in Colab
English (`en_v5`)	✓	✓	✓	link	Open in Colab
German (`de_v4`)	✓	✓	⌛	link	Open in Colab
English (`en_v3`)	✓	✓	✓	link	Open in Colab
German (`de_v3`)	✓	⌛	⌛	link	Open in Colab
German (`de_v1`)	✓	✓	⌛	link	Open in Colab
Spanish (`es_v1`)	✓	✓	⌛	link	Open in Colab
Ukrainian (`ua_v3`)	✓	✓	✓	N/A	Open in Colab

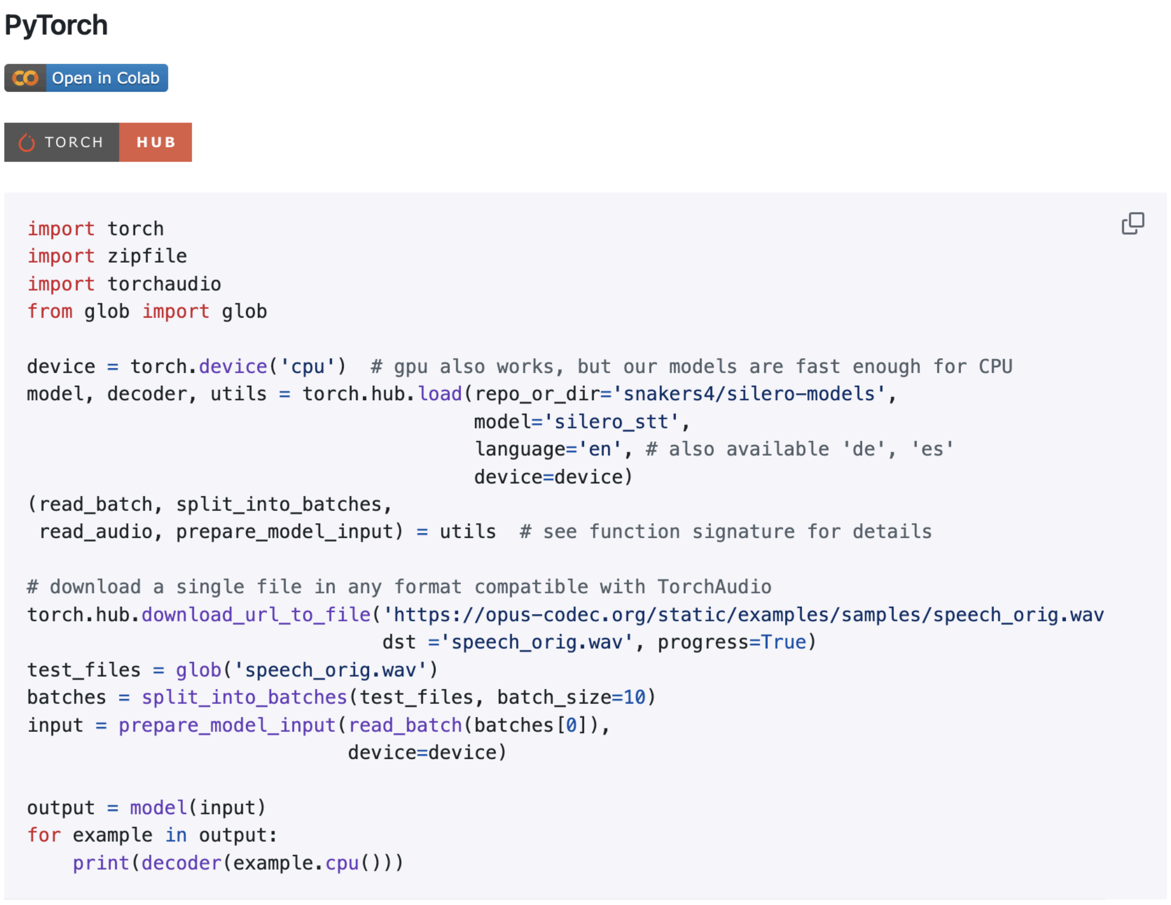
Model flavours:

	jit	jit	jit	jit	jit_q	jit_q	onnx	onnx	onnx	onnx
	xsmall	small	large	xlarge	xsmall	small	xsmall	small	large	xlarge
English `en_v6`		✓		✓		✓		✓		✓
English `en_v5`		✓		✓		✓		✓		✓
English `en_v4_0`			✓						✓	
English `en_v3`	✓	✓	✓		✓	✓	✓	✓	✓	
German `de_v4`			✓						✓	
German `de_v3`			✓							
German `de_v1`		✓					✓			
Spanish `es_v1`		✓					✓			
Ukrainian `ua_v3`		✓			✓		✓			

Figure 4: Available Model Checkpoints and Flavors

From the figure, the caption is a summary of the given variety of offered model checkpoints that are flexible for both PyTorch and ONNX integrations, quantization options, and are available in sizes with respect to JIT and ONNX models

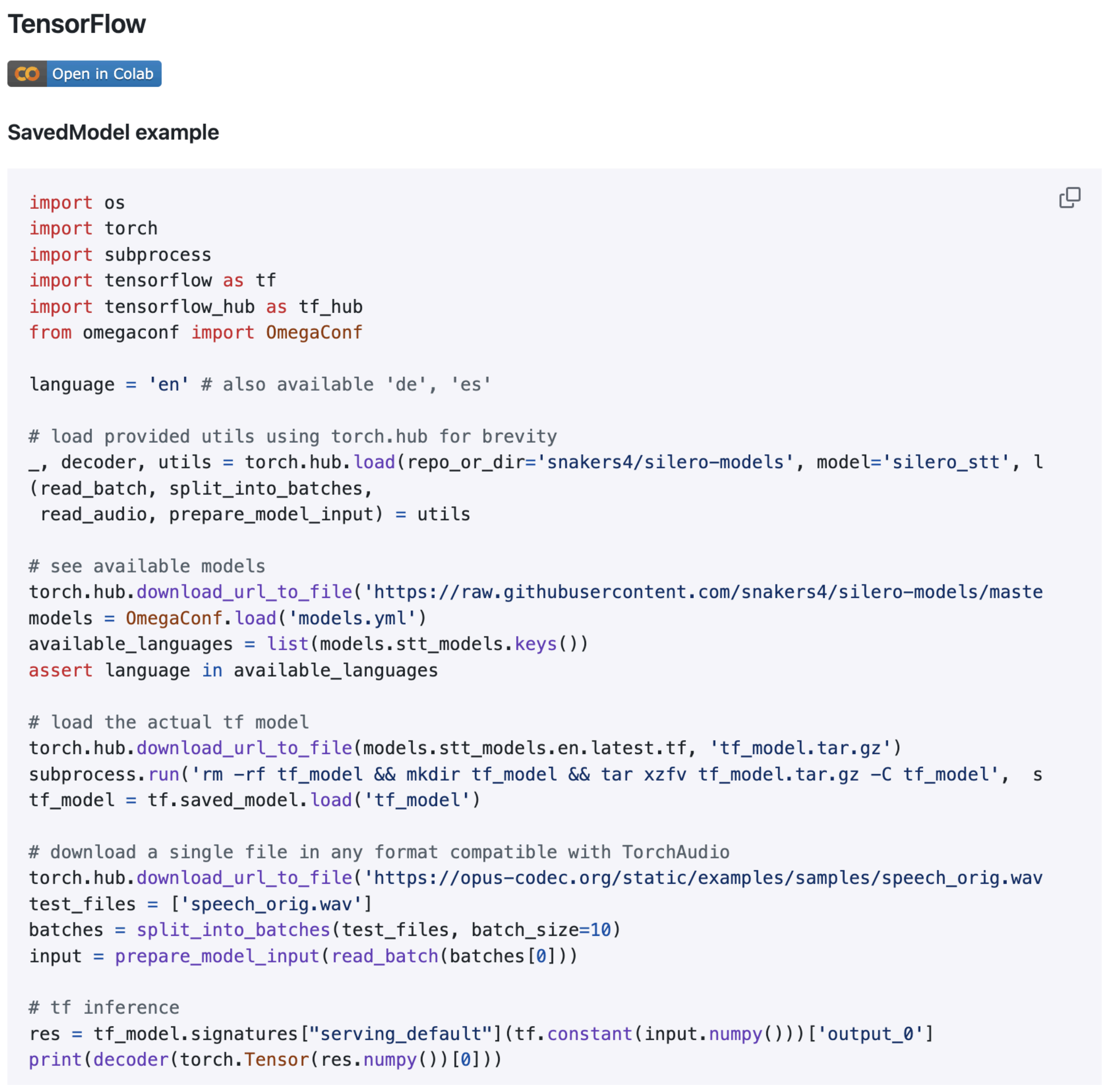
Below is a snippet in Python showcasing how to use the Silero STT model in applications. It shows the importing of required libraries, setting the computation device up, and loading a pre-trained model from the PyTorch Hub. This involves downloading a sample audio file, preparing the input file, and then transcribing speech to text using the model. Ands this is an evidence of the ease of running on the CPU, with simple code that would be easy to modify for languages supported by Silero's models.



```python
import torch
import zipfile
import torchaudio
from glob import glob

device = torch.device('cpu')  # gpu also works, but our models are fast enough for CPU
model, decoder, utils = torch.hub.load(repo_or_dir='snakers4/silero-models',
                                       model='silero_stt',
                                       language='en', # also available 'de', 'es'
                                       device=device)
(read_batch, split_into_batches,
 read_audio, prepare_model_input) = utils  # see function signature for details

# download a single file in any format compatible with TorchAudio
torch.hub.download_url_to_file('https://opus-codec.org/static/examples/samples/speech_orig.wav
                               dst ='speech_orig.wav', progress=True)
test_files = glob('speech_orig.wav')
batches = split_into_batches(test_files, batch_size=10)
input = prepare_model_input(read_batch(batches[0]),
                            device=device)

output = model(input)
for example in output:
    print(decoder(example.cpu()))
```

Figure 5: Sample Python Code for Silero Speech-to-Text Model Implementation.

The Silero uses a speech-to-text model that integrates into the TensorFlow framework. Below is an example of Python code showing how to import the necessary libraries, load the utility functions, and make the desired language accessible. This may consist of acquiring the SavedModel in TensorFlow, prepping the audio for input, and running the transcription. In this process, the streamlined model does well to point out that the model should operate well in the TensorFlow environment.



TensorFlow

Open in Colab

SavedModel example

```
import os
import torch
import subprocess
import tensorflow as tf
import tensorflow_hub as tf_hub
from omegaconf import OmegaConf

language = 'en' # also available 'de', 'es'

# load provided utils using torch.hub for brevity
_, decoder, utils = torch.hub.load(repo_or_dir='snakers4/silero-models', model='silero_stt', l
(read_batch, split_into_batches,
 read_audio, prepare_model_input) = utils

# see available models
torch.hub.download_url_to_file('https://raw.githubusercontent.com/snakers4/silero-models/maste
models = OmegaConf.load('models.yml')
available_languages = list(models.stt_models.keys())
assert language in available_languages

# load the actual tf model
torch.hub.download_url_to_file(models.stt_models.en.latest.tf, 'tf_model.tar.gz')
subprocess.run('rm -rf tf_model && mkdir tf_model && tar xzfv tf_model.tar.gz -C tf_model',  s
tf_model = tf.saved_model.load('tf_model')

# download a single file in any format compatible with TorchAudio
torch.hub.download_url_to_file('https://opus-codec.org/static/examples/samples/speech_orig.wav
test_files = ['speech_orig.wav']
batches = split_into_batches(test_files, batch_size=10)
input = prepare_model_input(read_batch(batches[0]))

# tf inference
res = tf_model.signatures["serving_default"](tf.constant(input.numpy()))['output_0']
print(decoder(torch.Tensor(res.numpy())[0]))
```

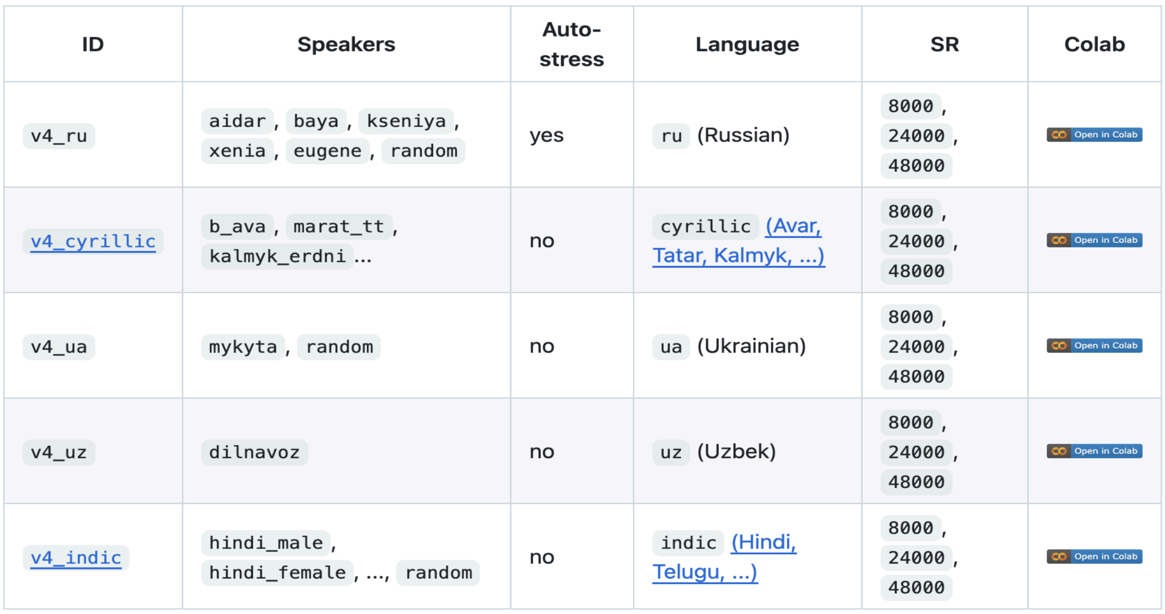
Figure 6:Sample codes for Silero Speech-to-Text Model Implementation for TensorFlow.

Silero was chosen because it was found to be much more accurate and faster compared to the original design which used Whisper for Speech-To-Text and SpeechRecognition for Voice-Activity-Detection. In the previous model using this combination, the SpeechRecognition software was run, listening to the microphone and recording what it believed to the speech before returning that to our software, which transcribed it to a temporary file before sending it to the Whisper model to turn into text. Switching from the SpeechRecognition VAD model to the Silero VAD model improved this pipeline as the Silero model merely detects when speech starts and stops, and does not record anything. This allowed us to cut out needing to transcribe the SpeechRecognition output to a temporary file, instead allowing us to use PyAudio to directly write the audio from input to a file on disc. This is then fed to the Silero STT model allowing for faster STT abilities.

4.2 Text-to-Speech

Text-to-Speech (TTS) technology is a form of speech synthesis that converts written textual content into audible speech. TTS systems equipped with deep learning models can generate synthesized speech and therefore have applications ranging from voice responses in virtual assistants to field help for users with visual impairment. Modern TTS models also come with different voices, and some even have SSML (Speech Synthesis Markup Language) to take control of the way speech sounds. The technology is rapidly growing to produce even much more natural and expressive voices.

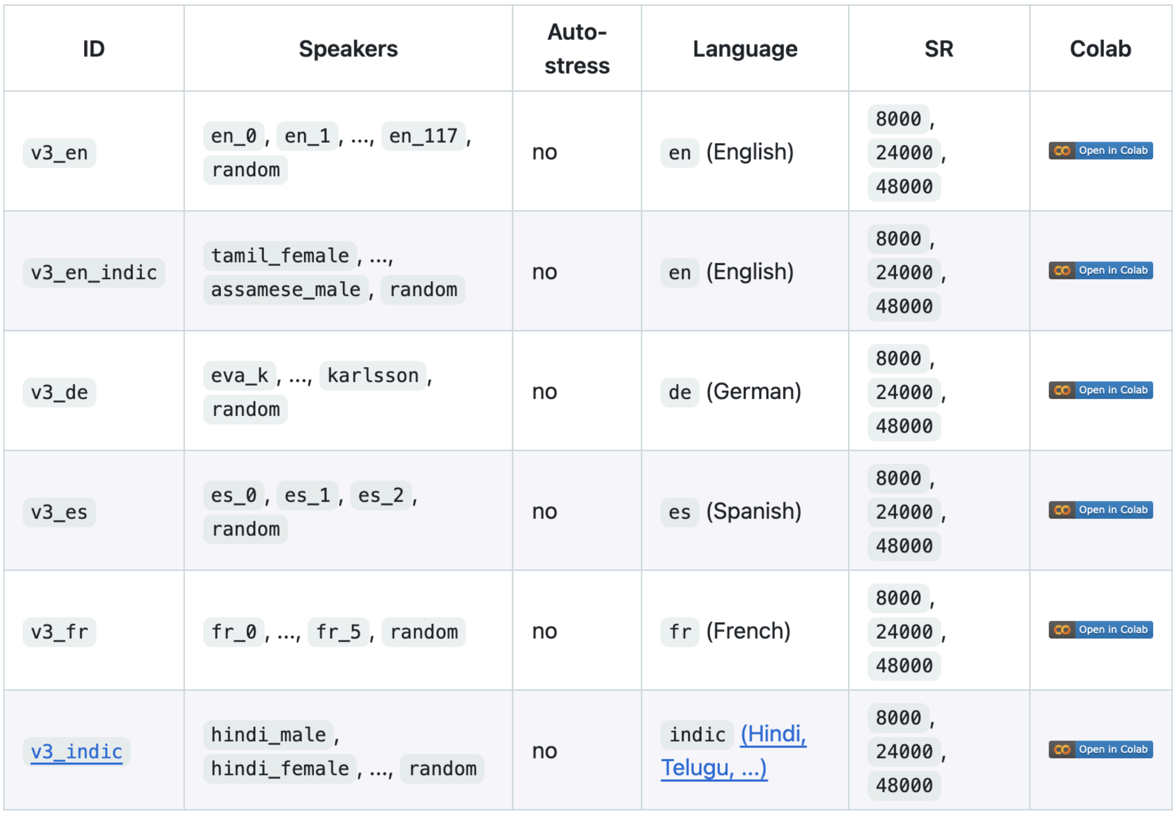
Like the figure shown below, the V4 Text-to-Speech models enhance the versatility of language support with many speakers, including those required for languages written in the Cyrillic script, besides other mostly spoken languages. Each model variant is identified by a model_id and provides options for different speaker voices to improve the naturalness of the synthesized speech. The models are also capable of supporting fine-tuning for produced speech using SSML (Speech Synthesis Markup Language) but still do not have auto-stress features for all languages. The sample rates range from 8000 Hz to 48000 Hz, making them usable for many applications and compatible with most platforms.



ID	Speakers	Auto-stress	Language	SR	Colab
`v4_ru`	`aidar`, `baya`, `kseniya`, `xenia`, `eugene`, `random`	yes	`ru` (Russian)	`8000`, `24000`, `48000`	Open in Colab
`v4_cyrillic`	`b_ava`, `marat_tt`, `kalmyk_erdni`...	no	`cyrillic` (Avar, Tatar, Kalmyk, ...)	`8000`, `24000`, `48000`	Open in Colab
`v4_ua`	`mykyta`, `random`	no	`ua` (Ukrainian)	`8000`, `24000`, `48000`	Open in Colab
`v4_uz`	`dilnavoz`	no	`uz` (Uzbek)	`8000`, `24000`, `48000`	Open in Colab
`v4_indic`	`hindi_male`, `hindi_female`, ..., `random`	no	`indic` (Hindi, Telugu, ...)	`8000`, `24000`, `48000`	Open in Colab

Figure 7: Overview of V4 Text-to-Speech Models

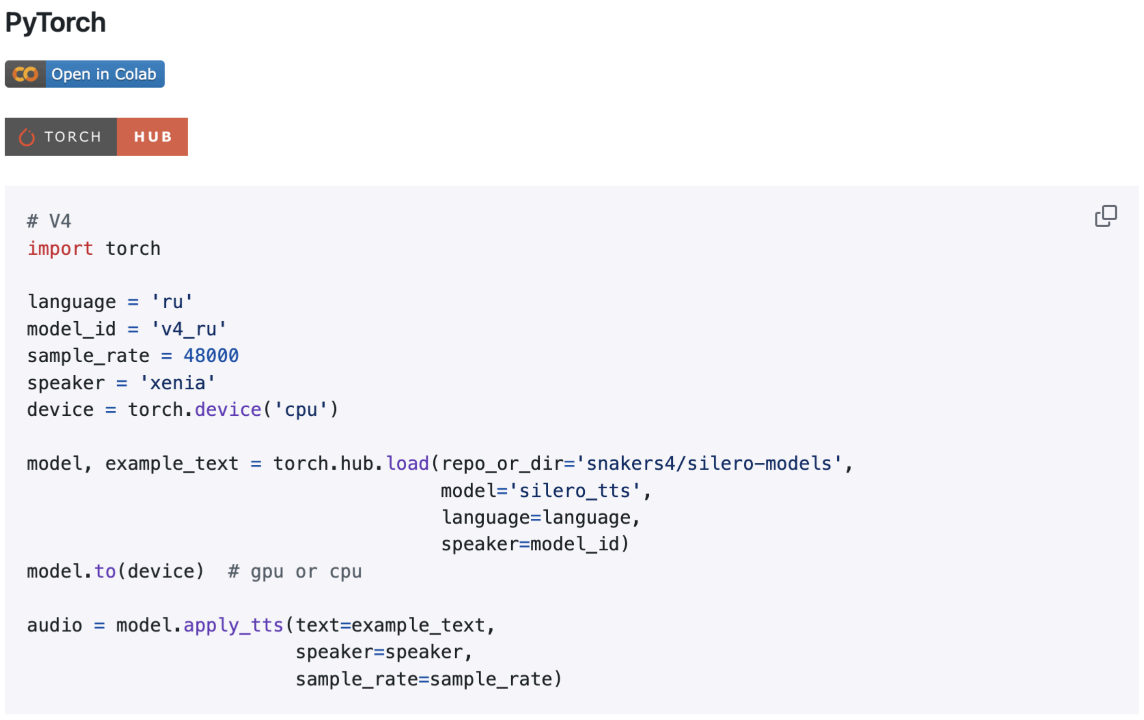
Like the figure of table shown below, The Silero TTS V3 models are further improved with SSML, so the synthesis of speech is more finely adjustable. The table contains the list of the models with different IDs that correspond to diverse speakers and languages, involving not only English and German but also Spanish, French, and Indic languages. The models do not inherently support auto-stress, but can work with speech rates (SR) of 8000, 24000, and 48000 Hz. Some language models in this list are going to have labeled speakers such as the names "African American" and "Australian." Others will only be labeled as "random" to ensure voice diversity for text-to-speech applications.



ID	Speakers	Auto-stress	Language	SR	Colab
`v3_en`	`en_0`, `en_1`, ..., `en_117`, `random`	no	`en` (English)	`8000`, `24000`, `48000`	Open in Colab
`v3_en_indic`	`tamil_female`, ..., `assamese_male`, `random`	no	`en` (English)	`8000`, `24000`, `48000`	Open in Colab
`v3_de`	`eva_k`, ..., `karlsson`, `random`	no	`de` (German)	`8000`, `24000`, `48000`	Open in Colab
`v3_es`	`es_0`, `es_1`, `es_2`, `random`	no	`es` (Spanish)	`8000`, `24000`, `48000`	Open in Colab
`v3_fr`	`fr_0`, ..., `fr_5`, `random`	no	`fr` (French)	`8000`, `24000`, `48000`	Open in Colab
`v3_indic`	`hindi_male`, `hindi_female`, ..., `random`	no	`indic` (Hindi, Telugu, ...)	`8000`, `24000`, `48000`	Open in Colab

Figure 8:Overview of V3 Text-to-Speech Models

Below is an example of the V4 Text-to-Speech model by using PyTorch, predefined settings for Russian, a defined speaker, and a sample rate of 48000 Hz. This code displays how the model is loaded, the device on which the computations will be made, and then how to generate audio from text. This clearly shows that the model can be used for high-quality synthesized speech using PyTorch.


PyTorch

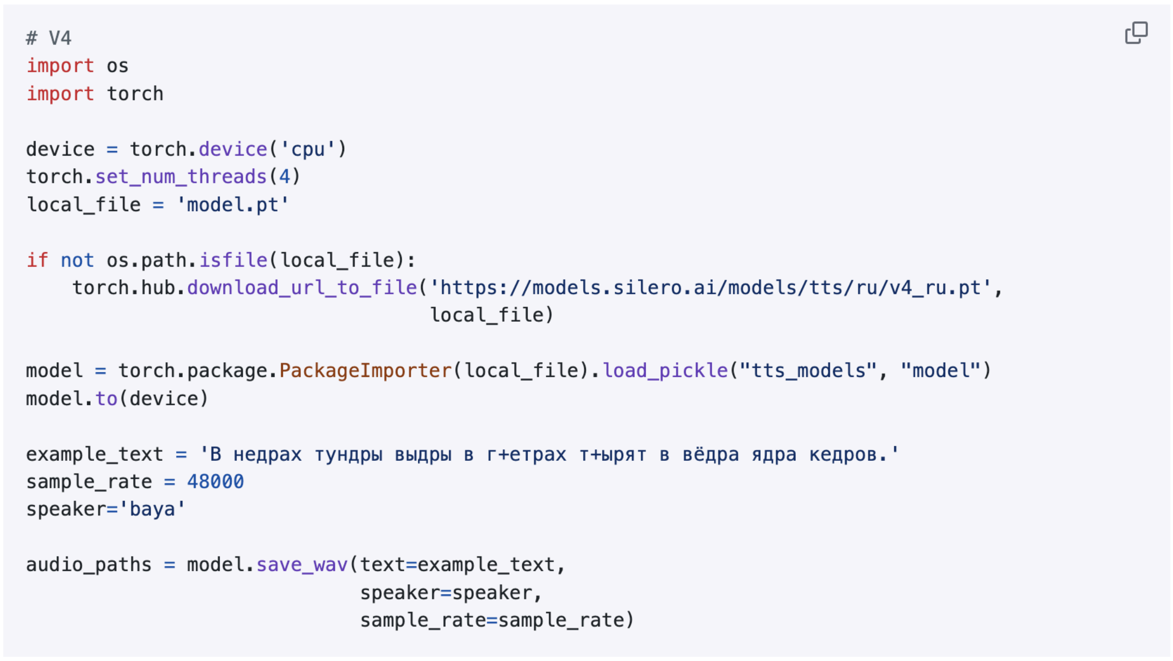
Open in Colab

TORCH HUB

```
# V4
import torch

language = 'ru'
model_id = 'v4_ru'
sample_rate = 48000
speaker = 'xenia'
device = torch.device('cpu')

model, example_text = torch.hub.load(repo_or_dir='snakers4/silero-models',
                                     model='silero_tts',
                                     language=language,
                                     speaker=model_id)
model.to(device)  # gpu or cpu

audio = model.apply_tts(text=example_text,
                        speaker=speaker,
                        sample_rate=sample_rate)
```

Figure 9: Implementation of V4 Text-to-Speech Model in PyTorch

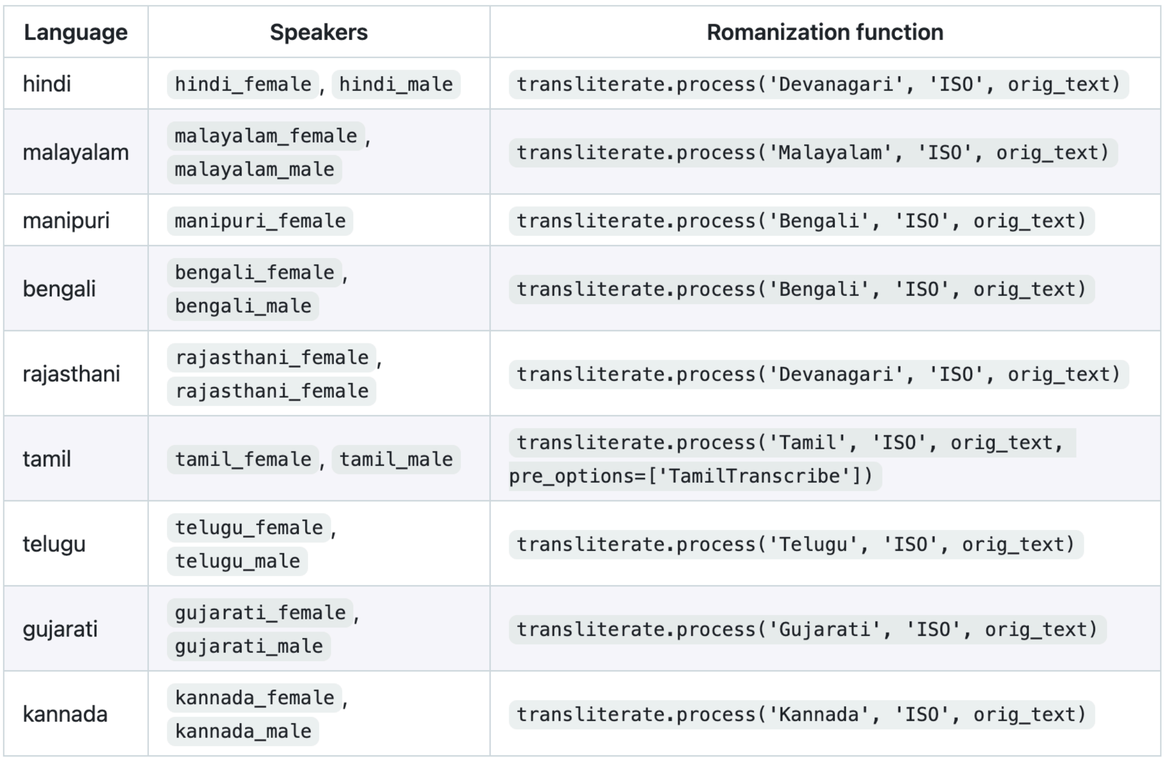
In Figure shown below, it provides a self-contained example of the use of the V4 Text-to-Speech model with the ability to inter-operate between PyTorch 1.10 and the Python standard library. This provides a way to configure the computational device, thread management for performance, and downloading the model file if not available locally. Then, it imports the model, sets the sample rate, chooses the speaker, and synthesizes speech from text. This could give the possibility to save the output as an audio file. This script was designed as an example of how the model is easy to access and provides very low entry-level for anyone willing to perform text-to-speech synthesis for his standalone application



```python
# V4
import os
import torch

device = torch.device('cpu')
torch.set_num_threads(4)
local_file = 'model.pt'

if not os.path.isfile(local_file):
    torch.hub.download_url_to_file('https://models.silero.ai/models/tts/ru/v4_ru.pt',
                                   local_file)

model = torch.package.PackageImporter(local_file).load_pickle("tts_models", "model")
model.to(device)

example_text = 'В недрах тундры выдры в г+етрах т+ырят в вёдра ядра кедров.'
sample_rate = 48000
speaker='baya'

audio_paths = model.save_wav(text=example_text,
                             speaker=speaker,
                             sample_rate=sample_rate)
```

Figure 10: Implementation of V4 Text-to-Speech Model in PyTorch for Standalone Use

Below is the figure of the table. This table lists the supported languages and the speaker options of a text-to-speech system, which includes male and female voices for Hindi, Malayalam, and few more. This language entry associates a romanization function through a transliteration process of converting the native-script-based text to a Latin alphabet in conformity with ISO standards to have a consistent pronunciation by the TTS system.



Language	Speakers	Romanization function
hindi	`hindi_female`, `hindi_male`	`transliterate.process('Devanagari', 'ISO', orig_text)`
malayalam	`malayalam_female`, `malayalam_male`	`transliterate.process('Malayalam', 'ISO', orig_text)`
manipuri	`manipuri_female`	`transliterate.process('Bengali', 'ISO', orig_text)`
bengali	`bengali_female`, `bengali_male`	`transliterate.process('Bengali', 'ISO', orig_text)`
rajasthani	`rajasthani_female`, `rajasthani_female`	`transliterate.process('Devanagari', 'ISO', orig_text)`
tamil	`tamil_female`, `tamil_male`	`transliterate.process('Tamil', 'ISO', orig_text, pre_options=['TamilTranscribe'])`
telugu	`telugu_female`, `telugu_male`	`transliterate.process('Telugu', 'ISO', orig_text)`
gujarati	`gujarati_female`, `gujarati_male`	`transliterate.process('Gujarati', 'ISO', orig_text)`
kannada	`kannada_female`, `kannada_male`	`transliterate.process('Kannada', 'ISO', orig_text)`

Figure 11: Supported languages and the speaker options of a text-to-speech system.

The precision-recall curve of the different Voice Activity Detection (VAD) models on the LibriParty dataset is presented below. The green one is the model Silero with a frames size of 30 ms, the blue one is a commercial model, and the red one is from the WebRTC model with a frames size of 30 ms as well. Precision is the proportion of the detected speech activity that is truly correct, while recall is the proportion of ability to detect all actual speech events. Ideally, a model will be more likely to return high precision and recall scores, whereby a perfect model results in both precision and recall being 1 or 100%. The graph suggests the Silero model balances precision and recall more effectively than the other models represented.

Moreover, Silero's Voice Activity Detection (VAD) accuracy is one of the best in this area and displays magnificent results in speech detection. This VAD makes a significant difference in speed: less than 1 ms of the CPU processing time is spent on audio chunks, while on a GPU or with ONNX optimization, this speed will be even faster. On the contrary, the model is lightweight in size—approximately one megabyte. From the other side, Silero VAD is versatile. This allows multilingual training and adaptation to practically all audio conditions by supporting both 8000 and 16000 Hz sampling rates with the corresponding sizes of the chunks. Compatible with the PyTorch and ONNX frameworks, portability is enhanced, and the permissive MIT-licensed free-to-use without-strings.

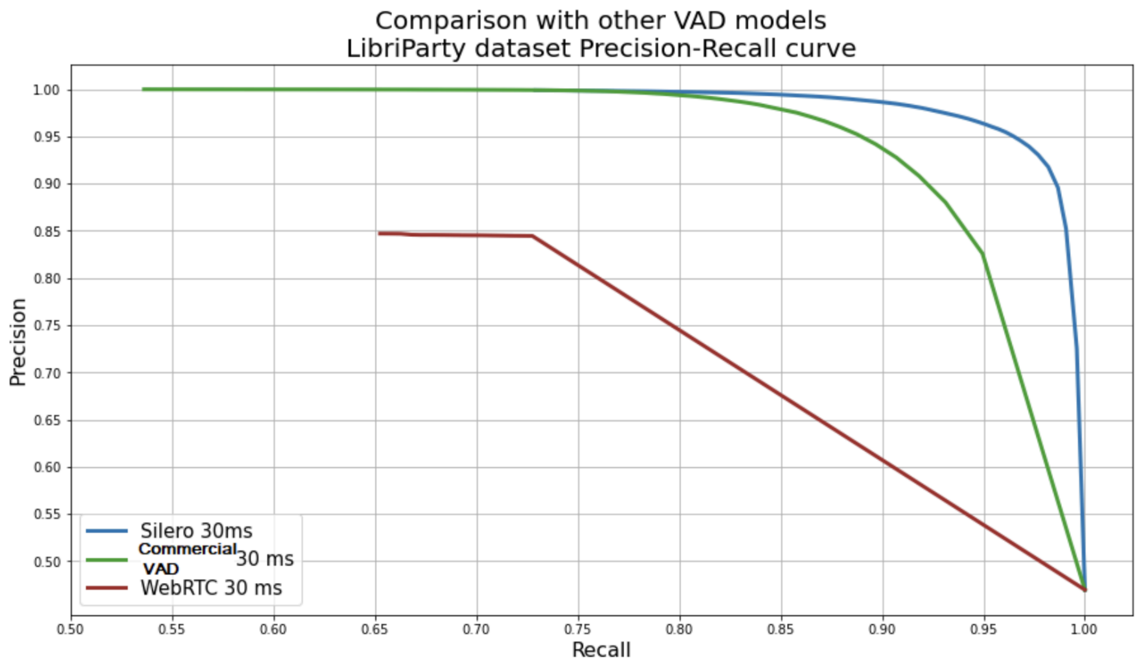


Figure 15:Precision-Recall Curve for VAD Models on LibriParty Dataset

4.3 Face Detection and Verification

Face detection and verification technologies are systems that verify a person's identification by his facial features. The capability of perfectly locating a face in images, further capable of verifying if two faces are the same, is hence very essential for applications in security, user authentication, and personalized services. Sophisticated algorithms evaluate facial features, such as distance between eyes, shape of jawline, and cheekbone contour, in order to come up with a facial signature. With AI growing stronger, we could very well see the reliability of these systems being put to use with AI technologies across the board, from smartphones to law enforcement.

The model we use is one of the biggest open-source projects on GitHub — DeepFace.And the model we used contains the largest Python library to be used for recognizing faces and analyzing facial attributes such as age, gender, emotion, and race. It wraps state-of-the-art models like VGG-Face, FaceNet, and DeepFace for high levels of accuracy surpassing human performance. It is simple to install via both PyPI and Conda, this library provides features for verification, finding, and analysis with very few lines of code. DeepFace was built in a way that users only need to make a single call for such complex operations that include detection, alignment, and normalization. It also offers a real-time analysis option using a webcam and can be deployed via Docker or consumed as an API.

In modern face recognition there are 4 steps: Detect, Align, Represent, and Classify. And the approach is more about alignment and representation of facial images.

For alignment, the steps will be shown below. The document we use illustrates the process of face alignment in such a way that from an input image, it generates a frontal face. It proposes that the 3D frontalization should use six key fiducial points on the face: the eyes, the tip of the nose, and points on the lips. Such points help in the detection of faces and, consequently, the alignment of faces in whatever poses and angles to a standard orientation, hence contribution to more accurate further face detection and further applications like recognition or verification. A 2D face image is then cropped from the original using these fiducial points for further processing. Then the third step is the spatially aligned 2D-enhanced and cropped face image is then obtained by aligning—mapping 67 fiducial points onto the face using Delaunay Triangulation for better spatial alignment. This further allows correction for out-of-plane rotations and is able to build a more accurate 3D face model. This careful plotting of fiducial points will contribute a lot to the precision in the subsequent tasks of face recognition. Then in constituting a 2D to 3D conversion in face recognition and verification, there is a relationship by which a 2D point is a projection of a 3D point, and in turn, gets transformed through multiplication with a transformation matrix. The improved transformation via optimization is to minimize an objective function: the sum of the squares of the residual errors between actual 2D points and projected 3D points. This is with regard to a covariance matrix in the consideration for data variability and further includes the Cholesky decomposition in the maximization of the ordinary least squares solution.

$$loss(\overrightarrow{P}) = r^T \Sigma r \quad \text{— } \textit{Equation 1}$$

where

$$r = (x2d - X3d\overrightarrow{P}) \text{ — } \textit{Equation 2}$$

Like shown above (Equation 1 and Equation 2), it is the relationship between 3D and 2D. And Σ is a covariance matrix and dimensions of (67 x 2) x (67 x2), X_{3d} is (67 x2) x 8 and [Gex]\overrightarrow(P) [/Tex] has dimensions of (2 x 4). We are using **Cholesky decomposition** to convert that loss function into ordinary least squares.

The last stage in the procedure of face alignment is referred to as frontalization: the correction of the 3D model towards a frontal view. In essence, this means adding the residual components to the x-y of the 3D warp such that there is less distortion on the face so that it is more accurate. Frontalization is carried out through piecewise affine transformations based on the 67 fiducial points, which had previously been marked through Delaunay triangulation. Such a rigorous process gives a more stable and standardized representation of the face that becomes applicable for recognizing and analyzing a variety of applications.

The architecture of DeepFace is designed for multi-class face recognition with input in the form of a 3D-aligned RGB image. It features convolutional layers with a large number of filters to capture low-level features, coupled with locally connected layers with various filter types to enhance discrimination across different facial regions. The model has 120 million parameters, mostly concentrated in the fully connected layers, and uses ReLU activation to promote sparsity. To prevent overfitting, dropout regularization is employed. Additionally, the network includes normalization steps and L2 regularization to ensure robustness against illumination variations.

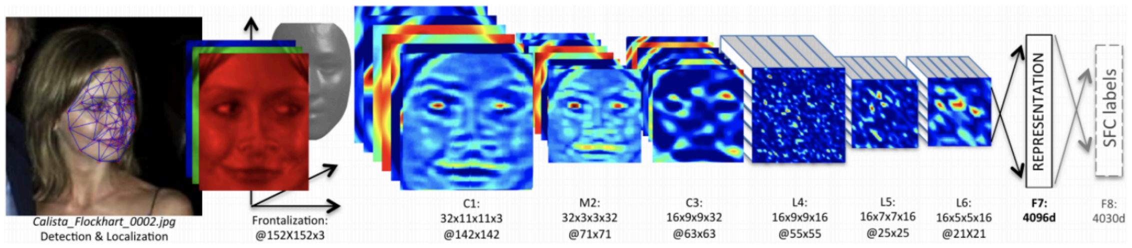


Figure 16: DeepFace full architecture

The results of DeepFace are also shown. The maximum accuracy is 97.35% shown in the document.

DeepFace provides a verification function that tests if two given face images represent the same person. The function accepts image paths or base64-encoded images as input and returns verification status along with distance metrics.

result = DeepFace.verify(img1_path = "img1.jpg", img2_path = "img2.jpg")

Additionally, the find function compares an input image with a database of facial images, outputting the results in a pandas DataFrame. This setup helps the system store facial embeddings to execute future searches more efficiently.

dfs = DeepFace.find(img_path = "img1.jpg", db_path = "C:/workspace/my_db")

Deepface wraps a variety of state-of-the-art face recognition models, offering the flexibility to choose according to the specific needs of a task. Its default model is VGG-Face, but it also includes models like FaceNet, OpenFace, and others, all of which provide a strong base for robust face verification and recognition. These capabilities allow operations such as verifying if two images show the same person, finding a face within a database, and representing faces as embeddings for further analysis.

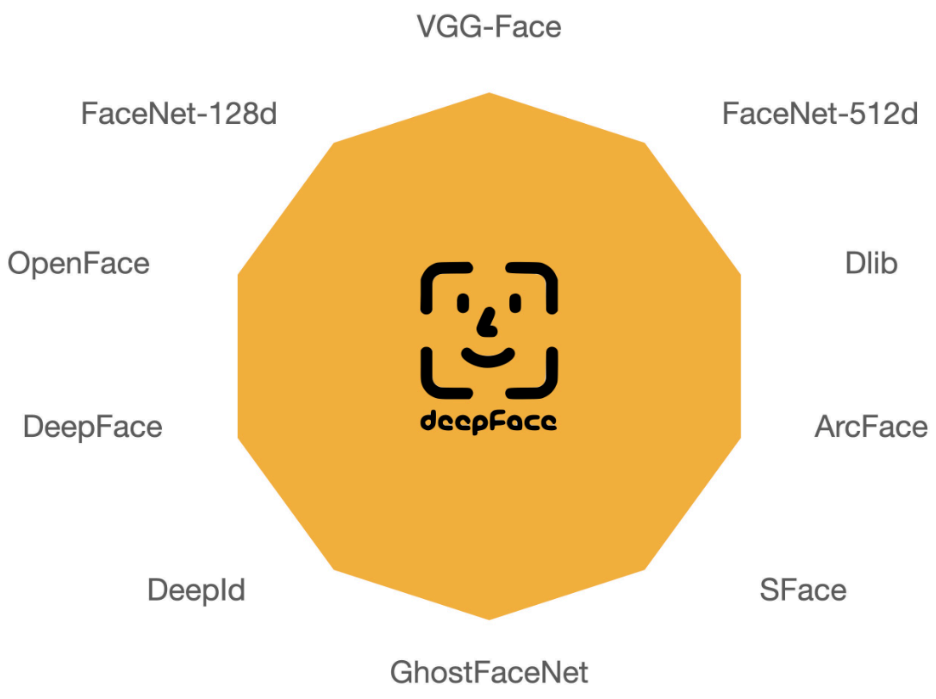
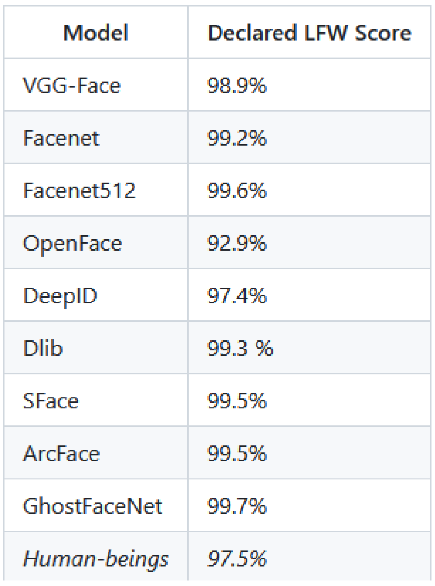


Figure 17: variety of face recognition models.

4.4 Face Recognition Model Selection

Out of all of the face recognition models provided by the DeepFace framework, we opted for working with VGG-Face. The VGG Face model, developed by the Visual Geometry Group at the University of Oxford, is particularly well-suited for facial recognition tasks due to its deep learning architecture, which closely mirrors that of the VGG-16 model known for image classification. It stands out for its training on a large and diverse dataset, which includes over 2 million images from approximately 2,622 distinct individuals. This extensive training helps the VGG Face model achieve high levels of accuracy in recognizing faces across various conditions such as different poses, lighting, and expressions. Its performance has established it as a benchmark model in facial recognition, often used in security systems and photo tagging applications due to its robustness.

Choosing the VGG Face model over alternatives like OpenFace, Dlib, or ArcFace can be justified on several grounds. Firstly, VGG Face's large training dataset and deep architecture tend to provide superior generalization capabilities, which is crucial in handling the vast diversity seen in real-world applications. While OpenFace and Dlib are also effective, they may not match the high accuracy and the robustness against diverse conditions provided by VGG Face. ArcFace offers competitive performance, particularly in terms of feature discrimination and is useful in highly accurate systems, but the simplicity and widespread use of the VGG Face model often make it a more accessible and readily applicable choice in many practical applications. Furthermore, the availability of pre-trained VGG Face models enables developers to implement facial recognition features quickly and with fewer resources, leveraging transfer learning to adapt to specific needs more efficiently than starting from scratch with other models. Below are all of the model's accuracy, measured by LFW metric (Labeled Faces in the Wild).



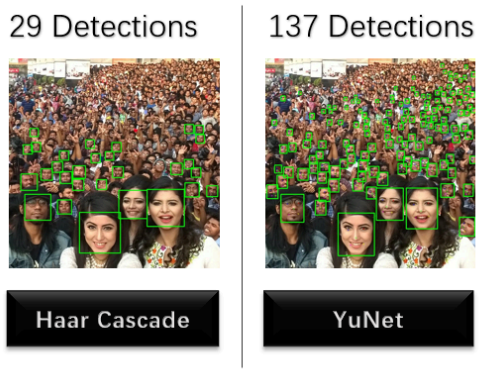
Model	Declared LFW Score
VGG-Face	98.9%
Facenet	99.2%
Facenet512	99.6%
OpenFace	92.9%
DeepID	97.4%
Dlib	99.3 %
SFace	99.5%
ArcFace	99.5%
GhostFaceNet	99.7%
Human-beings	*97.5%*

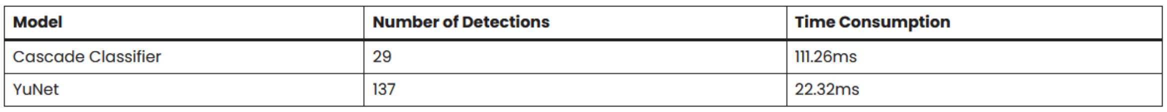
Figure 18: Different Models and declared LFW score.

4.5 Face Detection Model Selection

The YuNet face detection model stands out for its efficient architecture and real-time performance capabilities, designed specifically for use in environments where computational resources are limited. Built on a modified Single Shot Multibox Detector (SSD) framework, YuNet utilizes depthwise separable convolutions to reduce computational complexity while maintaining a competitive level of accuracy. This makes it exceptionally well-suited for real-time applications such as video surveillance and augmented reality on mobile or embedded systems. The model's ability to deliver robust face detection under varying conditions—such as changes in lighting, different poses, and partial occlusions—ensures its utility in both indoor and outdoor settings.

Compared to other popular face detection models like those available in OpenCV, SSD, Dlib, MTCNN, RetinaFace, and MediaPipe, YuNet offers a compelling balance of detection speed and accuracy with significantly lower computational costs. While OpenCV and Dlib are widely used for their ease of integration and broad feature set, they may not match the speed or efficiency of YuNet on constrained devices. SSD and RetinaFace, although highly accurate, require greater computational resources, which can be a limiting factor for real-time processing on less powerful hardware. MTCNN provides excellent accuracy through its multi-stage detection mechanism, but at the cost of increased computational overhead. Similarly, MediaPipe offers robust detection and additional features but might be overkill for applications that solely require face detection without the need for extensive tracking or landmark detection. Therefore, for projects where real-time performance, low power consumption, and minimal processing delay are critical, YuNet emerges as the optimal choice.





Model	Number of Detections	Time Consumption
Cascade Classifier	29	111.26ms
YuNet	137	22.32ms

Figure 19 : Numbers of detections for different models.

4.6 LLM Pipeline

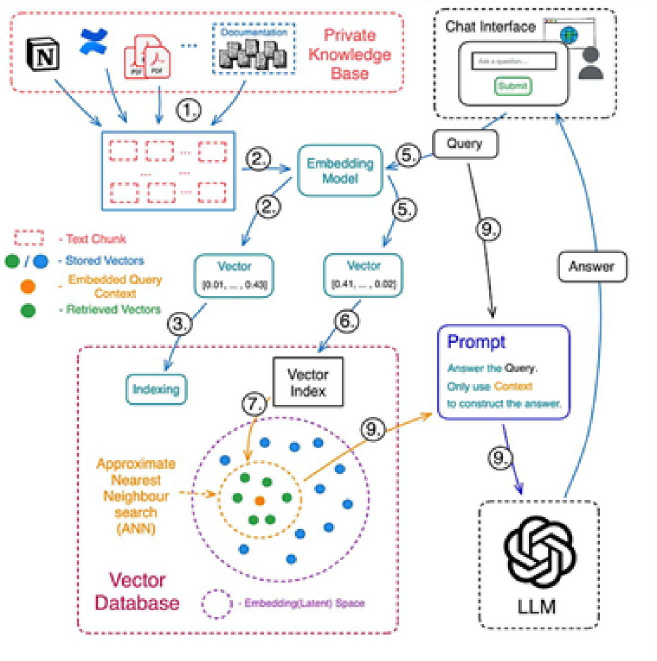


Figure 20: Language Learning Model (LLM) Pipeline graph.

4.6.1 Overview

The LLM pipeline is at the heart of our AI-enabled chat robot, designed to handle sophisticated interactions through an integration of advanced technologies. This pipeline utilizes a combination of deep learning models and intelligent systems to manage dialogues, access historical data, and utilize various external tools effectively.

4.6.2 Interaction Initialization

- **User Interaction Cache**: When a new interaction begins, the system automatically retrieves and caches relevant data about the user, including their previous interactions and any associated preferences or details. This preloading of data ensures that the robot can provide personalized and contextually relevant greetings, enhancing the user's experience from the moment of engagement.
- **Greeting and Continuation**: Depending on the cached information, the robot either initiates a new dialogue or seamlessly continues a previous conversation. This approach not only fosters a sense of continuity for returning users but also builds a deeper connection between the user and the robot by acknowledging past interactions.

4.6.2 Conversation Loop

- **Listening and Presence Detection**: Throughout the interaction, the robot continuously checks for the user's presence and attentively listens for any spoken inputs. This continuous monitoring is crucial for maintaining an engaging and responsive dialogue, ensuring the robot is always ready to proceed with the conversation as soon as the user speaks.
- **Query Classification**: Incoming queries are first processed by a specialized classification LLM, which determines the nature of the response required:

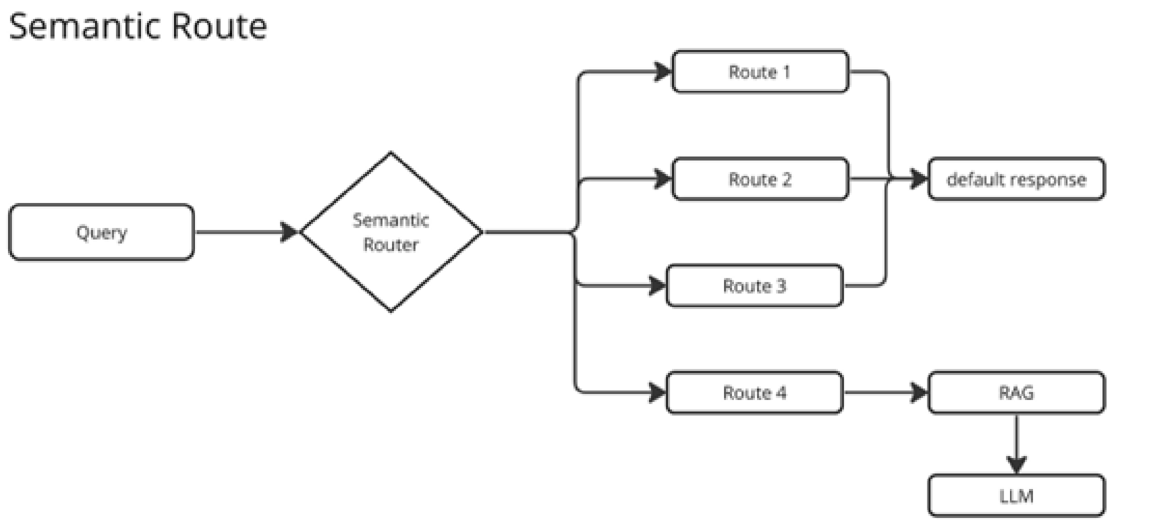


Figure 21: Semantic Route Flow chart.

  - **Naive LLM Response**: Straightforward queries are handled directly by the
  - LLM, providing quick and accurate answers without the need for additional contextual data.
    - The specific model used is also different depending on its underlying function; if the model is being used as an intermediary step and not the final dialogue generation step it will try to use a faster less smart model (3.5) but for generating speech it will use the more advanced but slower gpt 4 turbo.
  - **Information Retrieval via RAG**: More complex questions that require context or historical interaction data are routed to the Retrieval-Augmented Generation (RAG) system. This system pulls relevant information from a vast

database to enrich the robot's responses, making them more informed and contextually appropriate.

    - Uses ChromaDB as backend, explained further below.

- **Tool Utilization/Function Calling**: For queries that require external functionalities—such as accessing the internet, retrieving calendar events, or performing specific computations—the necessary tools are engaged. The results from these tools are then seamlessly integrated into the LLM's processing pipeline, enriching the response with accurate and useful information.
    - Some of these tools include: calendar access, math capabilities, access to current information from the internet, access to RAG, access to long term memory, change settings module. These can easily be expanded upon.

4.6.4 Response Generation and Handling

Data Integration and Response Formulation: Depending on the query classification, appropriate data is integrated into the response:

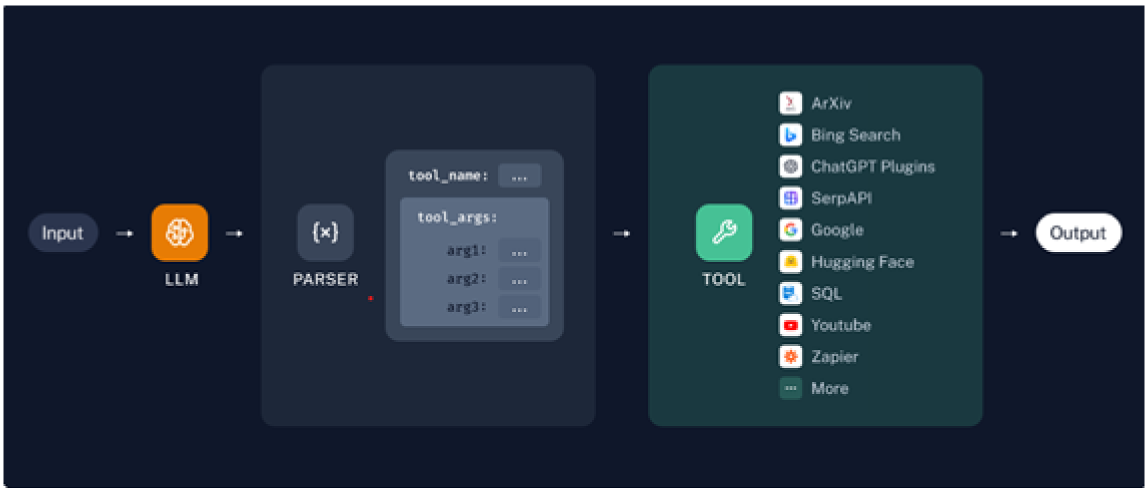


Figure 22 :Data integration and Response Formulation.

- **Tool Execution**: For operational queries needing external tools, the outcomes are synthesized into the LLM's responses, ensuring that the user receives comprehensive and practical information. This is done with the use of a special LLM fine tuned for function calling. Function calling is really what the underlying principle of tool usage is, its exposing useful functions to the LLM interface that allows the model to use predefined code such as looking something up on the internet via an API or doing a calculation.

    - "The latest models (gpt-3.5-turbo-0125 and gpt-4-turbo-preview) have been trained to both detect when a function should to be called (depending on the input) and to respond with JSON that adheres to the function signature more closely than previous models."

RAG System: *For complex inquiries, relevant information retrieved by the RAG (Retrieval Augmented Generation) system is incorporated into the LLM response, providing depth and context that enhance the relevance and accuracy of the robot's answers.*

    - **ChromaDB:** ChromaDB serves as the foundational vector storage system where embeddings of various documents, such as PDFs, articles, and other relevant textual data, are stored. These embeddings are preprocessed to capture semantic meanings, which are essential for effective information retrieval.
    - **Document Preprocessing**: Documents are first chunked into manageable pieces, then processed to extract vector embeddings that represent their semantic content. This preprocessing is critical for ensuring that the vectors are both comprehensive and efficient to search.

- **Retrieval part of RAG:**

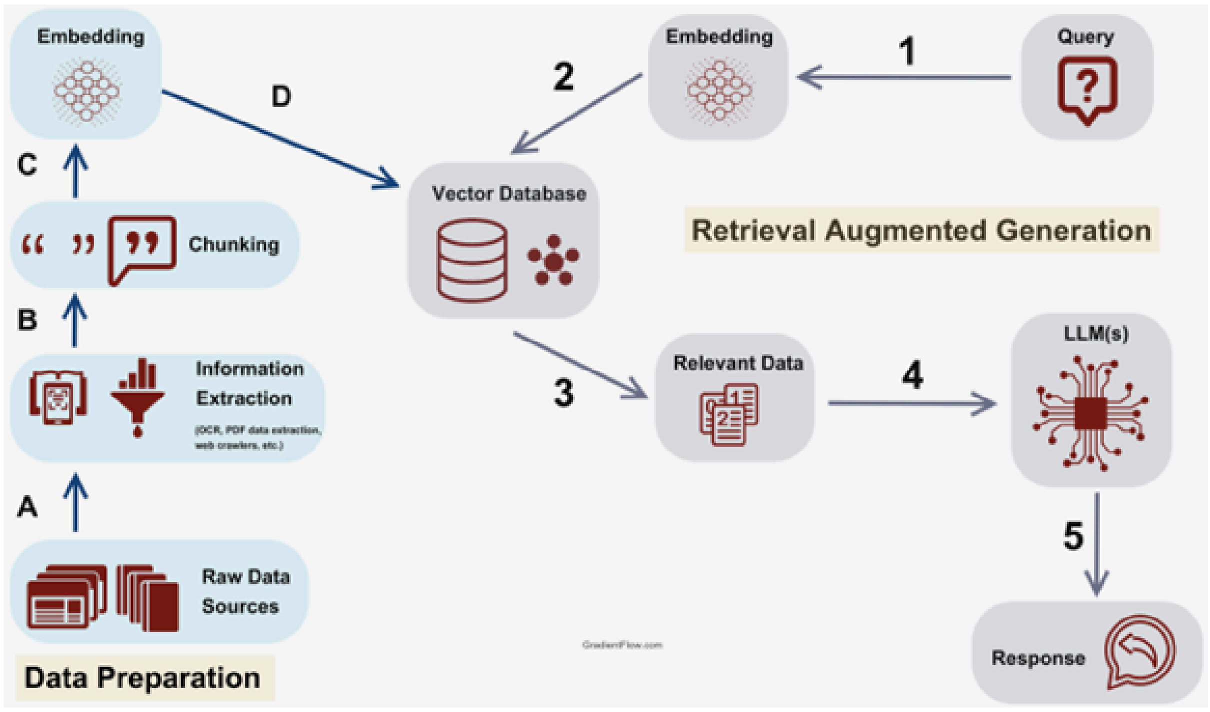


Figure 23: Flow chart of Retrieval augmented generation.

- **Semantic Search**: When the RAG system is activated by a query that requires additional contextual information, the system performs a semantic search within ChromaDB's collections depending on what information is queried (User data vs WPI data). Using the query's vector (embedding), the system searches for the most semantically similar document embeddings.
- **Cosine Distance Search**: The primary technique used for this search is cosine distance, which measures the cosine of the angle between the query vector and document vectors. This metric effectively identifies documents that are most similar in meaning to the query.
- **Further improvements:**
    - **Hybrid Search Approach**: To enhance the retrieval accuracy, a hybrid approach that combines keyword search and cosine distance measurements is employed. This method ensures that results are both relevant and precise.
    - **Result Integration and Re-ranking**: Once potential document matches are identified, they are potentially re-ranked based on their relevance to the current context, with the top results selected for

integration into the LLM prompt. This integration enriches the robot's responses, making them more informative and context-aware.

- **Response Processing**: After integrating all necessary data, the LLM crafts the final response, which is then refined for clarity and natural language flow. This refined response is converted into speech through a text-to-speech system, continuing the interaction loop and maintaining an engaging dialogue.

4.6.5 Long Term Memory System

- **Memory Storage**: The robot utilizes a sophisticated memory system that stores detailed logs of all interactions, segmented by individual user profiles. This long-term memory enables the robot to recall specific details or topics from previous conversations, enhancing subsequent interactions with personal relevance and continuity.

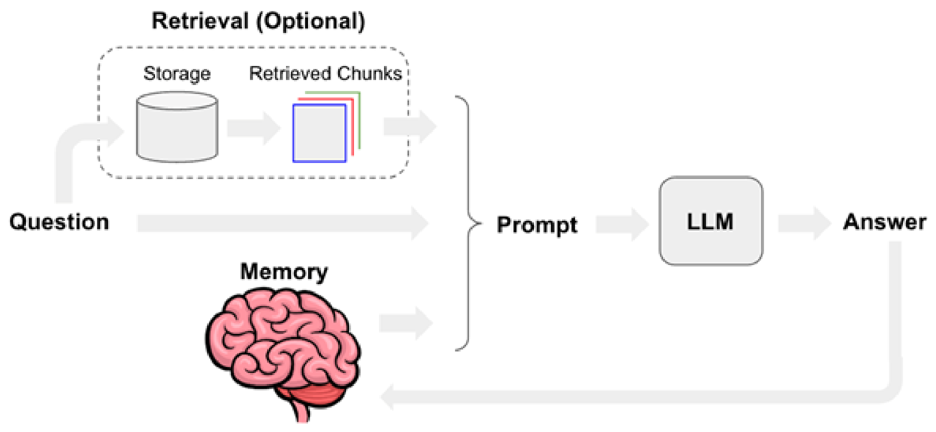
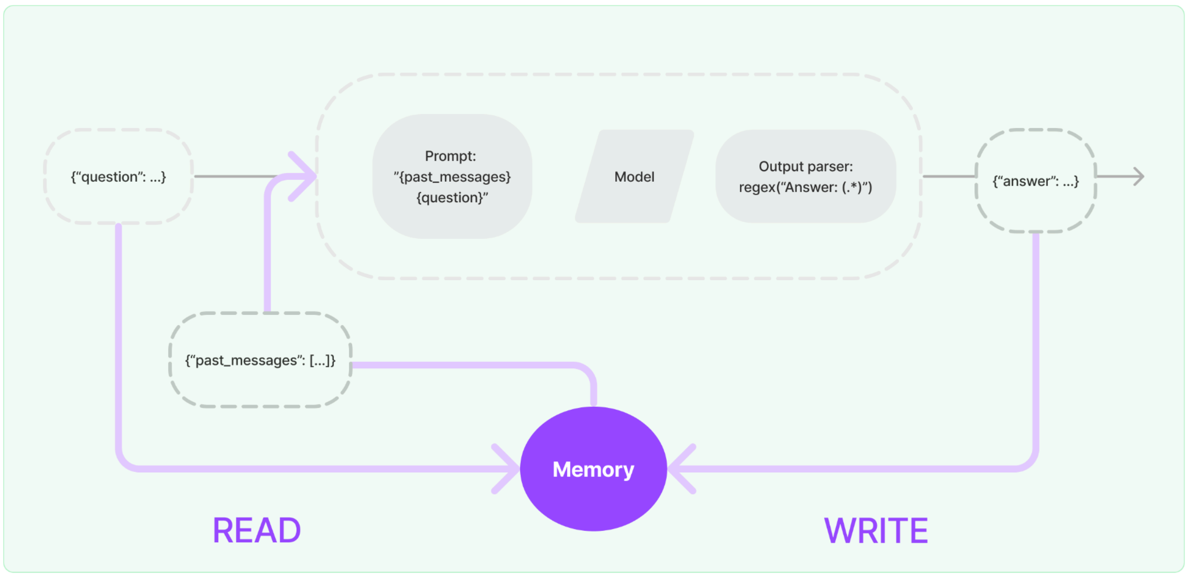


Figure 24 : Flow Chart of Memory Storage.

  - **Summary Memory**: At the end of each interaction session, a summary of the conversation is generated and stored. This summary captures key points and decisions, making it easy to retrieve and reference in future interactions.

- **Vector Store Memory**: Alongside summary memory, detailed interaction data is stored as vectors in ChromaDB. This dual approach ensures that both



high-level summaries and detailed conversational contexts are preserved.

Figure 25: Flow Chart of the vector store memory.

- **Challenges and Considerations**:
  - **Challenges with Vector Store Memory**: Initially, using vector-based memory for storing detailed interactions proved challenging, particularly in capturing the nuanced details that depend heavily on context. The fragmented nature of chunked text for embeddings sometimes resulted in loss of continuity and relevance.
  - **Advantages of Summary Memory**: To address these challenges, summary memory plays a crucial role by keeping coherent and contextually relevant records of each session. These summaries provide a reliable basis for recalling past interactions without the contextual fragmentation often seen with vector-based storage.

RAG and Memory

- **Combined Approach**: Both summary and vector-based memories are utilized strategically. For quick recall of specific facts or details, vector search is used, while summaries are employed for understanding broader context or session histories.

    - **Shared ChromaDB Infrastructure**: Both the RAG system and the long-term memory share the same ChromaDB infrastructure, which simplifies data management and enhances the efficiency of data retrieval processes.

- **Dynamic Settings Adjustment**: The system is designed to dynamically adjust various settings, including the robot's voice tone and interaction style (personality), based on user preferences and interaction history. This personalization capability is under continuous development, but should allow the user to change these things during conversations dynamically.

4.7 Database

The plan for the database of this project had 3 major parts. Assuming a user agreed to be recognized, they would be added to a centralized user database connecting their user_id and their name, as well as two specialized databases connecting their facial verification information and previous conversation context to their user_id. The decision to use a central userinfo database connecting their name to a numerical user_id was made in order to help differentiate between multiple users that might share the same name. Due to time constraints on this project, only the centralized user database and part of the plan for long term memory was finalized. In the current format user's facial recognition data is saved in the form of a .png file with a photo of their face and a filename of their user_id, and no long term memory for a specific user is stored. However, a vector database was implemented that stores information from documentation such as PDFs so that custom information - such as campus events, course catalogs, directions around campus, etc. can be loaded into the GPT prompt.

5 Experiments

5.1 Software Experiments

5.1.1 Silero vs Whisper STT

Before deciding which speech-to-test module to use for the robot, we must test two reliable modules and select one. To accomplish this, we had each module transcribe a complex sentence and convert it back to text. It was: "Despite its diminutive size, the intricate mechanism of a wristwatch contains an array of precisely calibrated gears, springs, and jewels, orchestrating the passage of time with remarkable position and elegance." The processes were timed and repeated five times to gain a comprehensive understanding of the data.

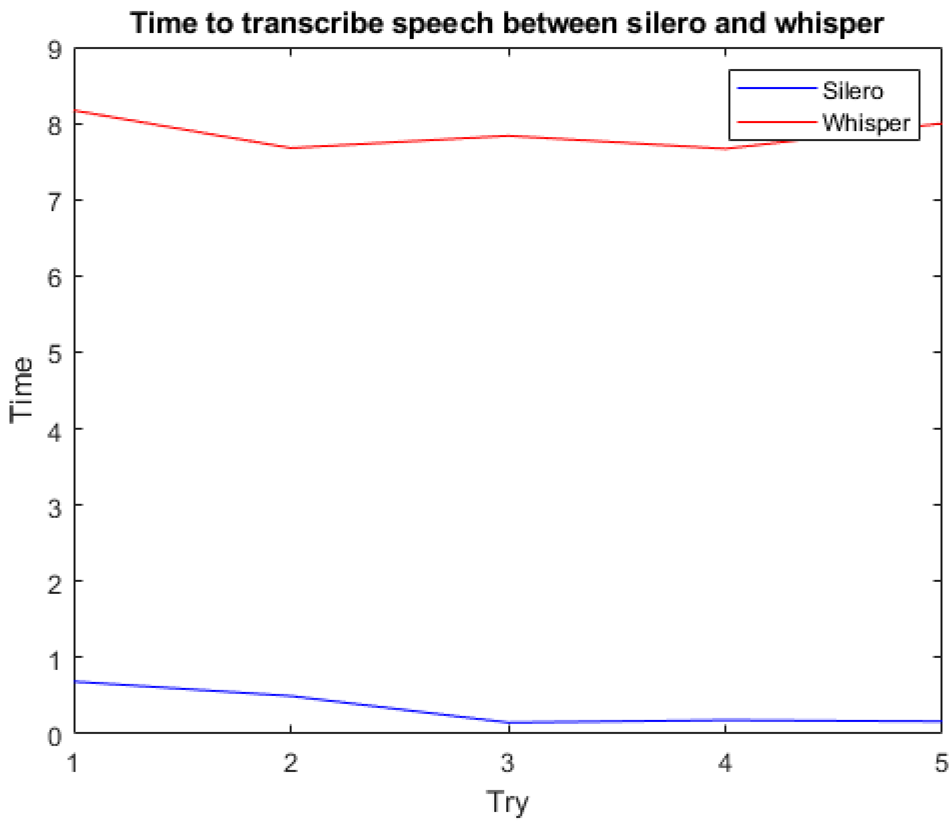


Figure 26: A graph showing the time taken to transcribe speech between Silero and Whisper

Upon reviewing the results, we found that Whisper was significantly slower than Silero when it came to the system's overall transcription time. However, the speech transcription contained numerous spelling errors and lacked accuracy. Our main objective was to identify the STT module that required the least amount of time to transcribe a given sentence, and it was evident that Silero was the best option. While it does tend to have a larger tendency for error, this can be counteracted in further versions with the use of the Silero Text Enhancement Models. In addition, GPT was found to generally be able to understand the meaning of a question even with some minor inaccuracies.

6 Conclusions

This report showed the successful development of the AI-enabled chatbot, which also shows the remarkable strides our team made in the realm of robotics and AI, especially in human-robot interaction. The proposed system introduced by us is a blend of leading-edge technology that transcends theoretical development, taking shape as a concrete interface capable not only of recognizing but also of communicating and interacting with humans.

Throughout the journey of detailed research, design, and implementation of this MQP, this report is going to show the integration of this AI-enabled chatbot with the integration of ChatGPT.

This MQP's objective is met as well. The robotic auto chat system has demonstrated its ability to handle some dialogues between human and robot. And it will respond with sensibility. We believe that it has brought us closer to the gap for the communications between human and robot.

In addition, it is also important to acknowledge the implication of this project. Even though the auto chat robot functions well in a prototype, there are also a lot of possibilities to refine and expand. For example, the signal processing is not used in this prototype like we planned to do. This auto chat robot could locate different people's positions by integrating the signal processing. And this could make the robot more efficient and communicate with humans more smoothly. Moreover, it is also possible to explore more AI models. And the enhancement of robot's empathetic responses could be made as well. Finally, exploring more applications of this technology in languages and dialects around the world could make the robot more universal.

In conclusion, the significance of this project is that it gives us a great opportunity to explore future research and more technology of human-robot interaction. Our project will pave the way for the following innovations between humans and robots. We hope to make them not just tools, but also partners in daily lives.

7 Reference

Hancock, P. A., Billings, D. R., Schaefer, K. E., Chen, J. Y. C., de Visser, E. J., & Parasuraman, R. (2011). A Meta-Analysis of Factors Affecting Trust in Human-Robot Interaction. Human Factors, 53(5), 517-527. https://doi.org/10.1177/0018720811417254

Sheridan, T. B. (2016). Human–Robot Interaction: Status and Challenges. Human Factors, 58(4), 525-532. https://doi.org/10.1177/0018720816644364

Dakhila, S., Ahmed, N., & Shaari, H. (2022). Comparison of two face recognition machine learning models. Journal of Pure & Applied Sciences, 21(4), Article 2120. https://doi.org/10.51984/JOPAS.V21I4.2120

snakers4. (n.d.). Silero Models [Software]. GitHub. Retrieved April 16, 2024, from https://github.com/snakers4/silero-models

snakers4. (n.d.). Silero VAD [Software]. GitHub. Retrieved April 16, 2024, from https://github.com/snakers4/silero-vad

Serengil, S. (n.d.). DeepFace [Software]. GitHub. Retrieved April 16, 2024, from https://github.com/serengil/deepface

LangChain. (n.d.). Introduction to LangChain. Retrieved April 16, 2024, from https://python.langchain.com/docs/get_started/introduction/

OpenAI. (n.d.). Introduction to the OpenAI API. Retrieved April 16, 2024, from https://platform.openai.com/docs/api-reference/introduction/

Aurelio Labs. (n.d.). Semantic Router [Software]. GitHub. Accessed April 16, 2024, from https://github.com/aurelio-labs/semantic-router

Chroma. (n.d.). Introduction. Chroma Docs. Retrieved April 16, 2024, from https://docs.trychroma.com/

Aurelio Labs. (n.d.). Semantic Router [Software]. GitHub. Retrieved April 16, 2024, from https://github.com/aurelio-labs/semantic-router