Implementing Voice Assistant for Visually Impaired Using LLMs and Vision Language Models

by

Jinke Jiang

A Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Implementing Voice Assistant for Visually Impaired Using LLMs and Vision Language Models

by

Jinke Jiang

Supervisory Committee

Prof. Hong-Chuan Yang, Supervisor
(Department of Electrical and Computer Engineering)

Prof. Issa Traoré, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

As a result of population aging, the number of visually impaired people is growing. Unfortunately, there is limited accessibility measures to help improve the quality of life of these people. The recent technological development in Artificial Intelligence (AI), especially Large Language Models (LLMs), should offer effective and efficient solutions. Recognizing the limitation of existing products, we design and implement a user-friendly and privacy-safe voice assistant for visually impaired people. Using LLMs and Vision Language Models, the assistant can recognize and identify objects through low-latency speech-to-speech interactions. The assistant can be deployed on offline edge computing devices with camera/microphone/speaker, with easily extendable functionalities. In this report, we present the design, adopted technologies, and adjustment that we applied to arrive at the final implementation.

Contents

List of Tables

List of Figures

ACRONYM

AI Artificial Intelligence

API Application Programming Interface

CNN Convolutional Neural Network

DaViT Dual-attention Vision Transformer

FFN Feed Forward Network

GIL Global Interpreter Lock

GAN Generative Adversarial Network

GPT-4 Generative Pre-trained Transformer-4

GPU Graphics Processing Unit

IP Internet Protocol

JIT Just-In-Time

LLMs Large Language Models

LSTM Long Short-Term Memory

MLP Multilayer Perceptron

NAT Network Address Translation

OCR Optical Character Recognition

ONNX Open Neural Network Exchange

PE Positional Encoding

RNNs Recurrent Neural Networks

SDP Session Description Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security

UI User Interface

UTF-8 Unicode Transformation Format-8

VAE Variational Autoencoder

WebRTC Web Real-Time Communication

ACKNOWLEDGEMENTS

I would like to thank:

My supervisor, Prof. Hong-Chuan Yang, for mentoring, understanding, and encouraging me, as well as his genuine concern for my personal and academic well-being, which allows me to progress effectively in my program.

Prof. Issa Traoré, for serving as a committee member for me.

Lab colleagues and classmates, for their valuable suggestions about this project.

DEDICATION

I first encountered visually impaired people in middle school when I was asked to develop a special computer diagnostic and repair software for them. This experience surprised me and allow me to realize that they are playing an important role in the society, and trying hard to fit in, even though not being able to see the world. This project is dedicated to this community and help them benefit from the technological developments in AI, such as Large Language Models, in recent years. I hope this project could reduce the costs of assistive devices for visually impaired individuals while enhancing their overall quality of life.

Chapter 1

Introduction

1.1 Background

According to the data published in 2019, approximated 1.2 million Canadians are living with visual impairment, accounting for 3.2% of the overall population [1]. Among them, around 4.1% individuals are blind. Figure 1.1 shows the increasing trend of the number of people with vision loss in Canada. By 2050, this number is expected to reach 2 million.

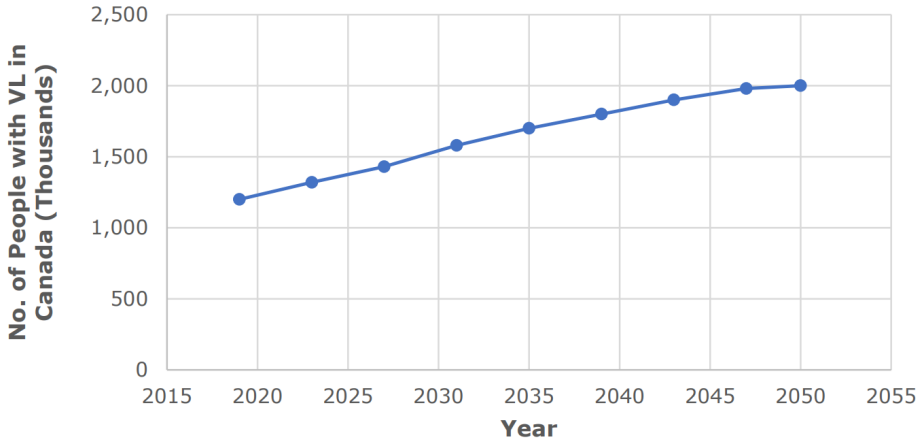

Figure 1.1: Projected prevalence of vision loss in Canada [1]

Currently, there are inadequate accessibility measures for them. For example, no accessible design on food packaging are widely available to help them recognize and identify products. There is limited efficient and accurate navigation tools for

them in both indoor and outdoor environments. Meanwhile, Generative Pre-trained Transformer-4 (GPT-4) and other open-source language models have achieved significant success in text and image comprehension and generation [2], enabling the possibility by utilizing these cutting-edge technologies to help visually impaired overcome daily barrier. In this project, using Large Language Models (LLMs) and Vision Language Models, we implement an intelligent and natural voice assistant to improve the quality of life for visually impaired.

1.2 Related Products

Some auxiliary devices, including various applications on the mobile phone and adapted hardware, have been developed over the past few decades to provide instructions or facilitate interactions between people and machines. However, many of these solutions are not user-friendly, as they are not optimized for individuals with vision loss. Before exploring further, we will evaluate and compare several products currently available in the market from the aspects of functionalities and cost.

1.2.1 "Be My Eyes" Application

In 2012, a Danish furniture craftsman Hans Jørgen Wiberg was inspired by the idea that family and friends could assist blind individuals with everyday tasks through video calls. Lately, he created the "Be My Eyes" mobile application in 2015 [3], establishing a large online community connecting blind users with volunteers as the name implies. It also integrated the GPT-4 vision model for image description in case of no response from the volunteers [4].

Earlier this year, although this company showcased a demonstration of video description in collaboration with OpenAI Inc., this feature is still not available for use as of September 20, 2024. However, there are some challenges faced by the video communication model, including:

- **Language Barriers**
  For speakers of minority languages, the likelihood of matching a suitable volunteer is low.

- **Unanswered Calls**
  Sometimes, the call may not be answered due to a lack of available volunteers

and time zone differences.

- **Privacy Concerns**
  There is no guarantee that the user's privacy will not be compromised or abused by criminals, as well as the data uploaded to AI company may be beyond the control.

- **Risk of Fraud**
  Blind people might be defrauded with incorrect information and instruction, leading to financial loss.

1.2.2 "AiSee" Device

A research group from National University of Singapore (NUS) developed a head-worn AI device called "AiSee" in 2018 [5]. This device contains a camera, a microphone, and bone conduction headphones. Visually impaired individuals can capture images of interest by pressing a hardware button, then ask relevant questions about those images, which are uploaded to a data center. Cloud-based LLMs infer the user's questions and provide accurate and timely responses.

The main limitation of such dedicated devices is their high costs. At present, the NUS research group is working on reducing costs to make the device more affordable. While this device enhances accessibility for people with visual impairments, its interactions is unnatural, and its functionalities are limited. Additionally, it relies on internet connectivity and cloud computing, which raises concerns about personal privacy.

1.3 Objectives

Considering the limitations of existing products and applications, our goal is to implement a voice assistant that serves as a guide for visually impaired individuals. The desired features of such assistant include:

1. Low-latency speech-to-speech communication

2. Capable of deploying on offline and/or edge computing devices regardless of Internet connectivity

3. Standard input-output interface to enable various third-party manufacturers to join the ecosystem

4. Applicable to different device forms such as mobile applications and wearable devices for greater flexibility and adaptability

5. Extensible and scalable architecture, allowing for easy addition of new functionalities

By leveraging state-of-the-art AI technologies, including LLMs and Vision Language Models, we develop a functional prototype of the voice agent. The prototype agent can understand user's intent as if engaging in a face-to-face conversation. It responds with answers including, but not limited to, reading text or labels, describing images, locating items, and identifying products in real time. Involving retrieval augmented generation technique in LLMs offers the memorization ability to our voice assistant.

Cost-effectiveness and privacy are other key consideration of our design. By utilizing open-source language models, we control the software costs. We are using LLMs with fewer parameters that do not sacrifice too much performance, reducing Graphics Processing Unit (GPU) memory consumption and allowing lower hardware resource requirements, as well as extending the battery life. Additionally, the fully controlled offline model design ensures that personal information and sensitive images, such as credit card security codes and body photos, are not exposed. As a result, more visually impaired people will benefit from this low-cost and privacy-safe design.

The remaining of the report is organized as following.

Chapter 2 introduces the system design and the methodology of customization of our voice assistant. It covers the functionalities of each components, describing the front-end and the back-end in our prototype agent. The audio/video processing steps and protocol design are elaborated, along with the challenges faced during development. Subsequently, we discuss how to customize LLMs using few-shot learning and retrieval augmented generation techniques.

Chapter 3 provides fundamentals of each modules of our system, including the interface for communication, voice activity detection, text-to-text and image-to-text model, speech recognition, and text-to-speech technologies. It also introduces the relevant principles of Transformer, Variational Autoencoder, and Generative Adversarial Network.

Chapter 4 examples the functionalities and use cases of the current prototype agent, such as recognizing text, identifying objects, locating items, and cleaning up duplicate data.

Eventually, Chapter 5 summarizes the project and looks forward to the future work that our voice assistant could be improved potentially.

Chapter 2

Methodology

2.1 System Design

As shown in Figure 2.1, our extensible voice assistant has the following workflow. The blue dot lines show how the voice streams are transmitted through different modules. Particularly, a Transport Layer Security (TLS) connection is established between the user and the Web Real-Time Communication (WebRTC) interface, considering for data security.

1. **User Interaction**
   The user interacts with the system via voice through WebRTC [6], which establishes encrypted real-time communication between the user and the assistant. The audio and video streams are split into several chunks and sent to the Voice Activity Detection module.

2. **Voice Activity Detection**
   This model identifies whether the user is speaking, automatically concatenating valid audio chunks into a segment then triggering the subsequent speech recognition.

3. **Speech Recognition**
   Once a piece of audio has arrived at this stage, the spoken input is processed to generate the text representation of user's speech.

4. **Intent Analysis**
   As the black solid lines shown in Figure 2.1, the Intent Analysis module is made

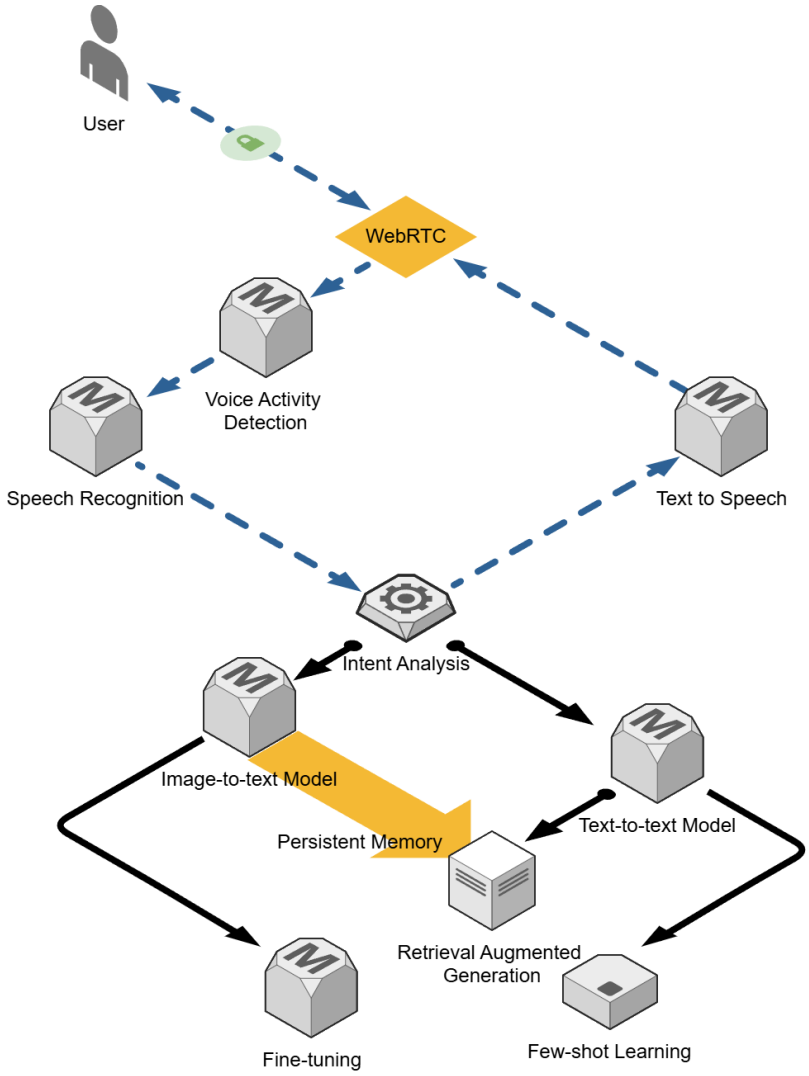

Figure 2.1: Extensible voice assistant framework

up of image-to-text and text-to-text model with some customization techniques. By utilizing the LLMs in natural language processing, the text obtained in the previous step is then passed to this part so that user's intent and keywords are extracted from the recognized speech. This module is responsible for understanding and generating appropriate responses, as well as determining which downstream tasks should be executed by using few-shot learning techniques.

5. **Text-to-text Model**
   As a part of Intent Analysis module, the text-to-text model performs natural language understanding and information extraction. It is also able to express

the comprehension from raw data and organize the speech in a fluent manner.

6. **Image-to-text Model**
   This module processes and understands the semantics of the input image, including Optical Character Recognition (OCR) and captioning.

7. **Retrieval Augmented Generation**
   This part extends the LLMs with the memorization generated by Vision Language Models from a database or external sources. As the yellow arrow shown in Figure 2.1, the descriptions in text form generated by image-to-text model are automatically recorded into a vector database, which enable the functionality of reviewing and summarization by the text-to-text model.

8. **Text to Speech**
   After processing the input and generating the appropriate response or performing the required task, the text output is converted back into speech using a Text to Speech model, and then communicated back to the user through WebRTC.

2.1.1 Front-end

Although our voice assistant is designed with a standard interface that various forms of client are able to access it, we implement it using Flutter [7] for the project demonstration, compiling it into WebAssembly [8]. Flutter is an open-source User Interface (UI) framework supporting compiling one single codebase into native code on web browser, desktop, and mobile systems. The high efficient rendering engine enables rapid response on user interaction and calculation, shipping with a unified native Application Programming Interface (API) and a set of modern UI widgets.

Once launching the application in web browser, it requests permissions to use the camera and microphone. Users will receive a voice notification that indicates whether the connection has been established or disconnected. In addition to web applications and mobile phones, the client can also be integrated into head-worn devices, such as smart glasses and helmets, providing additional feedback through vibration and tactile sensations.

2.1.2 Back-end

1. **Inference Service**

This service provides inference for LLMs and can be deployed on edge computing devices, personal computers, and cloud servers. The following hardware specifications are only intended for the demonstration environment.

- **GPU**
  The minimal requirement of GPU memory capacity is at least 11 gigabytes, based on the total parameters of LLMs, while the training machine for fine-tuning is equipped with dual NVIDIA A6000 GPUs.
- **CPU and RAM**
  CPU: AMD Ryzen Threadripper PRO 5945WX 12-Cores, 24 threads. RAM: 32 gigabytes
- **Operation System**
  Ubuntu is the preferred choice due to the well-developed support of GPU driver and compatibility with various deep learning framework.
- **Software Platform**
  We are utilizing PyTorch 2.4.1 as our deep learning framework, along with Python 3.12 and CUDA 12.4, running in Miniconda virtual environment [9] [10] [11] [12]. NumPy library is used for matrix calculation in image and audio processing [13].

2. **WebRTC Relay Server (Optional)**
   The relay server is deployed to address some challenges while the service side is running in a different machine from the client such as a data center. The server and client have difficulty in negotiation due to the specific network circumstance. In such cases, the relay server is utilized for relaying audio and video streams. However, this technique will increase the latency between the server and client.

2.1.3 Minimal Requirements

Our voice assistant can be deployed on any devices that support NVIDIA CUDA framework, including laptop, personal computers and edge embedded computers, with at least 11 Gigabytes GPU memory. GeForce RTX 3060 12GB and GeForce RTX 4070 12GB are the cost-effective options with the price starting from $399 at the end of 2024, which are affordable for visually impaired and have a major market share holder in personal computers.

We can also utilize the microphone, the speaker and the camera of our mobile phone as the hardware for capturing images and interaction. By developing an application on the phone, we connect the inference service through WiFi sharing or USB cable. This solution would further reduce the cost.

2.1.4 Audio and Video Processing

- **Media Segmentation**
  The audio and video streams are divided into chunks after decoding based on media frames, with audio chunks 20 ms in duration and video frames around 33 ms. The video frames are cached for later retrieval, while the audio frames are concatenated and adjusted to 32 ms to align with the input size required by the Voice Activity Detection model.

- **Audio Resampling**
  The default audio sampling rate for WebRTC is 48,000 Hz in stereo, while the Voice Activity Detection model only can process 16,000 Hz in mono. Therefore, it is necessary to resample the audio chunks to fit the input requirements of the Voice Activity Detection model.

- **Valid Audio Segment**
  At the stage of Voice Activity Detection, we maintained a state machine with cooldown and warm-up period. θ represents the confidence level of the audio frame being spoken and $\sigma 1$ denotes the count of frames for the warm-up period, while $\sigma 2$ denotes the count of frames for the cooldown period. As shown in Figure 2.2, the states inside the dot line can be thought as the valid audio segment. Table 2.1 shows the parameters for the state machine that is working in our functional prototype.

θ	$\sigma 1$	$\sigma 2$
0.6	0.2	0.7

Table 2.1: Parameters for the valid audio segments

The reason of including the changing state of speaking and non-speaking is to ensure that no valid speech is missed in the audio chunk. Specifically, even

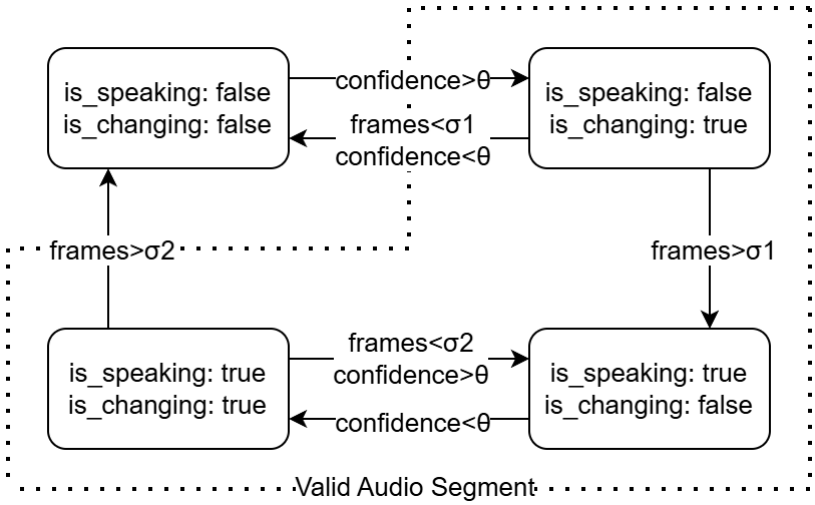

Figure 2.2: State machine of voice activity detection

the nearest 512 samples before the state of changing to speaking were cached circularly for robustness.

- **Audio Padding and Idle Generation**
  Once the output speech is generated by the text-to-speech model, the audio data will be upsampled and converted into a stereo stream to meet the WebRTC standard requirements. Since the audio length may not always be an integer multiple of the desired frame size, silence bytes will be appended to the end of the audio data as padding. Similarly, if there is no voice output, idle audio frames will also be generated. Throughout the entire audio stream, a timestamp must be maintained to ensure the correct sequence of audio packets.

- **Video Processing**
  There is no need to send the processed image back from the server since the desired user is visually impaired, therefore, only audio data will be transmitted to the client. For demonstration purposes, the application will include a local real-time video preview. The video sent to the server will also be trimmed as the image-to-text model input size to save bandwidth and speed up the processing.

2.1.5 Protocol

Since Python has become the most popular programming language in the field of machine learning [14], we have chosen it for all back-end model runtime. Additionally,

most open-source models are trained and run using Python, making a natural choice for our main programming language to use these modules seamlessly.

Although Python can handle threads, the Global Interpreter Lock (GIL) restricts its performance to utilize all CPU cores effectively. As shown in Figure 2.3, there is actually only one thread runing simultaneously. To address this limitation, we create multiple processes for each compute-intensive model, allowing them to run independently. These models communicate with each other via Transmission Control Protocol (TCP) socket connections. If the models are deployed on different machines, a TLS-encrypted connection can be implemented to ensure data security.

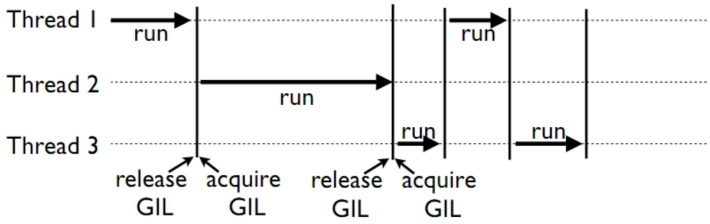

Figure 2.3: How multiple threads executes in Python ruled by GIL [15]

The preliminary data types transmitted over the TCP connection will include integers, bytes, and UTF-8 strings, covering images and commands. The following Table 2.2 illustrate the packet structure for each type of data.

Integer
big-endian bytes converted from integer

Bytes
big-endian bytes

Size	**String data**
big-endian bytes converted from integer	big-endian bytes of UTF-8 string

Table 2.2: Integer, Bytes, and UTF-8 string type packet

Due to the long generation time and extended playback duration of audio, we use asynchronous response to reduce latency in the progress of generating audio by text-to-speech model. As shown in Figure 2.4, while the audio chunk for Text 1 is still being generated, Text 2 is queued without waiting for a socket signal.

In this scenario, we only focus on the output audio chunk without checking whether the previous speech generation has completed. However, if the user wishes to stop

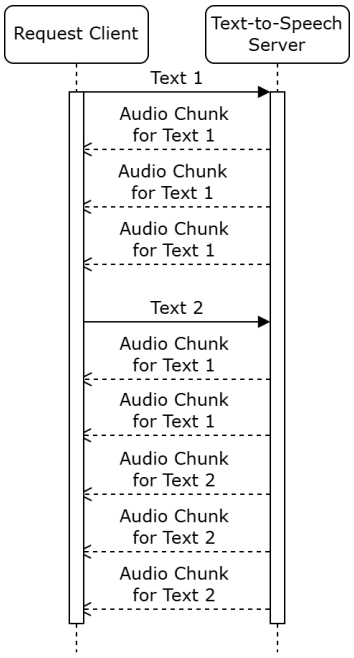

Figure 2.4: Asynchronous response in the text-to-speech model

the current speech, we must clear the queue of audio chunks that have not yet been encoded and sent to the WebRTC interface. To achieve this, an empty data packet is involved to indicate the asynchronous model to terminate the audio generation inference, representing that all subsequent packets are invalid. An event object is used as a bridge to notify that the generating loop should be stopped. Until the client eventually receives the empty packet, listening to the generated audio is resumed to normal state.

On the other side, all audio input should be ignored except for the stop speech command while the output is playing. Overall, this complicated interaction of voice and speech enhances the response speed, improving the experience of real-time communication.

2.1.6 Challenges and Solutions

During development, we encountered some performance issues. The main reason is related to CPU consumption. With a single user connection, CPU usage reached nearly 50% of total cores, causing lag and missed responses to user commands while playing generated speech.

To address this issue, we used a Python library called cProfile to analyze perfor-

mance during the entire runtime of specific code segments [16]. The sorted results highlights that the Just-In-Time (JIT) implementation of the Voice Activity Detection model was the primary source of CPU overhead. Consequently, we switched to the Open Neural Network Exchange (ONNX) model accelerator, which reduced CPU consumption to approximately 45% in one single core.

Although we significantly improved efficiency, the resampling function still consumes enormous resources due to an inefficient algorithm. By analyzing the strategy of audio downsampling, we observed that during the conversion from stereo to mono, we can simplify the calculation by using the mean value of samples, where the count is double the integral ratio of the source sample rate to the destination sample rate, eliminating the need for trigonometric functions while still achieving nearly the same audio quality.

We also noticed that the video encoding for the VP8 [17] encoder occupies 22% CPU time slices. Since people with vision loss do not rely on images from the server, we decided to omit the video component at the user side.

2.2 Customizing the LLMs

The pre-trained model typically requires significant costs in terms of hardware and training time, as well as the large number of data collections. To meet our specific application, we focus on customizing the LLMs based on open-source models to improve accuracy and performance for particular tasks. The following methods are two primary approaches to customization. While fine-tuning generally performs higher quality results comparing to few-shot learning and retrieval augmented generation, it still demands the same hardware resources as pre-training, longer training periods, having the risk to overfit.

2.2.1 Few-shot Learning

We use few-shot learning for intent analysis, task and object extraction, as LLMs can effectively handle these tasks with minimal data requirements. However, due to the limited token capacity, it is crucial to design prompts with an appropriate number of examples. The following case shows how we label our downstream tasks and extract optional keywords enclosed in brackets.

```
-Goal-
Given a user's input, classify the intent and append object
if the action is in [LOCATION]. If the intent does not match
any of the actions, output [NOT_RELATED]. Actions:  [CAPTION],
[LOCATION], [OCR], [STOP], [CONTINUE], [NOT_RELATED].

-Example-
Input:  What is this
Output:  [CAPTION]
Input:  Describe the picture
Output:  [CAPTION]
Input:  What can you see
Output:  [CAPTION]
Input:  Where is my key
Output:  [LOCATION](the key)
Input:  Find the red cup
Output:  [LOCATION](the red cup)
Input:  Tell me the location of the hat that is not on the wall
Output:  [LOCATION](the hat which is not on the wall)
Input:  Read the text
Output:  [OCR]
Input:  Can you see the labels
Output:  [OCR]

-Test-
Input:  <QUESTION>
Output:
```

In the prompts, we can indicate a off-topic response with a keyword such as NO_RELATED. If there are too many tasks that exceed the maximum context tokens, we can request another intent analysis to obtain the answers sequentially.

2.2.2 Retrieval Augmented Generation

Typically, the length of context tokens in most LLMs is from 4096 to 8192. It is not practical to include the entire history of previous image descriptions in a single infer-

ence. To solve this problem, we utilize the retrieval augmented generation technique, which extends the external knowledges as a form of long-term memory. Furthermore, retrieval augmented generation provide better performance on attention compared to long context tokens in LLMs [18].

As shown in Figure 2.5, the workflow for retrieval augmented generation is different from querying LLMs directly. The first step is embedding the query into vectors, followed by searching for the most similar item in the vector database using a specific algorithm, as illustrated in Formula 2.1. A_i and B_i denotes the position of each element of the two vectors A and B. The numerator $\sum(A_i \times B_i)$ calculates the dot product of the two vectors, measuring the degree of them aligning in the same direction. The denominator calculates the product of each vector's length. Consequently, d denotes the cosine similarity between two queried vectors, with a range from 0 to 2. The closer the value is to 0, the more similar the vectors are. After retrieving the context in the previous step, it is processed with LLMs to generate proper response.

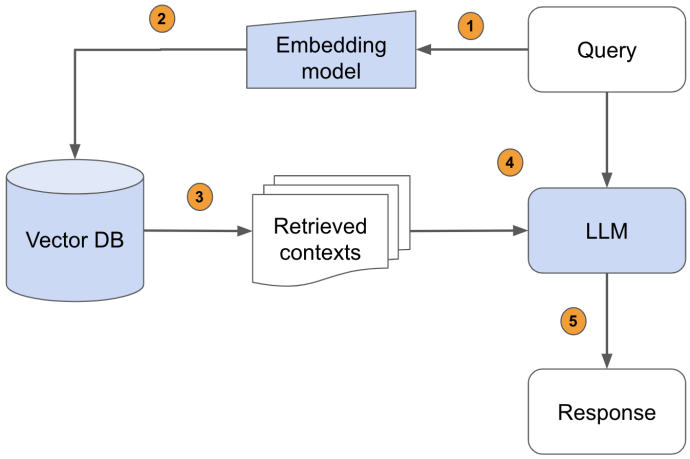

Figure 2.5: Retrieval augmented generation workflow [19]

$$d = 1.0 - \frac{\sum(A_i \times B_i)}{\sqrt{\sum(A_i^2)} \cdot \sqrt{\sum(B_i^2)}} \tag{2.1}$$

In our project, we chose Chroma [20] as it is an open-source AI embedding database specifically designed for building LLMs applications. Each session, the image description is automatically recorded in the Chroma database while in captioning and locating mode. When the user query the location of specific object, we firstly check whether it exists in the current image, otherwise, we extract the keyword and

query the history in the vector database. In case of oversize data, we periodically clean up outdated similar images to enhance the timeliness of information using the same algorithm of cosine similarity.

Chapter 3

System Modules

In this section, we elaborate on the principles of each module. Considering the system design, we prefer using multiple compact models for specific downstream tasks that are both flexible and extensible, rather than an end-to-end model. This approach allows us to add new features such as indoor navigation without retraining the entire model.

3.1 WebRTC

WebRTC is a unified solution for real-time communication with a standard protocol developed by Google LLC. It supports video calling and data exchange between two clients, optimized for various types of networks and devices, including desktops, mobile phones, and web browsers. The adaptive bandwidth design enhances call quality. especially in unstable network conditions.

Figure 3.1 shows the negotiation process between two WebRTC clients. At the beginning, the clients query the STUN server to check whether they are having a public Internet Protocol (IP) address. Subsequently, they exchange Session Description Protocol (SDP) messages, which describe media encoding, encryption, and other session information. After that, they share their public and local IP addresses named ICE candidate through a signaling server to determine the best network for data transmission. Once the connection is established, the video and audio stream can dynamically adjust, giving a seamless experience to users.

However, if there are issues such as firewalls or Network Address Translation (NAT) device strategies (e.g., symmetric NAT) that prevent direct connections, a

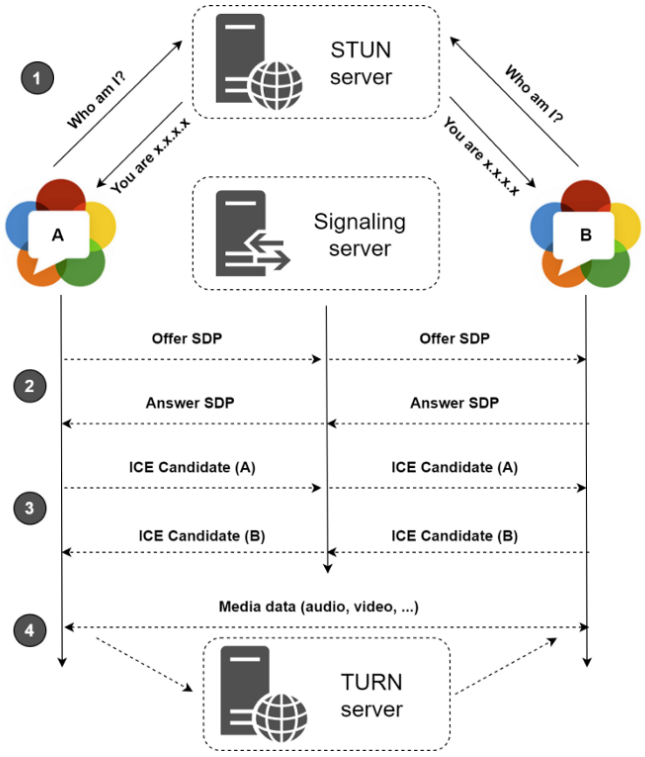

Figure 3.1: How WebRTC works
[21]

TURN server is involved to relay the communication, ensuring that both clients can connect even no valid pathway exists.

3.2 Voice Activity Detection

SileroVAD is preferred for voice activity detection task [22] in our project. It is a lightweight, pre-trained model that supports many languages and effectively distinguishes between various background noises and sound levels. While traditional algorithms like webrtc-vad [23] has been widely used in Python development for WebRTC, there is a growing trend that developers prefer to utilize SileroVAD based on nerual networks due to its better accuracy than webrtc-vad.

The model supports mono audio input at sampling rates of 8000 Hz and 16000 Hz, built on Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) network, enabling real-time segmentation on CPU.

3.3 Text to Speech

MeloTTS [24] is a lightweight and high-efficient text-to-speech engine based on VITS [25]. It supports multiple languages and can convert text into natural, fluent voice with various emotional styles. The VITS architecture is made up of two main components, Variational Autoencoder (VAE) and Generative Adversarial Network (GAN).

3.3.1 Variational Autoencoder

VAE compresses the data into a latent space with characteristic features and then reconstructs the data by decompressing it was proposed. Figure 3.2 shows this process, known as encoding and decoding.

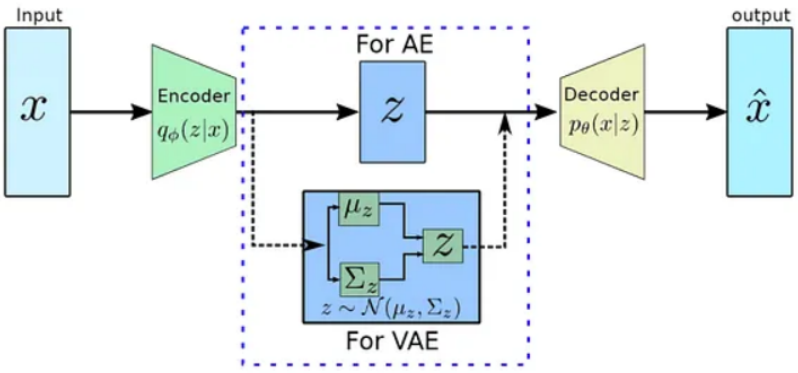

Figure 3.2: Variational Autoencoder comparing to Autoencoder [26]

For an autoencoder (AE), the information stored in latent space is typically a discrete value, meaning the space Z only represents what it has exactly seen during the training stage. As a result, the decoder struggles with poor generalization and a lack of diversity. For a VAE, the latent space incorporates a normal distribution with two parameters, mean μ_z and standard deviation Σ_z, to express the probability of data characteristics.

As shown in Figure 3.3, the decoder reconstructs data by sampling from the latent space, which enables the model to derive gradients that converts the predefined latent state distribution into a trainable representation. Kullback-Leibler divergence [27] is used in the loss function to facilitate backpropagation.

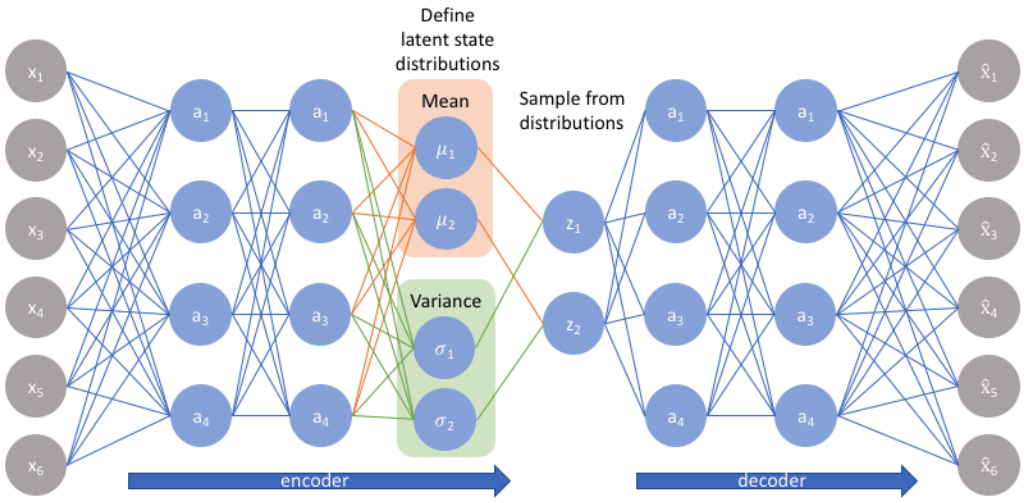

Figure 3.3: Encoder and decoder of VAE network
[28]

3.3.2 Generative Adversarial Network

As shown in Figure 3.4, two neural networks are built in the form of a minimax game, one is the generator G and the other is discriminator D. Here, x denotes the real voice, and z denotes the noise involved in the generation process $G(z)$. The generator aims to create higher fidelity voice while the discriminator tries to distinguish the differences between real and synthetic voice. During the training iterations, the generator and discriminator continuously evolve, competing against each other until reach a balanced state, referred to as Nash equilibrium. In this state, the discriminator can no longer recognize whether the generated voice is real or fake. Mathematically, that is the probability distribution of the real data $p_{data}(x)$ is equal to the one of generated data $p_g(x)$, and get the global optimum output value of 1/2.

The training process of GAN follows these steps. It randomly samples a vector from a latent space as the input for the neural network generator v1. Then the generator produces a synthetic voice that is evaluated against real voice by the discriminator v1. The discriminator outputs a probability score $D_G(x)$, ranging from 0 to 1, indicating a fake voice and a real voice, respectively.

$$D_G(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \tag{3.1}$$

Subsequently, the discriminator is continually trained using voice generated by the updated generator v2, with the goal of classifying generated voice as close to 1 as

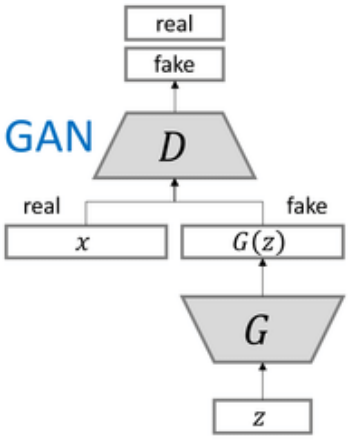

Figure 3.4: Generative Adversarial Network
[29]

possible, meaning that both networks are refined and cooperate towards producing high-fidelity voice.

To optimize both the generator and the discriminator in opposing directions, we chose a value function as such,

$$\mathbb{E}_{x\sim p_{data}}[\log D_G(x)] + \mathbb{E}_{x\sim p_g}[\log(1 - D_G(x))] \tag{3.2}$$

where $\mathbb{E}_{x\sim p_{data}}$ and $\mathbb{E}_{x\sim p_g}$ are the expectation of real data and generated data respectively, the logarithm is applied on the probability score of discriminator $D_G(x)$ for smooth gradients, with cross-entropy loss used. During GAN training, the discriminator learns to increase the value of expectation on real data $\mathbb{E}_{x\sim p_{data}}[\log D_G(x)]$ while decreasing the value on generated data $\mathbb{E}_{x\sim p_g}[\log(1 - D_G(x))]$, resulting the generator to improve its outputs to reach the high-fidelity level.

3.4 Introduction to Transformer

The following 3 modules are all built on Transformer architecture. Before introducing these models, let's explore the fundamentals of Transformer.

In the early stage of deep learning, neural networks had poor performance in processing text and audio data. To solve this issue, Recurrent Neural Networks (RNNs) [30] were proposed. They used recurrent units to capture the hidden state in sequential data. However, there was a critical problem that the gradient might

vanish or explode during the training period due to the limitation of back propagation. To overcome this challenge, the variant network LSTM [31] was proposed with the capability to handle the long-range dependencies. As the sequence length increased, the gradient problem shows again, and the time consumption of training and inference became significantly longer than usual.

Subsequently, the Transformer model changes the rules in the field of natural language processing. As the fundamental component of generative AI, the Transformer captures the context of sentences by utilizing multi-head self-attention and positional encoding mechanism, eliminating the sequential dependency of time steps, allowing it can process long data in parallel without losing contextual relationship throughout the text.

Figure 3.5 illustrates the Transformer model architecture. The inputs and the outputs are the two parts of the dataset during the training stage of the model. The outputs can be anything related to the input data, for instance, a shifted right version of input, translation of input, or even the generated image. After the model calculates the output probabilities, which represent the distribution of the next token, we concatenate the desired output with the input for the next iteration. In this way, the Transformer could generate diverse and relevant text and functions like an intelligent assistant for analyzing articles.

3.4.1 Input Embedding

To convert the language comprehension problem into a mathematical problem, we need to encode the words to a digital form. Normally, we build an N-sized vocabulary, where each element is a vector in which all N entries are 0 except for the entry corresponding to the specified word, which is 1, called a one-hot vector. Unfortunately, this method cannot express the similarity between the words. To solve this problem, word2vec [33] was proposed. As shown in Figure 3.6, similar words are positioned close to each other in the multi-dimensions space. Any group of words with the same vector offset reaches the consistency at the semantics level, meaning we can migrate and express the similar semantics by operating on vectors mathematically.

3.4.2 Positional Encoding

Now, we have our words encoded with word2vec. Unlike RNNs, where order is preserved, the Transformer architecture uses disordered tokens for parallel acceleration,

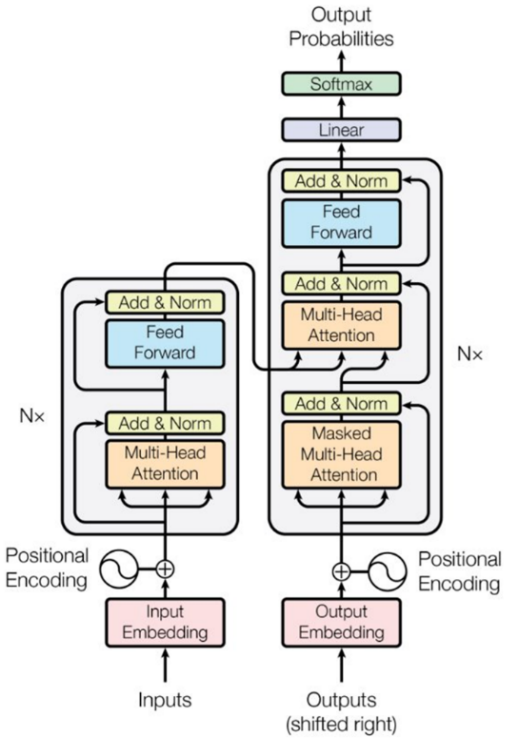

Figure 3.5: Transformer model architecture
[32]

resulting in the loss of positional information after input. By adding Positional Encoding (PE) information to the word embedding vector, the input vector retains the sequential order and semantics before processing. Formula 3.3 and 3.4 shows the method of sinusoidal and cosinusoidal position encoding respectively,

$$PE(pos, 2i) = \sin\left(pos/10000^{2i/d_{model}}\right) \tag{3.3}$$

$$PE(pos, 2i+1) = \cos\left(pos/10000^{2i/d_{model}}\right) \tag{3.4}$$

where pos represents the position index of input sentence, $2i$ and $2i$+1 are the dimensions indices, and d_{model} denotes the number of dimensions in the stage of word embedding. Choosing 10000 as the denominator avoids periodic value within the reasonable range of dimensions, as shown in Table 3.1 and Figure 3.7.

In position encoding, as the position indices increase, the digits in higher position change less frequently comparing to those in lower position. Therefore, we use the

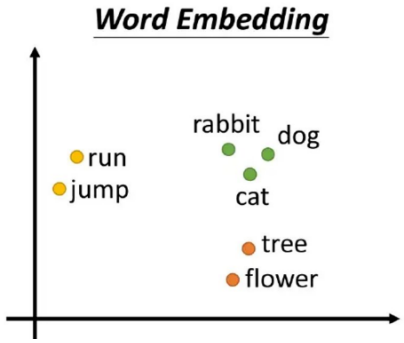

Figure 3.6: Word embedding in two dimensions
[34]

	dim0	dim1	dim2	dim3	dim4	dim5
pos0	$\sin(\frac{0}{10000^{\frac{0}{6}}})$	$\cos(\frac{0}{10000^{\frac{0}{6}}})$	$\sin(\frac{0}{10000^{\frac{2}{6}}})$	$\cos(\frac{0}{10000^{\frac{2}{6}}})$	$\sin(\frac{0}{10000^{\frac{4}{6}}})$	$\cos(\frac{0}{10000^{\frac{4}{6}}})$
pos1	$\sin(\frac{1}{10000^{\frac{0}{6}}})$	$\cos(\frac{1}{10000^{\frac{0}{6}}})$	$\sin(\frac{1}{10000^{\frac{2}{6}}})$	$\cos(\frac{1}{10000^{\frac{2}{6}}})$	$\sin(\frac{1}{10000^{\frac{4}{6}}})$	$\cos(\frac{1}{10000^{\frac{4}{6}}})$
pos2	$\sin(\frac{2}{10000^{\frac{0}{6}}})$	$\cos(\frac{2}{10000^{\frac{0}{6}}})$	$\sin(\frac{2}{10000^{\frac{2}{6}}})$	$\cos(\frac{2}{10000^{\frac{2}{6}}})$	$\sin(\frac{2}{10000^{\frac{4}{6}}})$	$\cos(\frac{2}{10000^{\frac{4}{6}}})$
pos3	$\sin(\frac{3}{10000^{\frac{0}{6}}})$	$\cos(\frac{3}{10000^{\frac{0}{6}}})$	$\sin(\frac{3}{10000^{\frac{2}{6}}})$	$\cos(\frac{3}{10000^{\frac{2}{6}}})$	$\sin(\frac{3}{10000^{\frac{4}{6}}})$	$\cos(\frac{3}{10000^{\frac{4}{6}}})$

Table 3.1: Position encoding for different positions and dimensions

same mechanism to represent positional information in sinusoidal positional encoding. Using sin and cos functions interchangeably could widen the difference between the adjacent positions.

3.4.3 Multi-Head Attention

This part involves the attention module which contains three input vectors, query (Q), key (K) and value (V) respectively. These vectors are obtained by transforming the encoded input vector though multiplication with a weight matrix. The weight matrix can learn the relationships between short and long-distance dependencies through backpropagation during the training. After that, the Q, K and V are sent to an h-head scaled dot-product attention module, followed by a linear layer. This operation reduces the computational cost and the number of parameters in the matrix, breaking down the knowledge block into several representation subspaces by linear projections. Figure 3.8 shows that those output of the attention heads are concatenated and aggregated into one linear layer, enabling the model to capture diverse perspectives

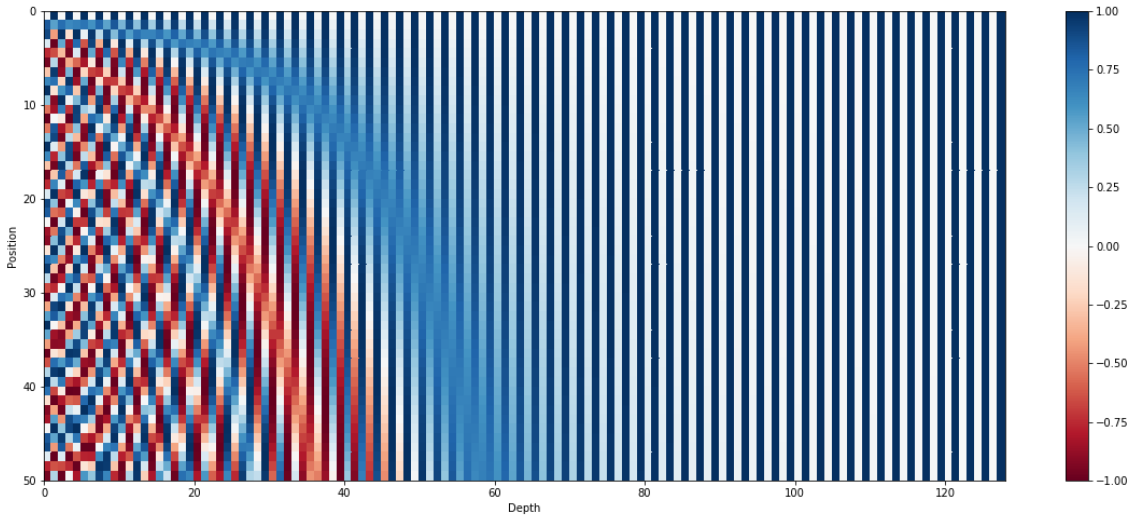

Figure 3.7: Heatmap for different positions and dimensions [35]

and better understand the input sequence.

3.4.4 Scaled Dot Product Attention

In this part, we apply linear operations on Q, K and V vectors. As shown in Formula 3.5, the input value of the SoftMax function is a matrix representing the similarity between Q and K vectors, calculated by their dot product.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.5}$$

Here d_k denotes d_{model}/h, h is the head count, K^T denotes the transposed matrix of K. The squared root of d_k is used to avoid gradient vanishing, scaling the value of evaluation function into an appropriate range. In Figure 3.9, the matrix, which highlights the most relevant features using the SoftMax function, is multiplied by the V vector to obtain a weighted sum of the value vectors.

3.4.5 Position-wise Feed Forward

Returning to Figure 3.5, we see a Multilayer Perceptron (MLP) following the multi-head attention. This is a non-linear Feed Forward Network (FFN), containing two linear layers with a ReLU activation and dropout function in between. Formula 3.6

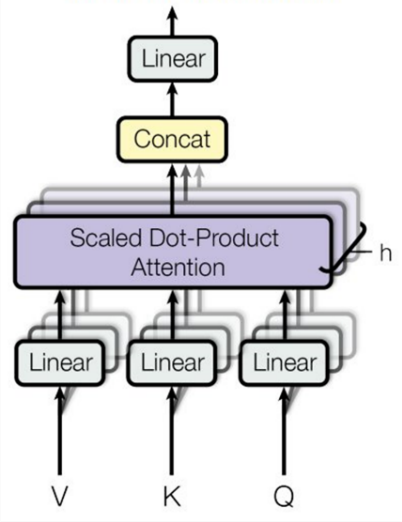

Figure 3.8: Multi-head attention and linear transform
[32]

illustrates the process of calculation.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3.6}$$

where x denotes the input, W_1 and W_2 denote the trainable weighted matrix, and b_1 and b_2 are the bias.

Since the position of the input is embedded in the encoding, the calculation is applied to each word in the sequence independently without considering the position relationship. This mechanism enhances the model's ability to capture complex patterns in the data that the dot product operation cannot.

3.4.6 Add and Normalize

We observe that after every operation of the feed forward and multi-head attention modules, there are addition and normalization steps. The addition step involves adding the input of the modules to the output, a process known as residual connection. This helps mitigate the problem of vanishing gradients and preserves the features of the original data. After the addition, layer normalization is applied to the residual result. Normally, we use batch normalization in deep learning to stabilize the training. However, batch normalization might disrupt the consistent relationships within a layer by affecting the sequence integrity. Figure 3.10 provides a concrete visualization

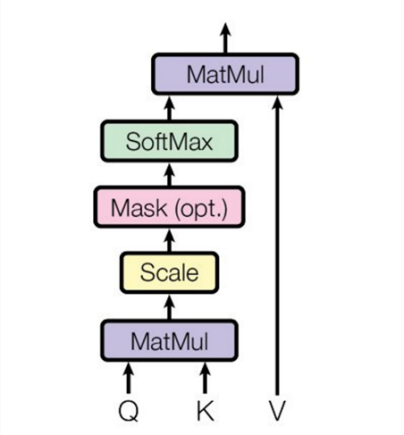

Figure 3.9: Scaled dot product attention
[32]

comparing batch normalization and layer normalization. Here H denotes height, W denotes width, C denotes the number of channels or dimensions, and N denotes the batch size.

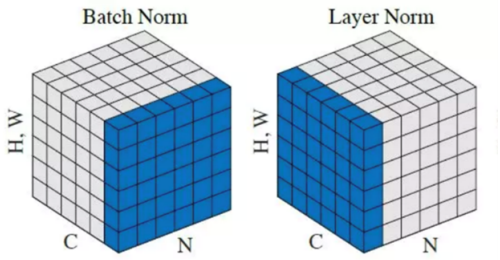

Figure 3.10: Batch normalization and layer normalization
[36]

The normalization operation involves the following steps. First, calculate the mean of input.

$$\mu_l = \frac{1}{d}\sum_{i=1}^{d} x_i \tag{3.7}$$

where d denotes the dimension, μ_l denotes the mean of input x_i. Next, calculate the variance σ_l^2 based on the mean μ_l.

$$\sigma_l^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu_l)^2 \tag{3.8}$$

We use Formula 3.9 to get the normalized value $\hat{x}_i$,

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \tag{3.9}$$

The ϵ is a tiny constant to ensure that the denominator is not zero. The final step is to scale and translate the normalized value using Formula 3.10.

$$y_i = \gamma \hat{x}_i + \beta \tag{3.10}$$

The initial value of γ and β are typically set to 1 and 0, respectively, they are trainable parameters for enhancing the model's expressiveness.

3.5 Speech Recognition

We use Whisper as the speech recognition engine, which is a Transformer-based sequence-to-sequence model open-sourced by OpenAI Inc., showing human-level robustness and accuracy. In addition to speech recognition, it also features translation capabilities. The architecture is illustrated in Figure 3.11. The audio input is processed through two convolutional layers followed by a gelu activation function.

3.6 Text-to-text Model

For intent analysis and information extraction, we chose Gemma-2 [38] as our text-to-text model. Unlike traditional Transformer architectures, Gemma-2 is a decoder-only model that performs well in text generation and translation. The smallest version, with 2 billion parameters, is adequate for our instructional needs.

3.7 Image-to-text Model

In the field of image-to-text generation, OCR is an early technique for extracting text from images that contains printed or handwritten text. Besides OCR, image captioning and segmentation are two other fundamental tasks, involving describing the visual content and understanding the target image. As a lightweight vision-language model, Microsoft Corporation open-sourced the Florence-2 [39] project, which has capabilities for a range of tasks such as captioning, OCR, classification, object detection,

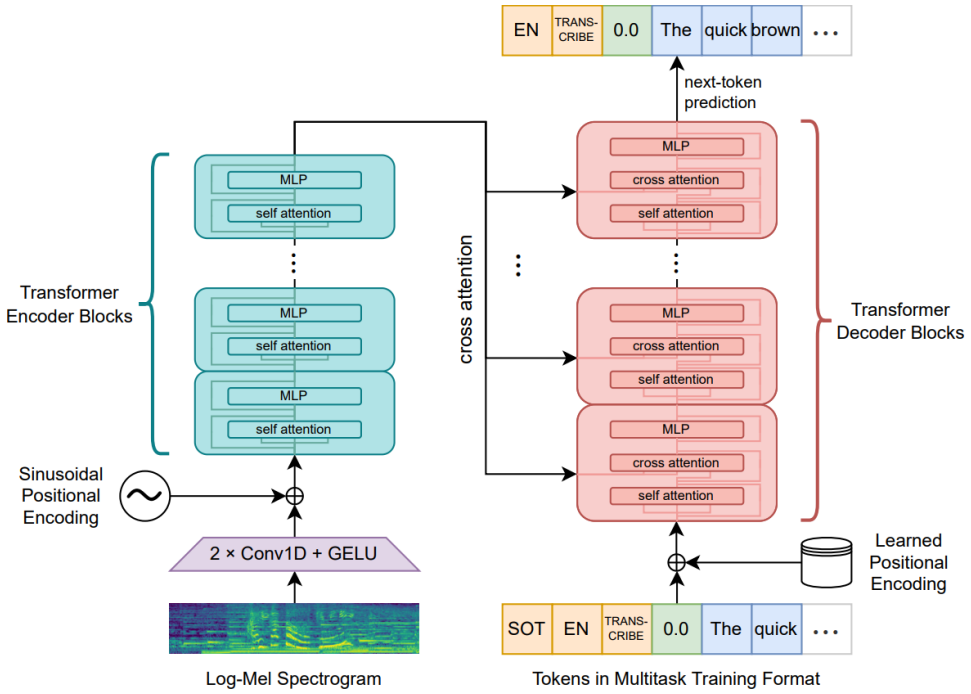

Figure 3.11: Architecture of Whisper [37]

and segmentation, covering most of the popular tasks in image-to-text generation. We selected it as our image-to-text model due to its memory requirement of only 1.8 gigabytes, along with the potential for fine-tuning in future developments.

This model is based on the Transformer architecture, encoding the image into visual embeddings by Dual-attention Vision Transformer (DaViT) while also encoding multi-task prompts as text and location embeddings, as shown in Figure 3.12. Florence-2 formulates each task as a translation problem to generate the response. The overall value function $\mathcal{L}$ is defined as follows,

$$\mathcal{L} = -\sum_{i=1}^{|y|} \log P_\theta(y_i | y_{<i}, x) \tag{3.11}$$

where θ denotes the model parameters, $|y|$ denotes the length of the output sequence y, y_i represents each element of y. $P_\theta(y_i|y_{<i}, x)$ means the conditional probability of predicting the element y_i by giving all previous elements $y_{<i}$ and the input sequence x.

Specifically, for tasks of processing regions, Florence-2 adds location tokens to the tokenizer's vocabulary list so that the model can produce location coordinates. For

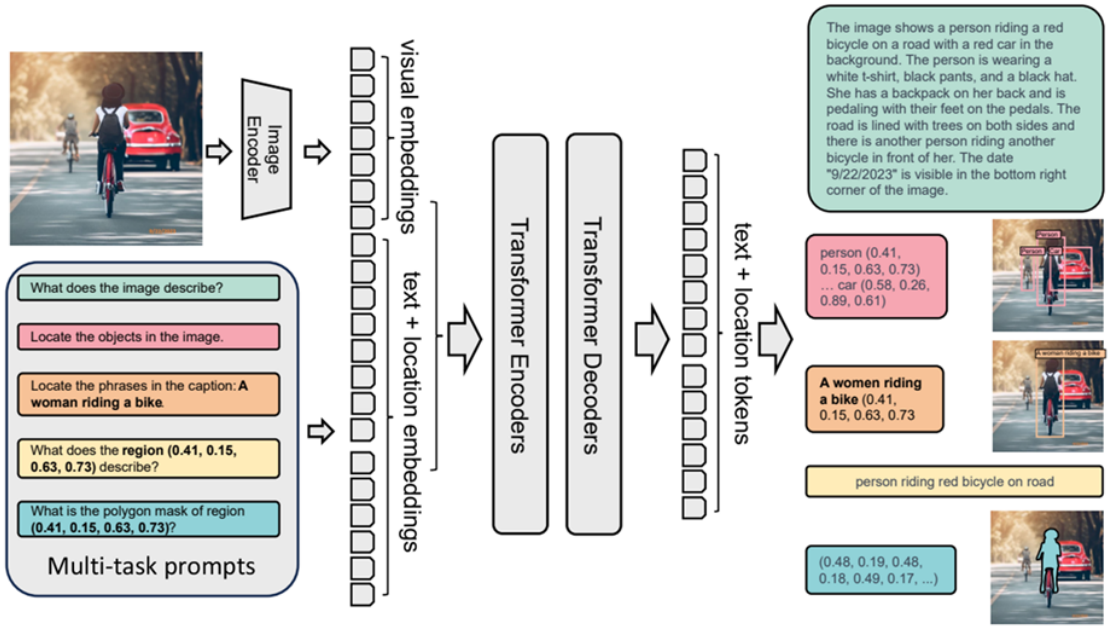

Figure 3.12: Architecture of Florence-2 [39]

the image embeddings in the multi-modality encoder and decoder, it adds a linear layer and layer normalization, concatenated with text embeddings for dimensional alignment.

3.7.1 Dual-Attention Vision Transformer

The DaViT [40] is a novel vision transformer model that involves dual-attention mechanism. The innovation architecture, as shown in Figure 3.13, enhances the capability to capture the global information in the images through two orthogonal attentions, including a spatial window for learning local characteristics and a channel group for learning global characteristics.

Specifically, for the spatial window multi-head self-attention, DaViT splits the spatial tokens into different windows, focusing on the spatial relationships between pixels in a specific region of the image, so that the model can process high-resolution images efficiently. On contrast, the channel group self-attention focuses on the relationships between different feature channels, transposing the tokenization from the spatial domain to the channel domain. Applying attention to the channels enhances the model's capability to capture the dependencies between different features such as

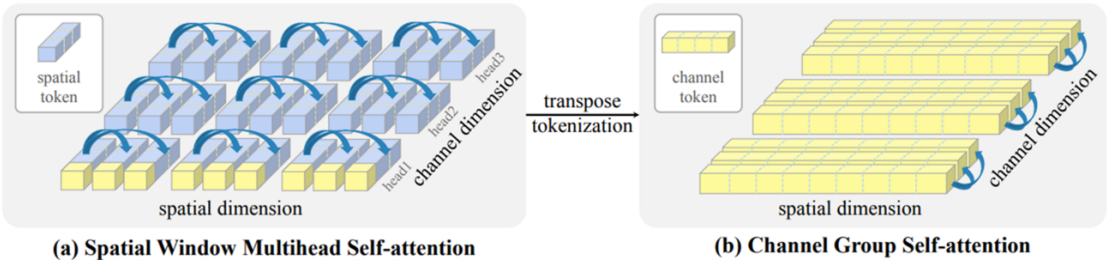

Figure 3.13: Spatial window and channel group self-attention in dual-attention vision transformer
[40]

edges, colors or textures in the image.

The network structure is illustrated in Figure 3.14, where P denotes the patches, which are similar to the tokens in text embeddings, created by splitting the image into several pieces. C denotes the dimensions, N denotes the number of items, while the subscript h denotes heads, w denotes window, and g denotes group. The projection module is applied to remap the original space to the target space, adapting it to the next input dimensions.

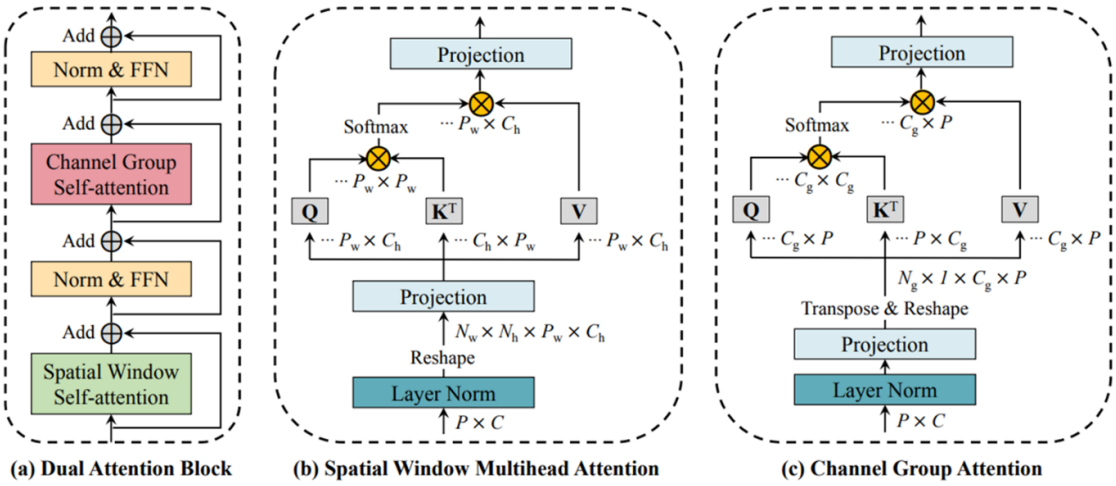

Figure 3.14: Dual-attention vision transformer network
[40]

Chapter 4

Demonstration

At present, our voice assistant supports following commands including captioning, locating, text recognition, interrupting speech output and continuing last action, as shown in Figure 4.1, 4.2, 4.3, 4.5.

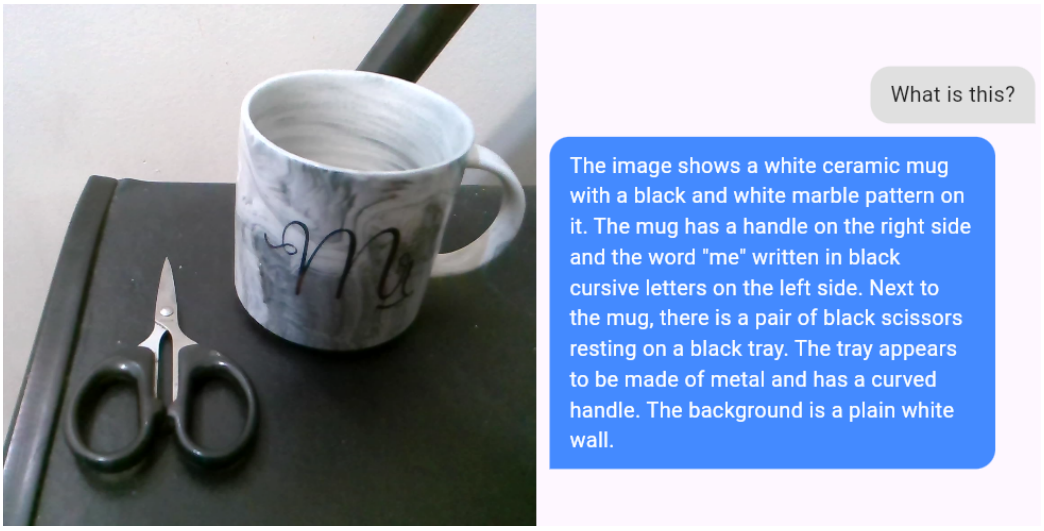

Figure 4.1: Image description

Figure 4.2: Locating item when it appears in the current frame

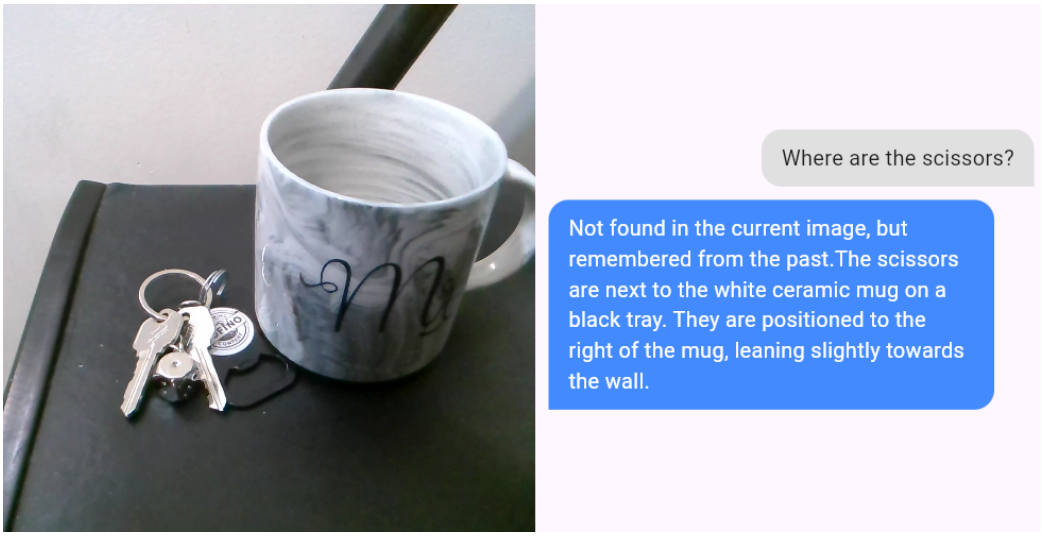

Figure 4.3: Locating item when it does not appear in the current frame by utilizing retrieval augmented generation technique

Figure 4.4: Text recognition

4.1 The Workflow of Locating Case

As shown in Figure 4.4, it illustrates the workflow of the last example mentioned above. When the user is asking a question about the image, the audio and video streams are split into several chunks and sent to the Voice Activity Detection module. The module identifies whether the user is speaking, automatically concatenating valid audio chunks into a segment. Once the piece of audio has arrived at Speech Recognition, the spoken input is processed to generate the text representation. Then the user's intent and keywords are extracted from the recognized speech. The LLMs can not find the item from the text generated by the image-to-text model but find it from the previous memory by retrieval augmented generation database. After that, it generates appropriate responses to the text-to-speech model. So that the user can hear the voice output.

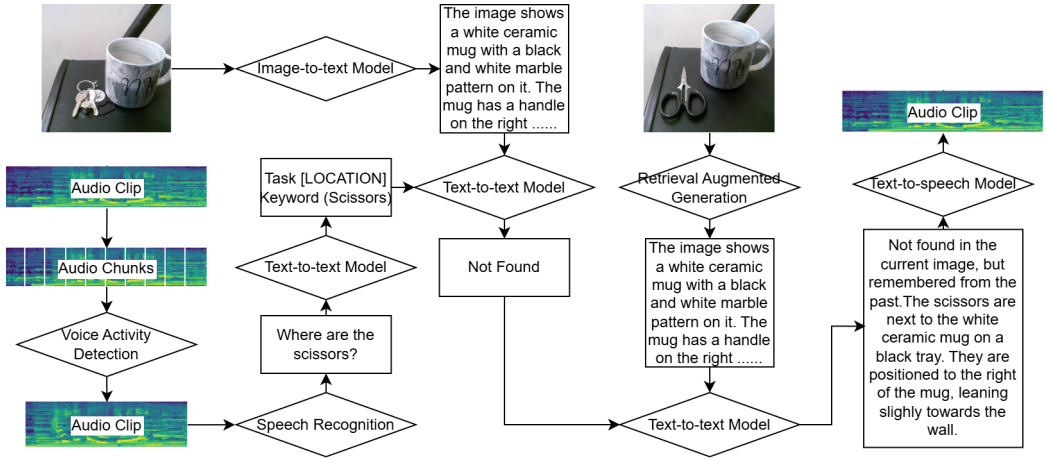

Figure 4.5: The workflow of locating case

4.2 Clean-up Duplicate Data

We now have three images in the same period time, each with the following text descriptions. The key difference is that the pen has been removed from the third image. We perform a similarity check between each pair of images, and the pair of first and second results the lowest similarity score shown in Table 4.2, indicating a higher probability that they were taken from the same position. Consequently, we can confidently remove the older item from the vector database to reduce duplicate data.

(1) The image shows a red desk with a black computer mouse, a pair of black scissors, and a black pen lying on it. The computer mouse is black and appears to be a wireless mouse with a round shape and a textured surface. The scissors are silver and have a curved handle. The pen is black with a silver tip and a clear body. The desk is also red and has a black cord attached to it.
(2) The image shows a red desk with a black computer mouse and a pair of black scissors on it. The mouse is black and has the brand name "Logitech" written on it in white letters. Next to the mouse, there is a black pen with a silver tip and a clear plastic tube. The scissors have a curved handle and sharp blades. The desk appears to be in a room with a wooden floor and a black wall in the background.
(3) The image shows a pair of black computer mouse and a black pair of scissors on a red surface. The mouse is on the left side of the image and the scissors are on the right side. The scissors have a curved handle and a pointed tip. The red surface appears to be a desk or table with a computer monitor in the background.

Table 4.1: The descriptions of three images

	Image (2)	**Image (3)**
Image (1)	0.09	0.119
Image (2)	-	0.164

Table 4.2: Similarity scores calculated by Formula 2.1

Chapter 5

Conclusions

Our voice assistant is a scalable framework designed to guide visually impaired in their daily life, offering functionalities such as reading labels, describing images, as well as locating and identifying items and products. It features low-latency speech-to-speech communication and can be deployed on edge computing devices.

By leveraging LLMs and Vision Language Models, the system offers a privacy-safe solution with cost-effective option, natural interaction, and extensibility for adding new features. We believe these advancements can significantly improve the quality of life for the 1.2 million visually impaired individuals in Canada.

5.1 Future Work

In future developments, we aim to enhance our system in several key areas. First, we will focus on fine-tuning the image-to-text model to improve the recognition of hand-held and touched objects, providing visually impaired individuals with clear operation instructions. The metrics of the recognition and response time of the whole system will also be measured to evaluate the performance and compare to other products that existing on the market. We might integrate some intrusion detection systems to ensure the data security when third-party manufacturers or Internet involved.

Additionally, integrating video analysis for indoor and outdoor navigation is essential, as it will enhance our ability to process time-series information. Operating electronic devices, such as remote controls or heating panels, will also become a higher priority. We plan to offer advice and step-by-step instructions, simplifying interactions and improving accessibility for users.

Furthermore, with advancements in the customization of LLMs, we will expand our support for more languages, ensuring individuals from diverse backgrounds can benefit from our system.

Bibliography

[1] Keith D. Gordon. *The Cost of Vision Loss and Blindness in Canada*. The Canadian Council of the Blind, May 2021.

[2] OpenAI. Gpt-4. `https://openai.com/index/gpt-4-research/`, 2023. [Online; accessed 21-Sep-2024].

[3] Be My Eyes. See the world together. `https://www.bemyeyes.com/about`, 2023. [Online; accessed 22-Sep-2024].

[4] OpenAI. Transforming visual accessibility. `https://openai.com/index/be-my-eyes/`, 2023. [Online; accessed 22-Sep-2024].

[5] NUSNEWS. Nus researchers develop ai-powered 'eye' for visually impaired people to 'see' objects. `https://news.nus.edu.sg/ai-powered-eye-for-visually-impaired-people/`, 2024. [Online; accessed 22-Sep-2024].

[6] Google for Developers. Webrtc. `https://webrtc.org/`, 2011. [Online; accessed 24-Sep-2024].

[7] Google LLC. Flutter. `https://flutter.dev/`, 2017. [Online; accessed 24-Sep-2024].

[8] WebAssembly. Webassembly. `https://webassembly.org/`, 2024. [Online; accessed 24-Sep-2024].

[9] The Linux Foundation. Pytorch. `https://pytorch.org/`, 2024. [Online; accessed 24-Sep-2024].

[10] Python Software Foundation. Welcome to python.org. `https://www.python.org/`, 2001. [Online; accessed 24-Sep-2024].

[11] NVIDIA Developer. Cuda toolkit - free tools and training. https://developer.nvidia.com/cuda-toolkit, 2024. [Online; accessed 24-Sep-2024].

[12] Inc Anaconda. Miniconda - anaconda documentation. https://docs.anaconda.com/miniconda/, 2018. [Online; accessed 24-Sep-2024].

[13] NumPy team. Numpy. https://numpy.org/, 2024. [Online; accessed 24-Sep-2024].

[14] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2020.

[15] Packt Publishing Limited. Learning concurrency in python. https://subscription.packtpub.com/book/programming/9781787285378/1/ch01lvl1sec13/the-limitations-of-python, 2024. [Online; accessed 28-Sep-2024].

[16] Michael Wagner, Germán Llort, Estanislao Mercadal, Judit Giménez, and Jesús Labarta. Performance analysis of parallel python applications. *Procedia Computer Science*, 108:2171–2179, 2017.

[17] Jim Bankoski, Paul Wilkins, and Yaowu Xu. Technical overview of vp8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*, pages 1–6. IEEE, 2011.

[18] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[19] Philipp Moritz Goku Mohandas. Building rag-based llm applications for production. https://www.anyscale.com/blog/a-comprehensive-guide-for-building-rag-based-llm-applications-part-1, 2023. [Online; accessed 26-Sep-2024].

[20] Chroma. Chroma. https://www.trychroma.com/, 2024. [Online; accessed 28-Sep-2024].

[21] Sagar Kava. Build your first flutter-webrtc app. `https://sagarkava.medium.com/build-your-first-flutter-webrtc-app-10cfff4b7b7c`, 2023. [Online; accessed 26-Sep-2024].

[22] Silero Team. Silero vad: pre-trained enterprise-grade voice activity detector (vad), number detector and language classifier. `https://github.com/snakers4/silero-vad`, 2024.

[23] John Wiseman. py-webrtcvad. `https://github.com/wiseman/py-webrtcvad`, 2024. [Online; accessed 28-Sep-2024].

[24] Wenliang Zhao, Xumin Yu, and Zengyi Qin. Melotts: High-quality multi-lingual multi-accent text-to-speech, 2024.

[25] Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech. In *International Conference on Machine Learning*, pages 5530–5540. PMLR, 2021.

[26] Sapna Naga. Unpacking variational autoencoders (vaes): A creative way to learn and generate, 2023.

[27] Wikipedia. Kullback-leibler divergence, 2024.

[28] Athiya Deviyani. Assessing dataset bias in computer vision. *arXiv preprint arXiv:2205.01811*, 2022.

[29] Ajkel Mino and Gerasimos Spanakis. Logan: Generating logos with a generative adversarial neural network conditioned on color. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 965–970. IEEE, 2018.

[30] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

[31] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[32] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[33] Tomas Mikolov. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[34] Hung yi Lee. Self-attention. `https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/self_v7.pdf`, 2021. [Online; accessed 20-Sep-2024].

[35] Amirhossein Kazemnejad. Transformer architecture: The positional encoding. `https://kazemnejad.com/blog/transformer_architecture_positional_encoding/`, 2019. [Online; accessed 20-Sep-2024].

[36] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[37] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR, 2023.

[38] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[39] Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4818–4829, 2024.

[40] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. Davit: Dual attention vision transformers. In *European conference on computer vision*, pages 74–92. Springer, 2022.