UNIVERSITI MALAYSIA SARAWAK
UNIMAS

Faculty of Computer Science and Information Technology

HEARSEE: A MULTIMODAL LARGE LANGUAGE MODEL-POWERED IMAGE-TEXT-TO-TEXT AND TEXT-TO-SPEECH WEB-BASED LEARNING TOOL FOR UNDERGRADUATE STUDENTS

FERDINAND MORIENTES ANAK RANTAI

Bachelor of Software

Engineering with Honours

(SE)

2024/2025

HEARSEE: A MULTIMODAL LARGE LANGUAGE MODEL-POWERED IMAGE-TEXT-TO-TEXT AND TEXT-TO-SPEECH WEB-BASED LEARNING TOOL FOR UNDERGRADUATE STUDENTS

FERDINAND MORIENTES ANAK RANTAI

This project is submitted in partial fulfilment of
the requirements for the degree of Bachelor of Software Engineering with Honours

Faculty of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK
2025

HEARSEE: ALAT PEMBELAJARAN BERASASKAN WEB UNTUK PELAJAR SARJANA MUDA YANG DIKUASAKAN OLEH MODEL BAHASA BESAR MULTIMODAL UNTUK IMEJ-TEKS-KE-TEKS DAN TEKS-KE-PERTUTURAN

FERDINAND MORIENTES ANAK RANTAI

Projek ini merupakan salah satu keperluan
untuk Ijazah Sarjana Muda Kejuruteraan
Perisian dengan Kepujian

Fakulti Sains Komputer dan Teknologi Maklumat
UNIVERSITI MALAYSIA SARAWAK
2025

Acknowledgement

I would like to express my heartfelt gratitude to my advisor, Dr. Azman, for his invaluable guidance, encouragement, and insightful feedback throughout the development of this project. I extend my thanks to Universiti Malaysia Sarawak for providing the tools and resources that made this work possible. Lastly, I thank everyone who directly or indirectly contributed to the completion of this project. Your support and contributions have been immensely appreciated.

Table of Contents

List of Figures

List of Tables

Abstract

HearSee is a web-based assistive learning tool designed to enhance undergraduate study by integrating multimodal large language models for image-text-to-text and text-to-speech processing. Implemented with a three-tier architecture utilising Gradio UI for presentation, Python service classes for application logic, and configuration utilities for data handling. The system leverages Qwen2-VL 7B via the Replicate API to perform OCR, generate descriptive captions, produce concise summaries, and support interactive Q&A on uploaded images. Processed outputs can be converted to natural-sounding speech using the Kokoro TTS engine (based on StyleTTS 2), with user-selectable voice profiles and adjustable playback speed. Deployed on Hugging Face Spaces, HearSee employs a prompt-engineering approach to optimize model responses and manages session state and temporary audio files for seamless user interactions. System efficacy is established through automated evaluation metrics—BERTScore for text quality and Mean Opinion Score for speech naturalness—alongside measured latency benchmarks. This was validated through automated testing which achieved 86% code coverage and user acceptance testing with undergraduate students, confirming that HearSee meets its objectives of improving accessibility and ease of use across diverse learning modalities with high user satisfaction and excellent semantic quality, while providing a scalable foundation for future enhancements in multimodal educational applications.

Abstrak

HearSee ialah sebuah alat pembantu pembelajaran berasaskan web yang direka untuk memperkasakan pengajian peringkat prasiswazah dengan menyepadukan model bahasa raya multimodal (LLM) bagi pemprosesan imej-teks-ke-teks dan teks-ke-ucapan. Sistem ini dibangunkan dengan seni bina tiga lapisan yang menggunakan antara muka pengguna (UI) Gradio untuk lapisan persembahan, kelas perkhidmatan Python untuk logik aplikasi, dan utiliti konfigurasi untuk pengendalian data.Sistem ini memanfaatkan model Qwen2-VL 7B melalui API Replicate untuk melaksanakan Pengecaman Aksara Optik (OCR), menjana kapsyen deskriptif, menghasilkan ringkasan yang padat, dan menyokong sesi soal jawab interaktif berdasarkan imej yang dimuat naik. Hasil yang telah diproses boleh ditukarkan kepada ucapan yang berbunyi semula jadi menggunakan enjin TTS Kokoro (berasaskan StyleTTS 2), dengan profil suara yang boleh dipilih oleh pengguna dan kelajuan main balik yang boleh laras. Setelah dilancarkan di platform Hugging Face Spaces, HearSee menggunakan pendekatan kejuruteraan prom (*prompt engineering*) untuk mengoptimumkan respons model, serta menguruskan keadaan sesi (*session state*) dan fail audio sementara bagi memastikan interaksi pengguna yang lancar. Keberkesanan sistem ini dinilai melalui metrik penilaian automatik—iaitu BERTScore untuk kualiti teks dan Skor Pendapat Purata (*Mean Opinion Score*) untuk keaslian bunyi ucapan—di samping penanda aras latensi yang diukur. Hal ini telah disahkan melalui ujian automatik yang mencapai 86% liputan kod dan ujian penerimaan pengguna (*user acceptance testing*) bersama pelajar prasiswazah. Ujian ini mengesahkan bahawa HearSee berjaya mencapai objektifnya untuk meningkatkan kebolehcapaian dan kemudahan penggunaan merentasi pelbagai modaliti pembelajaran, dengan tahap kepuasan pengguna yang tinggi dan kualiti semantik yang cemerlang. Di samping itu, sistem ini menyediakan asas yang boleh skala (*scalable*) untuk penambahbaikan aplikasi pendidikan multimodal pada masa hadapan.

Chapter 1: Introduction

1.1 Introduction

The rapid progression of educational technology has paved the way for innovations that cater to diverse learning needs. With modern digital learning tools, students can now access resources that work better for different ways of learning. For example, online videos help visual learners, audio recordings support auditory learners, and interactive websites allow hands-on learners to engage with educational content more easily. However, many existing tools fail to address the difficulty in approaching certain complexities of academic content, especially when it involves combining text, visuals, and auditory information (Morrant, 2022). The project aims to bridge this gap by developing HearSee, a web-based learning tool platform powered by a Multimodal Large Language Models (MLLMs).

Multimodal Large Language Models (MLLMs) are an advanced branch of generative AI designed to process and integrate information from multiple input types, such as text, images, and audio, enabling them to perform tasks like generating detailed image descriptions, summarizing text, and visual recognition (United States Artificial Intelligence Institute (USAII®), n.d.). By leveraging these capabilities, it introduces multimodal learning where it enhances traditional education by combining visual, auditory, and textual inputs to create an easier approach in learning new knowledge and experiences that improve memory retention and understanding (May, 2023).

Thus, HearSee offers a web-based learning tool using advanced Multimodal Large Language Models (MLLMs) which can be interfaced using a simple conversational user interface. It provides image analysis, text descriptions, and summaries and can be enhanced through text-to-speech generators for added capability. The goal is to create an inclusive learning environment that suits different learning styles and accessibility needs through the implementation of Multimodal Large Language Models (MLLMs) through multimodal learning.

1.2 Problem Statement

Many undergraduate students face difficulties retaining information. Traditional study methods, like textbooks and slides, rely heavily on visuals, which don't suit all learners. Those with learning difficulties, often struggle to follow along and may need additional support (Gupta, 2024.; Morrant, 2022). There’s a need for user-friendly tools that integrate features like image-text-to-text to handle visual and textual information. Image-text-to-text capability is selected as the critical feature for providing detailed text descriptions, accurate text transcriptions, and concise summarizations of visual content. This is further enhanced by the ability to ask questions about the content or instruct the tool to perform these tasks according to the students’ personal requirements. This web-based tool aims to integrate multimodal learning as the approach of consuming information through different sensory modalities (visual and auditory) to enhance learning outcomes as a solution.

1.3 Aim & Objectives

The aim of this project is to develop a multimodal large language model-based web application tool that enhances students' learning experiences by accurately processing images of educational materials, converting text and images into audio, and offering a user-friendly interface.

The objectives are as the following:

1. To develop a multimodal system using a Multimodal Large Language Model (MLLM) to extract, describe, and summarize educational image content with high semantic accuracy.
2. To implement a text-to-speech component using Text to Speech model for converting generated text into natural and intelligible speech output.
3. To design an accessible web-based interface using Gradio that supports image uploads, audio playback, and interactive user interface for enhanced multimodal learning.

1.4 Methodology

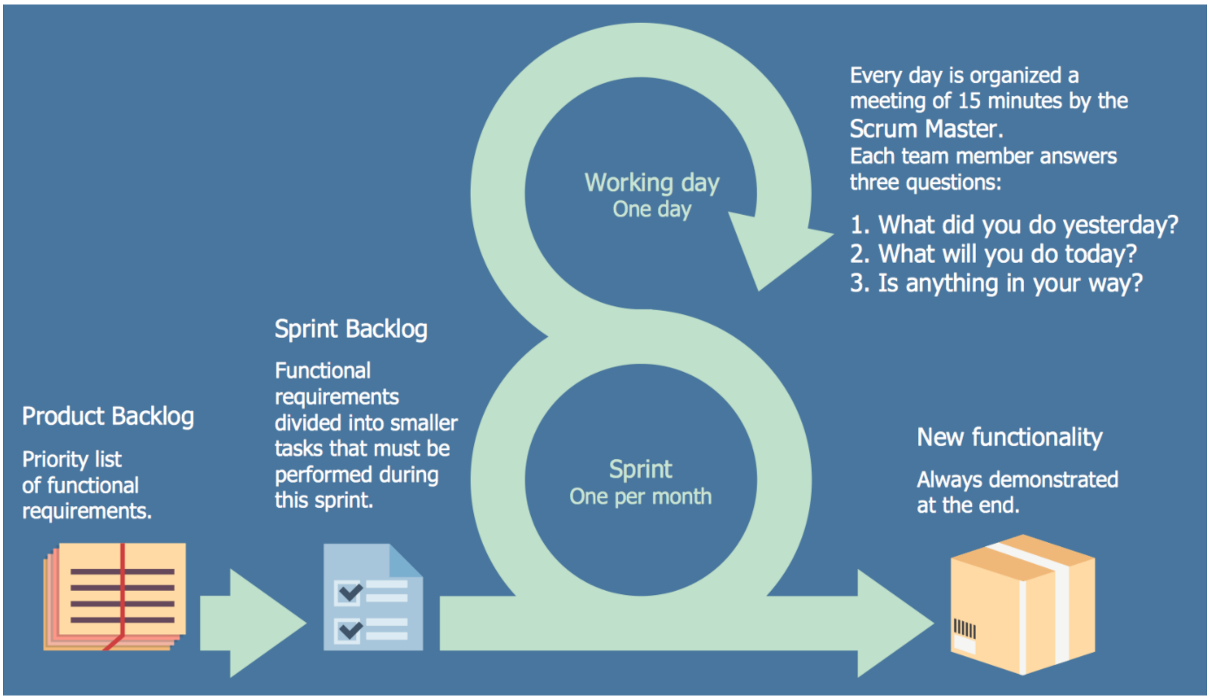


Figure 1 SCRUM Methodology (Scrum Workflow | Scrum Board | Scrum | Scrum Guide, n.d.)

SCRUM is an agile framework that will be adapted for this one-person project. It emphasizes flexibility, iterative progress, and continuous improvement. For this project, SCRUM will be utilised in managing the project throughout its whole lifecycle, allowing for regular reassessment and adjustment of priorities. This approach will ensure that the most valuable features are developed first and that the project remains aligned with its objectives throughout the development process.

The project will be divided into 2-week sprints. At the beginning of each sprint, tasks will be selected from the product backlog based on priority and estimated effort. These tasks will form the sprint backlog. Each sprint will have a clear goal, such as implementing a specific feature or improving a particular aspect of the web application.

During each sprint, work will be done for the tasks in the sprint backlog. Daily stand-ups will be replaced with brief self-reflection sessions to review progress and identify any obstacles. The implementation will follow best practices in web development, API integration, and AI model implementation. Regular commits to version control will ensure proper tracking of changes and allow for easy rollbacks if needed.

For each task to be considered done, it must meet the following criteria:

- Code is written and properly documented
- Unit tests are created and passed
- Feature is integrated into the main branch without conflicts
- Feature is tested on multiple browsers and devices
- Any necessary user documentation is updated

At the end of each sprint, a Sprint review will be conducted to ensure all completed tasks meet these criteria. A Sprint retrospective will be conducted for identifying weak points and possible future improvements.

1.5 Scope

The target demographic of this project and are university Undergraduate students who are taking degrees as full-time students. The system shall perform image-text-to-text and text-to-speech tasks only using Qwen2-VL 7B model and Kokoro TTS. Thus, no requirement for speech-based input is required. Only English language is supported.

1.6 Significance of Project

The project will further contribute to the learning accessibility for undergraduate students by leveraging generative Multimodal Large Language models and diffusion-based text to speech models to convert image-based educational materials into more easily digestible text and audio content.

1.7 Project Schedule

The project will be completed over nine months and divided into two phases: FYP 1 and FYP 2. Each phase spans approximately 14 weeks, aligning with a single semester timeframe. FYP 1 will take place from October 7, 2024, to January 12, 2025. FYP 2 will run in the second semester, from March 17, 2025, to June 29, 2025. Gantt charts for both phases are included in Appendix A and Appendix B, respectively.

1.8 Project Outcome

The outcome of the project is HearSee, a web application built with Gradio that uses Qwen2-VL 7B multimodal LLM for image analysis and Kokoro TTS for speech synthesis. This web application tool will help undergraduate students better understand and retain information from various study materials, supporting multimodal learning.

1.9 Project Summary

The project successfully developed HearSee, a web application designed to enhance learning accessibility for undergraduate students. Many undergraduate students face difficulties retaining information, especially those with auditory or visual processing difficulties. To address this issue, HearSee analyses images, provides text descriptions and summaries, and converts text to speech using generative multimodal AI and diffusion-based text to speech models.

The application was implemented using a three-tier architecture with Python and Gradio UI for the front-end, service-based application logic for the back-end, and Replicate API integration for AI model access. HearSee leverages Qwen2-VL 7B for multimodal image analysis and Kokoro TTS (based on StyleTTS 2) for natural speech synthesis, offering users selectable voice profiles and adjustable playback speeds. The system supports OCR text extraction, descriptive image captioning, content summarization, and interactive question and answering functionality through a conversational interface.

The project was implemented over a nine-month period, divided into two phases (FYP 1 and FYP 2), using an adapted SCRUM framework for project management. Full evaluation through automated testing (achieving 86% code coverage), BERTScore semantic quality assessment, and user acceptance testing with 10 undergraduate students validated the system’s effectiveness. The goal of creating an application that transforms educational materials into more easily digestible content was achieved, with users reporting high satisfaction rates and the system demonstrating excellent semantic quality in generated responses, thereby contributing to improved learning experiences for full-time undergraduate students.

Chapter 2: Literature Review

2.1 Introduction

This literature review aims to explore and conduct comparative analysis on the various research that utilizes technologies for the purpose of performing image-to-text tasks. In this chapter, similar works using image-to-text tasks will be reviewed.

Image-to-text tasks involve converting visual information from images into textual descriptions or representations. Image-to-text technologies can perform translation of the extracted visual information into natural language descriptions or captions that accurately and coherently describe the image as well as generating descriptive sentences for images (Karpathy & Fei-Fei, 2014). Researchers have developed various approaches to address image-to-text tasks, including the use of deep learning techniques such as convolutional neural networks (CNNs) for image understanding, recurrent neural networks (RNNs) or transformers for language generation. (Karpathy & Fei-Fei, 2014; Grand & Belinkov, 2019).

Text-to-speech (TTS) synthesis, a technology that transforms written text into natural-sounding speech, has evolved with recent advancements in artificial intelligence and finds widespread applications in virtual assistants and audiobooks (Li et al., 2023). Modern TTS systems are capable of generating highly natural and expressive speech through various approaches like style modeling and diffusion models, marking substantial progress from basic speech synthesis to sophisticated voice generation (Li et al., 2023).

In this literature review, the second section of this chapter will review similar works related to the proposed project and a subsection for the summary of reviewed works will be made. The fourth section will review the related technologies to be utilised for the proposed project and includes a subsection summary.

2.2 Review of Related Works

This section reviews existing research conducted using technologies to perform image-to-text tasks on a variety of images to provide textual data, adding educational and learning value to their specific domain. The following reviews will be analysed based on the context of the research problem each study addresses, the objectives or tasks they aim to achieve, the methods and technologies employed to tackle these tasks, and the results and analysis with limitations when present.

2.2.1 Adapting the Tesseract Open-Source OCR Engine for Tamil and Sinhala Legacy Fonts and Creating a Parallel Corpus for Tamil-Sinhala-English

Vasantharajan and Thayasivam (2021) developed an enhanced Optical Character Recognition (OCR) system motivated by the critical need to digitize and process documents in low-resource languages which are Tamil and Sinhala. Previously, text extraction from PDF documents with legacy fonts was challenging due to font encoding issues and lack of standardization. The researchers identified that most government and educational documents in these languages existed primarily in PDF format with non-standard legacy fonts, making them inaccessible for digital processing and educational use.

The researchers aimed to develop a simple but effective approach to create high-quality, large-scale trilingual data extraction system (Tamil, Sinhala, and English) using Deep Learning-based Printed Character Recognition (PCR), focusing on improving the Tesseract open-source OCR engine for better accuracy with legacy fonts.

The methodology employed in this research focused on developing a fine-tuned Tesseract OCR, Tamizhi-Net OCR for Tamil and Sinhala languages using the Tesseract open-source OCR engine. The researchers implemented a systematic approach that began with careful data preparation and training processes. Initially, they identified and collected 10 Tamil and 10 Sinhala fonts that were most commonly used in Sri Lankan portable documents, sourcing them from Free Tamil Font and Sinhala Fonts websites. For each font, they created extensive training data consisting of TIFF/Box pair files, with each font mapped to a TIFF file containing 250 pages of images.

Figure 2 The architectural diagram of Tamizhi-Net OCR.

For the training process, the researchers used training text files provided by Tesseract and created box files with precise coordinates specifications. To ensure accuracy, they utilized jTessBoxEditor to rectify misidentified characters and adjust letter tracking, effectively eliminating bounding box overlapping issues. The deep learning model was then implemented using Tesseract Long-Short Term Memory model and data, and trained using these carefully prepared TIFF/Box pair files. Figure 2 shows the architectural diagram of the Tamizhi-Net OCR.

The pre-processing pipeline utilises several key steps: image pre-processing using OpenCV, which involved grayscaling to convert coloured images to grayscale, followed by image resizing to standardize dimensions. The process continued with blurring operations to reduce noise, and thresholding to convert the grayscale image into a binary image where white pixels indicated the background and black pixels represented the foreground. The final pre-processing step involved contour detection, which joined continuous points along boundaries with the same colour intensity, making it easier to detect and localize edges of objects in the image.

For experimental evaluation, the researchers prepared test images, with each sample image consisting of 762 characters and 77 words, selected randomly from various fonts. They utilized the fastwer package to calculate Character Error Rate (CER) and Word Error Rate (WER) from the transcribed output and manually labelled ground truth text. The testing methodology was designed to compare the performance of the existing base Tesseract model against their fine-tuned model, Tamizhi-Net OCR, using these predefined error rates.

The fine-tuned OCR model displayed improvements in recognition accuracy:

- Character error rate improvements
    - Tamil: from 6.03% to 2.61% (-3.42% relative change)
    - Sinhala: from 7.61% to 4.74% (-2.87% relative change)
- Word-level error rate improvements:
    - Tamil: from 39.68% to 20.61% (-19.07% relative change)
    - Sinhala: from 35.04% to 26.58% (-8.46% relative change)
- Successfully created a parallel corpus containing:
    - Tamil: 185.4K sentences, 2.11M words
    - Sinhala: 168.9K sentences, 2.22M words
    - English: 181.04K sentences, 2.33M words

From the research work, Vasantharajan and Thayasivam (2021) revealed several key performance factors and limitations in the implemented OCR system. System accuracy was primarily dependent on input image pre-processing quality and showed variable performance across different legacy fonts, mainly those absent from the training dataset. While achieving improvements in character error rates (reducing to 2.61% for Tamil and 4.74% for Sinhala), the system faced challenges with mixed-language content and abbreviations, which complicated the creation of pure monolingual corpora. The substantial computational requirements, necessitating two weeks of training on an Nvidia GeForce MX350 GPU, presented a practical limitation for widespread implementation. Despite the overall success, the relatively higher word-level error rates (20.61% for Tamil and 26.58% for Sinhala) indicated room for further optimization in the system's word-level recognition capabilities.

Vasantharajan and Thayasivam (2021) concluded that their PCR approach successfully demonstrated the ability to produce large-scale quality parallel corpus from government PDFs, with particular effectiveness for educational and governmental document processing. The improved accuracy and ability to handle legacy fonts made the system valuable for preserving and digitizing educational content in low-resource languages.

2.2.2 LLaVA-docent: Instruction tuning with multimodal large language model to support art appreciation education

Lee et al. (2024) developed an AI-based art appreciation educational tool called LLaVA-Docent; a multimodal large language model (MLLM) specifically designed to support art appreciation education by performing image-to-text tasks using visual and textual input. The research was motivated by the lack of effective AI-assisted tools for art appreciation education, despite the growing adoption of AI systems in other educational domains. Previously, while some AI tools existed for art expression and creation, there was limited development of AI systems specifically for art appreciation pedagogy. Hence, the researchers aimed to develop an open-source MLLM that could serve as a personal tutor for art appreciation education while incorporating established pedagogical frameworks.

The methodology Lee et al. (2024) developed involved an all-inclusive design and development research (DDR) approach consisting of six phases:

1) Initial prototype development of LLaVA-Docent Version 1, using the LLaVA framework with a vision encoder, vicuna-13b-v1.5 as the LLM, clip-vit-large-patch14 as the image encoder, and incorporated Anderson’s critical stages theory of systemic educational change for art appreciation into the training framework

2) Literature review and expert validation;

3) Data design framework refinement;

4) Dataset generation using GPT-4 to create 1000 dialogue samples;

5) Model training of LLaVA-Docent Version 2 through pre-training and fine-tuning stages; and

6) Model evaluation by comparing to GPT-4’s zero-shot (single turn conversation) and few-shot (multi-turn conversation) capabilities.

The model architecture of LLaVA-Docent, defined by the components: Vision encoder (g(·)) for processing visual data, LLM (fφ(·)) based on Vicuna for language processing, Projection layer (W) to link visual and language modalities, and the prototype of LLaVA-Docent tool built using Hugging Face Space, are as shown in Figure 3 and 4 respectively.

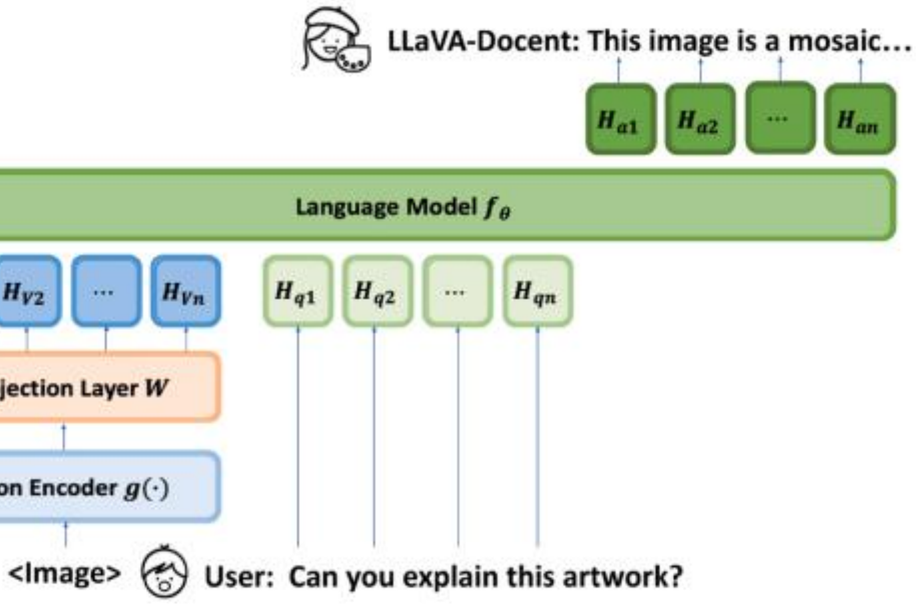
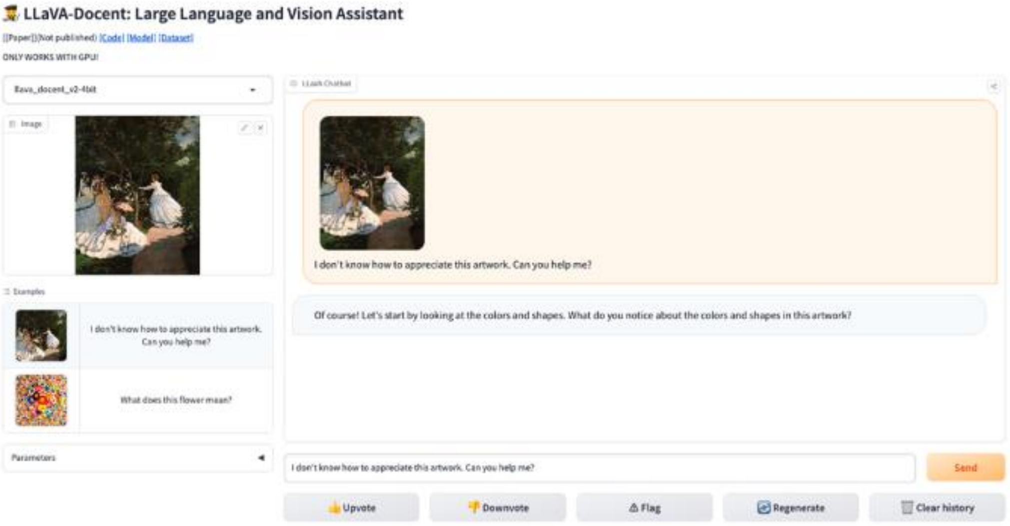


Figure 3 Architecture of LLaVA-Docent.



I don't know how to appreciate this artwork. Can you help me?

What does this flower mean?

I don't know how to appreciate this artwork. Can you help me?

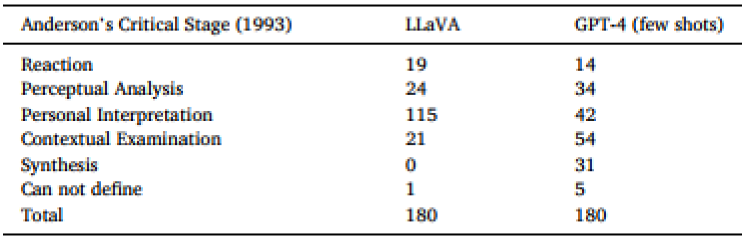
Of course! Let's start by looking at the colors and shapes. What do you notice about the colors and shapes in this artwork?

I don't know how to appreciate this artwork. Can you help me?

Figure 4 Prototype *of LLaVA-Docent.*

The results indicated that LLaVA-Docent proven several strengths compared to GPT-4, including: sequential progression through appreciation stages, limited cognitive load by asking one question at a time, and encouraging independent interpretation by providing fewer explanations. Quantified qualitative analysis showed the model successfully generated questions aligned with Anderson's critical stages, though with some stage distribution imbalances as shown in Table 1. It also revealed the model achieved classroom-like interactions through appropriate questioning and feedback mechanisms.

Table 1 Quantified qualitative analysis results



Anderson's Critical Stage (1993)	LLaVA	GPT-4 (few shots)
Reaction	19	14
Perceptual Analysis	24	34
Personal Interpretation	115	42
Contextual Examination	21	54
Synthesis	0	31
Can not define	1	5
Total	180	180

Analysis revealed that while LLaVA-Docent excelled in structured progression and cognitive load management, it had limitations including: restricted ability to handle recursive appreciation processes, limited dataset size affecting response variety, potential hallucination issues in artwork information, and lack of portfolio/documentation features. The researchers noted the model's smaller parameter size (13 billion) compared to GPT-4 affected some performance aspects.

Overall, Lee et al. (2024) concluded that LLaVA-Docent represents a noteworthy advancement in making art appreciation education more accessible and engaging through AI technology. The study demonstrates the potential for MLLMs to democratize art education while highlighting areas for future improvement, such as implementing retrieval-augmented generation, expanding dataset quality and quantity, and incorporating additional documentation features.

2.2.3 Purrfessor: A Fine-tuned LLaVA Diet Health Chatbot

Lu et al. (2024) developed an innovative AI chatbot that specialises in providing dietary guidance personalised for the user through engaging in an interactive, multimodal method. The researchers stated the limitations of existing health tracking apps, which often lack interactive and personalized health dialogues that adapt in real-time to user behaviors. Traditional apps like MyFitnessPal and Lifesum as examples provide static tracking but fall short in offering dynamic, personalized guidance. The researchers identified a need for AI-driven health chatbots that could provide on-demand guidance through multimodal interactions for supporting populations with limited resources and time constraints. The research leverages the Large Language-and-Vision Assistant (LLaVA) model, fine-tuned with food and nutrition data, to enhance user experience and engagement in dietary education.

Lu et al. (2024) was motivated by the growing need for advanced image-to-text systems capable of handling real-world, multimodal interactions, predominantly in domains such as food and nutrition. Prior work in this area, such as CLIP (Contrastive Language–Image Pretraining) and Vicuna, revealed the potential of integrating visual and linguistic understanding. However, these models were not specifically tailored for food-related tasks, leaving a gap in their ability to accurately interpret and describe complex food images. *Purrfessor* builds on these foundations by fine-tuning LLaVA, a model that combines CLIP's visual encoder with Vicuna's language decoder, to address this niche.

The primary objective of the study was to develop a robust image-to-text system capable of generating accurate and contextually relevant captions and Q&A responses for food-related images. The researchers aimed to enhance the model's ability to handle ambiguous and non-food images while maintaining high accuracy and user engagement in real-world scenarios.

The methodology involved a multi-step process, beginning with the integration of LLaVA-Vicuna-13b 1.6, a refined large language model (LLM) that bridges visual and linguistic understanding. The model was fine-tuned using a custom dataset constructed from multiple sources, including FoodData Central's Foundation Foods dataset and images collected via Google Image Search (GIS). The GIS method employed Python and the Google Custom Search API to retrieve food-related images using pre-defined search queries such as "raw food," "fresh produce," and "cooking raw meat and vegetables." Metadata, including collection dates, source websites, and image URLs, were systematically extracted to enrich the dataset. The chatbot's system architecture, depicted in Figure 5, was engineered to enable smooth interactions between users and a sophisticated conversational AI model, specifically tailored for dietary guidance and health recommendations. The architecture utilizes cloud-based infrastructure, a well-organized database, and intuitive front-end web applications to deliver an engaging and user-friendly experience.

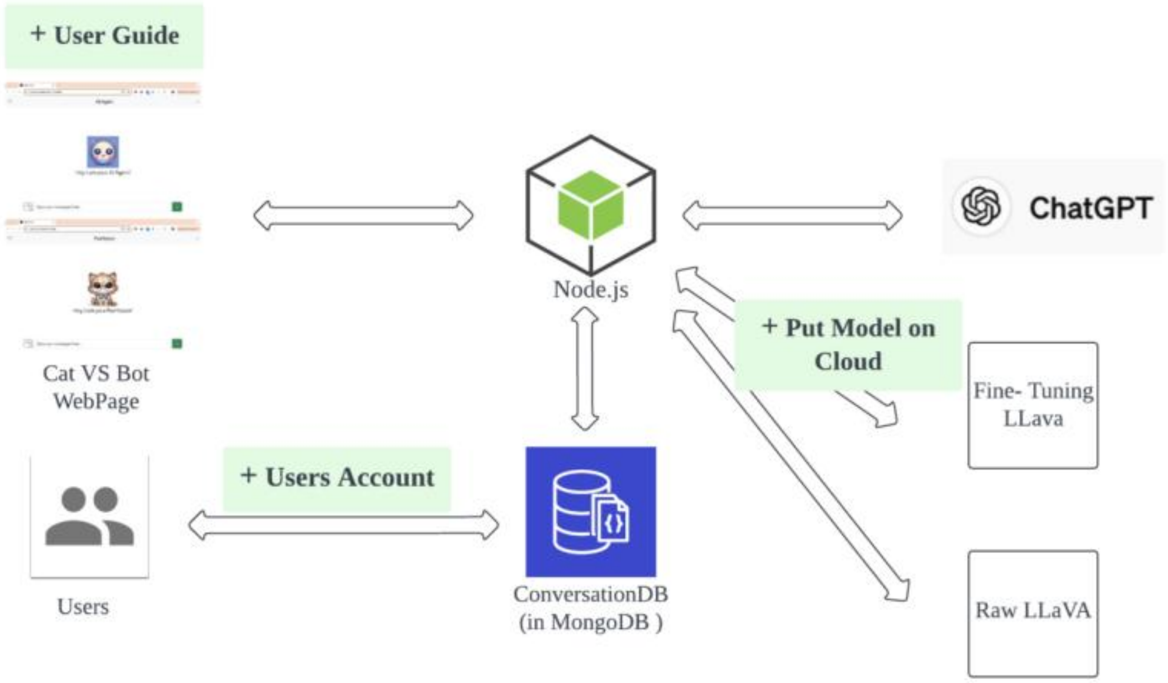
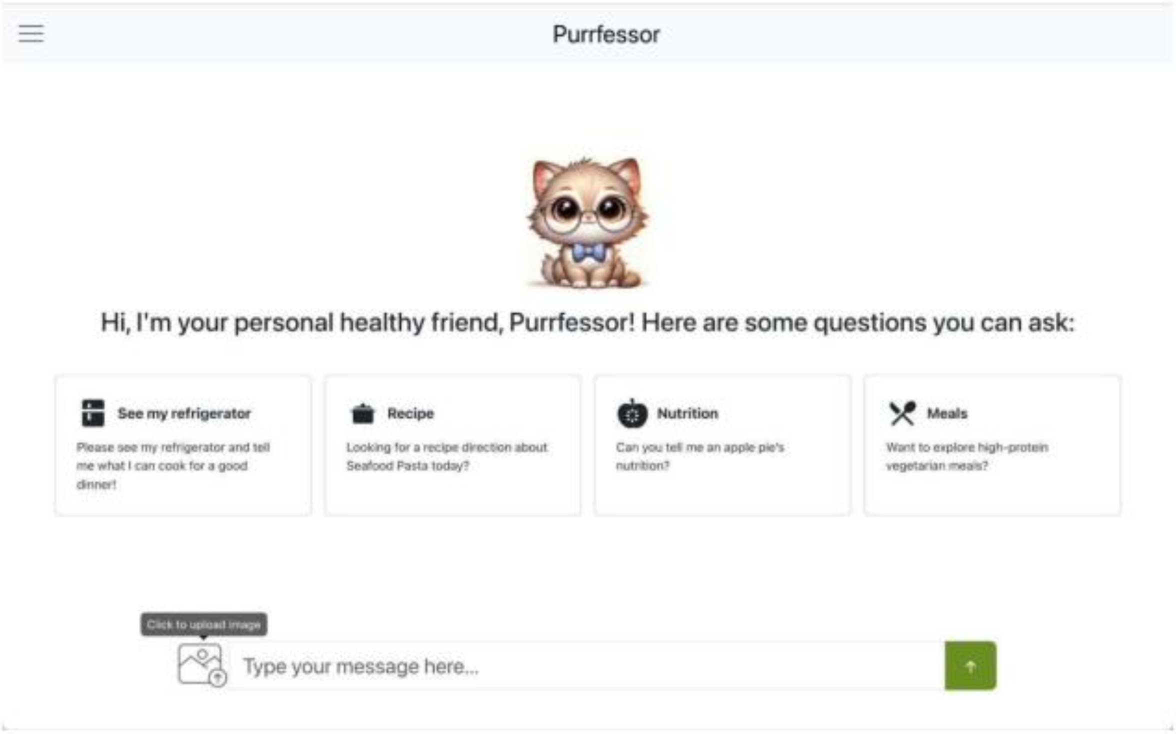


Figure 5 System Architecture for Purrfessor Chatbot.

The fine-tuning process incorporated a human-in-the-loop approach, where human annotators reviewed and refined captions and Q&A examples to ensure clarity, accuracy, and alignment with user expectations. This step was critical for handling edge cases, such as ambiguous food features (e.g., partially obscured ingredients) and non-food images. The annotators ensured that responses were neutral and factual, avoiding refusals even in ambiguous scenarios.

The experimental setup included a user interface, as shown in Figure 6 designed to facilitate interactions with the *Purrfessor* chatbot. The architecture of LLaVA, which integrates CLIP's visual encoder with Vicuna's language decoder, enabled the model to process diverse instruction types, including image descriptions, dialogue, and complex reasoning tasks.



Hi, I'm your personal healthy friend, Purrfessor! Here are some questions you can ask:

See my refrigerator
Please see my refrigerator and tell me what I can cook for a good dinner!

Recipe
Looking for a recipe direction about Seafood Pasta today?

Nutrition
Can you tell me an apple pie's nutrition?

Meals
Want to explore high-protein vegetarian meals?

Figure 6 Purrfessor Nov 24 Version User Interface.

The results demonstrated *Purrfessor's* ability to accurately identify and describe food-related images, even in challenging scenarios. For ambiguous food features, such1 as a pizza slice buried under toppings, the model successfully identified major ingredients but occasionally omitted secondary items (e.g., basil leaves), resulting in lower overlap scores. The text overlap score for image object detection tasks averaged 0.67 which indicated moderate alignment between the chatbot's outputs and expected responses. This score reflects the model's ability to accurately identify and describe food items in images. In non-food image scenarios, the chatbot flagged inappropriate content with appropriate disclaimers.

User experience testing revealed that the fine-tuned LLaVA model outperformed raw LLaVA and GPT-4 in terms of accuracy and user satisfaction. The study employed a 2x3 between-subject design, comparing chatbot profiles (Bot vs. Anthropomorphic) and models (GPT-4, raw LLaVA, fine-tuned LLaVA), with ChatGPT as a baseline. Users interacted with the chatbot by asking questions, uploading photos, and receiving personalized healthy recipe recommendations. The fine-tuned LLaVA model consistently achieved higher scores in clarity, relevance, and engagement.

Lu et al. (2024) highlighted the strengths of *Purrfessor* in handling diverse image-to-text tasks in the food domain. The integration of CLIP's visual encoder and Vicuna's language decoder proved effective for multimodal understanding, while the human-in-the-loop approach ensured high-quality training data. However, the study acknowledged limitations, such as the occasional omission of secondary ingredients in ambiguous food images, leading to lower overlap scores for secondary ingredients. There are also limitations present for the dataset in terms of bias. The custom dataset developed, while comprehensive, may not fully capture the diversity of real-world food images, potentially limiting the model's generalizability. Additionally, while the model performed well in non-food scenarios, further improvements could be achieved by incorporating a secondary verification step for ambiguous classifications and expanding the dataset to include a wider variety of food images.

In conclusion, Lu et al. (2024) demonstrated that *Purrfessor*, a fine-tuned LLaVA-based system, effectively bridges the gap between visual and linguistic understanding in image-to-text tasks for food-related applications. The study's findings underscore the importance of domain-specific fine-tuning and human-in-the-loop validation in enhancing model performance. Future work could focus on improving the model's handling of ambiguous classifications and expanding its capabilities to other domains beyond food and nutrition. Overall, *Purrfessor* represents a notable progress in image-to-text technologies, with promising implications for real-world applications.

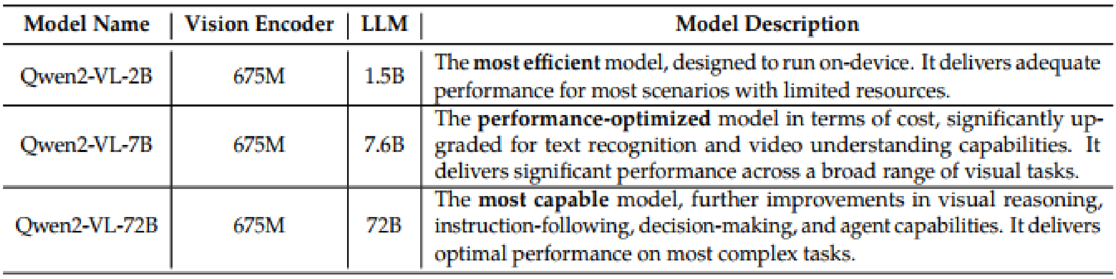
2.3 Review of Related Technologies

This section reviews existing related technologies to perform image-to-text and task-to-speech tasks for the proposed project. The following reviews will be analysed based on the context of the research problem each study addresses, the objectives or tasks they aim to achieve, the methods and technologies employed to tackle these tasks, and the results and analysis with limitations when present.

2.3.1 Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution

Wang et al. (2024) introduced a series of multimodal large language models named Qwen2-VL. It represents advancement in vision-language modelling by addressing the limitation of fixed-resolution processing in existing Large Vision-Language Models (LVLMs). While conventional models constrain images to predetermined sizes, Qwen2-VL introduces a dynamic resolution approach that adaptively processes images at their native resolutions, similar to human visual perception. This innovative technique, combined with a Multimodal Rotary Position Embedding (M-RoPE) mechanism and a unified training paradigm combining both images and videos, enables more efficient and accurate visual understanding. The model, available in three variants (2B, 8B, and 72B parameters), as shown in Table 2, demonstrated excellent performance across various benchmarks excelling in document understanding, mathematical reasoning, and multilingual OCR tasks.

Table 2 Descriptions of Qwen2-VL models



Model Name	Vision Encoder	LLM	Model Description
Qwen2-VL-2B	675M	1.5B	The **most efficient** model, designed to run on-device. It delivers adequate performance for most scenarios with limited resources.
Qwen2-VL-7B	675M	7.6B	The **performance-optimized** model in terms of cost, significantly upgraded for text recognition and video understanding capabilities. It delivers significant performance across a broad range of visual tasks.
Qwen2-VL-72B	675M	72B	The **most capable** model, further improvements in visual reasoning, instruction-following, decision-making, and agent capabilities. It delivers optimal performance on most complex tasks.

Wang et al. (2024) was motivated by the limitations of conventional vision-language models that use fixed-resolution approaches for visual processing. Previously, Large Vision-Language Models (LVLMs) were constrained by predetermined image input sizes, often requiring down sampling or up sampling of images to a fixed resolution (e.g., 224×224 pixels). This one-size-fits-all strategy limited the models' ability to capture information at different scales and led to noticeable loss of detailed information in high-resolution images. Wang et al. (2024) also stated that traditional LVLMs typically use static CLIP-style vision encoders, which may not adequately handle complex reasoning tasks or detailed image processing. While some researchers have improved performance by fine-tuning Vision Transformers (ViT) during training, Qwen2-VL takes this further by implementing dynamic resolution training and 2D Rotary Position Embedding (RoPE). This approach enables the model to better process visual information across varying spatial scales and resolutions.

The primary objective of the Qwen2-VL model is to advance image-to-text technologies by developing a multimodal model capable of understanding multilingual texts within images, performing complex reasoning tasks (e.g., code and math), and handling real-world applications such as video analysis and live chat at varying resolutions dynamically.

The methodology involved several innovative components:

1. Architecture: The researchers developed three model variants (2B, 8B, and 72B parameters), each using a 675M parameter Vision Transformer (ViT) combined with different-sized Language Models. Figure 7 shows the model architecture of Qwen2-VL.
2. Key Technical Innovations:
   - Naive Dynamic Resolution mechanism: Enables processing of images at their native resolutions
   - Multimodal Rotary Position Embedding (M-RoPE): Facilitates effective fusion of positional information across text, images, and videos, shown in Figure 8
   - Unified paradigm for processing both images and videos
3. Training Process:
   1. First Stage:
      - Focuses exclusively on training the Vision Transformer (ViT) component
      - Uses a large corpus of image-text pairs of around 600 billion tokens
      - Aims to enhance semantic understanding within the Large Language Model (LLM)
   2. Second Stage:
      - Unfreezes all parameters
      - Expands training to include a wider range of data, consists of additional 800 billion tokens of image-text content.
      - Enables more inclusive learning of visual and textual information

3. Final Stage:
    - Locks the ViT parameters
    - Performs exclusive fine-tuning of the LLM using instructional datasets
    - Implements ChatML format for instruction-following data

Additional Training Data:

- Processed approximately 1.4 trillion tokens in total
- Includes both text and image tokens
- Supervision is provided only for text tokens
- Uses diverse data sources including image-text pairs, OCR data, interleaved image-text articles, visual question answering datasets, video dialogues, and image knowledge datasets
- Initial pre-training phase involves around 600 billion tokens
- Data knowledge cut-off date is June 2023

The training approach balances computational demands with efficiency for video processing, by dynamically adjusting frame resolution while limiting total tokens per video to 16,384.

Based on the evaluations Wang et al. (2024) conducted, Qwen2-VL 72B as the flagship model achieved remarkable performance across multiple benchmarks:

- Surpassed state-of-the-art results on DocVQA (96.5), InfoVQA (84.5), and OCRBench (877), showing excellent document and diagrams reading capability

- Achieved 70.5 accuracy on MathVista, outperforming other LVLMs on mathematical reasoning and setting a new open-source record on MathVision benchmark with 25.9
- Demonstrated superior performance in multilingual OCR, exceeding state-of-the-art (SOTA) models such as GPT-4o across most languages as shown in Table 3
- Showed exceptional capabilities in video understanding, achieving top scores on MVBench (73.6), PerceptionTest (68.0), and EgoSchema (77.9)

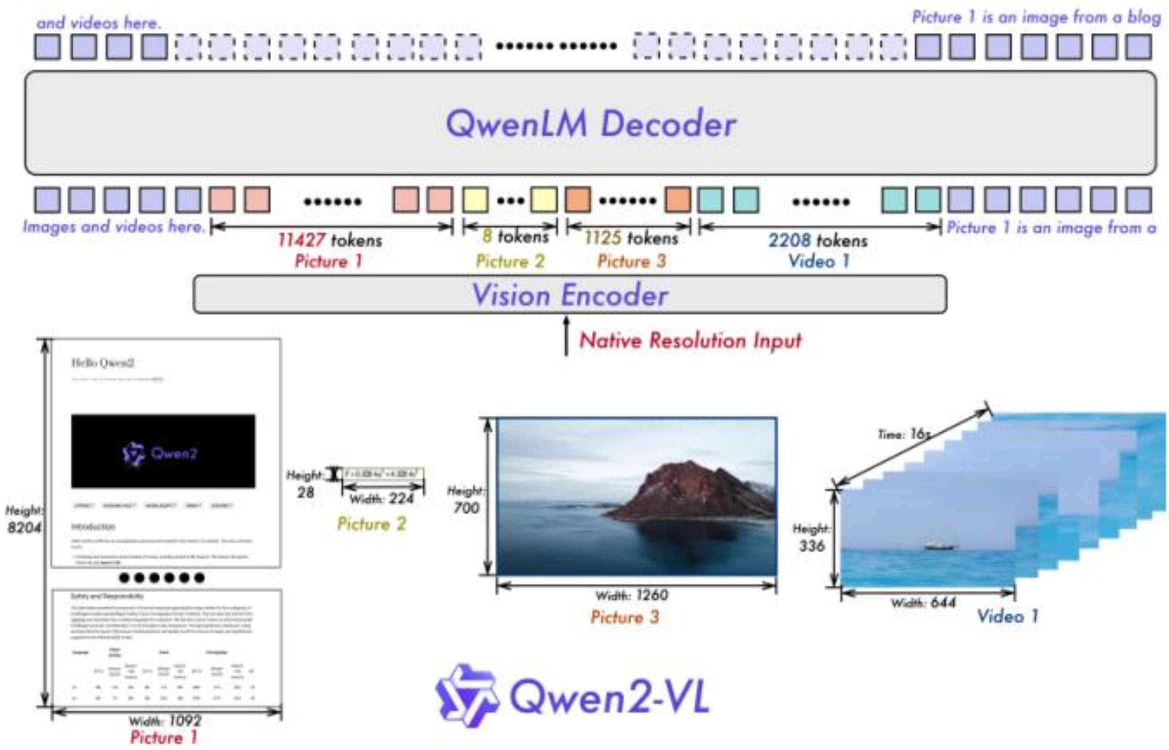


Figure 7 Qwen2-VL model architecture.

Figure 8 M-RoPE demonstration. By breaking down rotary embedding into temporal, height, and width components, M-RoPE is able to explicitly capture and model the positional information of text, images, and videos within a large language model (LLM) (Wang et al., 2024).

Table 3 Qwen2-VL performance comparison with GPT-4o on multilingual benchmarks

Language	Korean	Japanese	French	German	Italian	Russian	Vietnamese	Arabic
GPT-4o	87.8	88.3	89.7	88.3	74.1	96.8	72.0	**75.9**
Qwen2-VL-72B	**94.5**	**93.4**	**94.1**	**91.5**	**89.8**	**97.2**	**73.0**	70.7

From the analysis of the model Wang et al. (2024) developed, it was revealed that the dynamic resolution approach consistently achieved top-tier performance while consuming fewer tokens on average compared to fixed-resolution approaches. However, the model showed some limitations in handling extremely long video sequences due to context length constraints, and performance in complex reasoning tasks (like MMMU) still lagged behind GPT-4o to some extent. The researchers also noted that increasing image size alone didn't always lead to improved performance, suggesting the importance of appropriate resolution selection for different types of images.

The researchers concluded that Qwen2-VL successfully redefined the conventional predetermined-resolution approach in visual processing. The model's ability to handle dynamic resolutions, combined with its sophisticated position embedding system and unified processing paradigm, resulted in a more efficient and accurate visual representation system. The model series proved competitive performance against leading proprietary models while maintaining openness and accessibility for research and development.

2.3.2 StyleTTS 2: Towards Human-Level Text-to-Speech through Style Diffusion and Adversarial Training with Large Speech Language Models

Li et al. (2023) developed StyleTTS 2, an innovative text-to-speech (TTS) synthesis model that leverages style diffusion and adversarial training with large speech language models (SLMs) that builds upon its style-based predecessor StyleTTS. Previously, while substantial progress had been made towards human-level performance of speech, there is still room for improvement in terms of diverse and expressive speech, robustness for out-of-distribution texts, and reducing the massive datasets required for creating high-performing zero-shot TTS systems.

The motivation behind StyleTTS 2 stems from the need to improve the naturalness, expressiveness, and diversity of synthetic speech in TTS systems. Previous works have made notable developments in TTS with the advent of large-scale self-supervised speech language models (SLMs) such as WavLM, which have enhanced TTS quality and speaker adaptation. However, existing methods often rely on two-stage training processes or suboptimal SLM features for speech synthesis. Li et al. (2023) stated that StyleTTS 2 addresses these limitations by introducing a novel approach that combines style diffusion and adversarial training with SLMs, enabling end-to-end training and improved speech naturalness.

The primary objective of StyleTTS 2 is to achieve human-level TTS synthesis by modelling styles as latent random variables through diffusion models and leveraging large pre-trained SLMs as discriminators. The researchers aimed to eliminate the need for reference speech, ensure efficient latent diffusion, and maintain the benefits of diverse speech synthesis offered by diffusion models.

The methodology of StyleTTS 2 employs several innovative approaches:

1. Style Diffusion: The model uses a diffusion-based approach to model speech styles as latent random variables, allowing efficient sampling of style vectors without requiring reference audio.

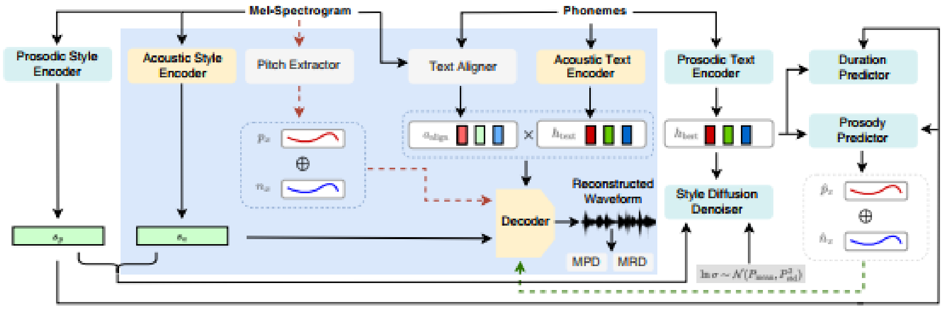
2. End-to-End Training: The system incorporates an end-to-end training process that jointly optimizes all components, including direct waveform synthesis and adversarial training with large speech language models (SLMs).

3. Speech Language Model Discriminators: The researchers uniquely employed pre-trained WavLM as discriminators with a novel differentiable duration modeling approach.

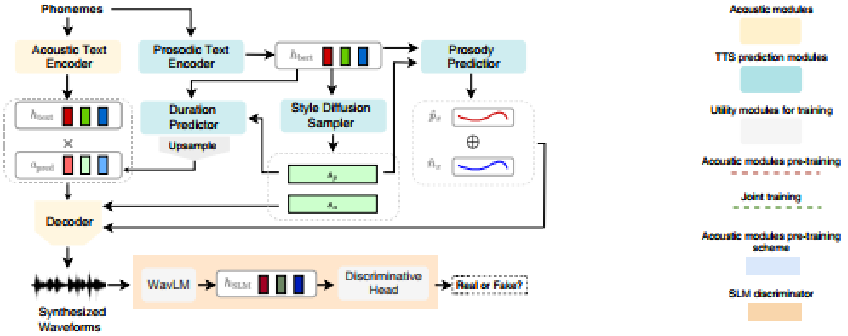
4. Architecture: The model consists of eight modules organized into three categories:

    - Speech generation system (acoustic modules)
    - TTS prediction system
    - Utility system for training

The key architectural figures and diagrams are presented in Figure 9, which shows the training and inference scheme for both single-speaker and multi-speaker cases.



(a) Acoustic modules pre-training and joint training. To accelerate training, the pre-training first optimize modules inside the blue box; the joint training then follows to optimize all components except the pitch extractor, which is used to provide the ground truth label for pitch curves. The duration predictor is trained with only $\mathcal{L}_{\text{dur}}$.



(b) SLM adversarial training and inference. WavLM is pre-trained and not tuned. Unlike (a), the duration predictor is trained E2E with all components using $\mathcal{L}_{slm}$ (eq. 5) via differentiable upsampling. This process is separate from (a) during training as the input texts can be different, but the gradients are accumulated for both processes in each batch to update the parameters.

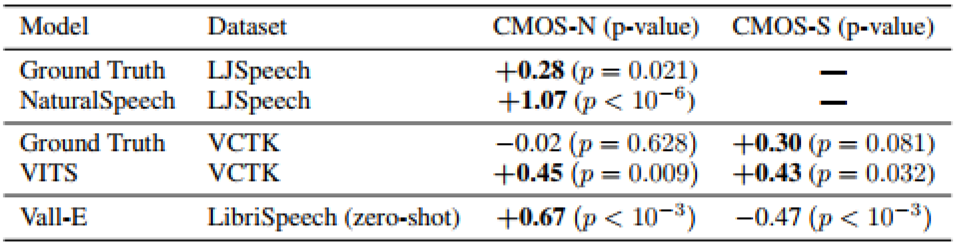
Figure 9 The training and inference framework of StyleTTS 2 varies between single-speaker and multispeaker scenarios (Li et al., 2024).

Li et al. (2024) evaluated StyleTTS 2 and the results demonstrate that StyleTTS 2 are remarkable as described below:

1. StyleTTS 2 surpassed human recordings on the LJSpeech dataset with statistically significant comparative mean opinion scores (CMOS) of +0.28 ($p < 0.05$), as shown in Table 4
2. It achieved human-level performance on the multispeaker VCTK dataset in terms of naturalness (CMOS = -0.02, $p > 0.05$)
3. When trained on the LibriTTS dataset, it outperformed previous publicly available models for zero-shot speaker adaptation

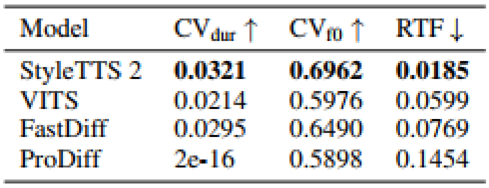
4. The model showed superior speech diversity with the highest coefficient of variation in duration (CVdur) and pitch curve (CVf0) compared to baseline models shown in Table 5
5. It maintained faster processing speeds than other diffusion-based TTS models

Table 4 Comparative mean opinion scores (MOS) for naturalness and similarity were evaluated for StyleTTS 2, with p-values derived from the Wilcoxon test relative to other models. Positive scores signify that StyleTTS 2 outperforms other models (Li et al., 2024).



Model	Dataset	CMOS-N (p-value)	CMOS-S (p-value)
Ground Truth	LJSpeech	**+0.28** ($p = 0.021$)	—
NaturalSpeech	LJSpeech	**+1.07** ($p < 10^{-6}$)	—
Ground Truth	VCTK	−0.02 ($p = 0.628$)	**+0.30** ($p = 0.081$)
VITS	VCTK	**+0.45** ($p = 0.009$)	**+0.43** ($p = 0.032$)
Vall-E	LibriSpeech (zero-shot)	**+0.67** ($p < 10^{-3}$)	−0.47 ($p < 10^{-3}$)

Table 5 The evaluation included speech diversity metrics and real-time factor (RTF). ProDiff and FastDiff were assessed using 4 diffusion steps.



Model	CV_{dur} ↑	CV_{f0} ↑	RTF ↓
StyleTTS 2	**0.0321**	**0.6962**	**0.0185**
VITS	0.0214	0.5976	0.0599
FastDiff	0.0295	0.6490	0.0769
ProDiff	2e-16	0.5898	0.1454

The analysis reveals that StyleTTS 2 substantially advances the state-of-the-art in TTS by combining style diffusion and adversarial training with SLMs. The model's ability to generate diverse and natural speech without requiring reference speech is a notable achievement. However, the researchers acknowledged limitations, mainly in handling large-scale datasets and robustness across diverse linguistic and acoustic conditions. The authors suggest that further improvements are needed to address these challenges and enhance the model's scalability and generalization capabilities.

In conclusion, Li et al. (2024) present StyleTTS 2 as a groundbreaking TTS model that achieves human-level performance through style diffusion and adversarial training with large SLMs. The model's ability to surpass human recordings on single-speaker datasets and match them on multispeaker datasets, along with its potential for zero-shot speaker adaptation, proves its significance in advancing TTS technology. The authors highlight the potential of their innovative approach for future research and applications in speech synthesis.

2.4 Summary of Related Works and Technologies

The following table shows the comparative summary of related image-to-text and text-to-speech works and technologies.

Table 6 Comparative Summary Table of Related Studies and Technologies

Works / Technology	Purpose	Core Technology	Key Features	Strengths	Limitations
Tesseract OCR (Tamizhi-Net) Vasantharajan & Thayasivam (2021)	OCR for low-resource languages (Tamil & Sinhala)	Fine-tuned Tesseract with custom training data	Pre-processing with OpenCV, custom font datasets, training on legacy fonts	Reduced CER & WER (Tamil: 6.03% → 2.61%; Sinhala: 7.61% → 4.74%), built large parallel corpus	Inconsistent font support, high word error rate, long training time

Works / Technology	Purpose	Core Technology	Key Features	Strengths	Limitations
LLaVA-Docent Lee et al. (2024)	AI tutor for art appreciation education	LLaVA + Vicuna + CLIP	Instruction tuning, educational stage modeling, GPT-4 for dialogue generation	Follows structured pedagogical stages, promotes independent thinking, accessible via Hugging Face	Limited dataset size, hallucination risk, lacks portfolio features
Purrfessor Lu et al. (2024)	Dietary chatbot using food images	Fine-tuned LLaVA on food data	CLIP + Vicuna, human-in-the-loop training, UI for interaction	Strong food recognition accuracy, high user satisfaction, outperforms GPT-4 in engagement	Does not capture secondary ingredients, potential dataset bias
Qwen2-VL Wang et al. (2024)	Multimodal perception model for vision-	Vision Transformer + LLM with M-RoPE	Dynamic resolution, video support, multilingual	SOTA in DocVQA, InfoVQA, OCRBench, MathVista,	Struggles with long videos, some reasoning

Works / Technology	Purpose	Core Technology	Key Features	Strengths	Limitations
	language tasks		OCR, 72B param version	multilingual OCR	tasks weaker than GPT-4o
StyleTTS 2 Li et al. (2023)	Text-to-speech generation	Style diffusion + adversarial training with WavLM	End-to-end, no reference speech, speaker adaptation	Surpasses human recordings on some benchmarks, diverse and natural speech	Challenges with large-scale data, OOD robustness issues

For HearSee project, Qwen2-VL 7B and Kokoro TTS (based on StyleTTS 2**)** are chosen as the core technologies because they align strongly with the goals of multimodal learning which is integrating vision and language understanding with natural language output. Qwen2-VL 7B stood out due to its ability to dynamically process images at their native resolutions and handle complex tasks like OCR, visual reasoning, and multilingual document analysis. This makes it ideal for extracting rich, structured information from diverse visual inputs. On the other hand, StyleTTS 2 enables high-quality, expressive text-to-speech synthesis using style diffusion and large speech language models, which allows the system to vocalize image-derived content in a natural and engaging way. Together, these models provide a capable framework for building HearSee, an end-to-end system that can see, understand, and speak thus capturing the essence of multimodal learning and creating a more accessible and interactive experience for users.

Chapter 3: Methodology

3.1 Introduction

This chapter presents a methodology for developing HearSee, an image-text-to-text and text-to-speech web-based application learning tool utilizing Multimodal Large Language Models. The systematic approach outlined herein encompasses the development framework, system architecture and flow, and implementation strategies for creating an effective learning tool that addresses the diverse needs of undergraduate students. The methodology has been structured around the SCRUM framework, an agile development approach that enables iterative development and continuous feedback integration from developer and users. The following sections detail the specific approaches, tools, and techniques that will be employed to ensure both technical and user requirements are met throughout the development process.

3.2 SCRUM Framework

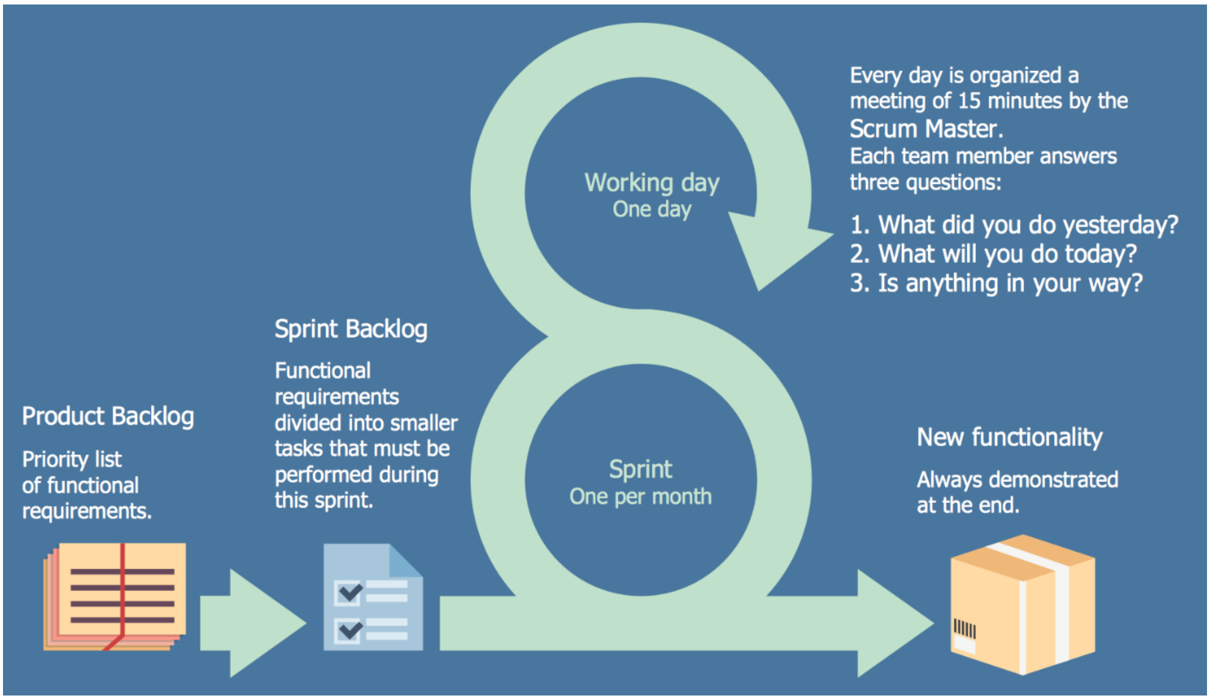


Figure 10 SCRUM Methodology (Scrum Workflow | Scrum Board | Scrum | Scrum Guide, n.d.)

SCRUM, an agile framework depicted in Figure 10, will be tailored for this solo project. It will guide the project's entire lifecycle, enabling regular reassessment and adjustment of priorities to ensure the most valuable features are developed first and the project stays aligned with its goals. The project will be organized into 2-week sprints. At the start of each sprint, tasks will be chosen from the product backlog based on priority and effort estimates, forming the sprint backlog. Each sprint will have a defined objective, such as developing a specific feature or enhancing a particular aspect of the web application.

During the sprint, work will focus on the tasks in the sprint backlog. Instead of daily stand-ups, brief self-reflection sessions will be held to assess progress and address any challenges. Development will adhere to best practices in web development, API integration, and AI model implementation. Regular version control commits will track changes and facilitate rollbacks if necessary.

For a task to be considered complete, it must meet the following criteria:

1. Code is written and documented.
2. Unit tests are created and passed.
3. The feature is integrated into the main branch without conflicts.
4. The feature is tested across multiple browsers and devices.
5. Any required user documentation is updated.

At the end of each sprint, a Sprint Review will ensure completed tasks meet these criteria, followed by a Sprint Retrospective to identify areas for improvement and potential enhancements.

3.3 System Architecture Implementation, Design and Requirements

After conducting the literature review in Chapter 2, the following detailed objectives were developed as the main guidelines for the system architecture, design and requirements for HearSee:

1) Develop a multimodal system for educational content processing:
    a) Conduct a systematic review of current research in multimodal content analysis utilising visual language models and optical character recognition
    b) Develop a system using Qwen2-VL 7B visual language model that is capable of:
        i) Text extraction from PNG and JPG images
        ii) Descriptive explanations of visual content
        iii) Concise summaries of visual content
    c) Conduct system processing testing with an image test set of 30 educational contents (consists of textbook pages, lecture slides, diagrams and charts of undergraduate level)
    d) Evaluate system performance using BERTScore (Zhang et al., 2019), to measure the quality of generated content (text descriptions, transcriptions, and summaries) against human-generated references through these metrics:
        i) Precision: measuring accuracy of generated content, ranges from 0.0 to 1.0
        ii) Recall: measuring completeness of generated content, ranges from 0.0 to 1.0
        iii) F1 Score: measuring overall performance balance, ranges from 0.0 to 1.0
2) Develop a text-to-speech conversion component for system generated text content:
    a) Review current text-to-speech technologies and their application contexts
    b) Implement a text-to-speech system using Kokoro TTS engine for converting processed text into intelligible speech as output

c) Evaluate speech output quality using Mean Opinion Score (MOS) (Streijl et al., 2014), with Likert scale for speech quality assessment through user testing with a minimum of 10 undergraduate students

3) Design and implement a web-based user interface for the learning tool:
    a) Create a responsive front-end interface using Gradio UI library that supports
        i) JPG and PNG image format file uploads with a maximum file size of 10MB
        ii) Audio playback controls with speed and voice selection features
        iii) Chat interface for user interaction with the system
    b) Develop a user guide for learning the functionalities of the website
    c) Establish performance metrics for system performance:
        i) Latency: The average duration of processing time in seconds between the submission of a request query and the receipt of a response
        ii) Total number of words successfully generated
    d) Validate the interface through:
        i) User experience testing with a minimum of 10 undergraduate students, measured through user feedback survey form utilising Likert Scale
        ii) Documentation of user interaction and pain points from the survey, identified through:
            (1) Task completion rate: Percentage of user goals successfully achieved
            (2) Time to completion: How long it takes users to accomplish their intended tasks

3.3.1 Three-tier Architecture

HearSee employs a three-tier architecture that separates the application into presentation, application, and data tiers. This established design pattern was chosen to ensure clear separation of concerns, enhance maintainability, improve scalability, and support future development. Table 7 details the implementation of each tier within HearSee, providing specific codebase examples that demonstrate these architectural principles.

Table 7 Three-tiered Architecture Implementation for HearSee

Aspects	Presentation Tier	Application Tier	Data Tier
Definition	User interface layer that handles user interactions and display	Business logic layer that processes data and implements core functionality	Data access layer that manages configuration, settings, and external data sources
Primary Purpose	Renders UI components and captures user inputs	Processes user requests and implements business rules	Provides data access, validation, and configuration
Technology	Gradio web interface components	Python service classes	Configuration files and utility modules

Aspects	Presentation Tier	Application Tier	Data Tier
Key Files	ui/chat_interface.py ui/components.py ui/guide_interface.py	services/image_service.py services/replicate_service.py services/tts_service.py	config/settings.py config/logging_config.py utils/validators.py utils/image_utils.py
Core Responsibilities	• Creating UI components • Defining layout and styling • Capturing user inputs • Displaying results	• Processing user inputs • Implementing business logic • Communicating with external APIs • Error handling and logging • Data transformation	• Centralizing configuration • Defining constants • Data validation • Providing utilities • Configuring system settings
Example Components	• Chat interface • Image upload • Text input • Buttons • Gallery displays	• Image processing • API integration • Text-to-speech conversion • Error handling	• Model constants • API configuration • Image size limits • Voice types

Aspects	Presentation Tier	Application Tier	Data Tier
	• Audio playback		• Initial chat history

3.3.2 C4 Architecture Design

This section presents a supplementary C4 architecture visualization for the HearSee web application. The C4 model provides a hierarchical approach to describing software architecture at different levels of abstraction. The design includes system context diagrams, container diagrams, and component diagrams. Additionally, detailed system analysis and design are supported by UML diagrams in the System Analysis and Design subsection.

Figure 11 system context diagram below illustrates HearSee and its relationships with users and external systems:

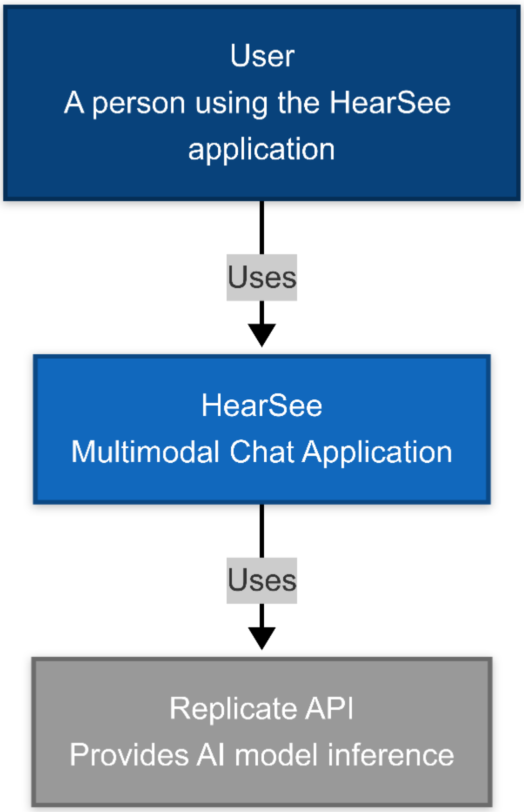


Figure 11 System Context Diagram of HearSee

Figure 12 container diagram below shows the high-level technical building blocks of the HearSee system:

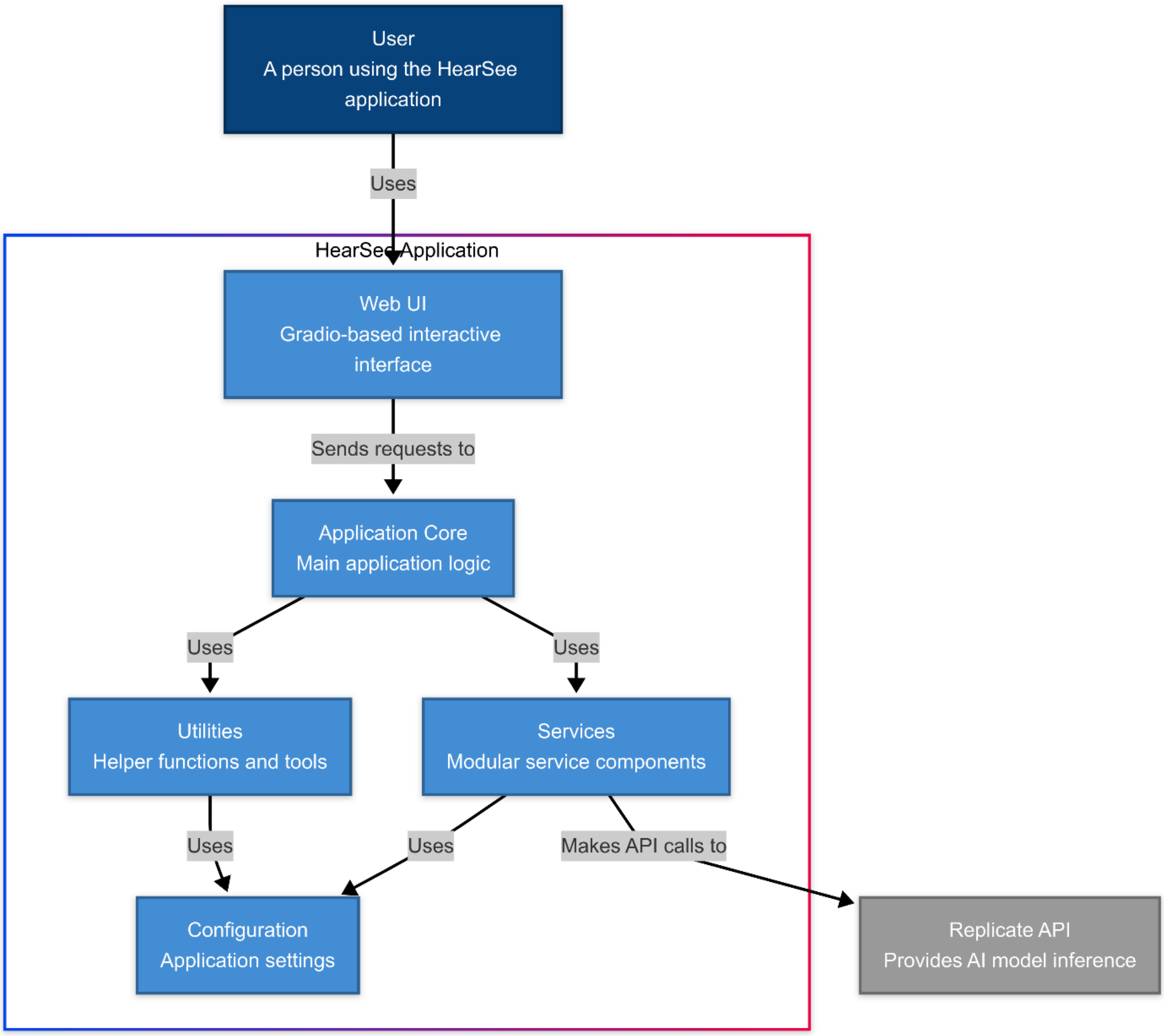


Figure 12 Container Diagram of HearSee

Figure 13 component diagram below breaks down the containers into their principal structural elements.

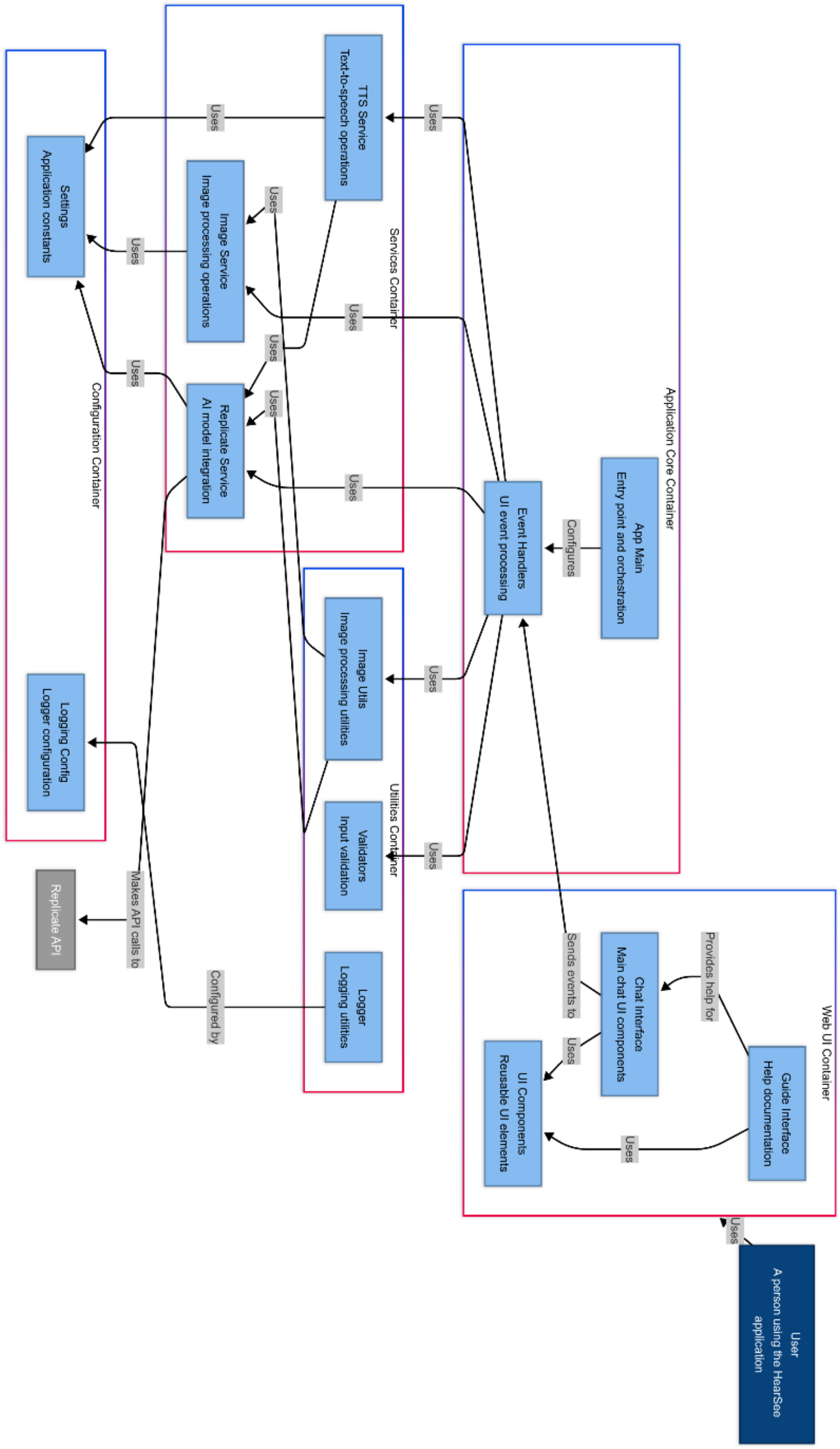


Figure 13 Component Diagram of HearSee

3.4 System Analysis and Design

The following subsections elaborates on the system analysis and design for HearSee. These serve as the technical framework for the system development.

3.4.1 Use Case Diagram

Figure 14 illustrates the use case diagram for HearSee that processes images and text to deliver results such as transcriptions, summaries, responses, and optional text-to-speech outputs. Table 8 presents the descriptions for the Use Case diagram.

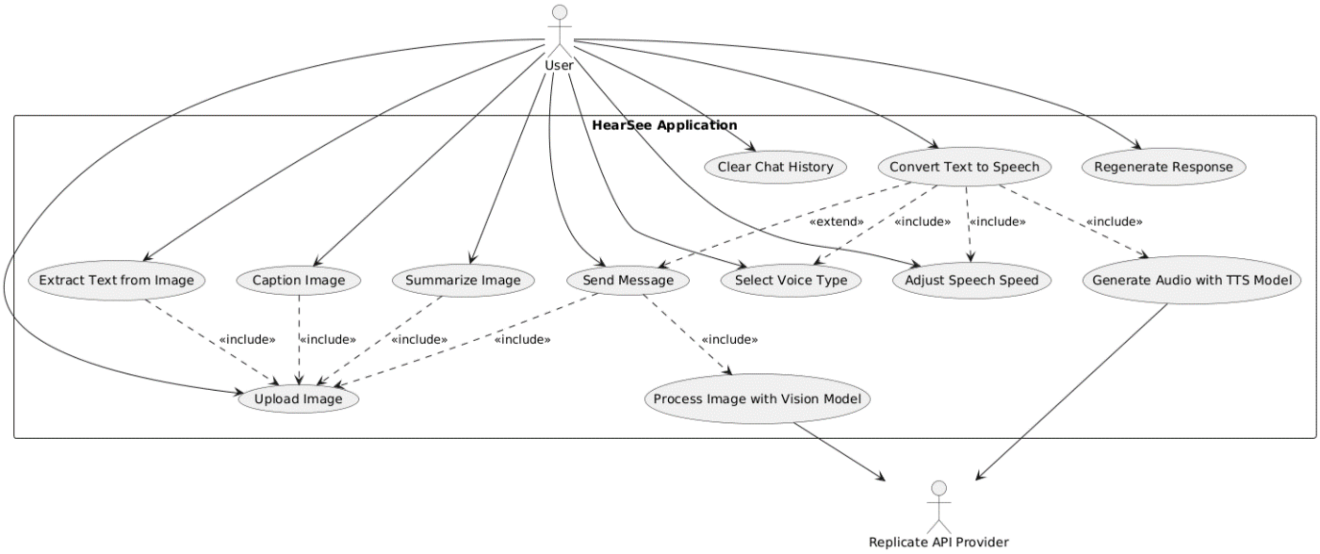


Figure 14 Use Case Diagram for HearSee

Table 8 Use Case Diagram Description

Use Cases	Actor	Description	Precondition	Postcondition
Upload Image	User	User uploads an image to the application for analysis	None	Image is displayed in the gallery and available for processing
Send Message	User	User sends a text message related to the uploaded image	Image must be uploaded	Message is processed by the vision model and a response is displayed
Regenerate Response	User	User requests a new response for the last message	At least one message exchange must exist	New response is generated and displayed
Clear Chat History	User	User clears the entire conversation history	None	Chat history is reset to initial state
Extract Image from Text	User	User requests text extraction (OCR) from the uploaded image	Image must be uploaded	Extracted text is displayed in the chat

Use Cases	Actor	Description	Precondition	Postcondition
Caption Image	User	User requests a brief caption for the uploaded image	Image must be uploaded	Image caption is displayed in the chat
Summarize Image	User	User requests a detailed analysis/summary of the uploaded image	Image must be uploaded	Image summary is displayed in the chat
Convert Test to Speech	User	User converts the last bot response to speech	At least one bot response must exist	Audio is generated and available for playback
Select Voice Type	User	User selects the voice type for text-to-speech conversion	None	Voice type preference is stored for TTS conversion
Adjust Speech Speed	User	User adjusts the playback speed for text-to-speech conversion	None	Speed preference is stored for TTS conversion
Process Image with Vision Model	Replicate API Provider	System processes the image and user message with the vision model	Valid image and message must be provided	MLLM-generated response is returned

Use Cases	Actor	Description	Precondition	Postcondition
Generate Audio with TTS Model	Replicate API Provider	System converts text to speech using the TTS model	Valid text, voice type, and speed must be provided	Audio file is generated and returned

3.4.2 Activity Diagrams

This section contains the activity diagrams for the HearSee application, showing the workflows, processes and activities within the system.

Figure 15 activity diagram below illustrates the process of uploading an image to the HearSee application, which is the initial step required for all image-based interactions. It shows how the system validates the uploaded image, stores it in memory, and updates the UI to enable further interaction options.

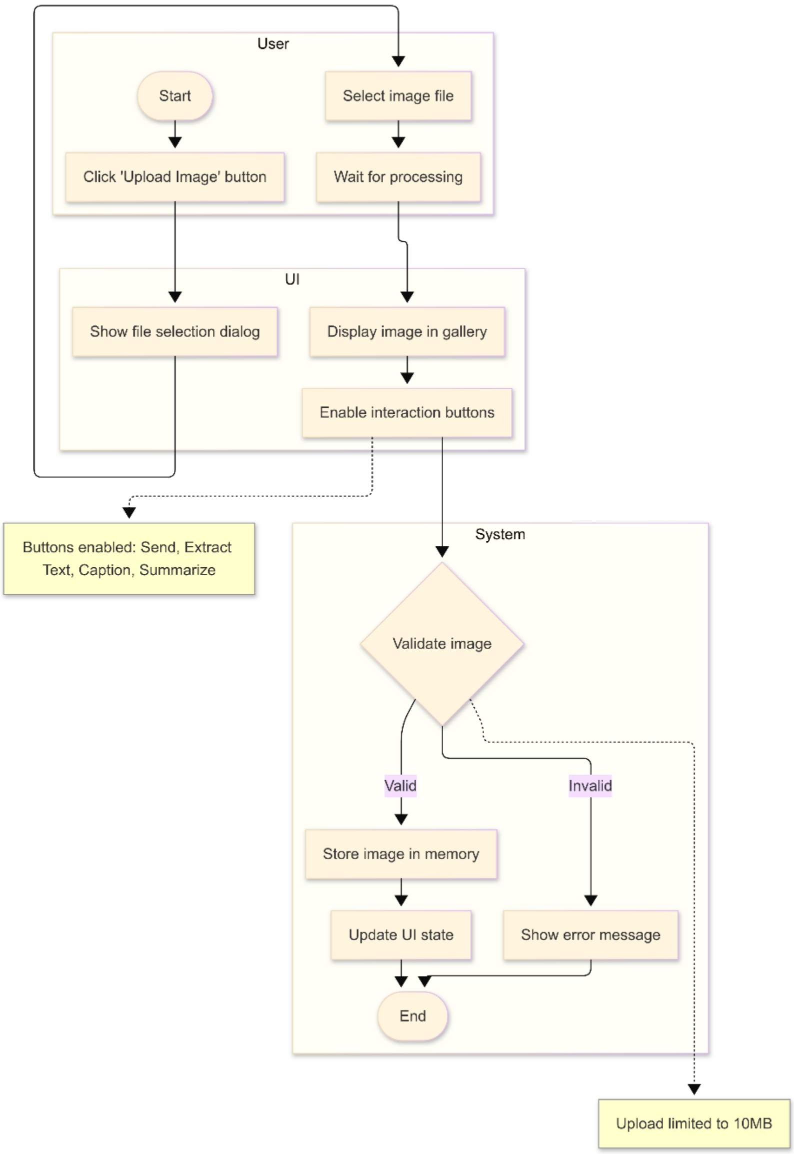


Figure 15 Image Upload Activity Diagram

Figure 16 activity diagram below illustrates the core conversational functionality of the HearSee application, showing how users can interact with the AI vision model after uploading an image.

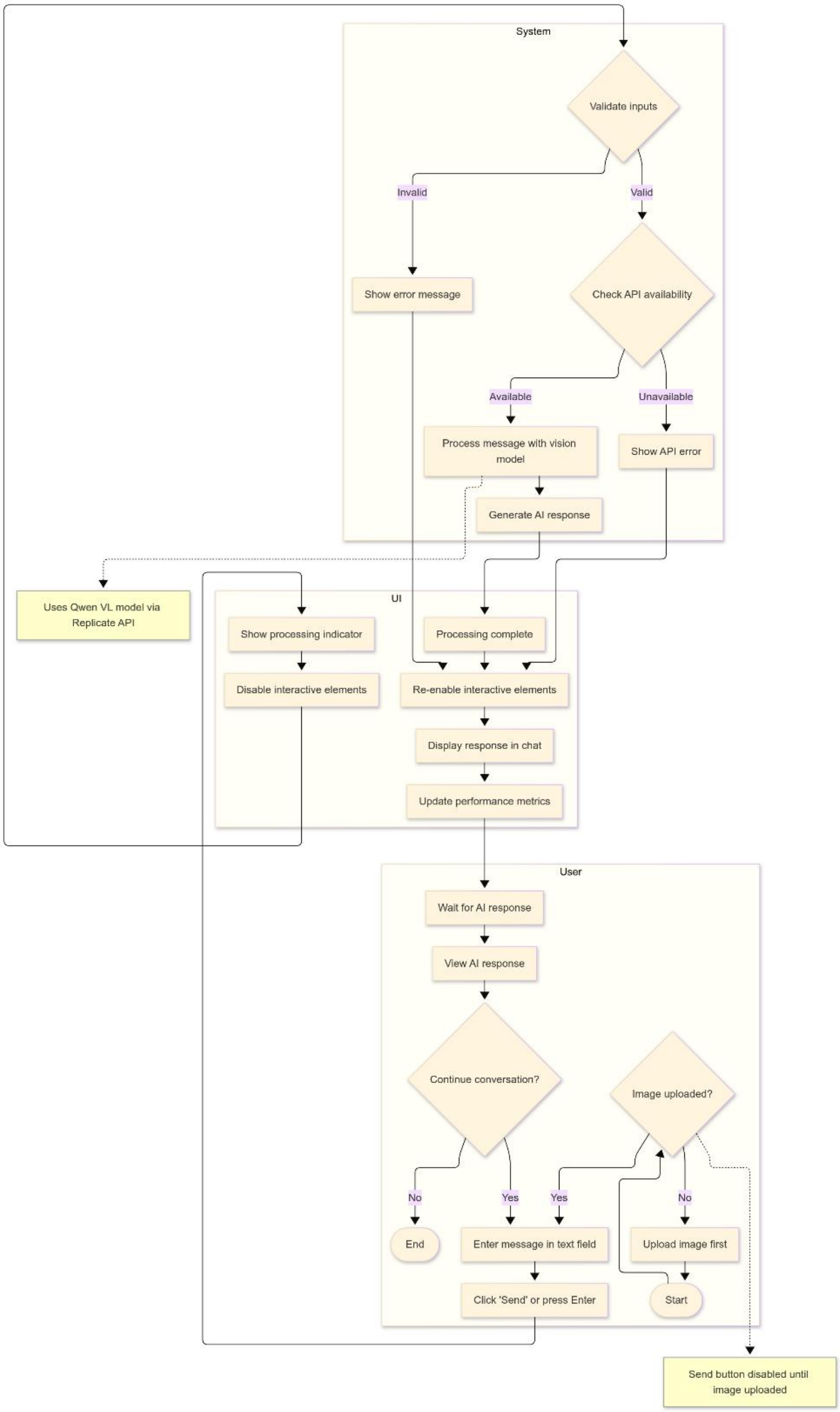


Figure 16 Chat Interaction Activity Diagram

Figure 17 activity diagram below outlines the specialized workflow for extracting text from images in the HearSee application. It shows how the system processes an uploaded image specifically for text recognition, sends it to the vision model with specialized prompts, and returns the extracted text to the user.

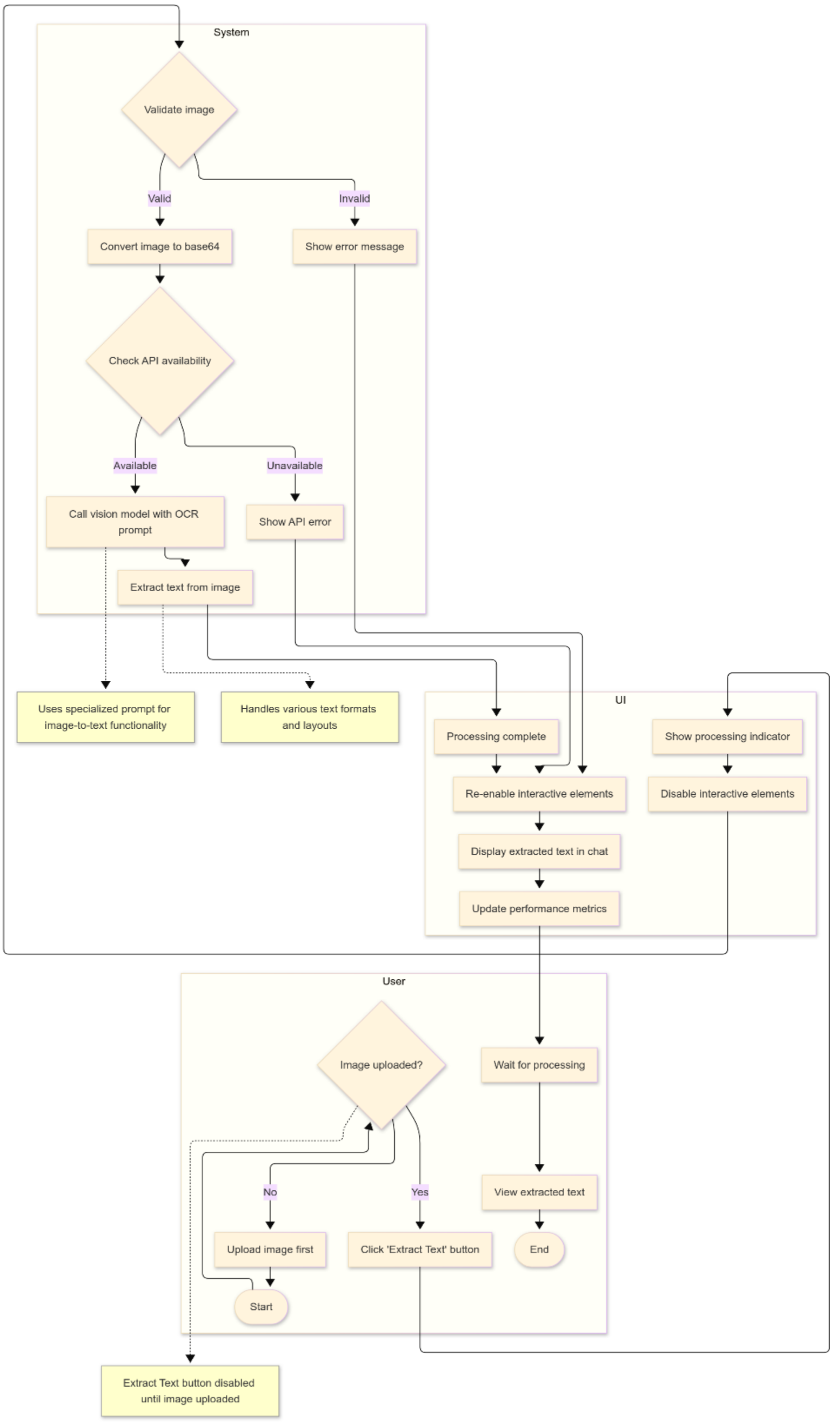


Figure 17 Text Extraction Activity Diagram

Figure 18 activity diagram below illustrates the process of generating descriptive captions for uploaded images. It shows how the system uses the vision model with specialized prompting to create detailed descriptions focusing on objects, people, scenery, colours, and composition.

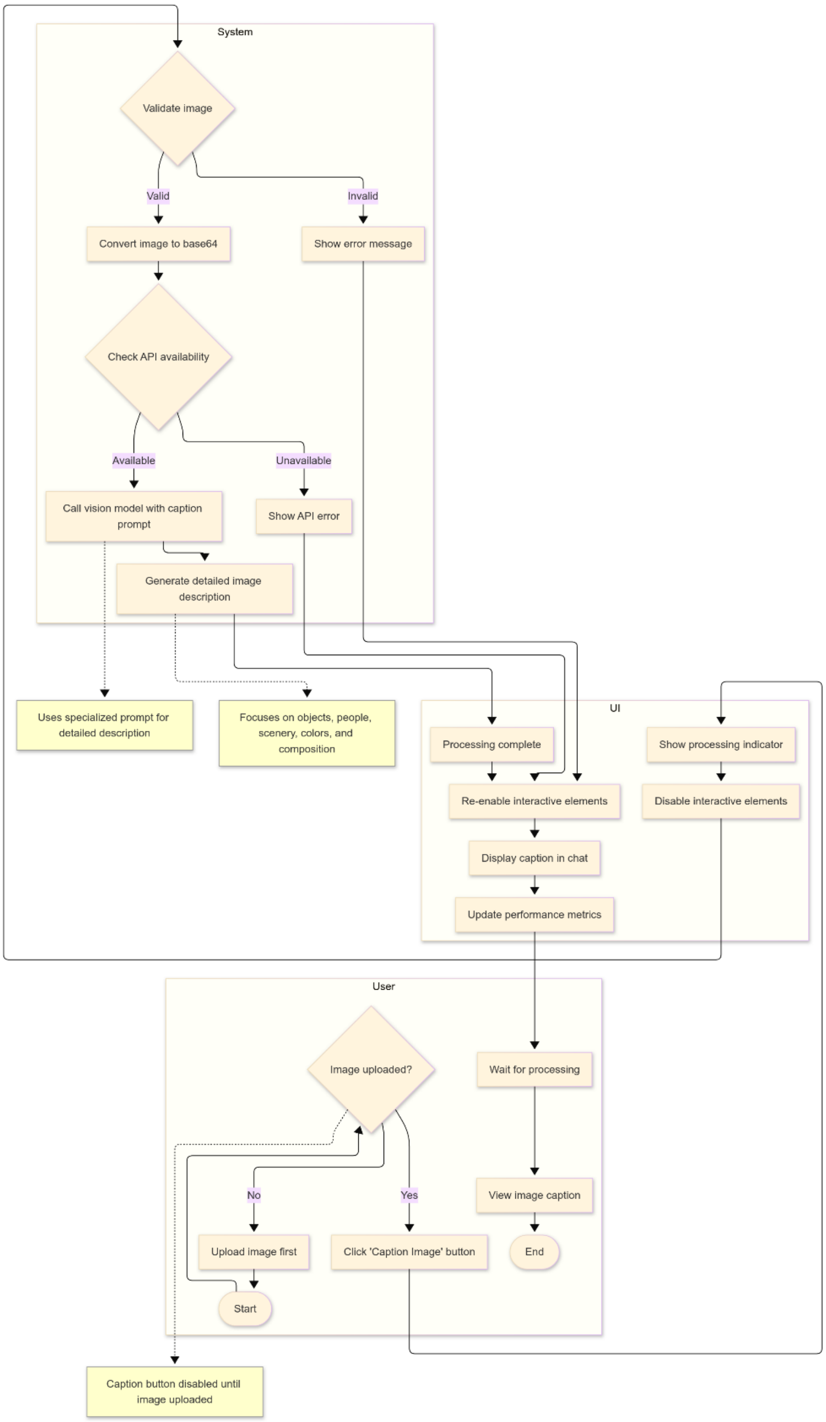


Figure 18 Image Captioning Activity Diagram

Figure 19 activity diagram below presents the workflow for generating contextual summaries of uploaded images. This process creates more in-depth analysis including context, interpretation, and significance of image elements.

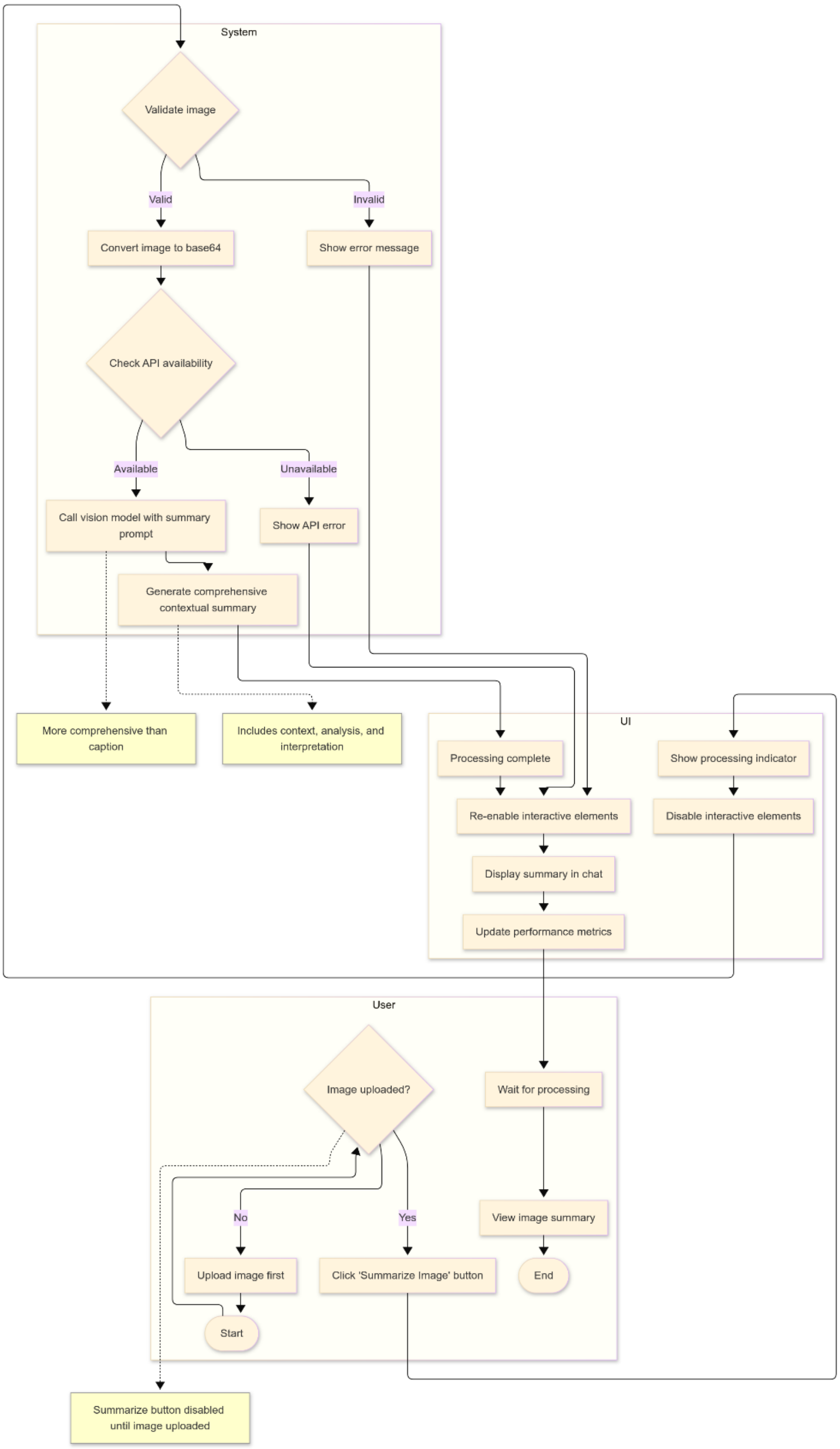


Figure 19 Image Summarization Activity Diagram

Figure 20 activity diagram below details the process of converting AI-generated text responses into spoken audio. It shows how users can select voice types and adjust speech speed before the system processes the text through a TTS model and delivers playable audio.

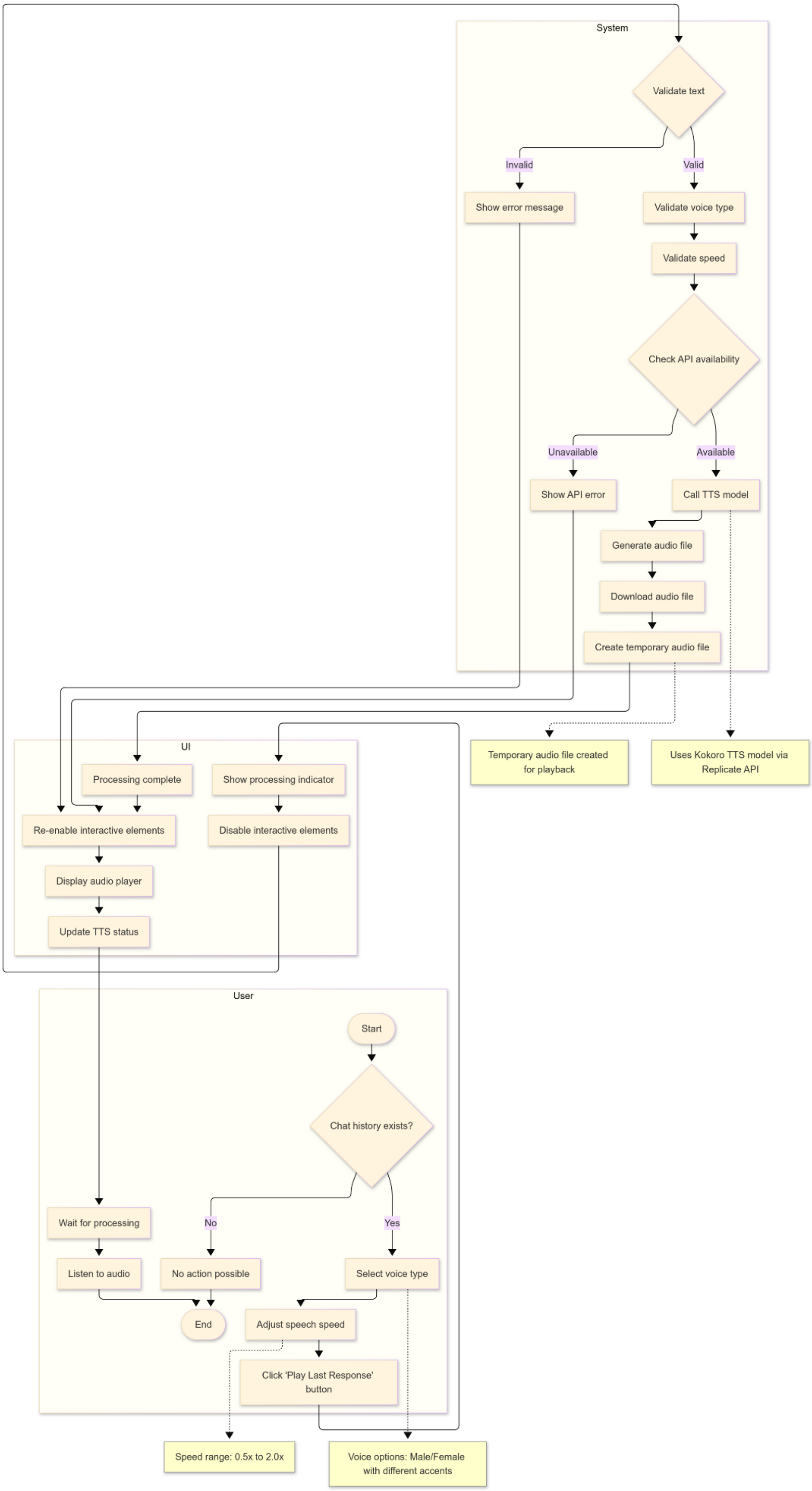


Figure 20 Text-to-Speech Conversion Activity Diagram

Figure 21 activity diagram below illustrates the process of regenerating an AI response to the last user message. It shows how the system extracts the previous user message, removes the last conversation pair, and processes the message again to generate a new response.

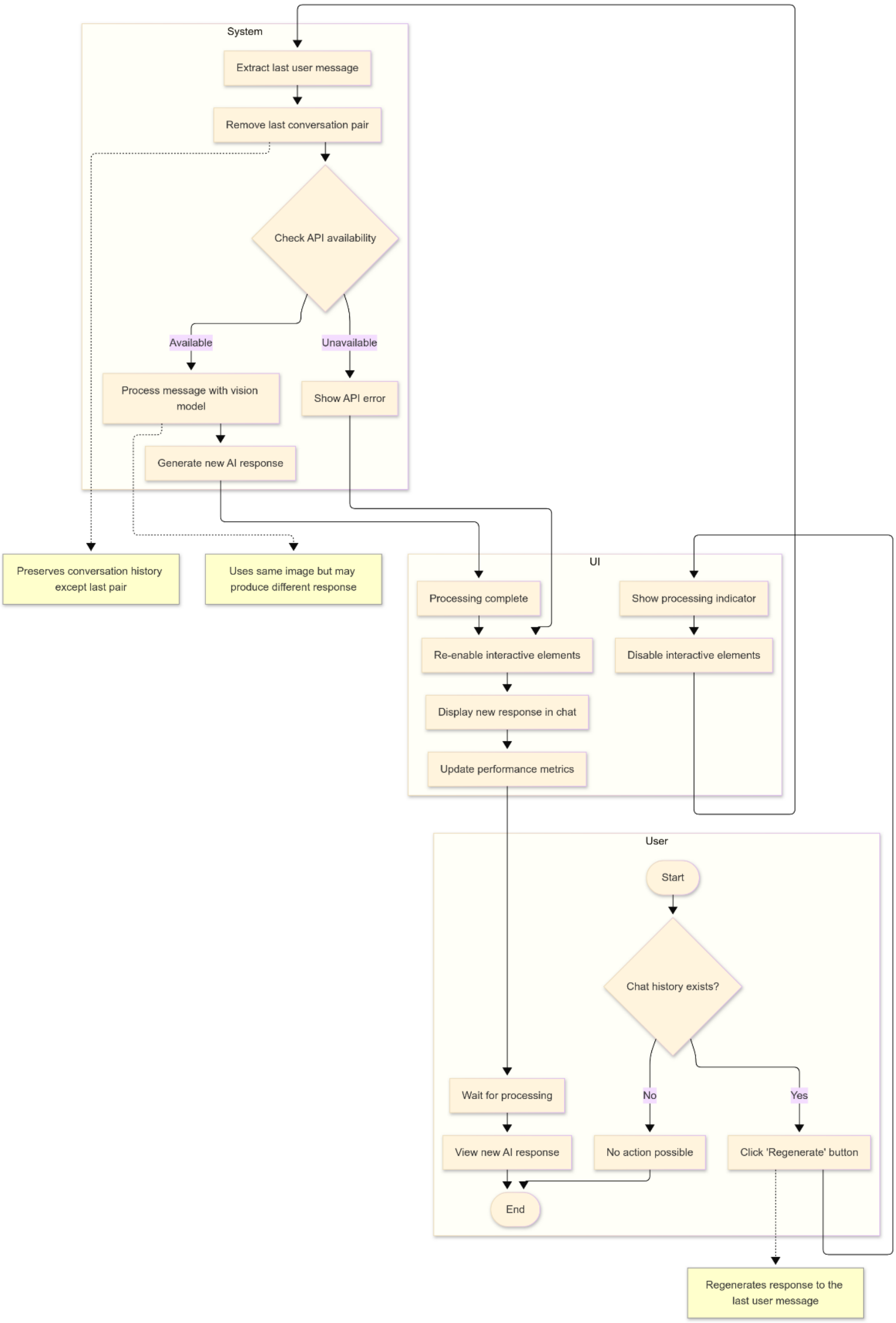


Figure 21 Regenerate Response Activity Diagram

Figure 22 activity diagram below outlines the process of resetting the application to its initial state. It shows how the system clears the chat history, image gallery, and performance metrics, starting a fresh session.

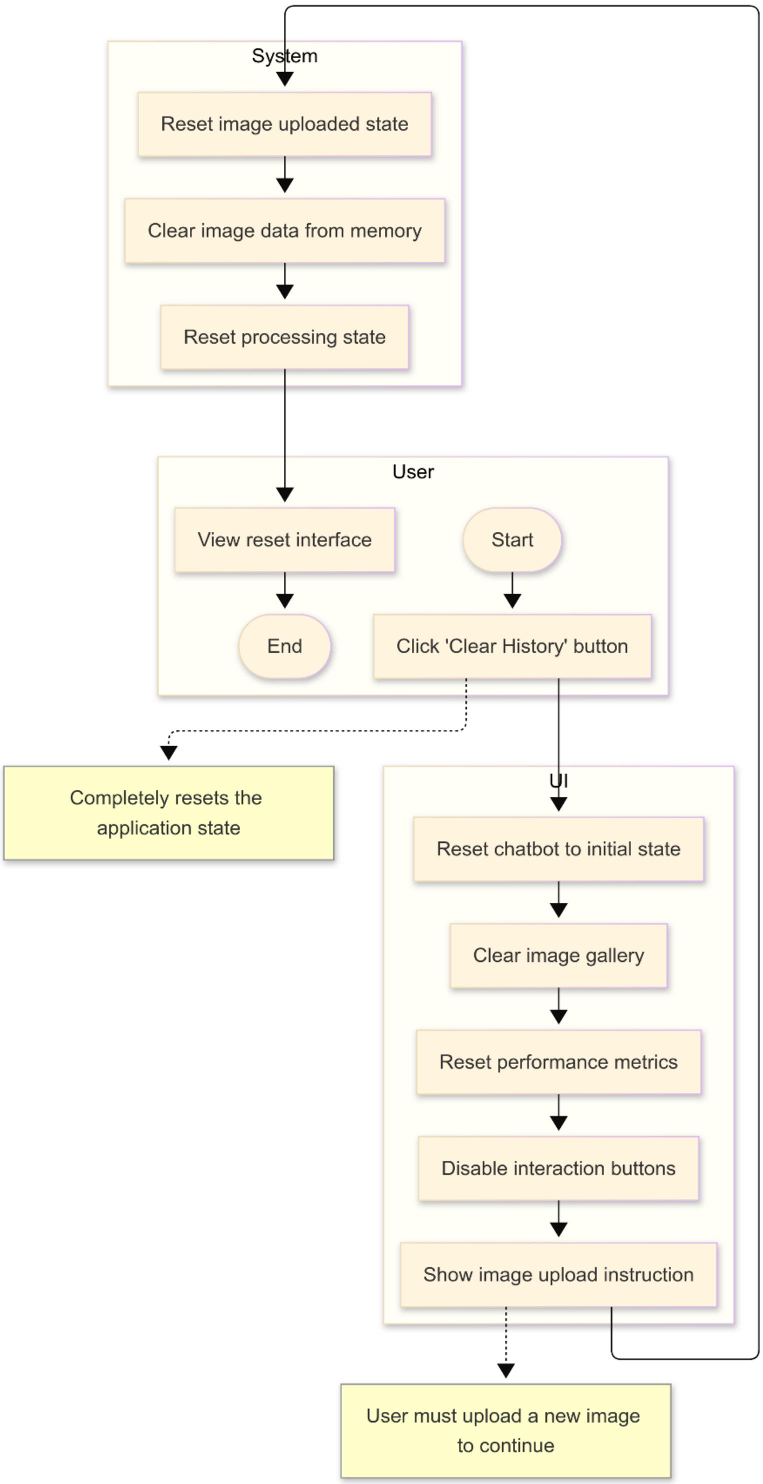


Figure 22 Clear History Activity Diagram

Figure 23 activity diagram below details the system's approach to handling various types of errors that may occur during operation. It illustrates how exceptions are caught, logged, categorized by type, and presented to users with appropriate messages.

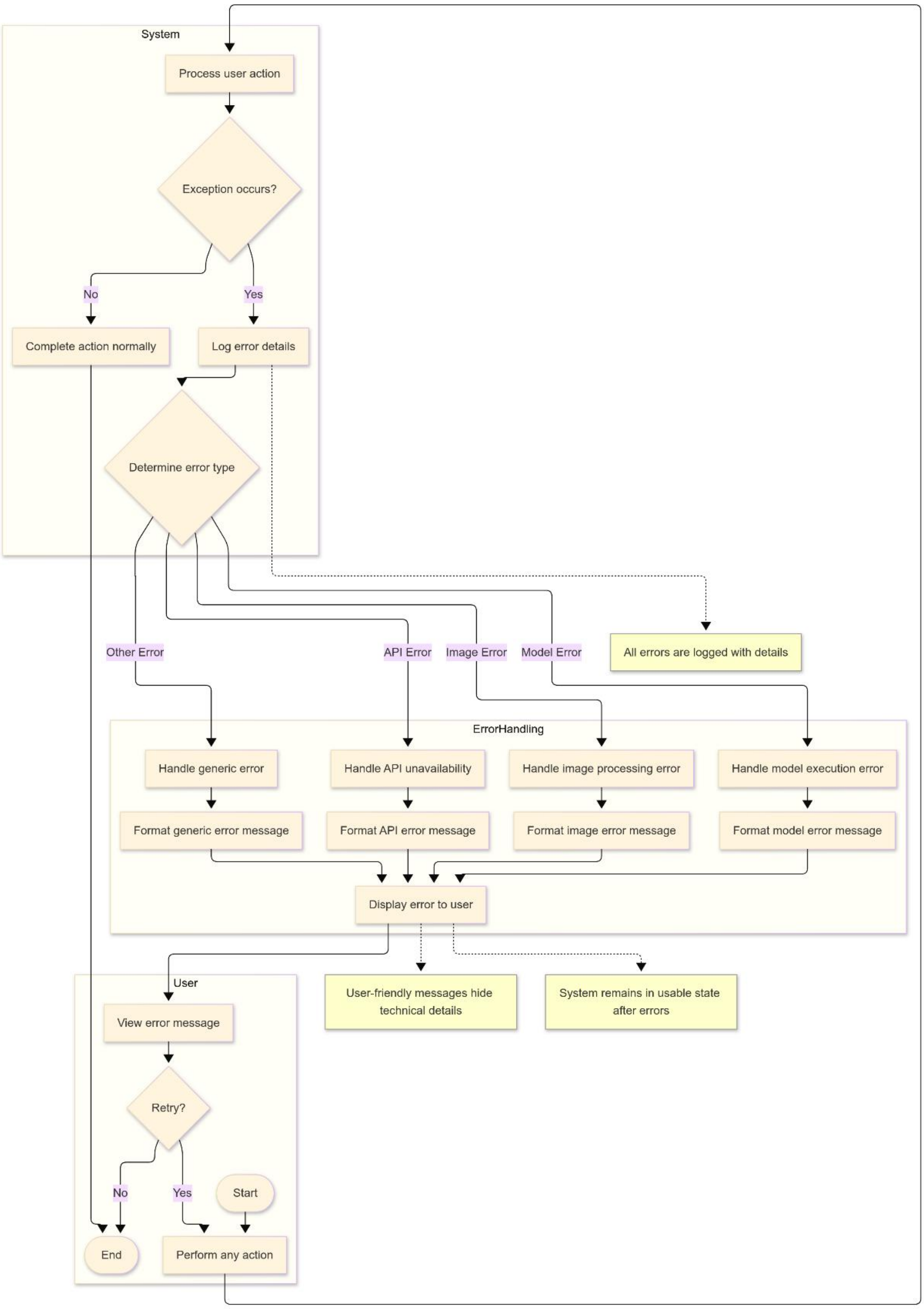


Figure 23 Exception Handling Activity Diagram

Figure 24 activity diagram below depicts the initialization sequence when the HearSee application is launched. It shows how the system loads environment variables, configures logging, creates the UI theme, initializes components, and sets up event handlers before launching the application.

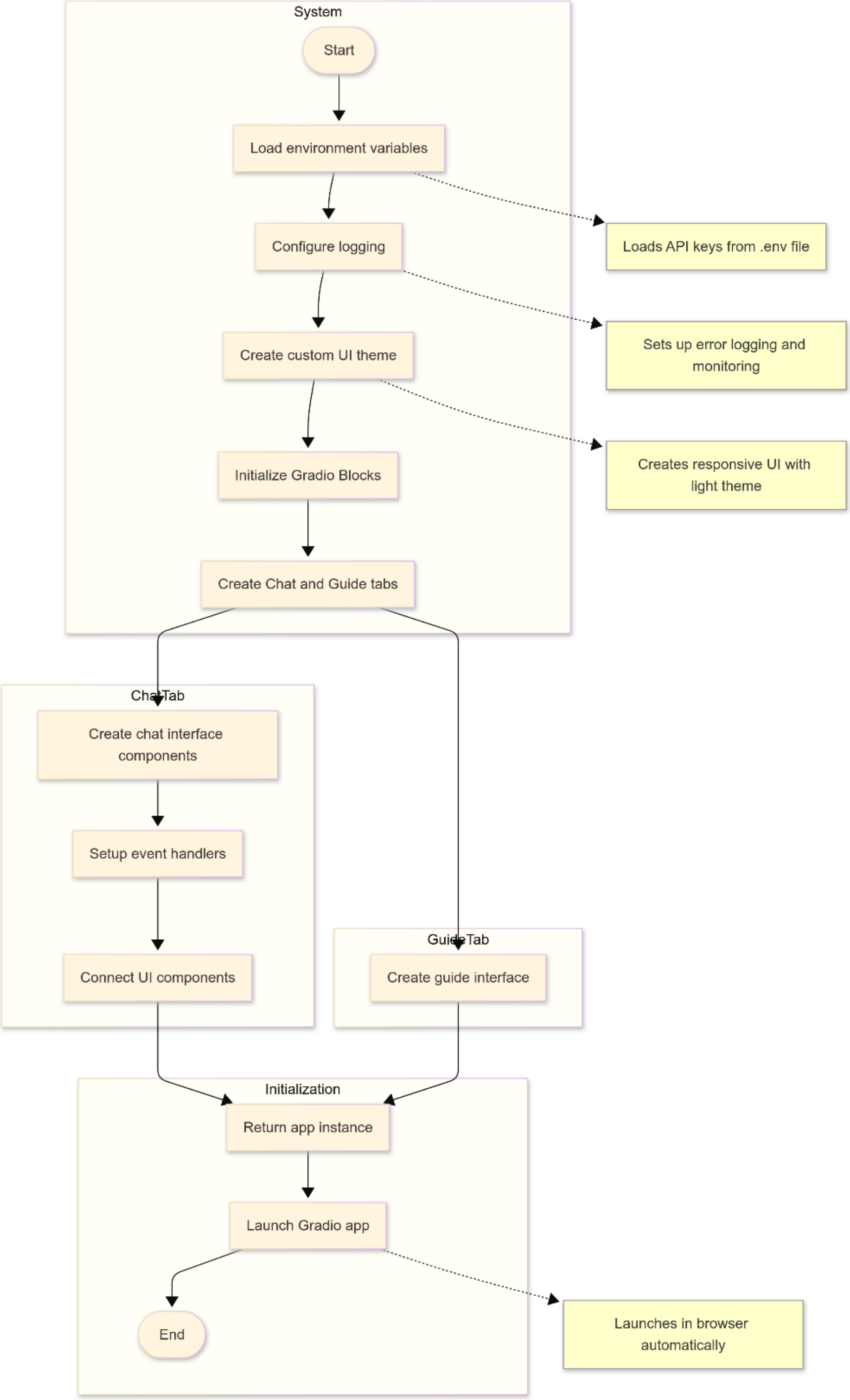


Figure 24 System Startup Activity Diagram

3.4.3 Class Diagram

Figure 25 presents the class diagram illustrating the relationships between classes, their attributes, and methods within HearSee, a multimodal chat application with vision and voice capabilities. The web application enables users to upload images, analyze them using AI vision models, and convert text responses to speech. The diagram is organized into four logical components: user interface, services, utilities, and configuration.

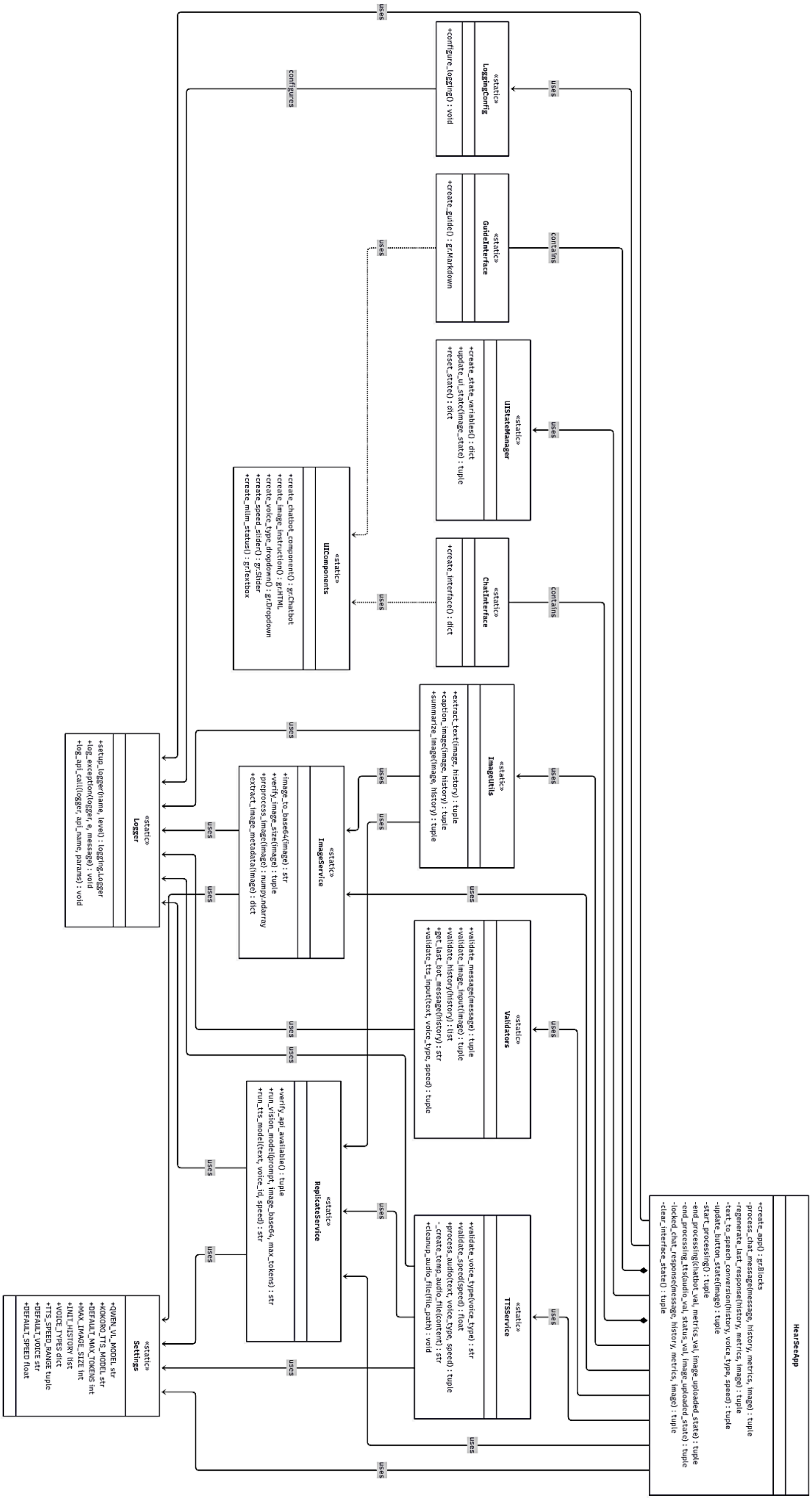


Figure 25 Class Diagram of HearSee

3.4.4 Data Flow Diagram

Figure 26 represents the data flow of HearSee which illustrates the movement of data through the system and the interactions between processes, external entities and data stores.

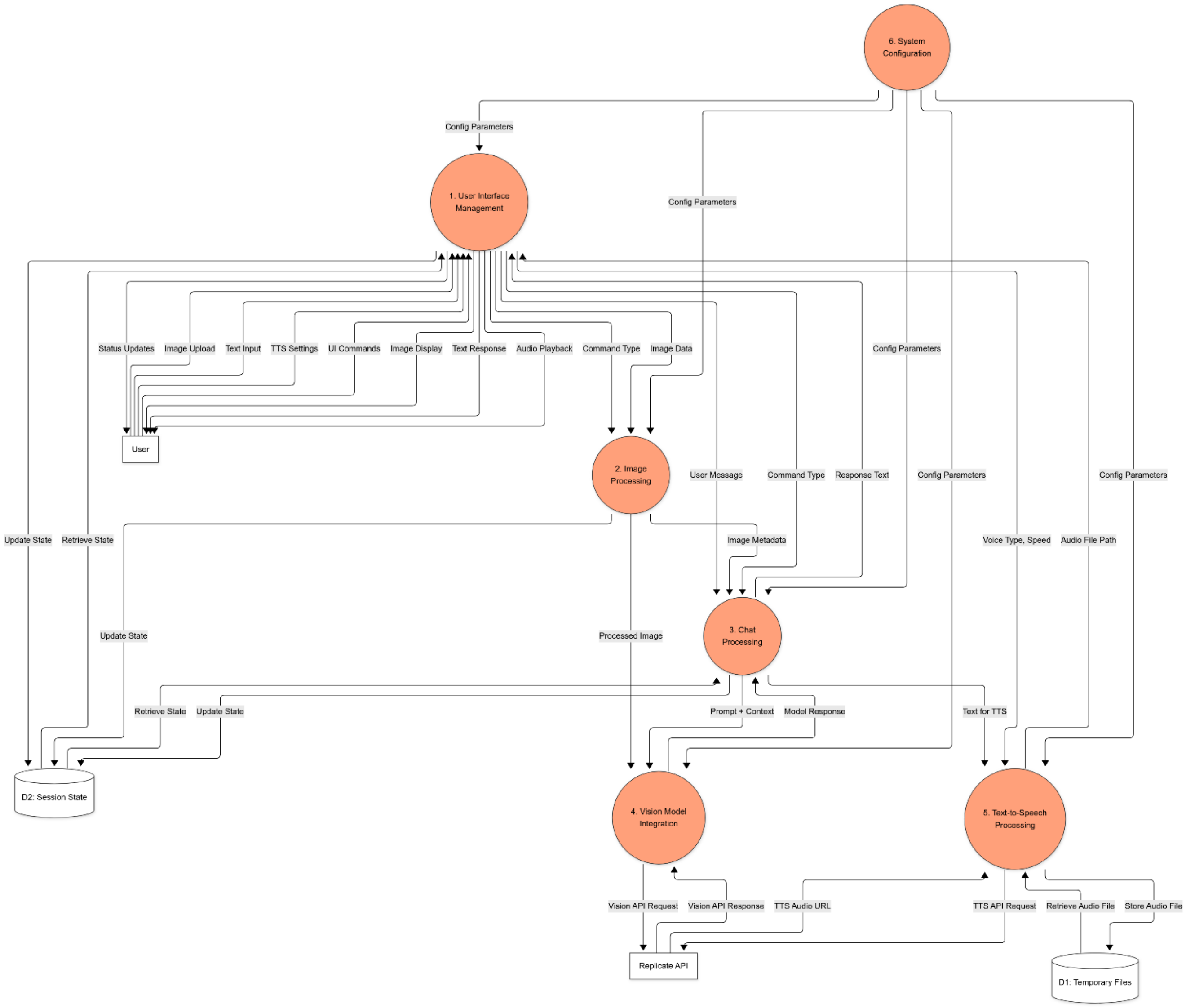


Figure 26 Data Flow Diagram of HearSee

Process Descriptions

Figure 26 breaks down the HearSee system into six major processes:

1. User Interface Management
    - Handles all user interactions through the Gradio web interface
    - Displays images, text responses, and status updates

    - Manages audio playback
    - Routes commands to appropriate processes
2. Image Processing
    - Validates and processes uploaded images
    - Converts images to appropriate formats (base64)
    - Extracts image metadata
    - Prepares images for AI model processing
3. Chat Processing
    - Manages conversation context and history
    - Formats prompts for the vision model
    - Processes model responses
    - Prepares text for TTS conversion
4. Vision Model Integration
    - Handles communication with Replicate API for vision models
    - Sends formatted requests with images and prompts
    - Receives and processes model responses
    - Handles API errors and retries
5. Text-to-Speech Processing
    - Validates TTS parameters (voice type, speed)
    - Sends TTS requests to Replicate API
    - Downloads and manages audio files
    - Provides audio files for playback
6. System Configuration
    - Manages application settings and parameters
    - Provides configuration values to all processes

    - Handles environment variables and defaults

Data Store Descriptions

The data stores in the system include:

- D1: Temporary Files
    - Purpose: Stores downloaded audio files temporarily
    - Content: WAV audio files from TTS processing
    - Persistence: Files are deleted after use
- D2: Session State
    - Purpose: Maintains the current application state
    - Content: Conversation history, uploaded images, UI state
    - Persistence: In-memory during the application session

Data Flow Descriptions

Figure 26 illustrates several key data flows through the system:

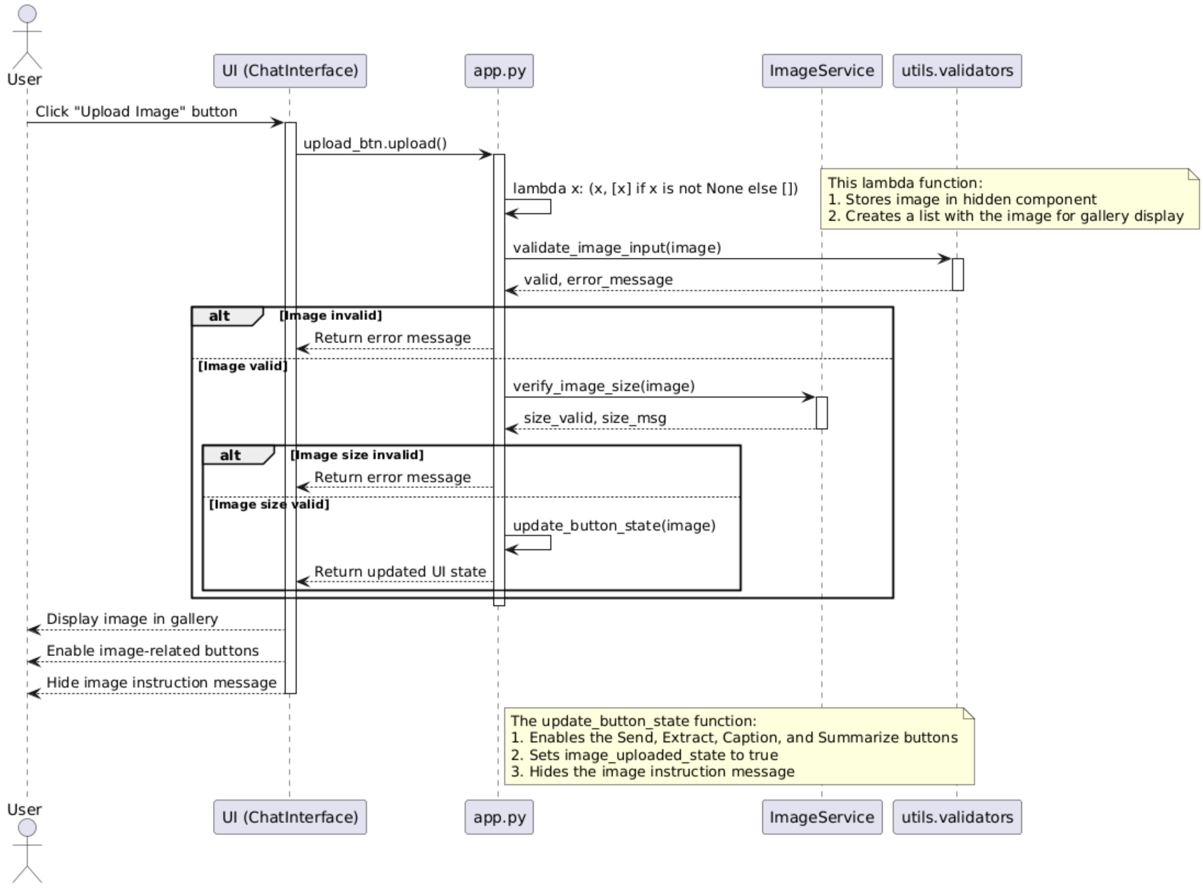
1. Image Processing Flow:
    - User uploads an image
    - UI Management
    - Image Processing
    - Vision Model Integration
    - Replicate API
    - Vision Model Integration
    - Chat Processing
    - UI Management
    - User
2. Text Input Flow:
    - User enters text

    - UI Management
    - Chat Processing
    - Vision Model Integration
    - Replicate API
    - Vision Model Integration
    - Chat Processing
    - UI Management
    - User
3. TTS Processing Flow:
    - Chat Processing provides text
    - TTS Processing
    - Replicate API
    - TTS Processing
    - Temporary Files
    - TTS Processing
    - UI Management
    - User
4. Configuration Flow:
    - System Configuration
    - All processes
    - Affects all data processing

3.4.5 Sequence Diagrams

This section contains the sequence diagrams for the HearSee application, showing the interactions between actors and system components for each use case defined in the use case diagram.

Figure 27 sequence diagram below shows the interactions when a user uploads an image to the application.



This lambda function:
1. Stores image in hidden component
2. Creates a list with the image for gallery display

The update_button_state function:
1. Enables the Send, Extract, Caption, and Summarize buttons
2. Sets image_uploaded_state to true
3. Hides the image instruction message

Figure 27 Upload Image Sequence Diagram

Figure 28 sequence below diagram shows the interactions when a user sends a message related to an uploaded image.

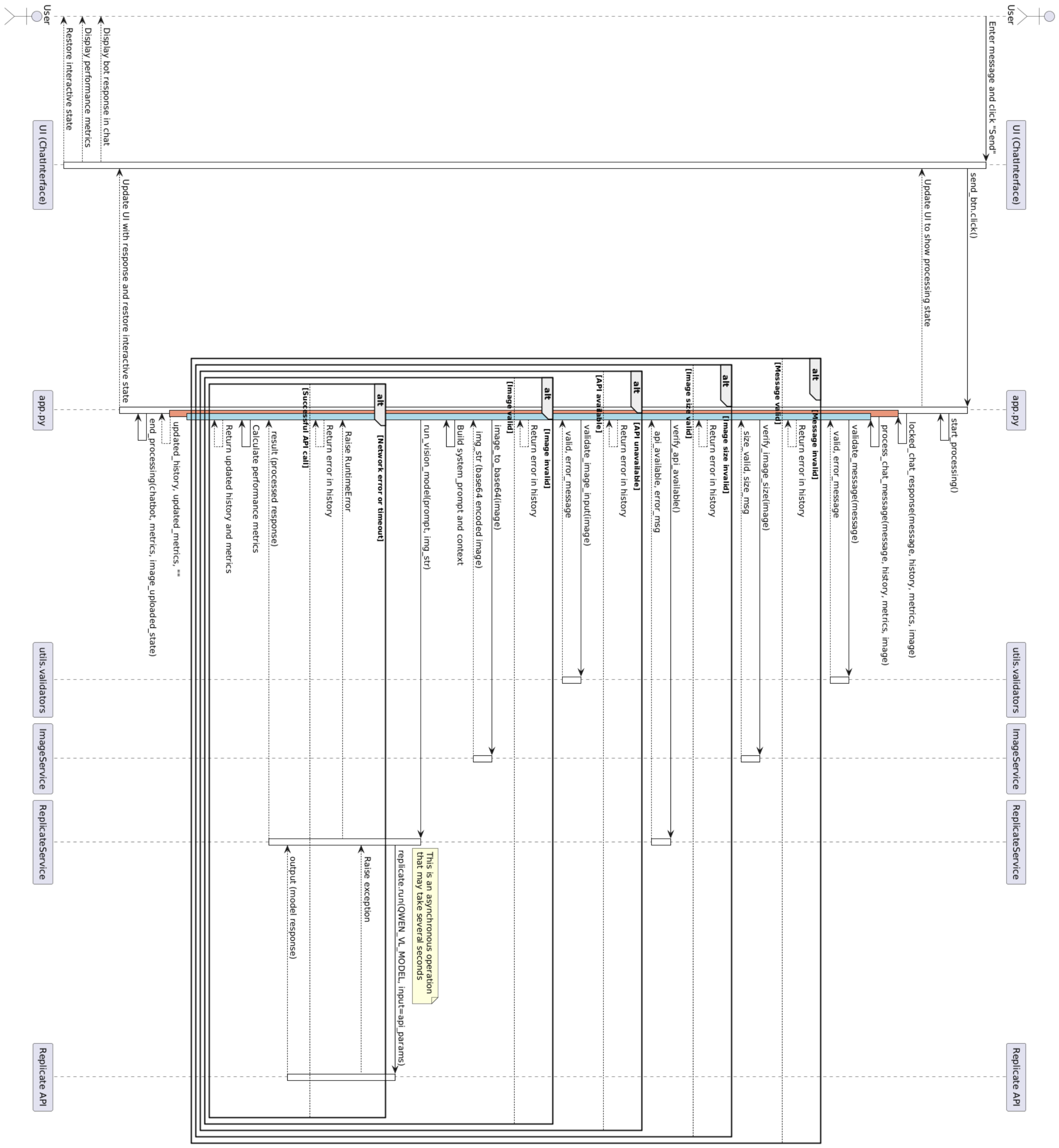


Figure 28 Send Message Sequence Diagram

Figure 29 sequence diagram below shows the interactions when a user requests to regenerate the last response.

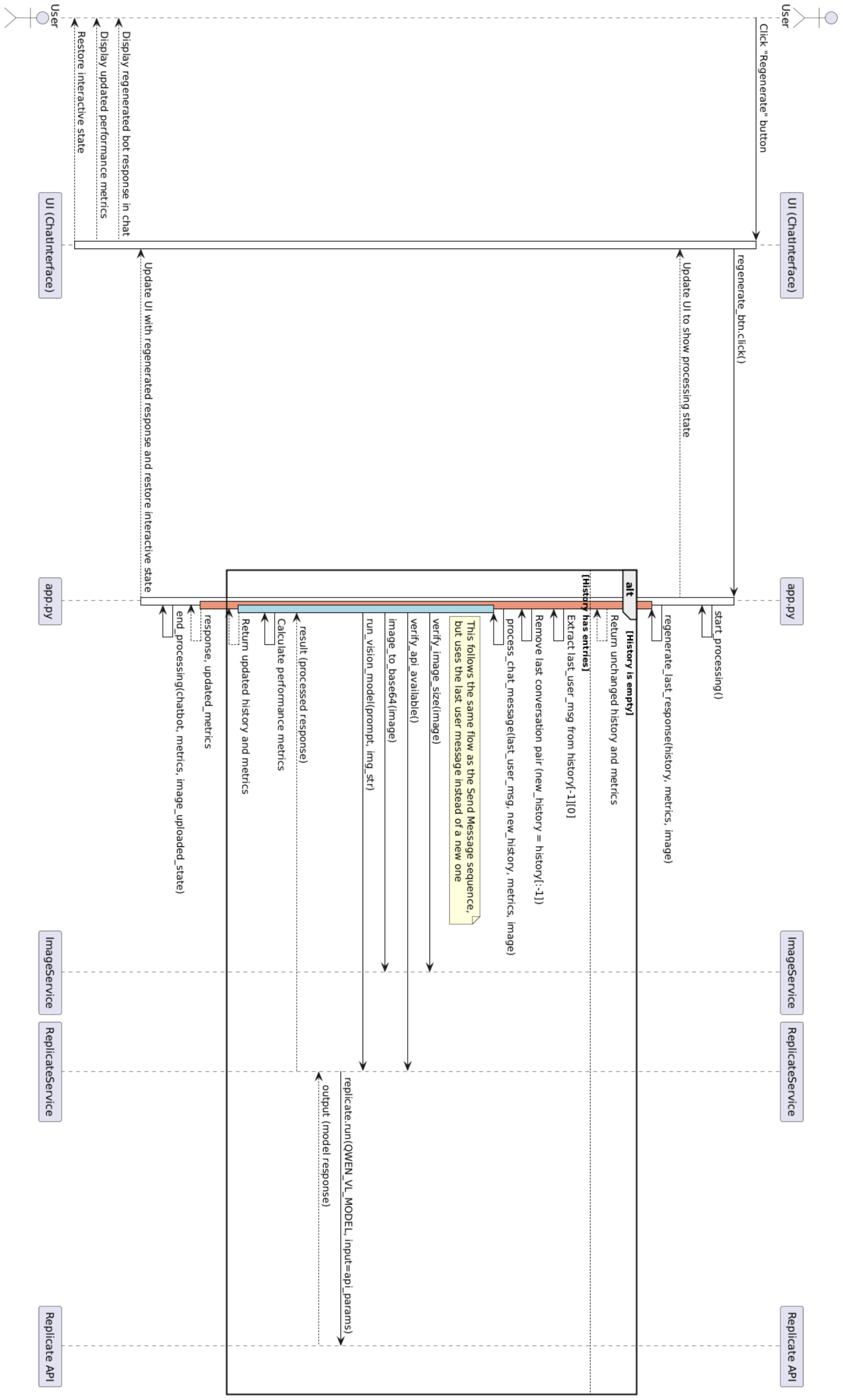


Figure 29 Regenerate Response Sequence Diagram

Figure 30 sequence diagram below shows the interactions when a user clears the chat history.

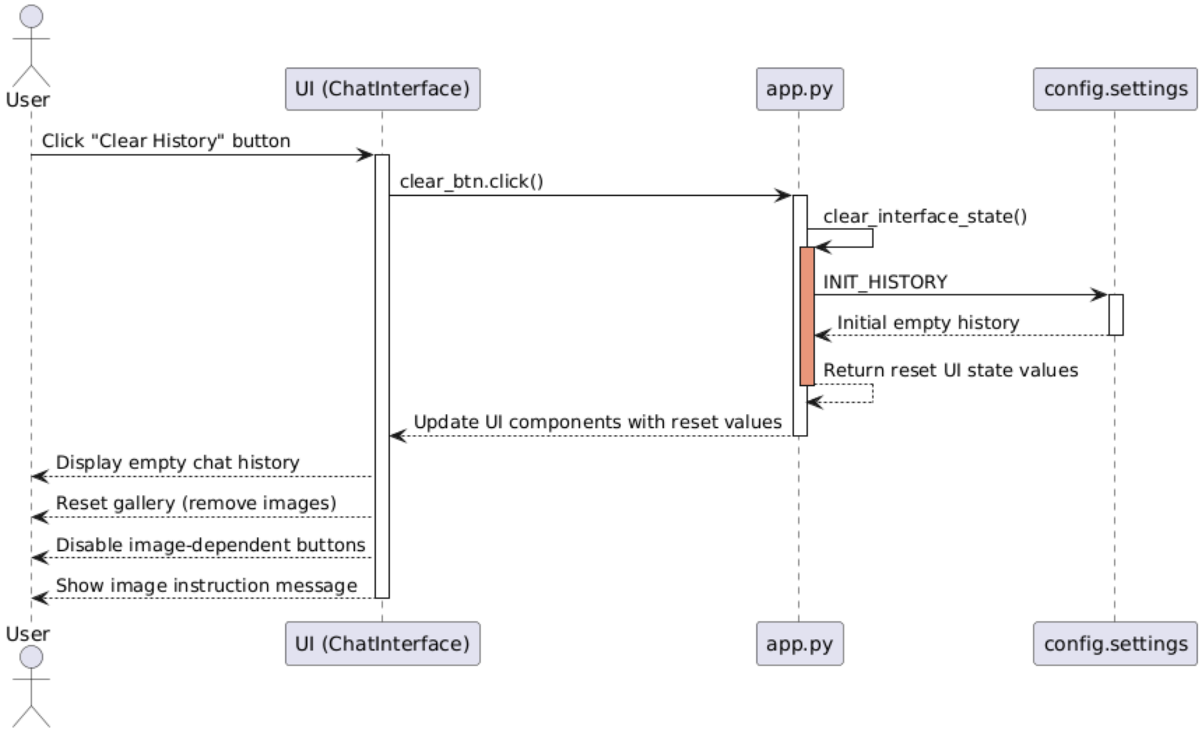


Figure 30 Clear Chat History Sequence Diagram

Figure 31 sequence diagram below shows the interactions when a user requests to extract text from an uploaded image.

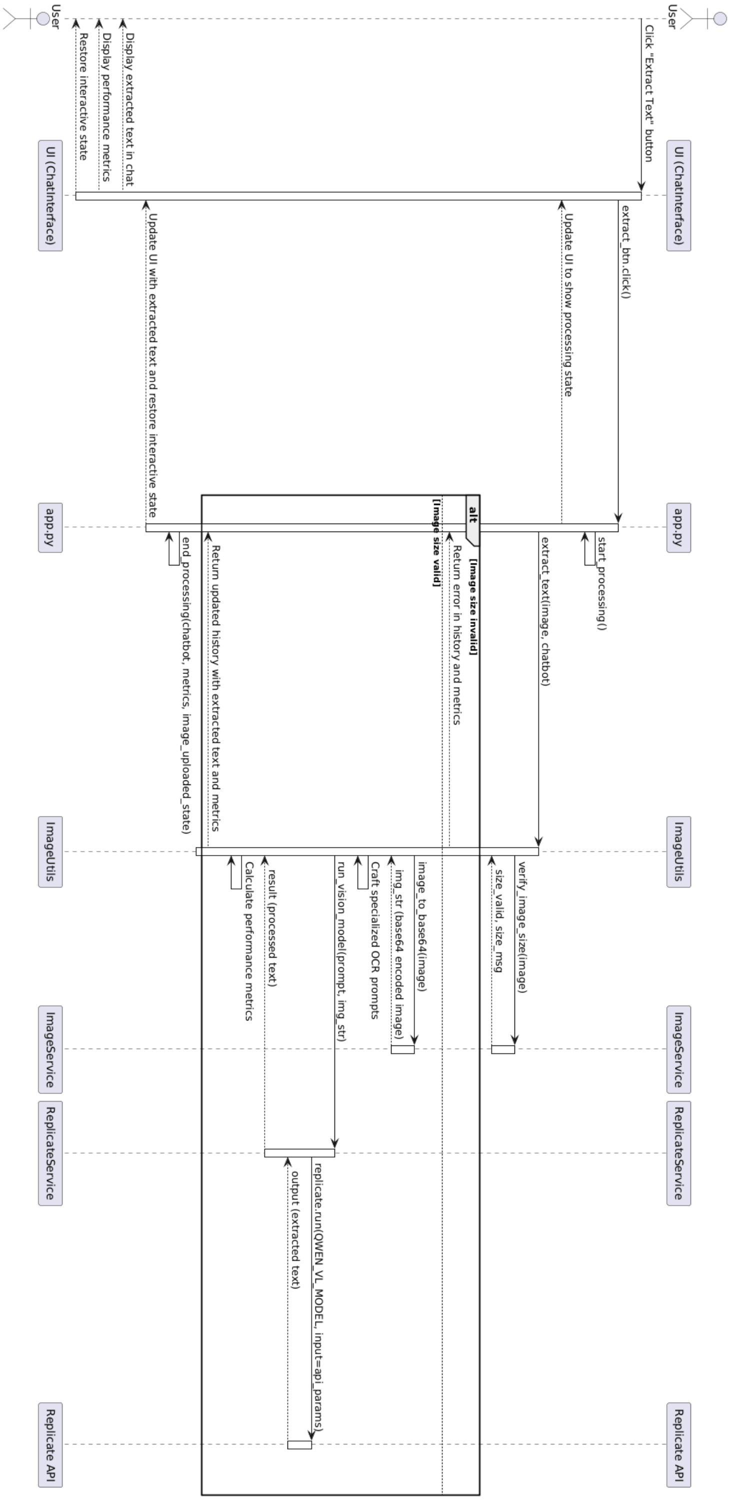


Figure 31 Extract Text Sequence Diagram

Figure 32 sequence diagram below shows the interactions when a user requests to generate a caption for an uploaded image.

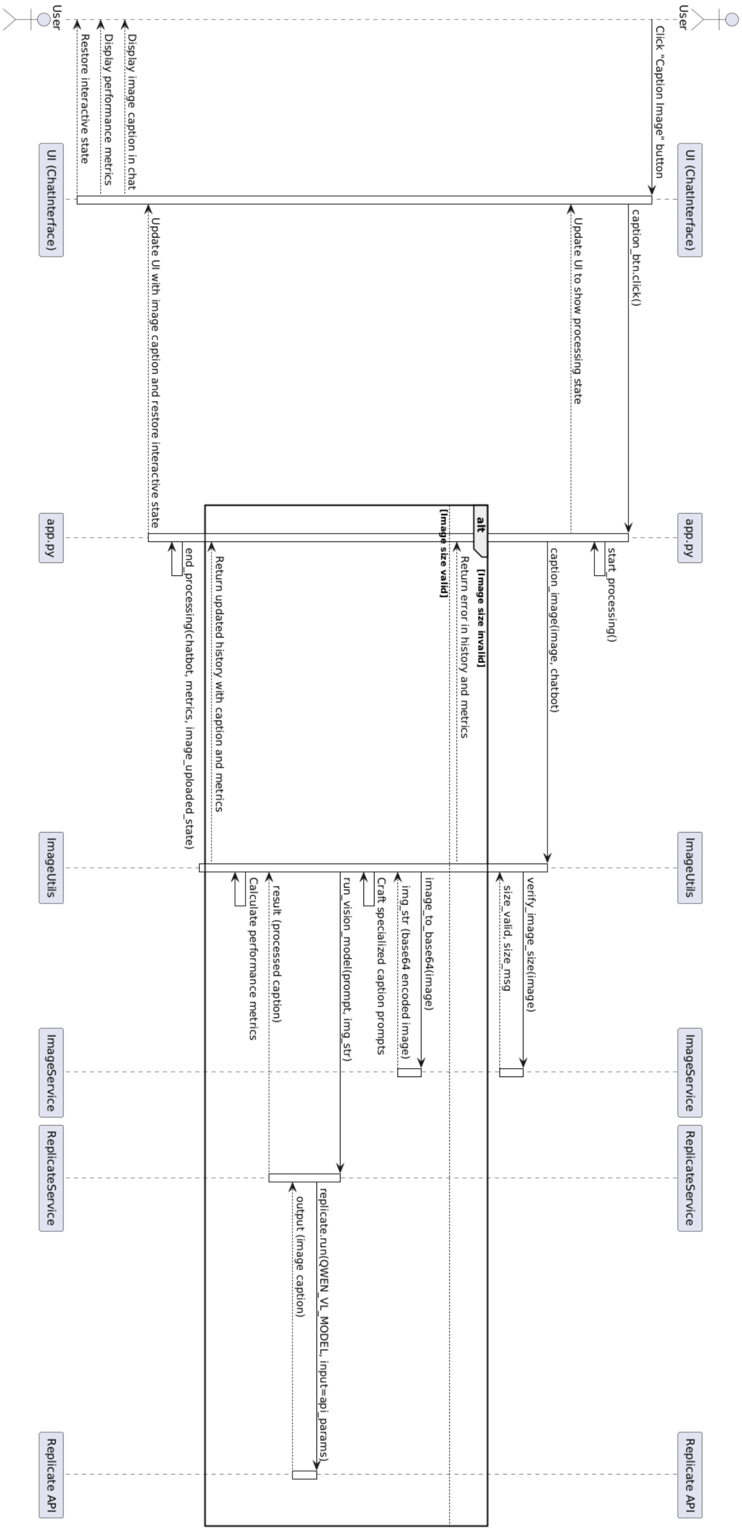


Figure 32 Caption Image Sequence Diagram

Figure 33 sequence diagram below shows the interactions when a user requests to generate a detailed summary of an uploaded image.

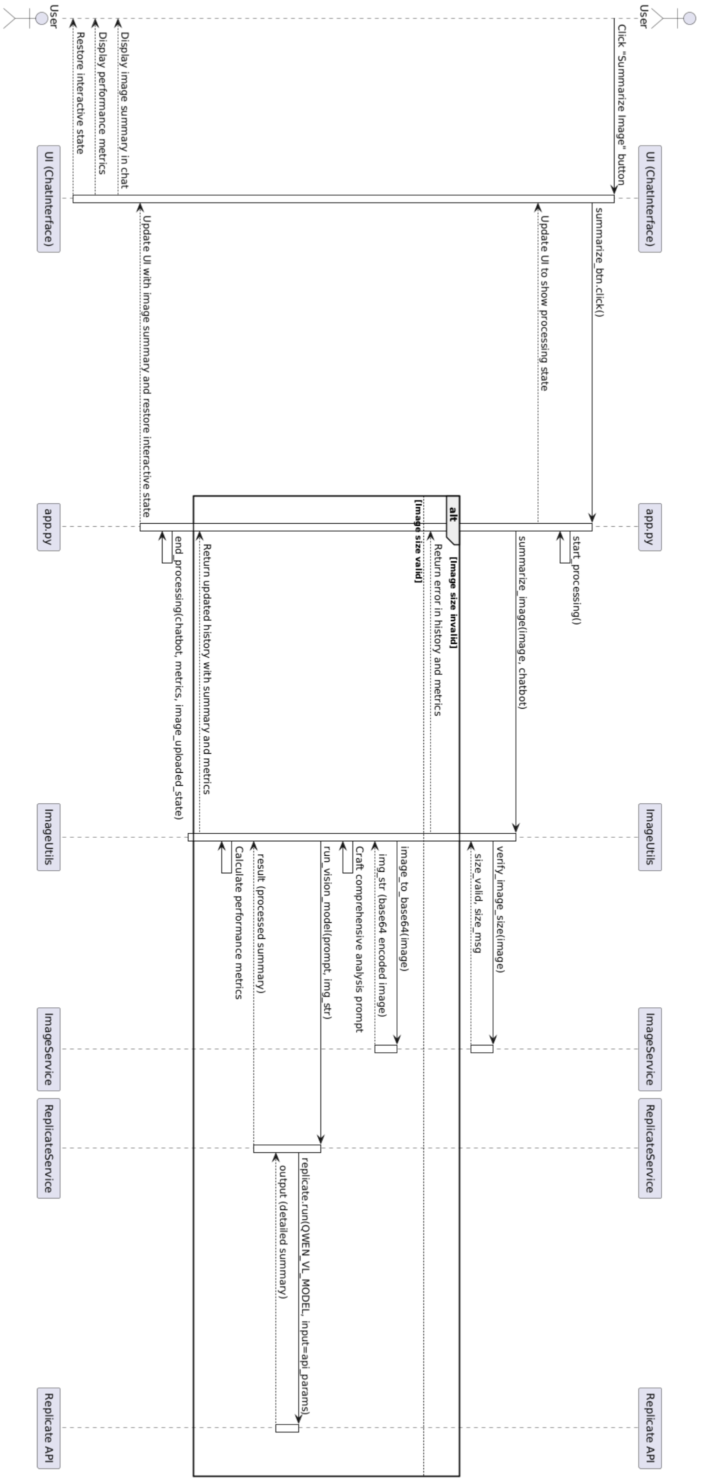


Figure 33 Summarize Image Sequence Diagram

Figure 34 sequence diagram below shows the interactions when a user converts the last bot response to speech.

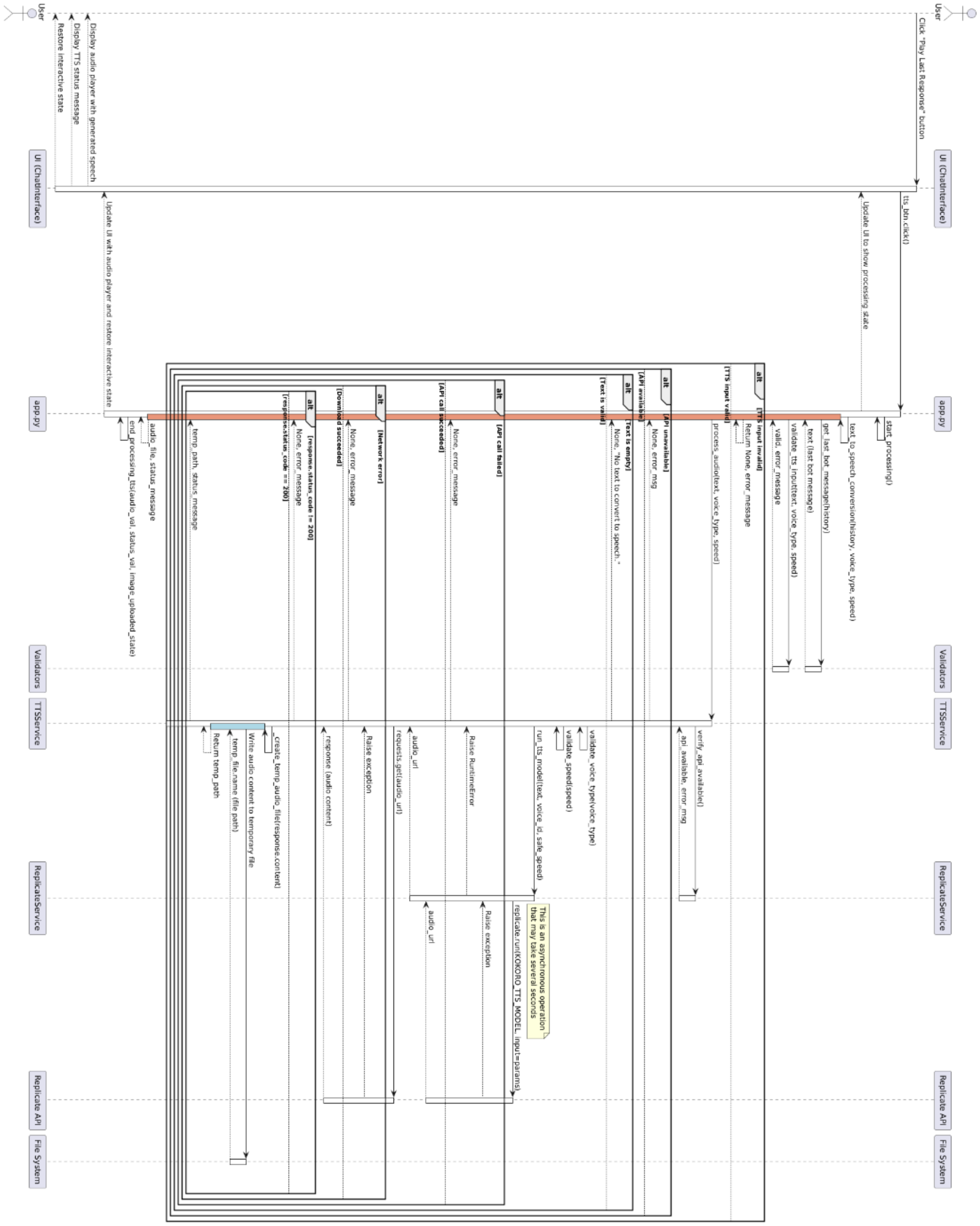


Figure 34 Convert Text to Speech Sequence Diagram

Figure 35 sequence diagram below shows the interactions when a user selects a voice type for text-to-speech conversion.

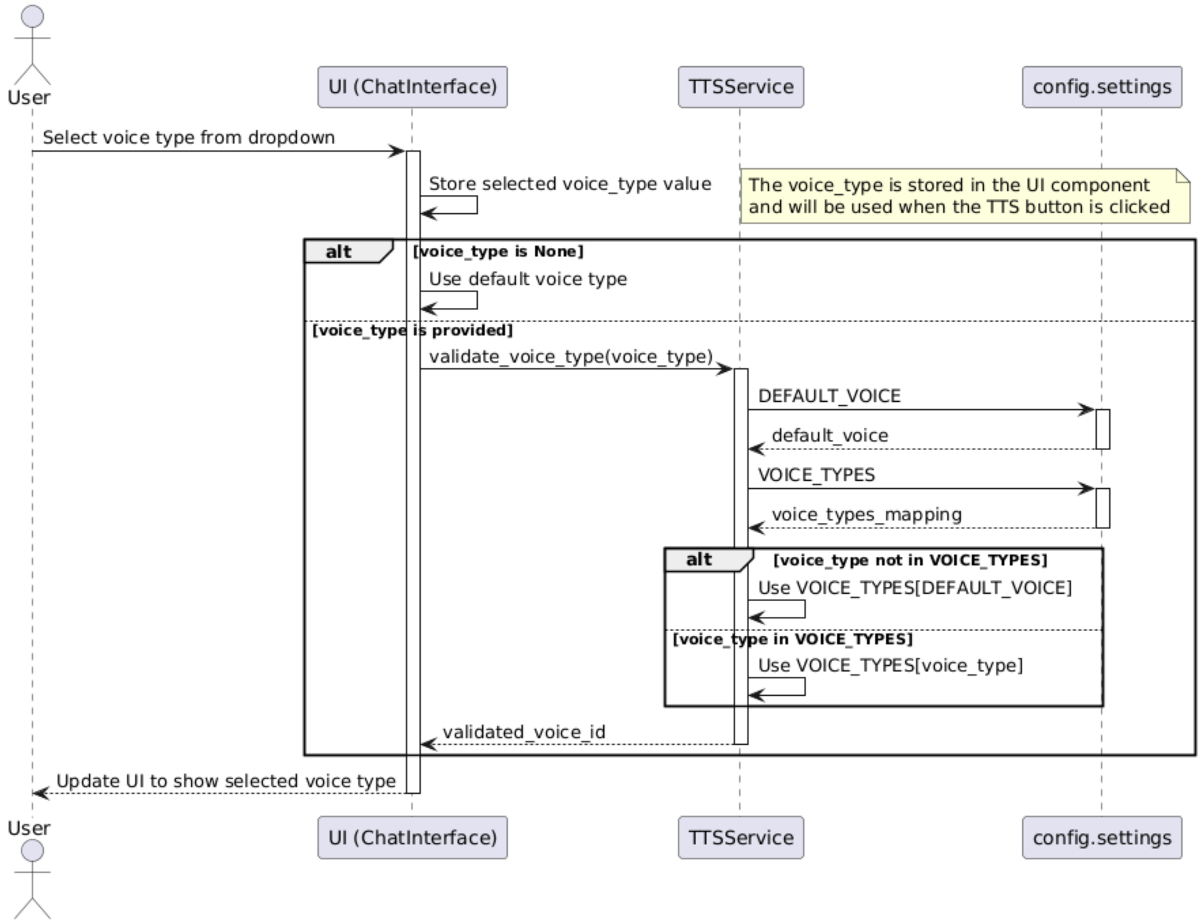


Figure 35 Select Voice Type Sequence Diagram

Figure 36 sequence diagram shows the interactions when a user adjusts the speech speed for text-to-speech conversion.

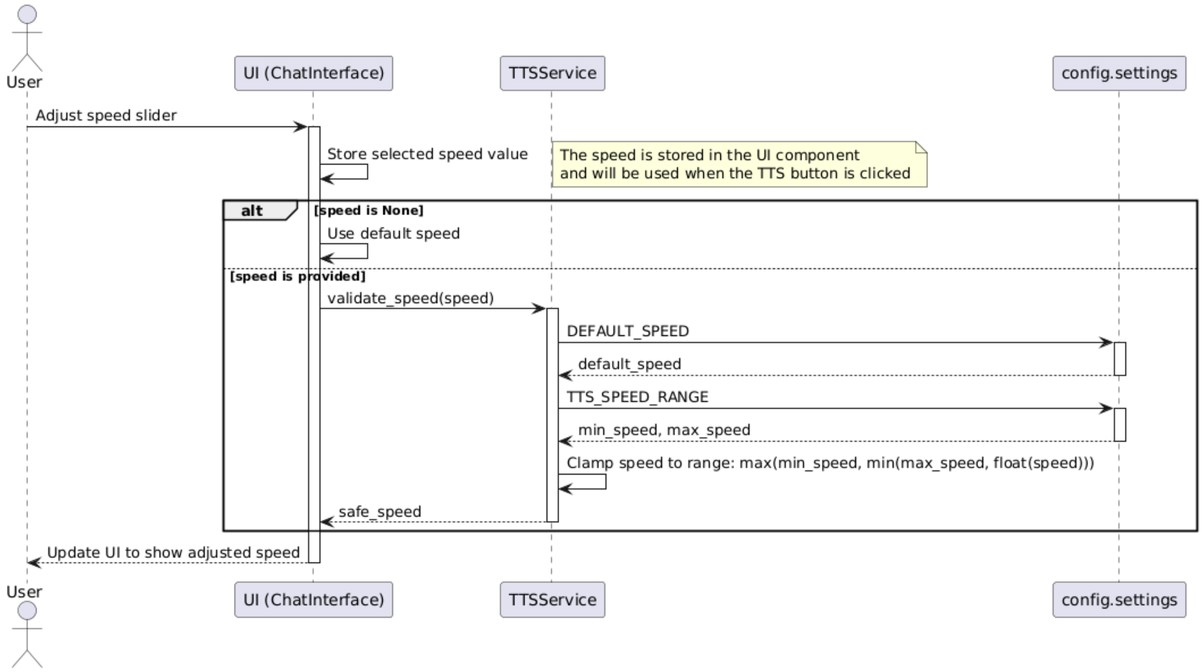


Figure 36 Adjust Speech Speed Sequence Diagram

Figure 37 sequence diagram shows the internal interactions when processing an image with the vision model.

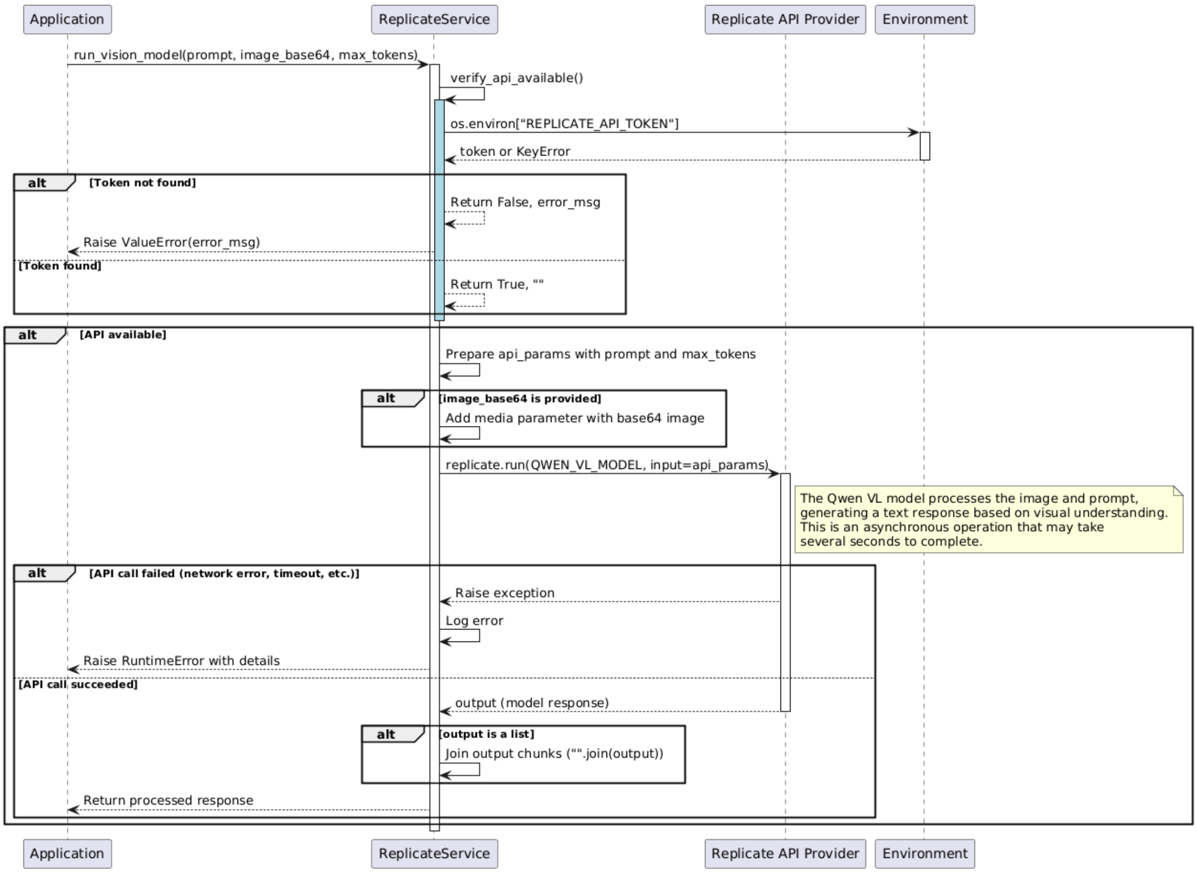
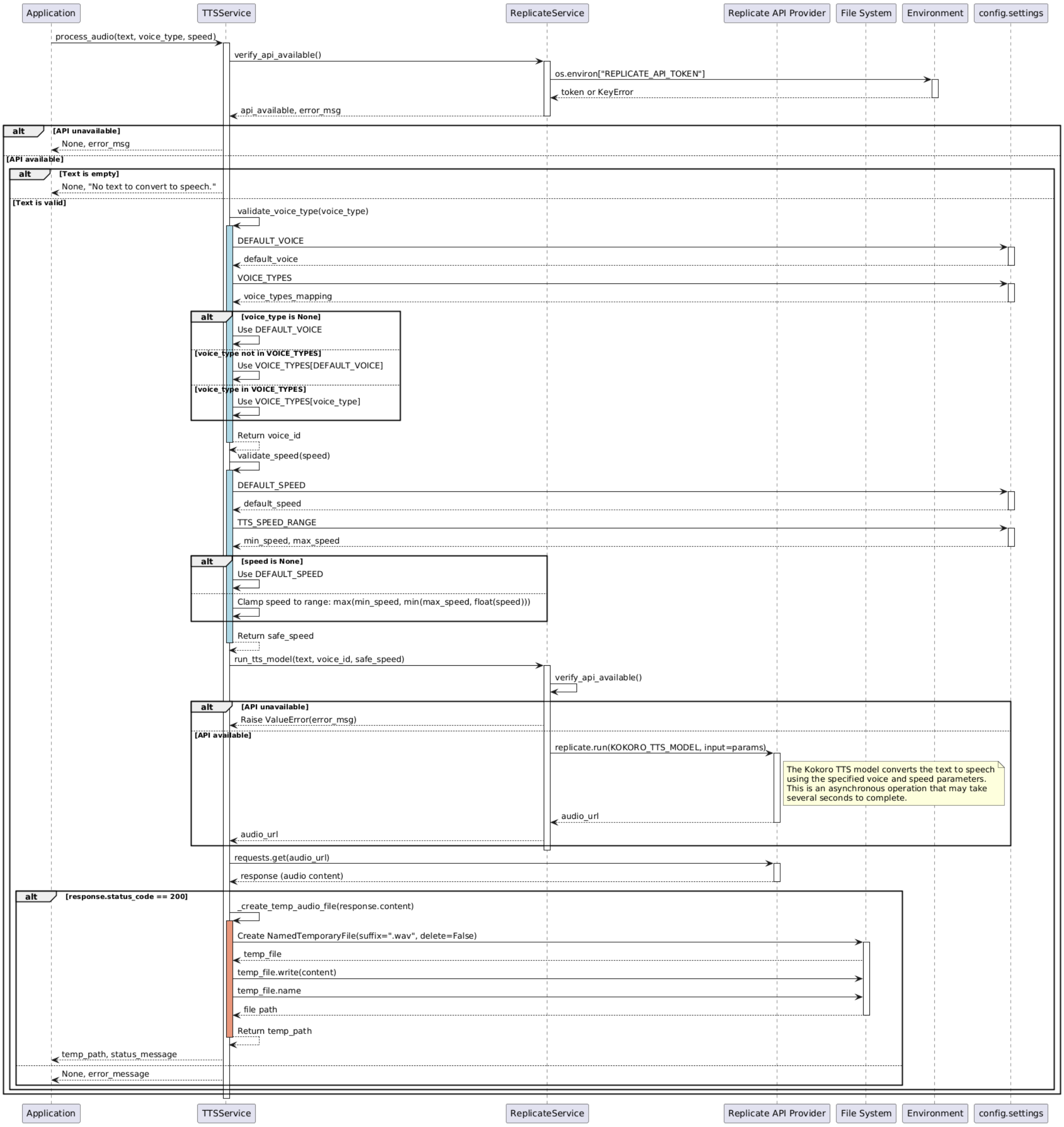


Figure 37 Process Image with Vision Model Sequence Diagram

Figure 38 sequence diagram shows the internal interactions when generating audio with the TTS model.

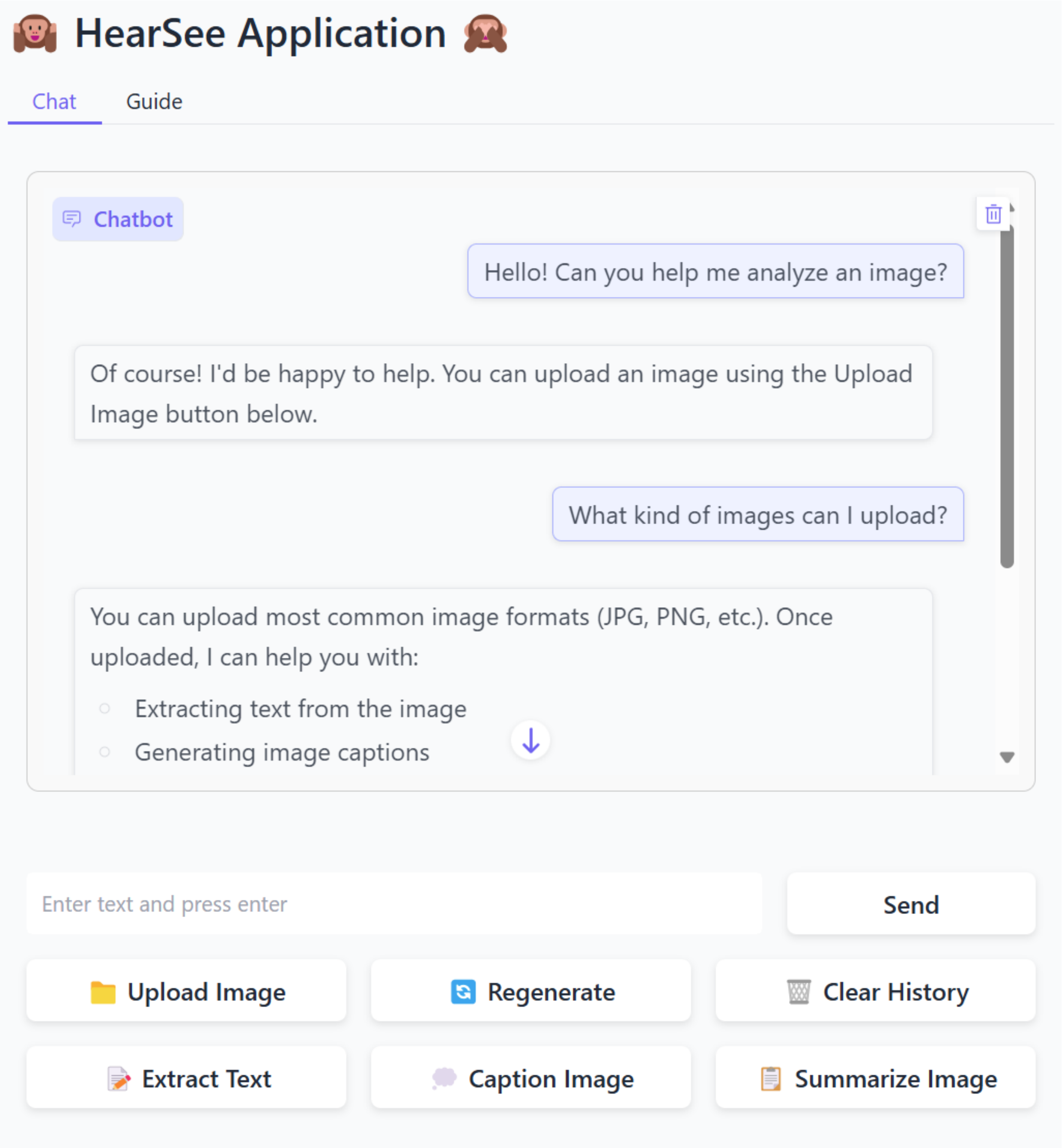


The Kokoro TTS model converts the text to speech using the specified voice and speed parameters. This is an asynchronous operation that may take several seconds to complete.

Figure 38 Generate Audio with TTS Model Sequence Diagram

3.5 User Interface Design

Figure 39, 40, and 41 illustrates HearSee’s main user interface design. The illustrated design is based the system requirements and architecture and which may be revised throughout the development of the system.



🙉 HearSee Application 🙈

Hello! Can you help me analyze an image?

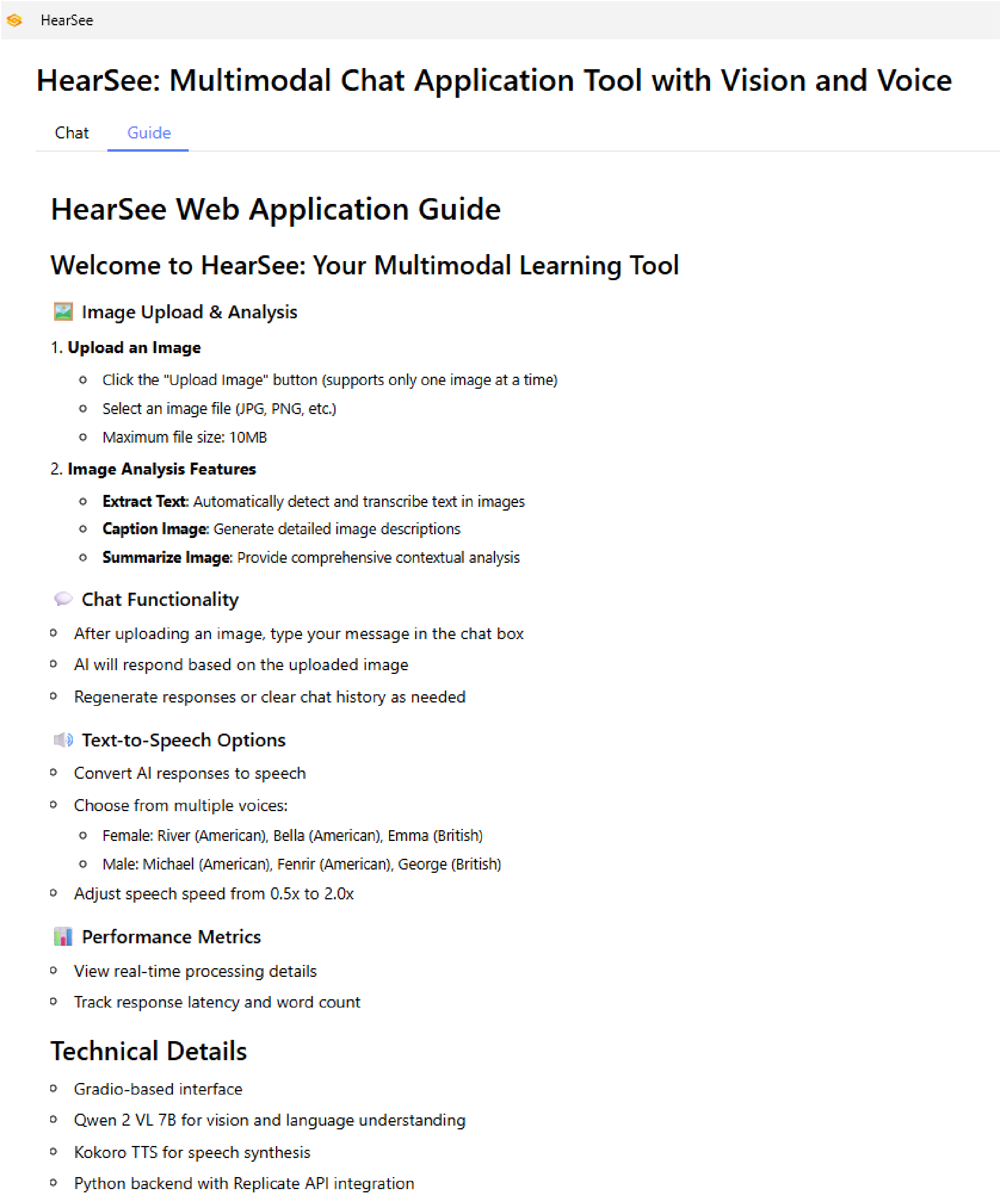
Of course! I'd be happy to help. You can upload an image using the Upload Image button below.

What kind of images can I upload?

You can upload most common image formats (JPG, PNG, etc.). Once uploaded, I can help you with:

- Extracting text from the image
- Generating image captions

Figure 39 HearSee’s user chat interface.



HearSee: Multimodal Chat Application Tool with Vision and Voice

HearSee Web Application Guide

Welcome to HearSee: Your Multimodal Learning Tool

Image Upload & Analysis

1. **Upload an Image**
    - Click the "Upload Image" button (supports only one image at a time)
    - Select an image file (JPG, PNG, etc.)
    - Maximum file size: 10MB
2. **Image Analysis Features**
    - **Extract Text**: Automatically detect and transcribe text in images
    - **Caption Image**: Generate detailed image descriptions
    - **Summarize Image**: Provide comprehensive contextual analysis

Chat Functionality

- After uploading an image, type your message in the chat box
- AI will respond based on the uploaded image
- Regenerate responses or clear chat history as needed

Text-to-Speech Options

- Convert AI responses to speech
- Choose from multiple voices:
    - Female: River (American), Bella (American), Emma (British)
    - Male: Michael (American), Fenrir (American), George (British)
- Adjust speech speed from 0.5x to 2.0x

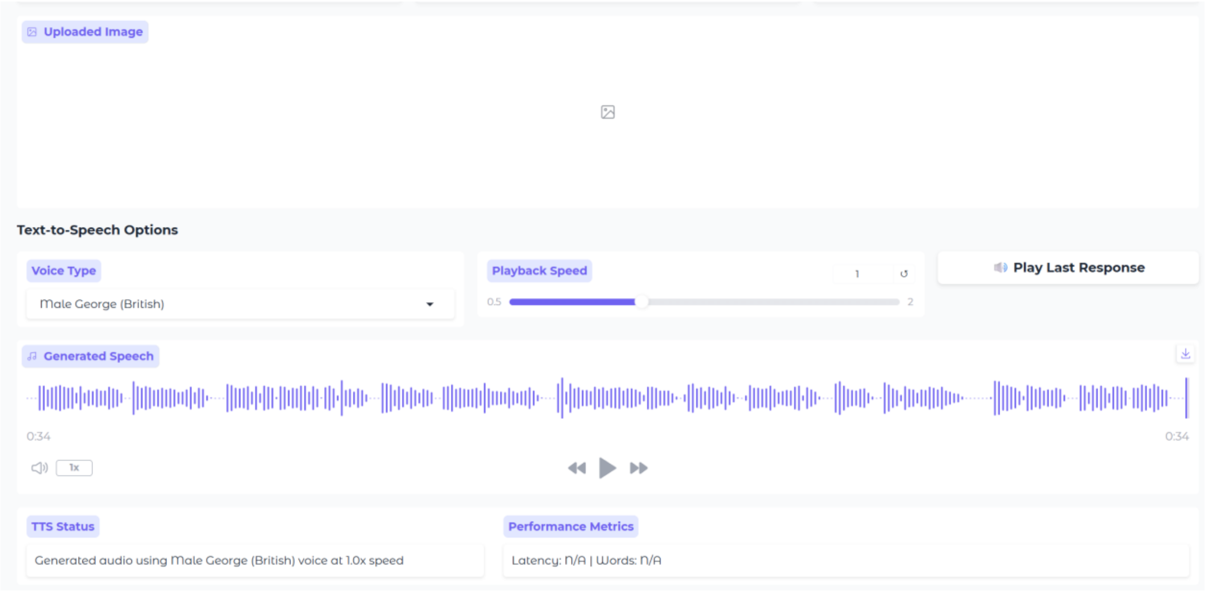
Performance Metrics

- View real-time processing details
- Track response latency and word count

Technical Details

- Gradio-based interface
- Qwen 2 VL 7B for vision and language understanding
- Kokoro TTS for speech synthesis
- Python backend with Replicate API integration

Figure 40 HearSee's user guide interface.



Text-to-Speech Options

Voice Type: Male George (British)

Playback Speed: 1 (0.5 – 2)

TTS Status: Generated audio using Male George (British) voice at 1.0x speed

Performance Metrics: Latency: N/A | Words: N/A

Figure 41 HearSee's Text-to-Speech user interface

3.6 Summary of Methodology

This chapter outlines the methodology used to develop *HearSee*, a web-based multimodal learning tool that processes image-to-text and text-to-speech tasks using advanced AI technologies. The development followed the SCRUM agile framework, adapted for a solo developer, and organized into two-week sprints to support iterative progress and continuous improvement. The system was built on a three-tier architecture—presentation, application, and data layers—using technologies like Gradio UI, Python services, and external APIs from Replicate. A detailed C4 model was used to structure the system's architecture and interactions. The system analysis and design were represented through use case diagrams, activity diagrams, class diagrams, data flow diagrams, and sequence diagrams, which illustrated processes such as image upload, text extraction, captioning, summarization, and speech generation. This project employed Qwen2-VL 7B for vision-language tasks and StyleTTS 2 (via Kokoro TTS) for speech synthesis. The user interface was designed with accessibility and functionality in mind, integrating voice controls, speech speed adjustments, and conversational chat features to support various learning preferences.

Chapter 4: Implementation

4.1 Introduction

This chapter details the implementation of HearSee, a web-based multimodal learning tool designed to enhance undergraduate students' learning experiences through AI-powered image analysis and text-to-speech capabilities. The system was developed to process educational materials through image-to-text conversion and provide audio output, making content more accessible across different learning preferences through multimodal learning.

The technical implementation of HearSee includes the technology stack, system architecture, core modules, external API integrations, and deployment process. The implementation utilizes Gradio for the user interface, Python for backend services, and integrated Qwen2-VL and Kokoro TTS models via Replicate's API for vision-language and speech synthesis tasks respectively. The three-tier architecture separated presentation, application, and data concerns as shown in the previous chapter.

The following subsections detail the specific approaches, tools, and techniques that were employed as well as actual implementations of the project’s methodologies to ensure the system design and requirements were met throughout the development process.

4.2 Development Environment and Hardware Specifications

The following subsections describes the hardware and software used for project development, deployment and installation. The demo of the project is hosted on Hugging Face Spaces for easy access on browsers through all modern devices.

4.2.1 Hardware Specifications

HearSee was implemented and tested on a Lenovo Legion 5i laptop using Microsoft Edge browser for testing. The workstation machine utilises an Intel Core i7-14650HX processor with 16 cores, 24 logical threads, 2.2GHz base clock, and 5.2GHz boost capability. The system was equipped with 32GB of DDR5-5600 RAM and a 1TB NVMe Gen 4 SSD for storage. Graphics processing was handled by an NVIDIA GeForce RTX 4060 with 8GB VRAM. The development environment ran on Windows 11 Pro (64-bit) 24H2 Release Preview operating system.

Hardware requirements for deployment and testing included webcam or camera for image capture, audio output devices for text-to-speech playback functionality, and a system specification of 8GB RAM with a minimum of 4-core CPU to ensure operation. A network connection was essential for API communication with Replicate services, as the system relies on hosted Generative AI models for image analysis and speech synthesis.

4.2.2 Software Dependencies

The implementation relies on a software stack centered around Python 3.10 as the programming language. The Gradio 4.0+ framework was selected for creating the web interface due to its simplicity and support for machine learning applications. Libraries were utilized throughout the implementation. The replicate 0.18.0 library provided the API client for interacting with AI models. Image processing capabilities were implemented using Pillow 10.0.0, while python-dotenv 1.0.0 handled environment variable management for configuration. HTTP communication was managed through the requests 2.31.0 library, and operations were performed using numpy 1.24.3. The Python logging library was used for application logging.

Development tools included Visual Studio Code with Python extensions for code editing and debugging, Git for version control and development, and pytest for implementing the testing framework. These tools provided a development environment that supported implementation and testing of the HearSee system.

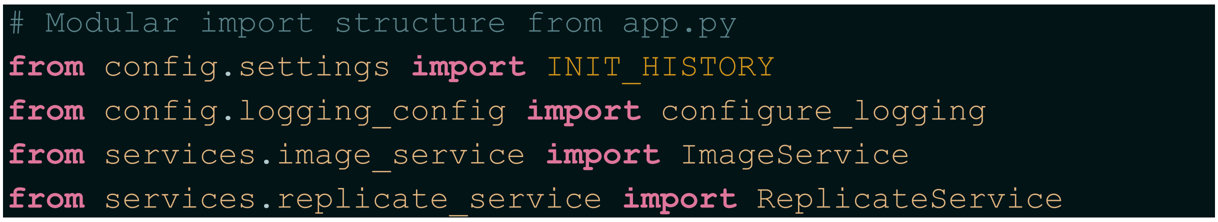
4.3 System Architecture Implementation

The following subsections describes the system architecture implementation of HearSee. This focuses on the overall architecture and the component integration using the system architecture and class diagrams as guidance.

4.3.1 Overall Architecture

HearSee system follows a three-tier architecture pattern with separation between the front-end interface, back-end processing logic, and data management components. The implementation is structured into five modules that work together to provide the functionality of the system. The Main Application (app.py) serves as the entry point that initializes the Gradio interface and connects all components. The Services module contains functionality for image processing, API integration, and text-to-speech conversion. UI Components module provides interface elements for chat, documentation, and UI elements. The Utilities module contains functions for image operations, validation, and logging. The Configuration module manages settings, constants, and logging configuration.

This approach allows for development and testing of components, making the system maintainable and extensible. The architecture follows the principle of separation of concerns, with each module having a responsibility. The import structure from app.py shows this organization:



```python
# Modular import structure from app.py
from config.settings import INIT_HISTORY
from config.logging_config import configure_logging
from services.image_service import ImageService
from services.replicate_service import ReplicateService
```

```
from services.tts_service import TTSService
from utils.validators import get_last_bot_message,
validate_image_input
from utils.image_utils import ImageUtils
from ui import ChatInterface, GuideInterface, UIStateManager
```

4.3.2 Component Integration

The components in HearSee are integrated through a combination of dependency injection and event-driven programming. The main application (app.py) serves as the orchestrator that initializes all components and connects them through event handlers. The integration follows principles that ensure the system functions. Service Abstraction ensures that functionality is encapsulated in service classes with interfaces. The Event-Driven UI approach means user interactions trigger events that are handled by functions. State Management is implemented through Gradio's state components to maintain application state across interactions. Error Handling at service boundaries ensures the system remains when encountering conditions.

The chat message processing flow exemplifies how components are integrated in the system. When a user submits a message with an image, the UI triggers the `process_chat_message` function. This function validates inputs using utility functions, then calls the ImageService to process the image. The image is sent to the ReplicateService for AI analysis, after which the response is formatted and returned to the UI. The TTSService converts the response to speech if requested by the user. This integration approach ensures coupling between components while maintaining a flow of data and control throughout the system.

4.4 Application Setup and Deployment

The following subsections describes the setup for the HearSee web application in local environment. It involves steps for installing dependencies and environment variables before initializing the Gradio application.

4.4.1 Local Development Setup

The application utilizes Gradio for creating a web-based user interface. The setup process involves steps to prepare the development environment. The Environment Configuration begins with setting up the directory, creating and activating a virtual environment, installing dependencies from the requirements.txt file, and setting up environment variables. The .env file must be configured with the Replicate API key to enable communication with the AI models. The following commands illustrate this setup process:

```
# Go to the project directory
cd hearsee

# Create and activate virtual environment
python -m venv venv
source venv/Scripts/activate  # On Linux: venv\bin\activate

# Install dependencies
pip install -r requirements.txt

# Set up environment variables
cp .env.example .env
# Edit .env to add your Replicate API key
```

Application Initialization is handled through the `create_app` function in app.py, which configures the Gradio interface with theming and sets up all event handlers. This function creates a predetermined appearance while keeping compatibility with Gradio versions through fallback mechanisms. The implementation includes version detection to handle differences in the theming API between Gradio versions:

```
def create_app():
    """
    Create and configure the HearSee application.

    This function initializes the Gradio interface with custom theming,
    sets up the chat and guide tabs, and connects all event handlers
    for the interactive elements.

    Returns:
        gr.Blocks: Configured Gradio application ready to be launched
    """
    # Create custom theming with compatibility for different Gradio versions
    try:
        # Try to create theme with the new method (for newer Gradio versions)
        system_theme = gr.themes.Soft(
            primary_hue="blue",
            secondary_hue="blue",
            neutral_hue="gray",
            text_size=gr.themes.sizes.text_md,
            font="system-ui",
        ).set(
            # Force light mode by setting these properties
            body_background_fill="white",
            background_fill_primary="white",
            background_fill_secondary="#f7f7f7",
            block_title_text_color="black",
            block_label_text_color="black",
            input_background_fill="#ffffff",
            body_text_color="black",
        )
    except (TypeError, AttributeError):
        # Fallback for older Gradio versions
        system_theme = gr.themes.Soft(
            primary_hue="blue",
```

```
            secondary_hue="blue",
            neutral_hue="gray",
            text_size=gr.themes.sizes.text_md,
            font="system-ui",
        )

    # Create the main application with theme
    with gr.Blocks(theme=system_theme, title="HearSee") as hearsee:
        # Application components and event handlers are defined here
        # ...

    return hearsee
```

4.4.2 Gradio UI Implementation

The Gradio framework was selected for its simplicity in creating machine learning interfaces and its sharing capabilities. The implementation includes features that enhance the user experience and development workflow. The local application runs on http://localhost:7860 by default and provides hot-reload functionality for development along with local file system integration for data processing. The public sharing configuration offers public sharing link generation using the share=True parameter, tunnel creation through Gradio's infrastructure, and URL generation for demonstration and collaboration purposes.

The application launch configuration is implemented in the main script as shown in the following code snippet:

```
# Example code snippet for Gradio setup from app.py
if __name__ == "__main__":
    # Create the application
    app = create_app()

    # Launch with optional public sharing
    app.launch(
        share=False,  # Set to True for public sharing
        server_name="0.0.0.0",
```

```
        server_port=7860
    )
```

The selection of Gradio as the UI framework was justified by the following factors: Its prototyping capabilities allowed for iteration on the UI design during development. The support for multiple input and output types (text, images, audio) aligned with the nature of HearSee. Configuration was required for web deployment, which simplifies development process. The sharing functionality eliminated the need for a remote hosting setup, simplifying demonstration and testing. The event-driven programming model simplified the implementation of user interactions.

4.5 Critical Function Implementation

The following subsections describes the implementation of the critical functions within the HearSee application to achieve its objectives.

4.5.1 Core Processing Functions

The core processing functions involves image analysis with processing and text-to-speech conversion, the two critical functionalities needed for the project to function as specified by the project's objectives.

4.5.1.1 Image Processing and Analysis

The image processing and analysis functionality is central to the HearSee system, enabling it to process user-uploaded images and prepare them for AI analysis. The implementation of this function is encapsulated in the process_chat_message function, which handles the functionality of processing user messages with the uploaded image. This function validates inputs, calls the Qwen 2 VL 7B vision model, and formats the response for display to the user.

```
def process_chat_message(message, history, metrics, image=None):
    """Process a chat message and return updated history and metrics.

    This function handles the core functionality of processing user
messages
    with the uploaded image. It validates inputs, calls the vision model,
    and formats the response.
```

```
    Args:
        message (str): The user's text message
        history (list): The conversation history as a list of [user, bot]
message pairs
        metrics (str): Current performance metrics string
        image (numpy.ndarray, optional): The uploaded image data. Defaults
to None.

    Returns:
        tuple: (updated_history, updated_metrics)
            - updated_history (list): Conversation history with new message
pair
            - updated_metrics (str): Updated performance metrics string
    """
    start_time = time.time()
    logger.info(f"Processing chat message: {message[:50]}{'...' if
len(message) > 50 else ''}")

    # Check image size
    size_valid, size_msg = ImageService.verify_image_size(image)
    if not size_valid:
        logger.warning(f"Image size validation failed: {size_msg}")
        return history + [[message, size_msg]], "Error: Image too large"

    # Check API availability
    api_available, error_msg = ReplicateService.verify_api_available()
    if not api_available:
        logger.error(f"API unavailable: {error_msg}")
        return history + [[message, error_msg]], "Error: API unavailable"

    # Validate image requirement
    valid_img, img_error = validate_image_input(image)
    if not valid_img:
        logger.warning(f"Image validation failed: {img_error}")
        return history + [[message, img_error]], "Error: Image required"

    # Process the message
    try:
        # Convert image to base64 for API transmission
        img_str = ImageService.image_to_base64(image)
```

```
        # Build system prompt that defines the AI assistant's role and capabilities
        system_prompt = "You are a helpful AI assistant specializing in analyzing images and providing detailed information."

        # Convert conversation history to a formatted context string
        # This preserves the conversation flow for the model
        context = ""
        for h in history:
            if h[0] is not None:  # Skip entries with no user message
                context += f"User: {h[0]}\nAssistant: {h[1]}\n\n"

        logger.debug("Running vision model")
        # Run the vision model with the complete prompt context and image
        result = ReplicateService.run_vision_model(
            f"{system_prompt}\n\nConversation History:\n{context}\nUser: {message}\nAssistant:",
            image_base64=img_str
        )

        # Calculate performance metrics for user feedback
        end_time = time.time()
        latency = end_time - start_time  # Total processing time in seconds
        word_count = len(result.split())  # Approximate word count of response
        updated_metrics = f"Latency: {latency:.2f}s | Words: {word_count}"

        logger.info(f"Chat message processed successfully in {latency:.2f}s")
        # Return updated history and metrics
        return history + [[message, result]], updated_metrics
    except Exception as e:
        logger.error(f"Error processing chat message: {str(e)}", exc_info=True)
        error_msg = f"Sorry, I encountered an error: {str(e)}"
        return history + [[message, error_msg]], "Error: System unavailable. Please try again."
```

This approach was chosen for the following reasons: The validation ensures that inputs meet requirements before processing, preventing errors and improving system reliability.

Context preservation through including conversation history provides the AI model with context for responses, enhancing the user experience. Performance metrics tracking for latency and word count provides transparency to users about system performance, setting expectations. The error handling with user feedback allows continued usage even when issues occur, preventing frustration and confusion.

Several implementation decisions shaped this function. Using base64 encoding for image transmission to the API ensures compatibility and reduces issues with file handling across environments. Formatting the prompt with system instructions, conversation history, and user message provides guidance to the AI model, resulting in responses. Logging at low levels facilitates debugging and monitoring after deployment, making the system easier to maintain and troubleshoot.

4.5.1.2 Text-to-Speech Conversion

The text-to-speech conversion functionality enables the system to convert AI-generated text responses to speech using the Kokoro TTS model, enhancing accessibility and user experience. The implementation is encapsulated in the text_to_speech_conversion function, which extracts the last bot message from the conversation history and converts it to speech using the voice type and speed parameters.

```
def text_to_speech_conversion(history, voice_type, speed):
    """Convert last bot message to speech.

    This function extracts the last bot message from the conversation
history
    and converts it to speech using the specified voice type and speed.

    Args:
```

```
        history (list): The conversation history as a list of [user, bot]
message pairs
        voice_type (str): The type of voice to use for TTS
        speed (float): The speed factor for speech playback

    Returns:
        tuple: (audio_file, status_message)
            - audio_file (str or None): Path to the generated audio file or
None if failed
            - status_message (str): Status message indicating success or
failure
    """
    text = get_last_bot_message(history)
    logger.info(f"Converting text to speech with voice: {voice_type},
speed: {speed}")
    result = TTSService.process_audio(text, voice_type, speed)
    if result[0] is None:
        logger.warning(f"TTS conversion failed: {result[1]}")
    else:
        logger.info("TTS conversion successful")
    return result
```

The TTS implementation follows a modular design which was achieved by delegating the TTS processing to the TTSService, maintaining separation of concerns and making the code maintainable. The interface is designed for simplicity, requiring only the current conversation history and TTS parameters, which makes it easy to use and understand. Status messages inform the user about the success or failure of the operation, enhancing the user experience. Logging helps with debugging and monitoring, making the system easier to maintain.

Implementation decisions for the TTS functionality include using temporary files for audio storage, which avoids the need for permanent storage while allowing playback. The system supports voice types and speed adjustments, enhancing user experience and accessibility for needs and preferences. Delegating to the TTSService allows for changes in the TTS backend without affecting the UI, making the system adaptable to requirements.

4.5.2 Error Handling and Validation

The implementation incorporates error handling to ensure system reliability. The TTSService's process_audio function demonstrates this approach:

```
# Example of robust error handling in the TTSService
def process_audio(text, voice_type=None, speed=None):
    """Process text to speech conversion."""
    # Check API availability
    api_available, error_msg = ReplicateService.verify_api_available()
    if not api_available:
        return None, error_msg

    # Validate inputs
    if not text or text.strip() == "":
        return None, "No text to convert to speech."

    try:
        # Get validated parameters
        voice_id = TTSService.validate_voice_type(voice_type)
        safe_speed = TTSService.validate_speed(speed)

        # Get audio URL from Replicate
        audio_url = ReplicateService.run_tts_model(text, voice_id,
safe_speed)

        # Download and save audio
        response = requests.get(audio_url)
        if response.status_code == 200:
            # Create temporary file with the audio content
            temp_path =
TTSService._create_temp_audio_file(response.content)
            return temp_path, f"Generated audio using {voice_type or
DEFAULT_VOICE} voice at {safe_speed}x speed"
        else:
```

```
            return None, f"Error downloading audio: HTTP status
{response.status_code}"

    except Exception as e:
        # Include "Error downloading audio" in the message if it's a
connection error
        if "ConnectionError" in str(type(e)) or "Network error" in str(e):
            return None, f"Error downloading audio: {str(e)}"
        return None, f"Error generating speech: {str(e)}"
```

The error handling strategy includes several components. Input validation ensures all user inputs are validated before processing, preventing errors from data. Parameter sanitization ensures parameters are sanitized to meet requirements, maintaining system integrity. Exception handling through try-except blocks catches and handles exceptions, preventing crashes. User feedback is provided through error messages formatted for user acknowledgement. Logging of errors with context supports debugging and system improvement. This approach ensures that the system degrades when errors occur, maintaining functionality even in the face of bad conditions or inputs.

4.6 Front-end Implementation

The following subsections describes the front-end implementation of HearSee using Gradio UI as the front-end development framework library.

4.6.1 User Interface Design

The front-end implementation focuses on creating a simplified user experience that makes the system accessible to users with varying levels of expertise. The interface components include a chat display for conversation history, text input field for user messages, image upload button and gallery for content, action buttons for functions (extract text, caption, summarize), text-to-speech controls with voice selection and speed adjustment, and status indicators for system performance and operation feedback.

The UI was implemented using Gradio components, organized into sections with visual hierarchy. The ChatInterface class encapsulates the UI creation logic, as shown in the following implementation:

```
@staticmethod
def create_interface():
    """
    Create the complete chat interface with all components and event
handlers.

    This method constructs the entire UI layout using Gradio components,
    organizing them into a logical structure with appropriate styling and
    configuration.
    """
    with gr.Column():
        # Main chat display and initial instruction message
        chatbot = create_chatbot_component()  # Main chat history display
```

```
        image_instruction = create_image_instruction()  # Warning to upload
image first

        # User input section with text field and send button
        with gr.Row():
            msg = gr.Textbox(
                label="Text Input",
                placeholder="Enter text and press enter key/send button",
                scale=9,
                container=True,
                show_label=False,
            )
            send_btn = gr.Button("Send", interactive=False)  # Initially
disabled until image upload

        # Primary action buttons for image handling and chat management
        with gr.Row():
            upload_btn = gr.UploadButton("📁 Upload Image",
file_types=["image"], file_count="single")
            regenerate_btn = gr.Button("🔄 Regenerate")
            clear_btn = gr.Button("🗑️ Clear History")

        # Specialized image processing action buttons
        with gr.Row():
            extract_btn = gr.Button("📝 Extract Text",
interactive=False)  # OCR functionality
            caption_btn = gr.Button("💭 Caption Image",
interactive=False)  # Image description
            summarize_btn = gr.Button("📋 Summarize Image",
interactive=False)  # Detailed analysis

        # Image display gallery
        with gr.Row():
            gallery = gr.Gallery(
                label="Uploaded Image",
                show_label=True,
                columns=2,  # Support for before/after comparisons
                rows=1,
                object_fit="scale-down",  # Maintains aspect ratio
            )

        # Text-to-Speech section header
```

```
        gr.Markdown("### Text-to-Speech Options")

        # Audio playback component for TTS output
        with gr.Row():
            audio_output = gr.Audio(label="Generated Speech",
interactive=False)

        # TTS configuration controls
        with gr.Row():
            voice_type = create_voice_type_dropdown()  # Voice selection
            speed = create_speed_slider()  # Playback speed adjustment
            tts_btn = gr.Button("🔊 Play Last Response")  # Trigger TTS
generation

        # System status indicators
        with gr.Row():
            tts_status = gr.Textbox(
                label="TTS Status",
                interactive=False,
                value="Idle",  # Default state
                scale=1  # Add scale parameter for responsive sizing
            )
            performance_metrics = create_mllm_status()  # Multimodal LLM
performance tracking
```

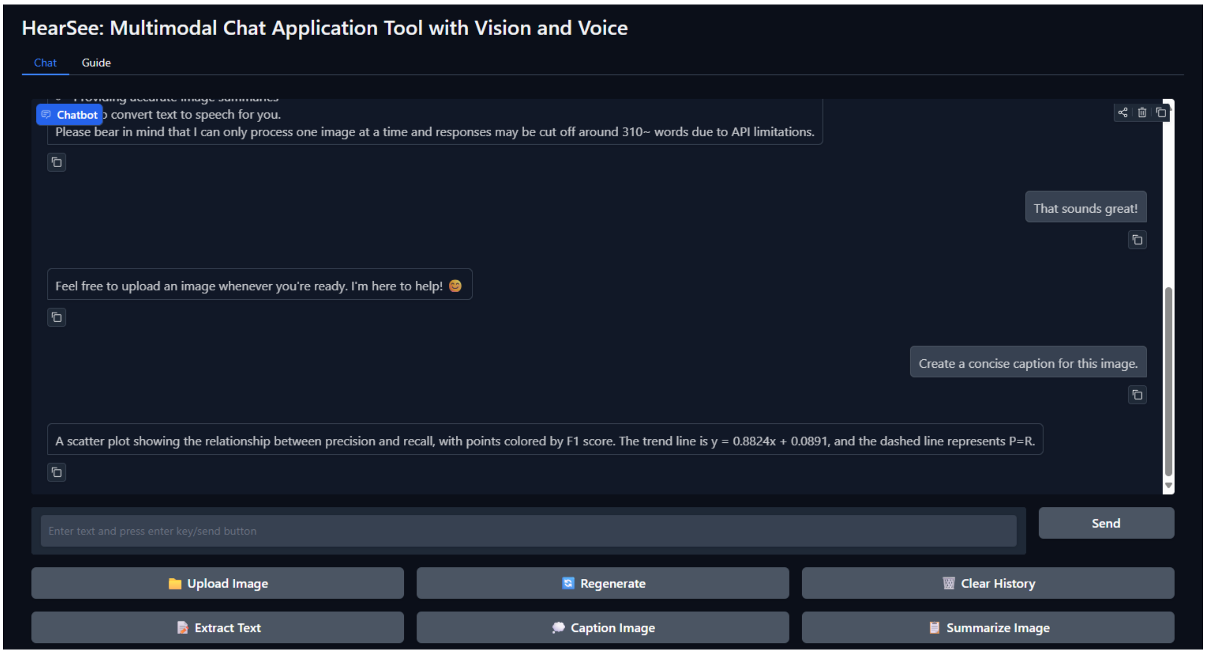
4.6.2 User Experience Considerations

The user experience design was designed to ensure the application is accessible and easy to use. Navigation is achieved through labelling and grouping of controls, making the interface easy to understand and navigate. Feedback through processing indicators and status messages keeps users informed about system state and operations, reducing uncertainty. Error message clarity is ensured through error messages with guidance, helping users recover from errors. Progressive disclosure presents features in a way that doesn't overwhelm users, making the system approachable.

Accessibility features were incorporated throughout the design to ensure the application is usable by users in most conditions and situations. Text-to-speech functionality provides audio access to AI responses, making the application accessible to users with sight impairments. UI elements with sizing provides readability for all users. Keyboard navigation support for all interactive elements ensures the application can be used without a mouse. Labels and alt text for UI components improve screen reader compatibility, enhancing accessibility.

4.6.3 Interface Screenshots

The main chat interface, shown in Figure 42, displays the conversation history at the top, with user messages on the right and AI responses on the left, each with a copy text button. Below the chat area are the text input field and action buttons for image upload, regeneration, and clearing history. The image processing buttons (Extract Text, Caption Image, Summarize Image) are arranged horizontally for quick access instead of typing in the text box.



o convert text to speech for you.
Please bear in mind that I can only process one image at a time and responses may be cut off around 310~ words due to API limitations.

That sounds great!

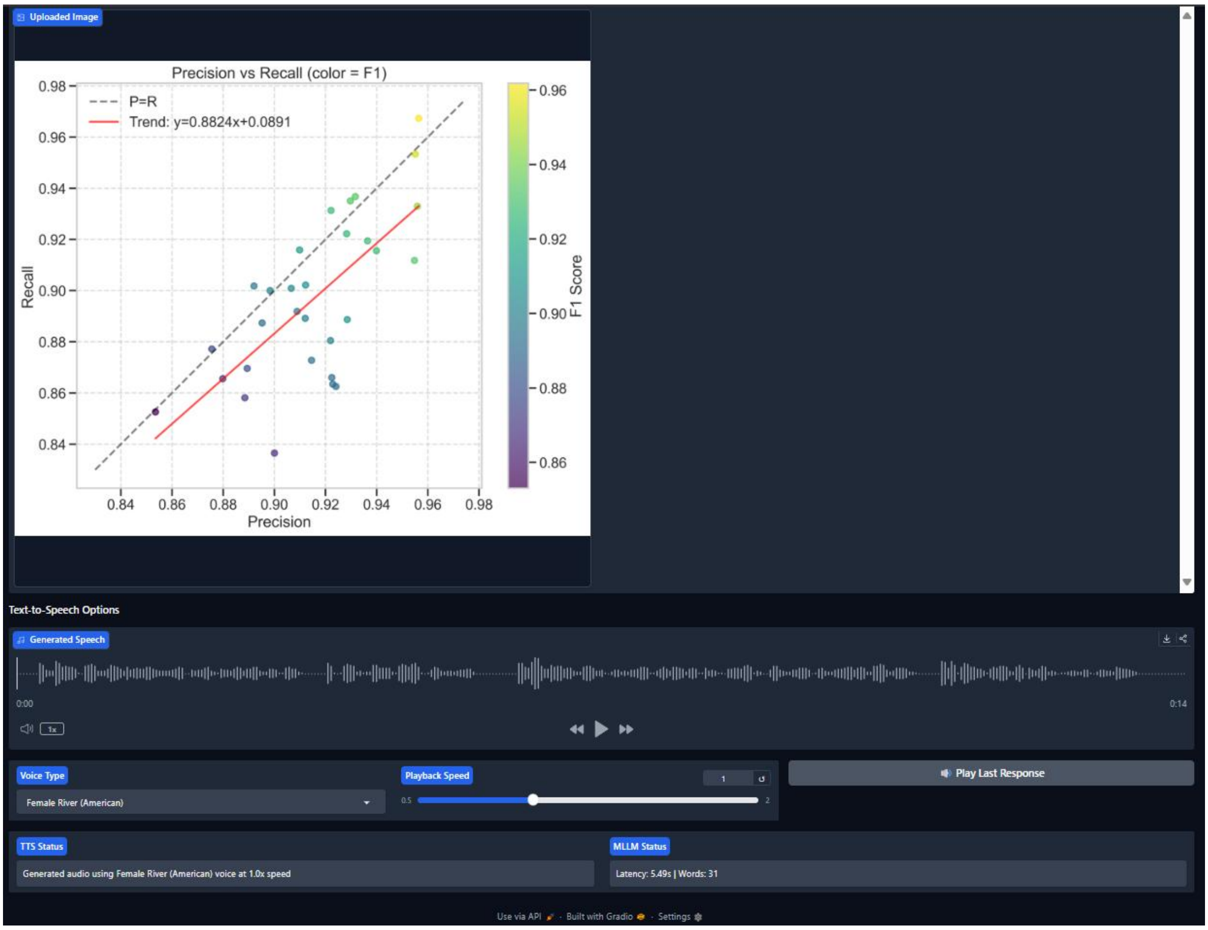
Feel free to upload an image whenever you're ready. I'm here to help! 😊

Create a concise caption for this image.

A scatter plot showing the relationship between precision and recall, with points colored by F1 score. The trend line is y = 0.8824x + 0.0891, and the dashed line represents P=R.

Figure 42 HearSee Chat User Interface

Figure 43 shows the interface after an image has been uploaded. The image is displayed in the gallery, and all image-related action buttons are now enabled. The text-to-speech section at the bottom shows the voice selection dropdown, speed slider, and play button. The generated speech section shows the post audio generation speed control, volume control, duration of audio and download file button. The status indicators at the bottom right show the TTS status and performance metrics.



Voice Type: Female River (American)

Playback Speed: 1

TTS Status: Generated audio using Female River (American) voice at 1.0x speed

MLLM Status: Latency: 5.49s | Words: 31

Figure 43 HearSee Image Upload and Text-to-Speech Options

4.7 Back-end Implementation

The following subsections describes the back-end implementation of HearSee. The backbone of the system consists of services, utilities and configuration files.

4.7.1 Server Architecture

The back-end implementation handles responsibilities that are essential to the system's functionality. Data processing and computation are managed through service modules that encapsulate functionality. File management and storage for images and audio ensure proper resource usage. API communication with Replicate enables AI model inference for image analysis and text-to-speech. Session management and state handling for the UI maintain a consistent user experience across interactions.

The server architecture is built around the concept of services that are instantiated as needed, with state maintained in the Gradio interface. This approach simplifies the implementation and avoids the need for state management, making the system maintainable and easier to reason about.

4.7.2 Data Processing Pipeline

The implementation includes a data processing pipeline that handles the flow of data from user input to AI processing and back to the user interface. The ReplicateService's run_vision_model method exemplifies this pipeline:

```
# Example of the data processing pipeline in the ReplicateService
```

```python
@staticmethod
def run_vision_model(prompt, image_base64=None,
max_tokens=DEFAULT_MAX_TOKENS):
    """
    Run the Qwen VL model with given prompt and optional image.

    Args:
        prompt (str): The text prompt for the model.
        image_base64 (str, optional): Base64 encoded image. Defaults to
None.
        max_tokens (int, optional): Maximum number of tokens to generate.
Defaults to DEFAULT_MAX_TOKENS.

    Returns:
        str: Model's text response.
    """
    # Validate API availability before running
    api_available, error_msg = ReplicateService.verify_api_available()
    if not api_available:
        logger.error(f"API not available: {error_msg}")
        raise ValueError(error_msg)

    # Prepare API parameters
    api_params = {
        "prompt": prompt,
        "max_new_tokens": max_tokens,
    }

    # Add image if provided
    if image_base64:
        api_params["media"] = f"data:image/png;base64,{image_base64}"
        logger.info("Image included in vision model request")
    else:
        logger.info("Running vision model without image")

    # Run the model
    try:
        logger.debug(f"Calling Replicate API with model: {QWEN_VL_MODEL}")
        output = replicate.run(QWEN_VL_MODEL, input=api_params)
        logger.info("Vision model API call completed successfully")
```

```
        # Replicate may return output as a list of string chunks or a
single string
        # We join the chunks if it's a list, otherwise return as is
        return "".join(output) if isinstance(output, list) else output
    except Exception as e:
        logger.error(f"Error running vision model: {str(e)}",
exc_info=True)
        raise RuntimeError(f"Error running vision model: {str(e)}")
```

The service-based architecture encapsulates functionality in service classes, making it easier to maintain and conduct tests by providing boundaries and responsibilities, as they don't maintain state between calls. Logging implementation provides visibility into the system's operation, allowing easier debugging and monitoring. Error propagation up the call stack with context enables handling at the UI level, providing informative feedback.

4.7.3 Integration Points

The front-end and back-end components integrate through an event handling system in the main application (app.py). The integration points include event handlers that respond to UI events and call back-end services, state management through Gradio state components that maintain application state across interactions, data transformation functions that convert data between UI and service formats, and error handling patterns that provide feedback to users.

The event chain for sending a message demonstrates how these integration points work together:

```
# Event chain for message submission via Enter key
send_handler = msg.submit(
    # Step 1: Show processing state and disable all interactive elements
    start_processing,
    inputs=None,  # No inputs needed
```

```
    outputs=[processing_indicator, msg, send_btn, upload_btn, extract_btn,
             caption_btn, summarize_btn, regenerate_btn, tts_btn,
processing_status]
).then(
    # Step 2: Process the message with the AI model
    locked_chat_response,
    inputs=[msg, chatbot, performance_metrics, image_output],  # Message
and context
    outputs=[chatbot, performance_metrics, msg],  # Updated conversation
and metrics
    show_progress="full"  # Show progress bar during processing
).then(
    # Step 3: Restore UI state after processing completes
    end_processing,
    inputs=[chatbot, performance_metrics, image_uploaded_state],  # Current
state
    outputs=[chatbot, performance_metrics, processing_indicator, msg,
send_btn,
             upload_btn, extract_btn, caption_btn, summarize_btn,
             regenerate_btn, tts_btn, processing_status]  # UI elements to
update
)
```

This event chain demonstrates how the UI and back-end components work together to provide a functional web application to the users, with state transitions and feedback throughout the process.

4.8 Configuration and Customization

The following subsections describes the configuration variables and customization options for HearSee alongside explanations for implementation decisions.

4.8.1 Configuration Management

The system implements configuration management through the settings.py module, which centralizes all constant values and configuration parameters. This approach provides a default instantiation of values for system settings and makes the application maintainable and configurable:

```
# Configuration structure example from settings.py
# Replicate Model Constants - specific model versions for reproducibility
QWEN_VL_MODEL = "lucataco/qwen2-vl-7b-
instruct:bf57361c75677fc33d480d0c5f02926e621b2caa2000347cb74aeae9d2ca07ee"
KOKORO_TTS_MODEL = "jaaari/kokoro-
82m:f559560eb822dc509045f3921a1921234918b91739db4bf3daab2169b71c7a13"

# API Configuration - controls response length
DEFAULT_MAX_TOKENS = 512  # Balances between detailed responses and API
costs

# Image Processing Settings - prevents uploading excessively large images
MAX_IMAGE_SIZE = 10 * 1024 * 1024  # 10MB in bytes (10 * 1024KB * 1024B)

# TTS Configuration - voice options available from Kokoro TTS for text-to-
speech conversion
VOICE_TYPES = {
    "Female River (American)": "af_river",
    "Female Bella (American)": "af_bella",
    "Female Emma (British)": "bf_emma",
    "Male Michael (American)": "am_michael",
    "Male Fenrir (American)": "am_fenrir",
```

```
    "Male George (British)": "bm_george"
}

# Valid range for speech speed adjustment - prevents unintelligible output
TTS_SPEED_RANGE = (0.5, 2.0)  # (min_speed, max_speed)

# Default TTS Settings - used when user doesn't specify preferences
DEFAULT_VOICE = "Female River (American)"  # Must match a key in VOICE_TYPES
DEFAULT_SPEED = 1.0  # Normal speaking rate (1.0x)
```

The configuration approach makes that all constants are defined in one place, providing a source of truth that prevents inconsistencies. Each setting is documented with comments explaining its purpose, making the configuration self-documenting. Settings can be changed without modifying code, simplifying configuration updates. All modules access settings through the same interface, ensuring consistent usage throughout the application.

4.8.2 Customization Options

The implementation provides customization options that allow the system to be adapted to use cases and requirements. Model selection can be updated through the Replicate model identifiers to use AI models as they become available. Voice options can be extended or modified to support different voices and languages, enhancing the text-to-speech experience. UI theming can be customized to match branding requirements, providing a visual identity. Response length can be adjusted through the maximum token limit to balance detail and cost based on needs. Image size limits can be modified based on system capabilities and performance requirements.

These customization options allow the system to be adapted to use cases and requirements without code changes, enhancing its flexibility and longevity.

4.9 Security Considerations

The following subsections describes the security considerations of HearSee. Due to the simplicity of possible user inputs, input validation is the most important step to ensure the web application continues to function in most situations. Data protection is also highlighted as the web application features using user images and text as well as API keys for AI inferencing requests.

4.9.1 Input Validation

All user inputs undergo validation to ensure system security and stability. The image validation function represents this approach:

```python
def validate_image_input(image):
    """
    Validate that an image is provided and in the correct format.

    Args:
        image: The image to validate

    Returns:
        tuple: (is_valid, error_message)
    """
    if image is None:
        return False, "Please upload an image first. The system requires an image to analyze."

    try:
        # Additional validation could be added here
        # For example, checking image dimensions, content type, etc.
        return True, ""
    except Exception as e:
        return False, f"Invalid image format: {str(e)}"
```

The validation strategy includes several components. Existence checks ensure that required inputs are provided, preventing null reference errors. Type checks verify that inputs are of the expected type, to make sure it works with processing functions. Range checks ensure numerical values are within ranges, preventing performance issues and behaviour. Format validation verifies that inputs meet format requirements to be compatible with downstream processing. Size limits are enforced on uploads to prevent resource exhaustion, protecting system stability.

4.9.2 Data Protection

Data protection measures were implemented to ensure user privacy and system security. API key protection is implemented by storing the Replicate API key in environment variables rather than being hardcoded inside the main source code, preventing exposure in version control systems. Ephemeral storage is used for images and audio files, which are cleaned up after use to minimize data retention. This means that the system has no permanent storage of user data, enhancing privacy by ensuring that user related inputs and outputs are not kept after exiting the web application. Communication is ensured through HTTPS for data protection in transit when communicating with external APIs. User inputs are sanitized before use to prevent injection attacks and security vulnerabilities. These security measures help to keep user data protected throughout the application lifecycle, preserving privacy and security while keeping the system functional.

4.10 Implementation Challenges and Solutions

The following subsections describes the implementation challenges faced during development and the solutions found.

4.10.1 Technical Challenges

The implementation of the HearSee system encountered several technical and design challenges that required solutions. One technical challenge was handling large image files, specifically more than 10MB. The solution involved implementing size validation and limits to prevent resource exhaustion. Images are validated before processing, and users are provided with feedback when images exceed size limits. This approach improved system stability and prevented crashes due to invalid API requests.

Another challenge was managing API rate limits and costs associated with the Replicate service. The solution implemented token limits and prompt construction to minimize API usage. The system also provides performance metrics to users, making them aware of resource usage. This approach reduced API costs while maintaining response quality and improved user awareness of system performance. Though token limits do causes long responses to be cut off, the user can prompt to continue the response.

Creating a user interface that handles asynchronous operations presented another challenge. The solution involved implementing a state-based UI with visual feedback during processing. The UI disables interactive elements during processing and shows progress indicators to keep the users informed. This enhanced the user experience by providing feedback about the state of the system and preventing operations that could lead to conflicts or errors.

Ensuring compatibility across Gradio versions was also challenging. The solution implemented version detection and fallback mechanisms for theme creation and component configuration. This improved compatibility across environments and simplified deployment, making the more system robust and adaptable.

4.10.2 Design Trade-offs

Trade-offs were made during implementation to balance requirements and constraints. The choice between a web interface and a native application was resolved in favour of a web interface with Gradio. This provided development and cross-platform compatibility at the cost of performance and offline capabilities. The trade-off was justified by the need for deployment and sharing to many users, which aligned with the project's goals.

Another trade-off involved choosing between cloud API and local models for AI processing. Using Replicate's API for AI models instead of running models locally reduced system requirements and simplified deployment, but introduced API dependency and costs. The benefits of accessing models without hardware requirements outweighed these drawbacks, making this a trade-off.

The decision between temporary and persistent storage was another consideration. Using temporary storage for images and audio files enhanced privacy and reduced storage requirements, but limited the ability to save and retrieve past interactions. This aligned with the project's focus on privacy and simplicity, making it a trade-off for the system's goals.

Finally, the decision to focus on only image-text-to-text and text-to-speech was also a consideration. Limiting the application to core image analysis and TTS features allowed for faster implementation of these features, rather than a broader but slower to implement feature sets. This trade-off ensured that the implemented features worked well, rather than attempting to implement too many features with lower quality or non-functional.

4.11 Summary of Implementation

This chapter has presented the implementation of the HearSee system. The implementation translates the design into a system using Python, Gradio, and Replicate API integration. Key implementation includes a three-tier architecture, which enhances maintainability and scalability. The user interface for image and text-based analysis and text-to-speech provides ease of use and accessible. Error handling and validation improve reliability so that the system degrades when issues occur. Integration with AI models through the Replicate API enables image analysis and speech synthesis capabilities. Logging and performance metrics enhance transparency and facilitate debugging.

The implementation provides a foundation for the testing and evaluation that will be discussed in the following chapter. The architecture and error handling ensure system reliability and maintainability, while the design principles enhance usability and accessibility. These characteristics makes HearSee a tool for interaction combining vision and voice capabilities for improving learning capabilities for undergraduate students.

Chapter 5: Testing and Evaluation

The testing and evaluation of HearSee project employed a three-pronged approach encompassing automated testing, semantic evaluation, and user acceptance testing. These methodologies were selected to ensure thorough validation of both technical robustness and functional capabilities of the web application. The combination of these testing strategies provided additional perspectives on system quality, with each approach addressing distinct aspects of the HearSee's performance.

Automated testing focused on code correctness, reliability, and functional integrity through the unit, integration and functional testing of individual components and their interactions. Semantic evaluation assessed the quality of the Multimodal Large Language Model (MLLM) generated content by comparing outputs from the implemented Qwen2 VL 7B model against a frontier model (Claude 4 Sonnet) using established natural language processing metric (BERTScore). User acceptance testing gathered direct feedback from potential users to validate usability, effectiveness, and educational value in real-world scenarios. Together, these testing methodologies created an evaluation framework that validated both technical implementation and practical utility of HearSee as an educational tool.

Appendix E is included for the visual evidence of test execution and data collection for HearSee project testing. Table 9 presents the summary of the project’s testing scope.

Table 9 Summary of Testing Scope

Testing Type	Data Source	Method	Quantity
Automated Testing	Source code	Pytest Test Suite (Unit, Integration, Functional)	139 Tests

Semantic Quality Evaluation	Educational Image-to-Text Pairs	BERTScore	30 Sentence Pairs
User Acceptance Testing	Undergraduate Students participants	Google Forms Survey	10 users, 19 questions

5.1 Automated Testing Framework

The following subsections describes the implementation of the testing architecture, code coverage analysis and the test suite composition for HearSee. Verification is achieved by running a comprehensive test suite consisting of 139 tests using Pytest. Screenshots of the terminal output from the test run is provided in Appendix E as proof of automated testing.

5.1.1 Testing Architecture

The automated testing framework for HearSee was implemented using pytest, a Python-based testing framework for Python applications. The testing approach utilised unit, integration, and functional tests, each targeting different aspects of the system. Unit tests focused on validating individual functions and methods separately so that each component behaves correctly. Integration tests verified the correct interaction between multiple components, such as the image processing pipeline and text-to-speech conversion workflow. Functional tests were made to test complete user workflows and system behaviours from an end-user perspective, including responsive design and UI interactions. The testing framework uses pytest fixtures to create consistent test environments and reduce code duplication across test cases. Mock objects were used to separate components from external dependencies for API services to enable controlled testing of error handling and edge cases. This structured approach allowed high test coverage of the codebase.

5.1.2 Code Coverage Analysis

Verification of the system's technical robustness was performed by executing the full automated test suite, which comprises 139 individual tests. Each test is designed with specific pass/fail assertions that validate expected outcomes. The system's behaviour is considered verified when all 139 tests pass without error. The automated test suite achieved an overall code coverage of 86%, representing a high level of test verification across the HearSee codebase. This coverage metric indicates that the majority of the application code was executed during test runs, reducing the risk of undetected bugs or regressions. Table 10 shows the coverage analysis by component files which shows the overall coverage in critical areas of the application. The services layer, responsible for core functionality including image processing, API interactions, and text-to-speech conversion, maintained coverage more than 89%, with the image service achieving 95% coverage, replicate service 91%, and text-to-speech service 89%. The UI components revealed even higher coverage at over 98%. Utility functions achieved near-complete coverage. The coverage analysis identified app.py as an area of lower coverage at 15%, which was an intentional testing decision. This main application file primarily contains initialization code and Gradio interface setup that is hard to test in isolation.

Table 10 Code Coverage Report by Files

File	**statements**	**missing**	**excluded**	**coverage**
app.py	129	110	6	15%
utils__init__.py	43	35	0	19%
utils\logger.py	63	45	10	29%
ui__init__.py	35	20	0	43%
tests\fixtures\test_data.py	37	12	0	68%
tests\conftest.py	53	14	0	74%
services\tts_service.py	57	6	0	89%

File	statements	missing	excluded	coverage
services\replicate_service.py	55	5	0	91%
services\image_service.py	56	3	0	95%
config\logging_config.py	36	1	0	97%
tests\unit\config\test_settings.py	58	2	0	97%
tests\unit\services\test_image_service.py	65	2	1	97%
tests\unit\ui\test_chat_interface.py	60	1	0	98%
utils\image_utils.py	97	1	0	99%
config__init__.py	2	0	0	100%
config\settings.py	9	0	0	100%
services__init__.py	7	0	0	100%
tests__init__.py	0	0	0	100%
tests\fixtures__init__.py	0	0	0	100%
tests\functional__init__.py	0	0	0	100%
tests\functional\test_responsive_design.py	64	0	0	100%
tests\integration__init__.py	0	0	0	100%
tests\integration\test_error_handling.py	82	0	0	100%
tests\integration\test_image_processing_pipeline.py	62	0	0	100%
tests\integration\test_tts_pipeline.py	101	0	0	100%
tests\integration\test_ui_interactions.py	79	0	0	100%
tests\test_config.py	13	0	0	100%
tests\unit__init__.py	0	0	0	100%
tests\unit\config__init__.py	0	0	0	100%
tests\unit\services__init__.py	0	0	0	100%
tests\unit\services\test_replicate_service.py	35	0	0	100%
tests\unit\services\test_tts_service.py	86	0	0	100%
tests\unit\ui__init__.py	0	0	0	100%
tests\unit\ui\test_components.py	78	0	0	100%
tests\unit\ui\test_guide_interface.py	54	0	0	100%

File	statements	missing	excluded	coverage
tests\unit\utils__init__.py	0	0	0	100%
tests\unit\utils\test_image_utils.py	87	0	0	100%
tests\unit\utils\test_validators.py	61	0	0	100%
ui\chat_interface.py	34	0	0	100%
ui\components.py	12	0	0	100%
ui\guide_interface.py	7	0	0	100%
utils\validators.py	62	0	5	100%
Total	1779	257	22	86%

5.1.3 Test Suite Composition

The HearSee test suite comprises a total of 139 individual tests distributed across the three testing tiers. Based on detailed analysis of the test execution results done after automated testing as shown in Table 11, unit tests form the foundation of the testing procedures with 99 tests (71.22% of the total), focusing on detailed verification of individual functions and methods. Integration tests account for 33 tests (23.74%), validating component interactions and workflows. Functional tests complete the suite with 7 tests (5.04%), testing end-to-end functionality from a user perspective.

Table 11 Test Distribution by Category

Test Category	Count	Percentage	Key Focus Areas
Unit	99	71.22%	Individual function validation, edge cases, error handling
Integration	33	23.74%	Component interactions, workflow validation, API integration

Test Category	**Count**	**Percentage**	**Key Focus Areas**
Functional	7	5.04%	End-to-end workflows, UI responsiveness, user journeys
Total	139	100.00%	

The unit tests concentrated on validating core functionality within isolated components, with emphasis on the services layer (image processing, API interactions, text-to-speech conversion) and utility functions (validators, image utilities, logging). Integration tests focused on key workflows including the image processing pipeline, text-to-speech pipeline, error handling across components, and UI interactions. Functional tests addressed responsive design across different viewport sizes, complete user journeys, and cross-component interactions. Other test implementations included error handling tests that verified the system's resilience against various failure scenarios such as API unavailability, network failures, invalid inputs, and resource constraints. The image processing pipeline tests validated the complete workflow from image upload through text extraction, captioning, and summarization. The text-to-speech pipeline tests ensured correct voice synthesis across different voice types and speech parameters.

5.2 Semantic Quality Evaluation using BERTScore

The following subsections describes the BERTScore methodology, results and semantic quality analysis of the MLLM used for HearSee project. The semantic quality of the generated text was verified by calculating the precision, recall and F1 score using BERTScore for each of the 30 text pairs against a reference model. This score quantitatively measures semantic similarity, and the results are detailed in Section 5.2.2. The generation of the candidate dataset and the subsequent semantic evaluation were performed using custom-developed Python scripts. Appendix E contains screenshots showing the execution of these tools as proof of the evaluation process.

5.2.1 BERTScore Methodology

BERTScore (Zhang et al., 2019) was selected as the primary metric for evaluating the semantic quality of HearSee's image-to-text capabilities due to its ability to capture deeper semantic relationships between texts beyond simple lexical matching. Unlike traditional metrics such as BLEU or ROUGE that rely on exact word matches, BERTScore leverages contextual embeddings from transformer models to compute token-level similarities using cosine similarity between embeddings (Zhang et al., 2019). This approach allows for more nuanced evaluation of semantic equivalence, which is important for assessing AI-generated descriptions of visual content where multiple valid phrasings may convey the same meaning. The benchmark setup consisted of 30 diverse text pairs generated from a representative sample of educational images, included as proof of testing in Appendix D, comparing outputs from the implemented Qwen2-VL 7B model against those from Claude Sonnet 4 (Anthropic, 2025), which served as the reference model. Claude Sonnet 4 was chosen as the reference model due to its position as a frontier multimodal model with high-quality image understanding capabilities. While using human-generated references would be ideal, the use of a frontier model as reference provided an automated approach to semantic quality evaluation, establishing a benchmark against current state-of-the-art capabilities in multimodal understanding.

30 educational images were processed by the frontier model, Claude Sonnet 4, to create a high-quality "reference" dataset. The same 30 images were processed by HearSee's implemented Qwen2-VL 7B model to create the "candidate" dataset. The candidate text was programmatically compared against the reference text using the BERTScore metric. The resulting Precision, Recall, and F1 scores, detailed in Section 5.2.2, serve as the quantitative verification of the system's semantic performance against a state-of-the-art model. Two custom tools were developed for generating the candidate dataset from Qwen2-VL 7B and the reference dataset from Claude Sonnet 4, and for performing the semantic evaluation using BERTScore with RoBERTa large model (Liu et al., 2019) as the transformer model for similarity computation.

5.2.2 BERTScore Results

The BERTScore evaluation revealed strong semantic similarity between Qwen2-VL 7B outputs and the reference Claude Sonnet 4 outputs across the test dataset. The key metrics exhibited high performance: Based on the histogram (Figure 44), precision achieved 0.9156, indicating that Qwen2-VL 7B's outputs contained highly relevant content with minimal extraneous information compared to Claude Sonnet 4. Recall reached 0.8969, showing that Qwen2-VL 7B captured most of the relevant information present in Claude Sonnet 4's outputs. The F1 score, which balances precision and recall, was 0.9060, reflecting strong overall semantic similarity.

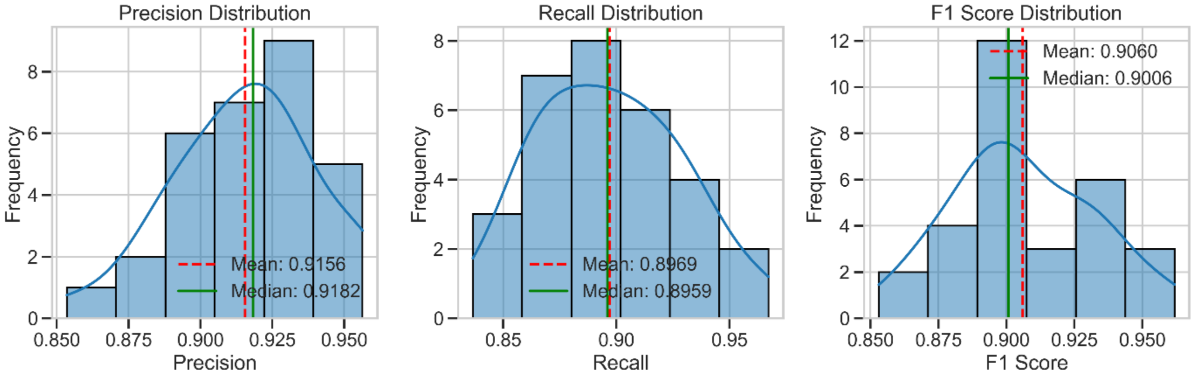


Figure 44 Precision, Recall & F1 Score Histogram

The precision-recall scatter plot (Figure 45) revealed consistent performance across samples with minimal outliers, indicating reliable semantic quality across different types of educational content. The precision-recall relationship followed a positive linear trend represented by the equation y = 0.8824x + 0.0891, suggesting a balanced trade-off between these metrics with a slight tendency toward higher precision than recall. This relationship indicates that Qwen2-VL 7B prioritizes accuracy over completeness when compared to Claude Sonnet 4, which is preferable for educational applications where factual correctness is important.

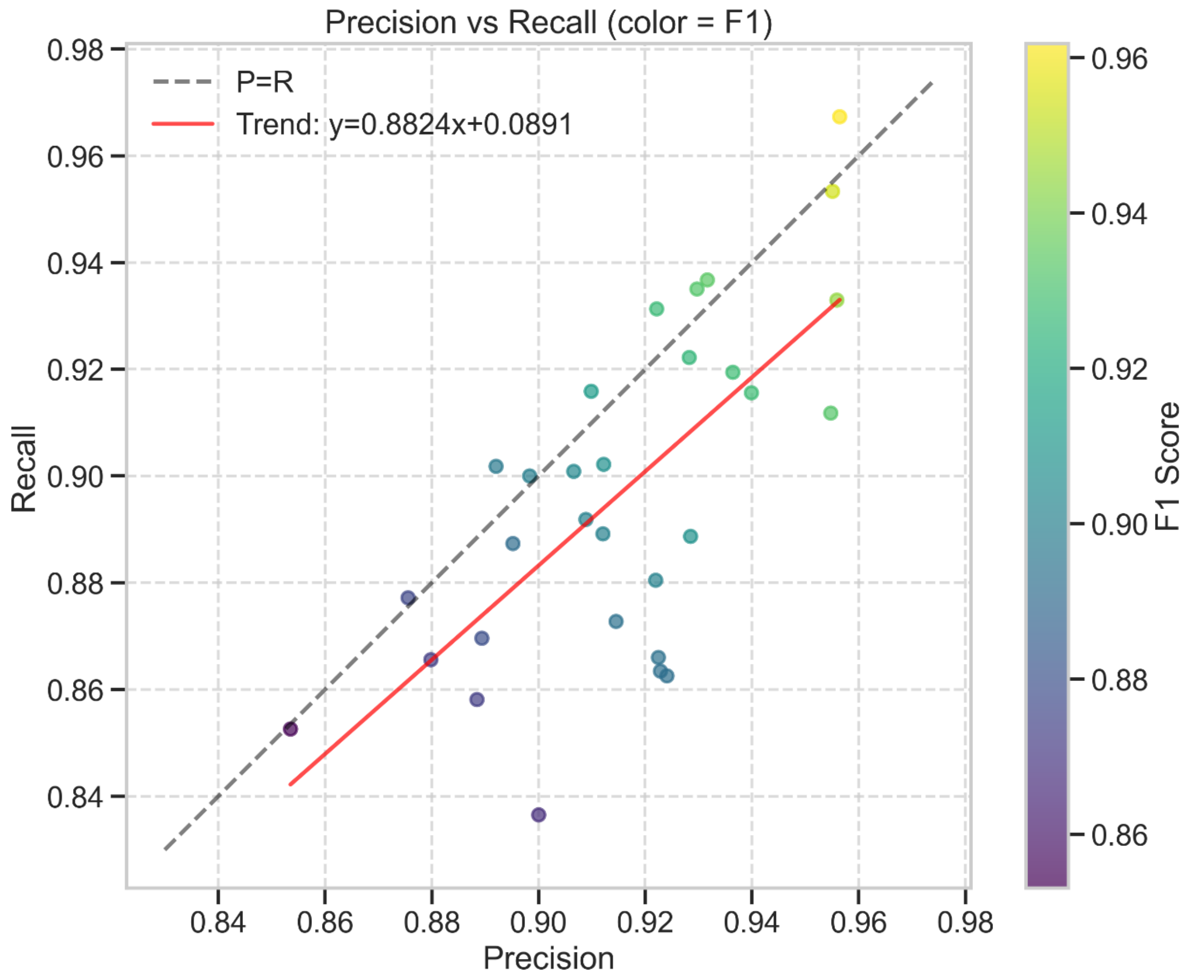


Figure 45 Precision & Recall Scatter Plot

5.2.3 Semantic Quality Analysis

The precision-recall trade-off observed in the BERTScore results shows important characteristics of Qwen2-VL 7B's performance in the context of educational content processing. The consistently higher precision than recall across samples indicates that Qwen2-VL 7B prioritizes accuracy over completeness, making it well-suited for educational applications where factual correctness is critical. When evaluated against established quality thresholds (Figure 46), both precision (0.9156) and F1 (0.9060) scores fell within the "Excellent" range (>0.9), while recall (0.8969) was at the high end of the "Good" range (0.8-

0.9), just 0.0031 points below the "Excellent" threshold. These results validate the choice of Qwen2-VL 7B for educational content processing, demonstrating that it achieves semantic quality comparable to frontier models like Claude Sonnet 4 despite having substantially fewer parameters. This efficiency makes it suitable for deployment in educational settings where computational resources may be limited but high-quality content analysis is still essential.

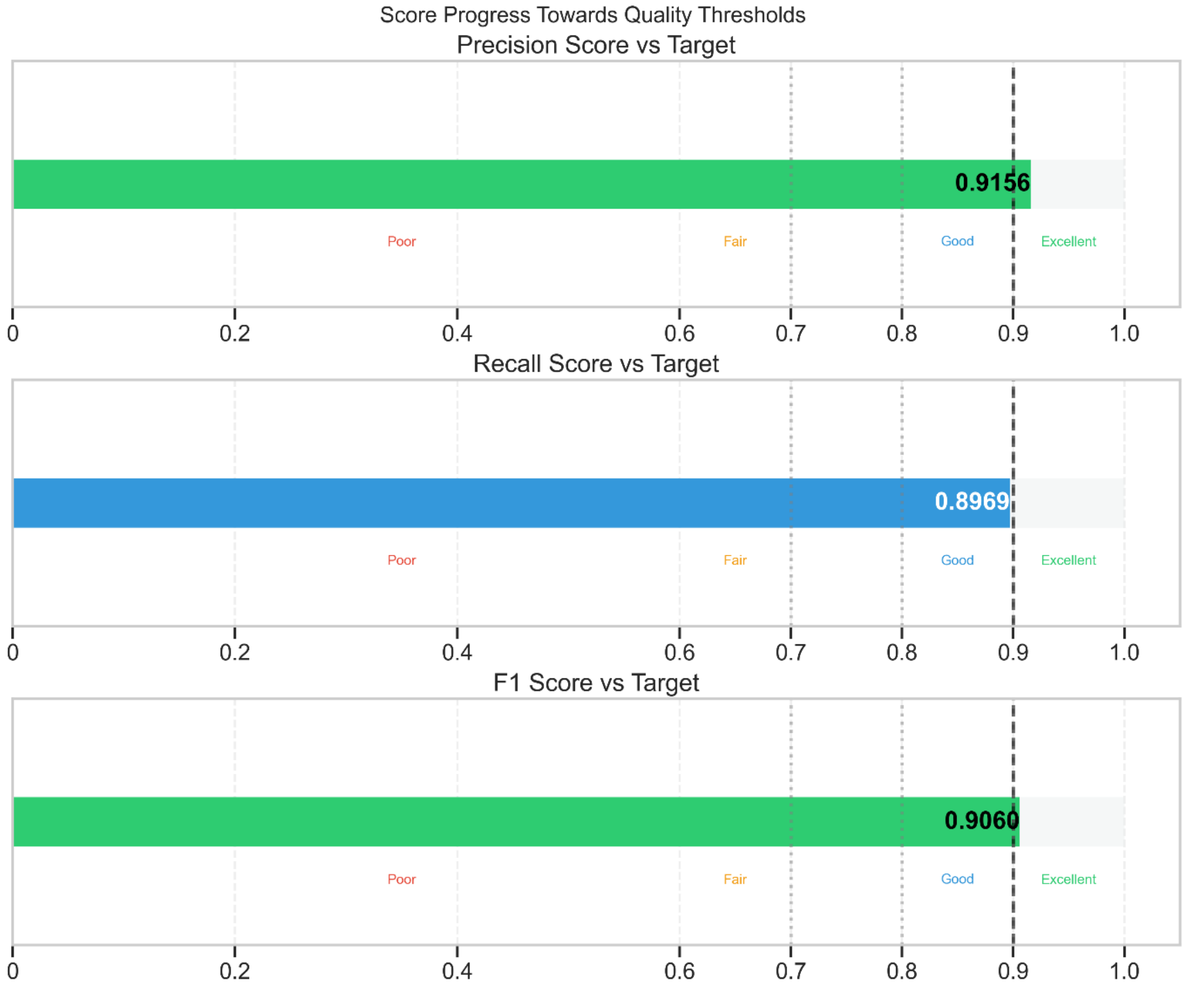


Figure 46 Score Thresholds Graph for Precision, Recall & F1 Score

5.3 User Acceptance Testing

The following subsections describes the user testing methodology, user testing results and the qualitative user feedback from the user acceptance testing conducted. The application's usability and educational value were verified through direct user feedback, collected via a structured survey. The quantitative results were aggregated and are presented in Table 12, while qualitative feedback was analyzed for common themes. Specific questions used to gather this feedback are provided in Appendix C. Screenshots of the survey form is included in Appendix E as proof of the testing instrument.

5.3.1 User Testing Methodology

To verify the application's real-world usability, effectiveness, and educational value, user acceptance testing was conducted with 10 participants from the target demographic of undergraduate students. The testing was designed to evaluate the system against key performance criteria which are usability, effectiveness, and educational value through a series of structured tasks that exercised all major features of the system. Quantitative verification was done through user ratings on the 5-point Likert scale were aggregated to calculate the average score and standard deviation for each evaluation criterion, as presented in Table 12. Qualitative verification was through open-ended user feedback was thematically analysed to identify recurring strengths and areas for improvement, corroborating the quantitative findings.

Users were instructed to fill in their response in the feedback form by completing tasks including uploading various educational images, extracting text from images, generating summaries, asking questions about the content, and converting text to speech with different voice options and speed settings.

Feedback was collected through a Google Forms survey using a 5-point Likert scale, where value of 1 indicates the poorest rating and 5 for indicating the excellent rating. This is to evaluate specific aspects of the application as well as optional text inputs for their thoughts or insights. The survey assessed the user experience, including ease of use, processing speed, accuracy of text extraction, quality of content analysis, speech synthesis quality, and overall educational value. The 5-point Likert scale was selected to provide sufficient granularity for analysis while being easy for participants to provide their input.

5.3.2 User Testing Results

The user testing results demonstrated positive overall satisfaction with HearSee across multiple evaluation criteria. The application received high ratings for ease of use, with image upload functionality scoring an average of 4.9 out of 5, indicating near-perfect usability for the feature. Content analysis capabilities also gained high ratings, with text extraction accuracy scoring 4.7 out of 5 and content explanations averaging 4.3 out of 5. The chat interface functionality was rated at 4.3 out of 5 for intuitiveness and 4.6 out of 5 for response relevance, confirming the effectiveness of the conversational aspect of the model. Speech synthesis features received high ratings, with clarity of speech output scoring 4.6 out of 5 and naturalness of speech averaging 4.1 out of 5. Voice preferences showed some variation among participants, with Male Michael and Male George being the most frequently preferred voice options, followed by Female River and Female Bella. The educational value of the application was affirmed with participants rating the tool’s helpfulness for understanding educational content at 4.6 out of 5 and the enhancement of learning experience through the multimodal approach at 4.6 out of 5. Areas identified for improvement included processing speed, which received the lowest average rating at 3.8 out of 5, which is mainly affected by the quality of the participants internet and image size to be uploaded. Table 12 shows the evaluation criteria, average score and standard deviation of each criterion as a summary of the user acceptance test results.

Table 12 User Testing Results Summary

Evaluation Criteria	**Average Score (1-5)**	**Standard Deviation**
Ease of image upload	4.9	0.32
Image processing speed	3.8	0.42
Text extraction accuracy	4.7	0.48
Quality of content explanations	4.3	0.48
Quality of content summaries	4.1	0.74
Chat interface intuitiveness	4.3	0.48
Response relevance	4.6	0.52
Instruction understanding	4.4	0.52
Speech output clarity	4.6	0.52
Speech naturalness	4.1	0.57
Speech speed adjustment utility	4.3	0.67
Educational content understanding	4.6	0.52
Multimodal learning enhancement	4.6	0.70
Likelihood of future use	4.2	0.79
Task completion rate of success	100%	0.00

5.3.3 Qualitative User Feedback

The survey feedback collected during user testing revealed several ideas regarding HearSee's strengths and opportunities for improvement. Participants rated high for the multimodal approach, essentially how the combination of visual, textual, and audio modalities enhanced their understanding of complex educational content. The accuracy of text extraction from images was noted as impressive based on the scores, especially for complex educational diagrams and charts that OCR systems may fail to perform with equivalent results. Several participants rated high on the natural-sounding speech synthesis, as well as the range of voice options that allowed personalization of the learning experience. The most frequently mentioned area for improvement was language support, with two participants explicitly suggesting expansion to handle multiple languages beyond English. Processing speed was another area identified for enhancement for large images. Despite these suggestions for improvement, all participants reported successful completion of their assigned tasks, indicating that the current implementation is functionally complete and usable as no notable pain points are identified from the user feedbacks.

5.4 Summary of Testing and Evaluation

The testing of HearSee through automated testing, semantic evaluation, and user acceptance testing provides a multidimensional assessment of the web application's quality and effectiveness as an educational tool. The automated testing results demonstrate technical robustness with 86% overall code coverage and high coverage in critical components including services, UI, and utilities. The 139 tests across unit, integration, and functional levels provide systematic verification of the application's behaviour under various conditions, including error scenarios and edge cases. The semantic evaluation using BERTScore validates the quality of generated content, with Qwen2-VL 7B achieving high precision and F1 scores when compared to the frontier model Claude 4 Sonnet. This affirms that the selected vision-language model delivers high-quality content analysis suitable for educational applications. User testing results corroborate these technical findings from an end-user perspective, with high ratings for usability, content quality, and educational value.

Chapter 6: Conclusion

6.1 Introduction

The HearSee project was proposed and developed for addressing the challenge of information retrieval and retention among undergraduate students. The project utilises Multimodal Large Language Models (MLLMs) to transform visual educational content into accessible text and audio formats, thus directly responds to the growing need for diverse learning modalities in higher education. The value of this work lies in its practical application of Generative AI technologies to create demonstrable improvements in educational accessibility, including students with visual or auditory processing difficulties.

6.2 Objectives and Achievements

The primary objective of developing a web application that leverages multimodal LLMs to analyse images and provide text descriptions was successfully achieved through the implementation of the HearSee system. The application is capable of processing educational images using the Qwen2-VL 7B model, demonstrating excellent semantic quality from the BERTScore benchmark when compared against frontier models.

The second objective of implementing text-to-speech functionality was accomplished through integration of the Kokoro TTS engine, achieving high speech clarity ratings and naturalness ratings from user testing, with six distinct voice options and adjustable speed controls.

The third objective of creating an accessible web-based interface was met through the development of a Gradio-powered application with 100% task completion rates during user testing and an exceptional usability rating for core functionality. The system architecture achieved 86% code coverage through automated code testing, to ensure reliability while maintaining a clean separation of concerns across the three-tier architecture. User acceptance testing with undergraduate students confirmed the educational value of the tool, with ratings of 4.6 out of 5 for both educational content understanding and multimodal learning enhancement.

6.3 Discussion

The technical implementation of HearSee showed in detail about the application of MLLMs in educational contexts. The decision to use Qwen2-VL 7B rather than larger, more expensive models proved effective, demonstrating that cost efficient models with fewer parameters can achieve comparable performance for educational applications when properly implemented. This finding suggests that high-quality learning experiences can be delivered without requiring frontier-level capability and resources.

The three-tier architecture separating UI, services, and utilities made the system easier to maintain and test thoroughly, resulting in a more reliable application. Users shown interest in the multimodal approach, specifically on how combining visual analysis, text extraction, and speech synthesis improved their learning. Using temporary file storage for images and audio effectively balanced functionality with privacy protection, which is essential for educational applications that must protect student data. The integration of performance metrics directly in the interface provided transparency to users about system operations.

6.4 Constraints and Limitations

HearSee faced several technical constraints that impact its functionality and deployment. The reliance on cloud-based API services introduces dependency on internet connectivity and Replicate service availability, preventing offline usage in environments with limited connectivity. The current implementation processes images with a high latency which, while still acceptable, could impact user experience during extended sessions with multiple images. The 512-token limit occasionally truncates longer responses, requiring users to request continuations for broader analyses of complex educational materials.

Resource constraints include funding requirements for cloud-based processing that incur API costs, potentially limiting scalability in resource-constrained environments. The temporary file storage approach, while beneficial for privacy, prevents session persistence across user sessions, requiring reprocessing of previously analysed content upon loading the web application.

Scope limitations include the current restriction to English language processing, limiting the tool's applicability in multilingual educational contexts, and the absence of video processing capabilities that would benefit lecture-based learning scenarios.

6.5 Future Works and Recommendations

For future works, HearSee should first introduce the ability to use local model options to reduce network-based delays and be able to work offline, which is important for students with little to no internet access. Adding support for multiple languages would make it useful in more educational settings, especially where students speak different languages.

Future development should include video analysis for recorded lectures and educational videos, creating a more complete tool for modern learning. New sharing features could allow students share processed content and notes with classmates, encouraging collaborative learning. To help institutions adopt the system, integration plugins for popular Learning Management Systems should be created.

Future research study could include how multimodal learning tools affect student performance over time and create guidelines for using AI tools in teaching. For widespread use, the system requires deployment options that work for learning institutions, including server-based versions that don't require powerful devices and can be accessed by more students.

6.6 Summary

HearSee successfully shows how multimodal large language models can improve undergraduate education through AI-powered content processing. The project achieved its three main goals: analyzing images, converting text to speech, and creating an easy-to-use interface. This was achieved by combining multiple modalities into one learning tool that's both user-friendly and educationally viable. Testing and user feedback confirmed that the technical implementation works well, providing a blueprint for future educational technology that combines advanced AI with practical accessibility. The project revealed specific areas for improvement, particularly offline functionality, language support, and integration with educational institutions. As institutions look for new ways to support different learning styles, HearSee demonstrates how artificial intelligence can create more inclusive and effective learning environments.

References

Anthropic. (2025). *Claude Sonnet 4*. https://www.anthropic.com/claude/sonnet

Gradio-App. (n.d.). GitHub - gradio-app/gradio: Build and share delightful machine learning apps, all in Python. 🌟 Star to support our work! GitHub. https://github.com/gradio-app/gradio

Grand, G., & Belinkov, Y. (2019). Adversarial regularization for visual question answering: strengths, shortcomings, and side effects. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1906.08430

Gupta, S. (2024, May 13). What are learning disabilities? Verywell Mind. https://www.verywellmind.com/learning-disabilities-types-causes-symptoms-and-treatment-6386232

hexgrad/Kokoro-82M · Hugging Face. (n.d.). https://huggingface.co/hexgrad/Kokoro-82M

Karpathy, A., & Fei-Fei, L. (2014). Deep Visual-Semantic alignments for generating image descriptions. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1412.2306

Lee, U., Jeon, M., Lee, Y., Byun, G., Son, Y., Shin, J., Ko, H., & Kim, H. (2024). LLaVA-Docent: Instruction Tuning with Multimodal Large Language Model to Support Art Appreciation Education. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2402.06264

Li, Y. A., Han, C., Raghavan, V. S., Mischler, G., & Mesgarani, N. (2023). StyleTTS 2: Towards Human-Level Text-to-Speech through Style Diffusion and Adversarial Training with Large Speech Language Models. arXiv (Cornell University). https://doi.org/10.48550/arXiv.2306.07691

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). ROBERTA: A robustly optimized BERT pretraining approach. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1907.11692

Lu, L., Deng, Y., Tian, C., Yang, S., & Shah, D. (2024). Purrfessor: a fine-tuned multimodal LLAVA diet health chatbot. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2411.14925.48550/arxiv.2306.07691

May, M. (2023, May 18). 7 reasons to love — and leverage — multimodal learning in your classroom. SolidProfessor. https://solidprofessor.com/blog/multimodal-approach-learning

Morrant, B. (2022, May 18). What do memory, processing and retention problems look like? The Speech and Langu. https://www.thespeechandlanguagegarden.com/post/what-do-memory-processing-and-retention-problems-look-like

Qwen/Qwen2-VL-7B-Instruct · Hugging face. (2024, December 6). https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct

Replicate. (n.d.). GitHub - replicate/replicate-python: Python client for Replicate. GitHub. https://github.com/replicate/replicate-python

Scrum workflow | Scrum board | Scrum | Scrum Guide. (n.d.). https://www.conceptdraw.com. https://www.conceptdraw.com/examples/scrum-guide

Singh, G. (2023, February 23). ACR (Absolute Category Rating). Telecom Trainer. https://www.telecomtrainer.com/acr-absolute-category-rating/

Spaces. (n.d.). https://huggingface.co/docs/hub/spaces

Streijl, R. C., Winkler, S., & Hands, D. S. (2014). Mean opinion score (MOS) revisited: methods and applications, limitations and alternatives. Multimedia Systems, 22(2), 213–227. https://doi.org/10.1007/s00530-014-0446-1

United States Artificial Intelligence Institute (USAII®). (n.d.). Understanding multimodal AI: benefits, working, and applications. https://www.usaii.org/ai-insights/understanding-multimodal-ai-benefits-working-and-applications. https://www.usaii.org/ai-insights/understanding-multimodal-ai-benefits-working-and-applications

Vasantharajan, C., & Thayasivam, U. (2021). Adapting the Tesseract Open-Source OCR engine for Tamil and Sinhala legacy fonts and creating a parallel corpus for Tamil-Sinhala-English. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2109.05952

Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., Chen, K., Liu, X., Wang, J., Ge, W., Fan, Y., Dang, K., Du, M., Ren, X., Men, R., Liu, D., Zhou, C., Zhou, J., & Lin, J. (2024). QWEN2-VL: Enhancing Vision-Language Model's perception of the world at any resolution. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2409.12191

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2019). BERTScore: Evaluating Text Generation with BERT. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1904.09675

Appendices

Appendix A: Gantt Chart Schedule for FYP 1

Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
Submission of Brief Proposal	🟩	🟩												
Submission of Full Proposal		🟩	🟩	🟩	🟩	🟩								
Submission of Chapter 1			🟩	🟩	🟩	🟩	🟩							
Submission of Chapter 2						🟩	🟩	🟩	🟩					
Submission of Chapter 3									🟩	🟩	🟩	🟩		
Submission of FYP 1 Final Report & Paper for Assessment											🟩	🟩	🟩	

Appendix B: Gantt Chart Schedule for FYP 2

Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
Submission of Proposed/Revised FYP Report, Title, Gantt Chart	🟩	🟩	🟩											
Design, Implementation, and Integration of LLM, front-end and back-end components	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩						
Submission of first draft of Chapter 4							🟩	🟩	🟩					
User Acceptance Testing and Evaluation									🟩	🟩	🟩			
Submission first draft of Chapter 5 and the FYP 2 Full Report & Paper											🟩	🟩	🟩	

Submission of FYP 2 Final Report, Source Code, User Manual & Paper for Assessment												🟩	🟩	🟩
FYP Symposium														🟩

Appendix C: Table of Questions and Ratings for User Acceptance Test

No.	Feedback Question	Rating Label / Format
1	How easy was it to upload images to the system?	1 (Very Difficult) – 5 (Very Easy)
2	How would you rate the speed of image processing? (Image appears on user interface)	1 (Very Slow) – 5 (Very Fast)
3	Did you encounter any errors or difficulties during image upload?	Yes / No
4	How accurately did the system extract text from the uploaded images?	1 (Not Accurate) – 5 (Very Accurate)
5	How helpful were the explanations of visual content?	1 (Not Helpful) – 5 (Very Helpful)
6	How would you rate the quality of the content summaries?	1 (Poor) – 5 (Excellent)
7	What specific aspects of the text extraction or analysis could be improved? (Optional)	Open-ended (Optional)
8	How intuitive was the chat interface for interacting with the system?	1 (Not Intuitive) – 5 (Very Intuitive)
9	How relevant were the responses to your questions about the uploaded images?	1 (Not Relevant) – 5 (Very Relevant)
10	How would you rate the system’s ability to understand your instructions?	1 (Poor Understanding) – 5 (Excellent Understanding)
11	How would you rate the clarity of the speech output?	1 (Not Clear) – 5 (Very Clear)

No.	Feedback Question	Rating Label / Format
12	How natural did the speech sound to you?	1 (Very Robotic) – 5 (Very Natural)
13	Which voice type did you prefer? (Select one)	Voice A / Voice B / etc.
14	How useful was the ability to adjust speech speed?	1 (Not Useful) – 5 (Very Useful)
15	How helpful was this tool for understanding the content in your educational materials?	1 (Not Helpful) – 5 (Very Helpful)
16	Did the multimodal approach (combining visual, text, and audio) enhance your learning experience?	Yes / No
17	How likely are you to use this tool again for your studies?	1 (Not Likely) – 5 (Very Likely)
18	Were you able to successfully complete all the tasks you attempted with the tool?	Yes / No
19	Please share any other feedback or suggestions for improving HearSee (Optional)	Open-ended (Optional)

Appendix D: Table of Reference and Candidate Dataset for BERTScore Benchmark

Benchmark Prompt: "Describe this image in one detailed sentence."

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
1	This educational slide describes plant growth and development, explaining that growth is typically accompanied by energy-consuming metabolic processes, emphasizing the indeterminate nature of plant growth where plants retain unlimited growth capacity throughout their lives due to the presence of meristems at specific locations in the plant body, and noting that this continuous addition of new cells is referred to as the "open form of growth."	The image is a slide from a presentation about plant growth and development, discussing the indeterminate nature of plant growth, the role of meristems, and the open form of growth.
2	This image displays a neatly organized table with a dark blue header and gray body detailing the team composition for a Google Apps training project, showing four key roles: a Project Manager responsible for overseeing implementation, stakeholder communication, budget management, and timeline adherence; an IT Trainer tasked with conducting Google Apps training and developing training	The image is a table that outlines the roles and responsibilities of team members in a project. The table has three columns: Team Member, Role, and Responsibilities. The roles listed are Project Manager, IT Trainer, Curriculum Specialist, and Technical Support. The

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	materials; a Curriculum Specialist focused on educational content design and teaching methodologies; and Technical Support providing IT assistance, troubleshooting, and system maintenance.	responsibilities for each role include stakeholder communication, budget management, timeline adherence for the Project Manager; conducting Google Apps training, developing training materials, and delivering training sessions for the IT Trainer; curriculum development and teaching methodologies for the Curriculum Specialist; and technical assistance, troubleshooting, and maintenance for the Technical Support.
3	This table displays language performance scores for two AI models (GPT-4o and Qwen2-VL-72B) across eight different languages, with Qwen2-VL-72B generally outperforming GPT-4o in most languages except for Vietnamese and Arabic, where GPT-4o shows slightly better results, and both models achieving their highest scores in	The image is a table comparing the performance of two language models, GPT-4o and Qwen2-VL-72B, across different languages. The table lists the models' scores for each language, with scores ranging from 70.7 to 97.2.

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	Russian (96.8 for GPT-4o and 97.2 for Qwen2-VL-72B) while showing their lowest performance in Vietnamese and Arabic.	
4	This image displays a slide titled "Abiotic and Biotic Stress" that lists eight abiotic stress components affecting plant growth, including sunlight, salt concentration, drought condition, temperatures, soil condition, oxidative agents, cold condition, and metal toxicity.	The image is a slide from a presentation titled "Abiotic and Biotic Stress," which lists various abiotic components that can cause stress to organisms. The abiotic components include sunlight, salt concentration, drought condition, temperatures, soil condition, oxidative agents, cold condition, and metal toxicity.
5	This flowchart illustrates a document digitization and OCR (Optical Character Recognition) process that begins with three input sources - scanned documents, PDF documents, and JPEG images - which are then processed through an input handling module that converts PDFs to images if necessary, followed by a preprocessing module that enhances the original image quality, and	The image depicts a workflow for converting scanned documents, PDF documents, and images into editable text documents. The process begins with input documents, which are then processed by a preprocess module to create images if the input is a PDF. These images are

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	finally through an OCR system that converts the processed image into an editable text document in TXT format.	then fed into an OCR (Optical Character Recognition) module, which extracts text from the images. The resulting text is then saved as an editable document in a TXT format.
6	This diagram illustrates the complete life cycle of a flowering plant, showing the circular progression from seed germination and vegetative growth (depicted by a small green seedling) through floral transition to reproductive growth and fertilization (shown by a mature plant with flowers on the left), with a detailed cross-section of a flower in the center revealing its reproductive parts including the carpel, stamen, petals, and sepals, ultimately leading back to seed formation to complete the cycle.	The image illustrates the life cycle of a flowering plant, depicting the stages from seed germination and vegetative growth, through floral transition, to reproductive growth and fertilization, and finally back to seed production, highlighting the key components of a flower such as the carpel, stamen, sepals, and petals.
7	This image shows a comparison table of three Qwen2-VL multimodal AI models (2B, 7B, and 72B variants), all featuring a 675M parameter vision encoder paired with increasingly larger language models (1.5B,	The image is a table comparing three different models, each with specific features and descriptions. The columns are labeled "Model Name," "Vision Encoder,"

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	7.6B, and 72B parameters respectively), where the smallest 2B model is optimized for efficient on-device performance, the 7B model offers significantly enhanced text recognition and video understanding capabilities across diverse visual tasks, and the largest 72B model provides the most advanced visual reasoning, instruction-following, and decision-making capabilities for handling complex tasks.	"LLM," and "Model Description." The rows list three models: Qwen2-VL-2B, Qwen2-VL-7B, and Qwen2-VL-72B. The Vision Encoder for Qwen2-VL-2B is 675M, for Qwen2-VL-7B it is 765M, and for Qwen2-VL-72B it is 675M.
8	This image is a continuation slide titled "Abiotic and Biotic Stress cont." that lists eight biotic components in blue text including virus, fungal pathogens, bacteria, nematodes, insects, parasitic angiosperms, weeds, and cultivated or native plants, all formatted as bullet points against a white background.	The image is a slide from a presentation titled "Abiotic and Biotic Stress cont." It lists various biotic components that can cause stress to plants, including viruses, fungal pathogens, bacteria, nematodes, insects, parasitic angiosperms, weeds, and cultivated or native plants.
9	This detailed anatomical diagram illustrates a cross-section of a flower showing all its major reproductive and structural components,	The image is a detailed diagram of a flower, labeled with various parts and their functions. The

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	including the colorful petals that attract birds and insects, the male reproductive parts (stamen with anthers that produce pollen and filaments for support), the female reproductive parts (pistil containing the stigma that catches pollen, style that provides a passage for pollen, and ovary where seeds develop), the protective sepals that guard the flower before opening, and the supporting stem that transports water and nutrients to the entire flower structure.	petals are color-coded to attract birds and insects, the anther produces pollen, the filament supports the anther, the sepals protect the flower before it opens, the stigma is sticky to catch pollen, the style is the passage way for pollen, the ovule develops into seeds, the ovary becomes the fruit, and the stem supports the flower and transports water and food.
10	The image shows a project management table with four main tasks for what appears to be a Google Apps training initiative: Budget Allocation (assigned to Project Manager for 2 weeks to research and propose allocation for training resources), Module Development (assigned to IT Trainer for 3 weeks to design and develop specific Google Apps modules for teaching), Curriculum Enhancement (assigned to Curriculum Specialist for 3 weeks to update existing curriculum to incorporate	The image is a table with four columns: Task, Assigned To, Deadline, and Notes. The tasks listed are Budget Allocation, Module Development, Curriculum Enhancement, and Technical Assessment. The Budget Allocation task is assigned to the Project Manager with a deadline of 2 weeks and notes to research and propose an

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	Google Apps), and Technical Assessment (assigned to Technical Support for 1 week to evaluate technical readiness and potential challenges).	allocation for training resources. The Module Development task is assigned to the IT Trainer with a deadline of 3 weeks and notes to design and develop specific Google Apps modules for teaching. The Curriculum Enhancement task is assigned to the Curriculum Specialist with a deadline of 3 weeks and notes to update the existing curriculum to incorporate Google Apps.
11	This image shows two detailed flowcharts comparing acoustic module training approaches: (a) depicts a pre-training and joint training methodology where modules are first optimized independently (shown in the blue box) before joint training with components like prosodic style encoder, acoustic style encoder, pitch extractor, text aligner, and various predictors, while (b) illustrates a multi-adversarial training and inference approach using WavLM that involves separate	The image depicts a complex neural network architecture for speech synthesis, specifically focusing on the integration of acoustic and prosodic modules for generating realistic speech waveforms. The network consists of several key components: an acoustic style encoder, a pitch extractor, an acoustic text encoder, a prosodic text encoder,

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	training processes with phonemes and acoustic modules, including discriminative components and end-to-end training with differentiable upsampling.	a duration predictor, a prosody predictor, a decoder, and a style diffusion denoiser. The pre-training and joint training process involves optimizing the modules inside the blue box, followed by optimizing all components except the pitch extractor, which provides ground truth labels for pitch curves. The duration predictor is trained with only Lbar.
12	This image shows a text slide with a teal header titled "Environmental Factors that Affect Plant Growth" followed by four bullet points explaining how plant growth and distribution are constrained by environmental conditions, including that any suboptimal environmental factor becomes a limiting factor, these limiting factors determine geographical plant distribution patterns, and specifically that only water-adapted plants can survive in desert environments.	The image is a slide from a presentation titled "Environmental Factors that Affect Plant Growth," which discusses how plant growth and distribution are limited by environmental factors, how any one environmental factor that is less than ideal can become a limiting factor in plant growth, and how limiting factors are

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
		responsible for the geography of plant distribution, exemplified by the fact that only plants adapted to limited amounts of water can live in deserts.
13	This image displays a comprehensive text description of system architecture and web development practices, beginning with a definition of logical or system architecture as a method for describing system structure and component interactions, then providing examples like three-tier architecture and system environments such as web development setups, and concluding with five key team goals for website development projects including understanding client needs, creating user-friendly designs, delivering on time and budget, maintaining effective communication, and thoroughly testing before launch.	The image is a text-based description of logical or system architecture, including examples of different types of architectures and team goals for developing a website for a client.
14	The image shows a screenshot of the LLaVA (Large Language and Vision Assistant) web interface, where a user is engaging with an AI	The image shows a screenshot of a user interface for a large language and vision assistant

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	assistant about Claude Monet's famous painting "Le D�jeuner sur l'herbe" (The Luncheon on the Grass), which is displayed both as the main uploaded image and in smaller thumbnails, with the AI responding to questions about appreciating artwork and discussing the colors and shapes visible in the painting, while the interface shows various controls including upvote/downvote buttons, flag option, regenerate function, and clear history button at the bottom.	named LLaVA-Docent, which is designed to help users appreciate and understand artworks by analyzing their colors, shapes, and other visual elements.
15	This image shows a scientific slide about photoperiodism, defining it as the mechanism by which organisms measure seasonal changes in relative day and night length, and lists three categories of flowering responses: long-day plants, short-day plants, and day-neutral plants.	The image is a slide from a presentation about photoperiodism, explaining how organisms measure seasonal changes in relative day and night length, and it specifically discusses flowering responses in plants, categorizing them into long-day plants, short-day plants, and day-neutral plants.
16	This image presents a detailed flowchart outlining the Industrial Training Work Process	The image is an infographic detailing the Industrial Training

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	for an internship period from February 26, 2024, to August 26, 2024, consisting of six sequential steps with specific date ranges: starting with requesting a student application letter via FCSIT LI System (Step 1, January 1-26, 2024), followed by sending the application and transcript to companies (Step 2, January 1 - March 1, 2024), submitting placement offer letters (Step 3, January 12 - March 1, 2024), submitting duty forms and agreements (Step 4, February 26 - March 1, 2024), registering at UNIMAS LI System (Step 5, February 1-15, 2024), and finally registering for the TMF39412 course (Step 6, March 12-15, 2024).	Work Process for a specific period from 26.02.2024 to 26.08.2024. It outlines six steps, each with a corresponding date range and a brief description of the action required. The steps are as follows: 1. Request for "Student's Application for Industrial Training Placement" Letter via FCSIT LI System (01.11.23 - 26.02.24) 2. Send Student's Application for Industrial Training Placement" letter + pre transcript to companies
17	This detailed cross-sectional diagram of a leaf shows the internal anatomy from top to bottom, including the upper epidermis at the surface, followed by the palisade mesophyll tissue with elongated cells containing chloroplasts, then the spongy mesophyll tissue with irregularly shaped cells and air spaces, the lower epidermis at the bottom with guard	The image is a detailed diagram of a cross-section of a leaf, showing various cellular and tissue structures. It includes the upper and lower epiderms, palisade mesophyll tissue, spongy mesophyll tissue, vascular bundles (xylem and

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	cells containing chloroplasts that form stomatal openings, and centrally located vascular bundles composed of xylem and phloem tissues for water and nutrient transport.	phloem), air spaces, and guard cells with chloroplasts. The chloroplasts are indicated within the cells, particularly in the mesophyll tissue, which is responsible for photosynthesis.
18	This image shows a "Risk Assessment" flowchart that defines risk assessment as the process for identifying threats to a given information system and estimating their risk values, with the flowchart detailing the systematic steps including describing the system, identifying threats and vulnerabilities, analyzing security controls, estimating both the likelihood and severity of threats, calculating overall risk, and finally recommending appropriate security controls.	The image is a flowchart titled "Risk Assessment," which outlines the steps involved in assessing the risks to a given Information System (IS). The process begins with describing the system, followed by identifying the threats, vulnerabilities, and security controls. It then involves analyzing the security controls, estimating the likelihoods and severities of threats, and finally estimating the risk and recommending security controls.
19	This image illustrates a system architecture diagram for a "Cat VS Bot WebPage"	The image depicts a system architecture diagram for a chatbot

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	application that includes a user guide component, Node.js server managing user accounts and storing conversation data in MongoDB, with connections to ChatGPT for AI interactions and the ability to deploy models to the cloud with options for both fine-tuned LLaVa and raw LLaVa processing capabilities.	application, with various components and interactions highlighted. It includes a Node.js server at the center, connected to a ConversationDB (MongoDB) database, which stores user accounts and chat history. The system also integrates with a user guide, a chatbot webpage, and a chatbot model (Fine-Tuning LLaVA and Raw LLaVA) on the cloud. Users can interact with the chatbot through the webpage, and the system provides a user guide for assistance.
20	This document discusses Section 5 titled "How to Escape the Method Prison," explaining how the journey from idea to tangible result in software engineering requires finding common ground, and detailing the development of Essence as a unified foundation that emerged from work beginning in 2006 at Ivar Jacobson	The image is a screenshot of a page from a book or document titled "5 How to Escape the Method Prison." The page discusses the process of transitioning from an idea to a tangible result, emphasizing the importance of establishing a

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	International, evolved through the SEMAT community founded in 2010, and became an adopted OMG standard in 2014, with the text including inspirational quotes from Michelangelo about seeing the essence within marble and Antoine de Saint-Exup�ry about achieving perfection through elimination rather than addition.	common ground. It mentions the work on an escape route from the "method prison" starting in 2006 at Ivar Jacobson International (IJI) and the subsequent formation of the SEMAT community in 2009, which led to the development of Essence, a standard common ground in software engineering.
21	This academic page discusses the unification of software development and operation by presenting a model-driven approach that emphasizes agility as an iterative process, featuring a circular development cycle diagram (Figure 1) that shows how software engineers work through phases of requirements gathering, formal verification, implementation, and runtime environment monitoring, with feedback loops connecting operations back to development in an adaptive, continuous improvement process	The image is a page from a book or document, titled "Towards a Unified View of Development and Operation," discussing the integration of agility and formal methods in software engineering. It outlines a process involving software engineers, requirements, environment, and software models, formal verification, iterative design loops, implementation, and a runtime self-adaptive loop, with feedback

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	based on rigorous mathematical foundations and formal verification methods.	loops for operations to development and run-time environment.
22	The image displays a presentation slide on "Security Coding Standards" with a red header, showing "Code 1" which demonstrates a vulnerable C code snippet that creates a fixed-size destination array and copies integers from source to destination using memcpy without bounds checking, followed by an explanation of why this code is non-compliant due to the lack of verification that the number of elements doesn't exceed the buffer size, potentially leading to dangerous buffer overflows and security vulnerabilities.	The image is a slide from a presentation on security coding standards, focusing on a specific code snippet labeled "Code 1." The slide explains that the code creates a fixed-size destination array of integers and copies a specified number of integers from a source array to the destination array using the memcpy function. It highlights that the code is non-compliant because there is no check to ensure that the number of elements to be copied is less than or equal to the buffer size, which can lead to buffer overflows and potential security vulnerabilities.
23	This technical diagram illustrates the architecture of Qwen2-VL, a multimodal AI	The image illustrates the architecture of Qwen2-VL, a

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	system that processes native resolution visual inputs through a Vision Encoder, which handles various media types including images (Picture 1 showing a document interface, Picture 2 displaying a simple interface, and Picture 3 featuring a scenic water and mountain landscape) and Video 1 (showing a pixelated blue scene), before feeding the encoded visual information along with text tokens to the QwenLM Decoder that processes the combined multimodal input to generate responses.	model that processes both text and multimedia data, including images and videos, by encoding them into tokens and then decoding them through a Vision Encoder and QwenLM Decoder, respectively.
24	This image shows Figure 3, a selection of five informational cards from the "User Story Essentials" practice used in software engineering education, featuring cards with yellow and green headers that define key concepts including "User Story" (describing software functionality from an end-user perspective), "Customer Team" (explaining stakeholder roles), "Find User Stories" (identifying valuable software features), and "Story Card" (an index card format for	The image is a screenshot from a presentation or document titled "Escaping Method Prison � On the Road to Real Software Engineering," and it focuses on a section labeled "User Story Essentials." The screenshot shows a selection of five cards that form part of this practice. The cards are titled "User Story Essentials," "Customer Team,"

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	capturing user story details), all designed to help students transition from academic learning to real-world software development practices.	"Find User Stories," "User Story," and "Story Card." Each card provides a brief explanation of its purpose and content. The "User Story Essentials" card explains the importance of capturing what users want from a software system in an informal way. The "Customer Team" card discusses how user stories are written by
25	This image shows instructions for internship students regarding the Industrial Training Portal, explaining that students must update their activities in the log book through the portal using Mozilla Firefox or Google Chrome at https://estudent.unimas.my/PenyeliaanPelajar/, upload hardcopy log books signed by company supervisors monthly (requiring a total of six signatures over six months), keep uploaded files' names short and simple when updating files, and be aware that the system	The image contains a text excerpt about internship activities, specifically detailing the process for students to update their log books through the Industrial Training Portal using Mozilla Firefox or Google Chrome, uploading hardcopy log books signed by the company supervisor monthly, and keeping all uploaded files' names short and simple, with the portal

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	will automatically update to show only the latest file version to accommodate all university students while maintaining small file sizes.	automatically updating the latest file and catering to all university students, suggesting that students should keep each file size as small as possible.
26	This image shows a page from a document titled "Escaping Method Prison - On the Road to Real Software Engineering" that introduces four popular agile scaling methods - SAFe (Scaled Agile Framework), Scaled Professional Scrum (SPS), Disciplined Agile Delivery (DAD), and Large Scale Scrum (LeSS) - with accompanying visual diagrams illustrating each framework's structure and components, while the text explains that despite their popularity and value to organizations, these methods essentially represent overlapping approaches with similar practices but different terminology, creating a problem where organizations often remain unaware of alternative methodologies.	The image is a page from a book or document titled "Typical Methods and Their Problems," which discusses four well-known methods for scaling agile: The Scaled Agile Framework (SAFe), Scaled Professional Scrum (SPS), Disciplined Agile Delivery (DAD), and Large Scale Scrum (LeSS). The page includes a diagram comparing these methods, highlighting their similarities and differences, and a section titled "Escaping Method Prison On the Road to Real Software Engineering."
27	This image shows the detailed meeting minutes from an SE Lab Meeting on 26/10,	The image is a list of topics and sections that should be included

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	outlining a comprehensive software engineering project structure that includes sections on client and team introductions, company and team backgrounds with logos and roles, problem statements and project objectives covering all development phases, system architecture perspectives with 3-tier architecture and sequence diagrams, project scope and system environment specifications, functional modules with infographic descriptions, Scrum methodology implementation, task allocation among team members, team goals (maximum 4), expected deliverables including working systems and various documentation artifacts (proposals, SRS, SDS, STD, user manuals, and forms), project planning with Gantt charts, and APA 7th style references.	in a meeting minutes document for a SE Lab Meeting on 26/10, covering project introduction, client information, team background, problem statement, project objectives, system perspective, project scope, system environment, system functionality, software methodology, task allocation and contribution, team goals, expected results, project planning, and references.
28	This educational diagram illustrates the concept of Multimodal Rotary Position Embedding (M-RoPE) by showing how different temporal positions of a video sequence featuring a Shiba Inu dog are	The image illustrates a multimodal rotary position embedding (M-RoPE) for video features, specifically focusing on a dog, with a particular emphasis

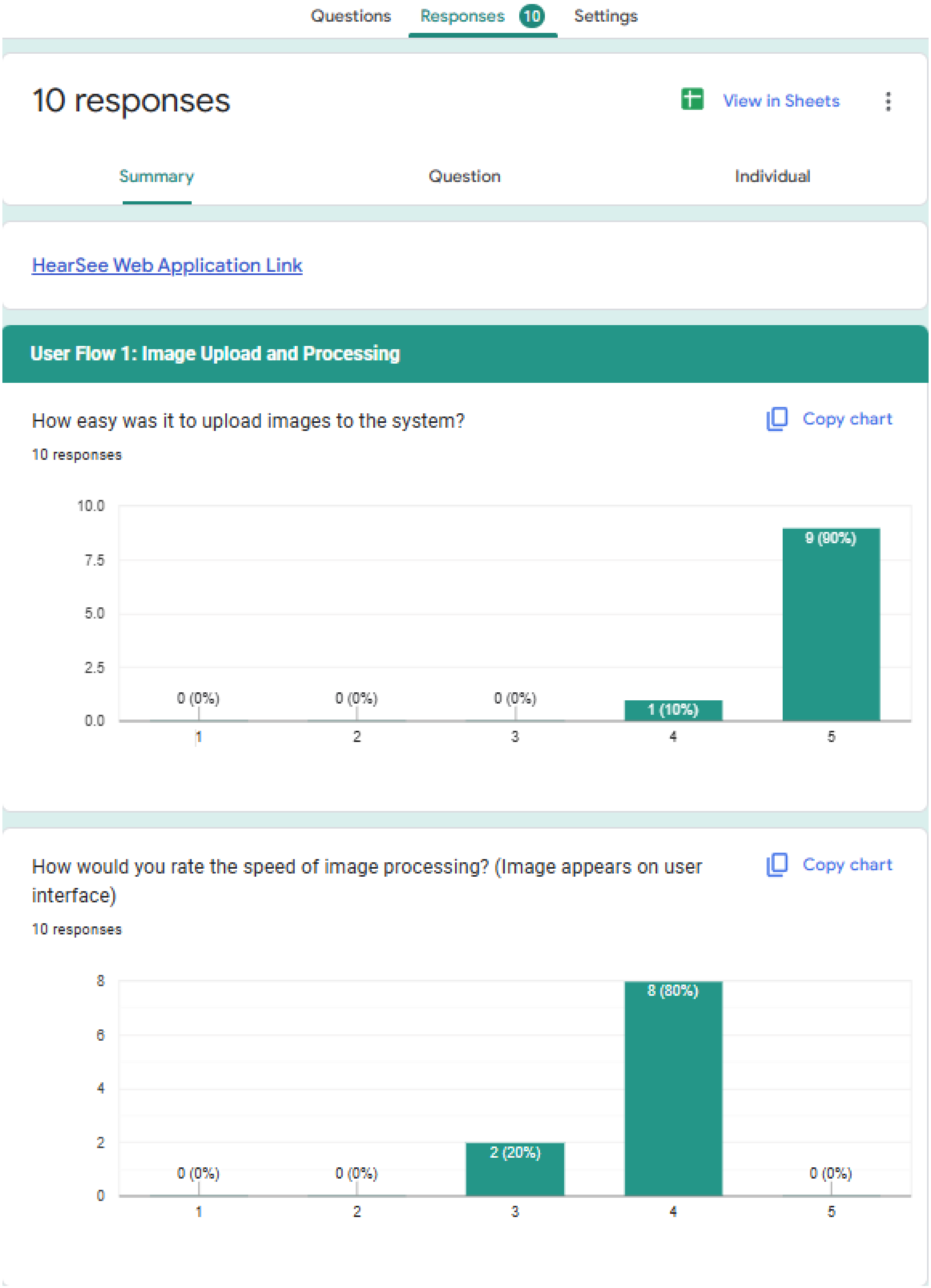
No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
	processed through various rotary position embedding layers (indicated by the blue matrix structures on the right), with the left side displaying multiple frames arranged in a grid pattern that demonstrates the temporal and spatial relationships between video frames.	on a Shiba, and includes position ids, time, height, and width dimensions.
29	This image shows the title page of an academic paper titled "Using Design Thinking for Requirements Engineering in the Context of Digitalization and Digital Transformation: A Motivation and an Experience Report" by authors Angela Carell, Kim Lauenroth, and Dirk Platz, which includes an introduction and motivation section discussing how digitalization and digital transformation are pervasive in the software engineering community, arguing that traditional requirements engineering approaches may be insufficient for addressing the challenges of digital transformation projects and proposing the adoption of design thinking methodologies to better cope with these evolving demands.	The image is a page from a document titled "Using Design Thinking for Requirements Engineering in the Context of Digitalization and Digital Transformation: A Motivation and an Experience Report." The authors are Angela Carell, Kim Lauenroth, and Dirk Platz. The page is divided into sections, with the first section titled "Introduction and Motivation," discussing the omnipresence of digitalization and digital transformation in the software engineering community and the

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
		challenges they pose. The text mentions that many governments consider digitalization as the primary challenge of this decade and that people inside the software engineering community often consider both terms as buzzwords.
30	The Break-Even Point Analysis graph displays a production unit versus revenue chart with two intersecting lines - a blue line representing total revenue that increases proportionally with units sold and a black line showing total costs (fixed and variable expenses) - which meet at approximately 5,000 production units to indicate the break-even point where revenue equals costs, with areas below this intersection marked as "LOSS" and areas above marked as "PROFIT."	The image is a Break-Even Point Analysis graph that illustrates the relationship between production units, revenue, and costs to determine the point at which a business starts generating profit. The X-axis represents production units, while the Y-axis represents revenue. The blue line shows total revenue, increasing proportionally with the number of units sold, while the black line represents total costs, which include both fixed and variable expenses. The Break-Even Point

No.	Claude Sonnet 4 Sentences (Reference)	Qwen2-VL 7B Sentences (Candidate)
		(BEP) occurs where the total revenue and total cost lines intersect, approximately at 5,000 units, indicating the level at which revenue equals costs, resulting in no profit or loss.

Appendix E: Proof of Test Execution and Data Collection

Google Survey Form Image



10 responses

HearSee Web Application Link

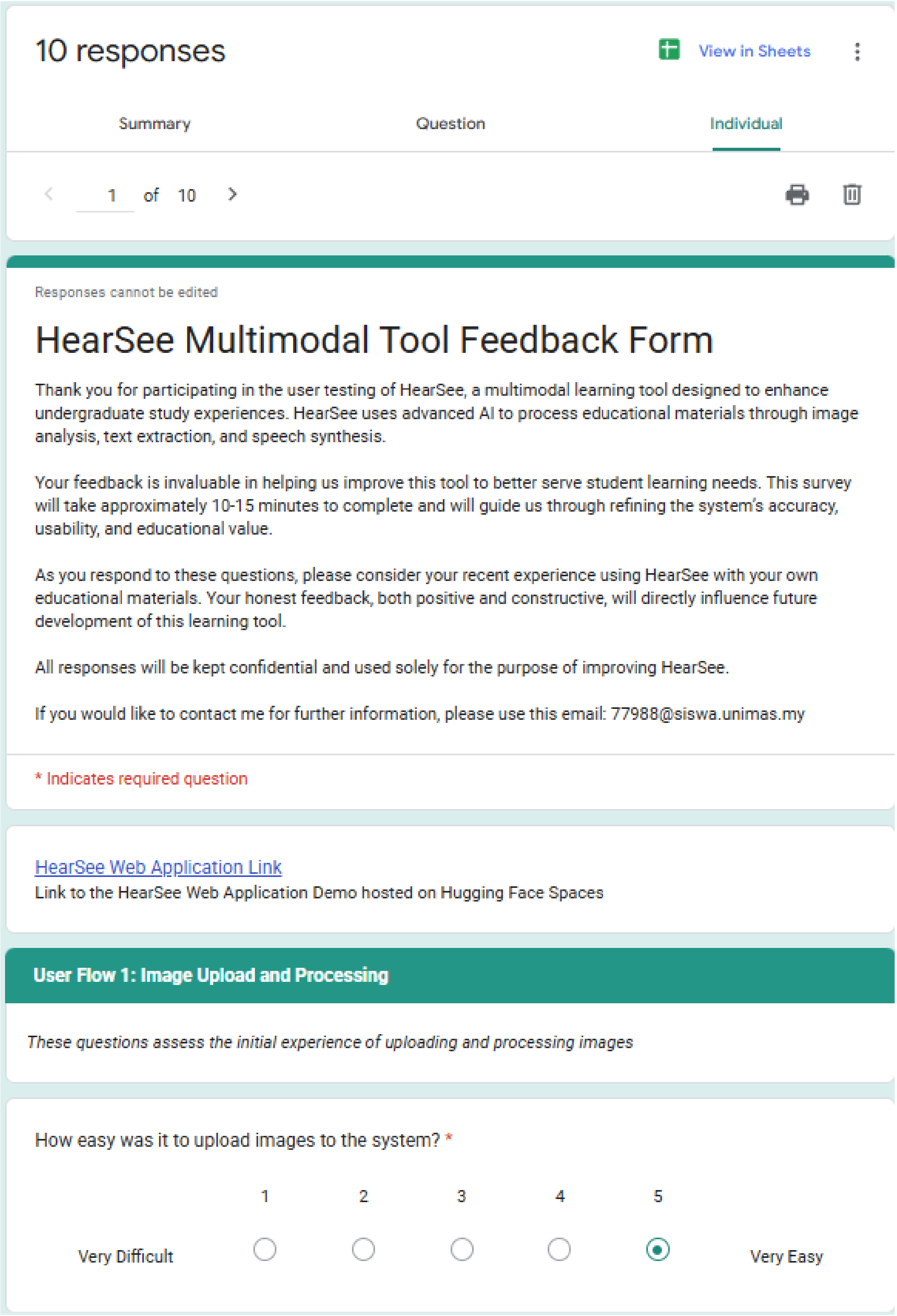
User Flow 1: Image Upload and Processing

How easy was it to upload images to the system?

10 responses

How would you rate the speed of image processing? (Image appears on user interface)

10 responses



HearSee Multimodal Tool Feedback Form

Thank you for participating in the user testing of HearSee, a multimodal learning tool designed to enhance undergraduate study experiences. HearSee uses advanced AI to process educational materials through image analysis, text extraction, and speech synthesis.

Your feedback is invaluable in helping us improve this tool to better serve student learning needs. This survey will take approximately 10-15 minutes to complete and will guide us through refining the system's accuracy, usability, and educational value.

As you respond to these questions, please consider your recent experience using HearSee with your own educational materials. Your honest feedback, both positive and constructive, will directly influence future development of this learning tool.

All responses will be kept confidential and used solely for the purpose of improving HearSee.

If you would like to contact me for further information, please use this email: 77988@siswa.unimas.my

* Indicates required question

HearSee Web Application Link

Link to the HearSee Web Application Demo hosted on Hugging Face Spaces

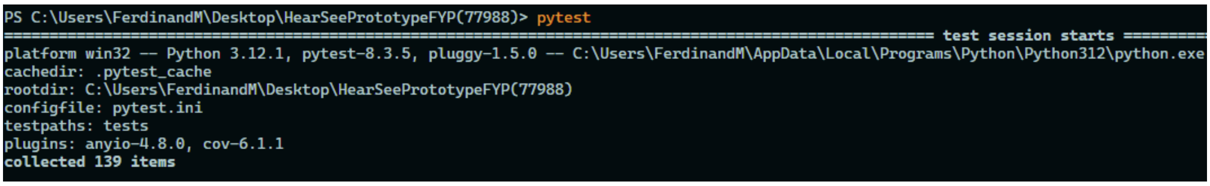
User Flow 1: Image Upload and Processing

These questions assess the initial experience of uploading and processing images

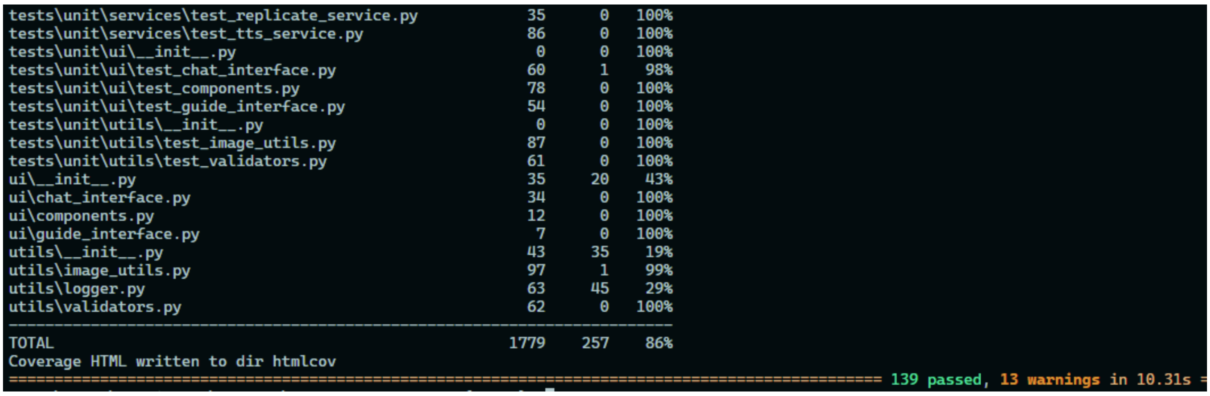
How easy was it to upload images to the system? *

	1	2	3	4	5	
Very Difficult	○	○	○	○	●	Very Easy

Automated Test Suite Execution Image

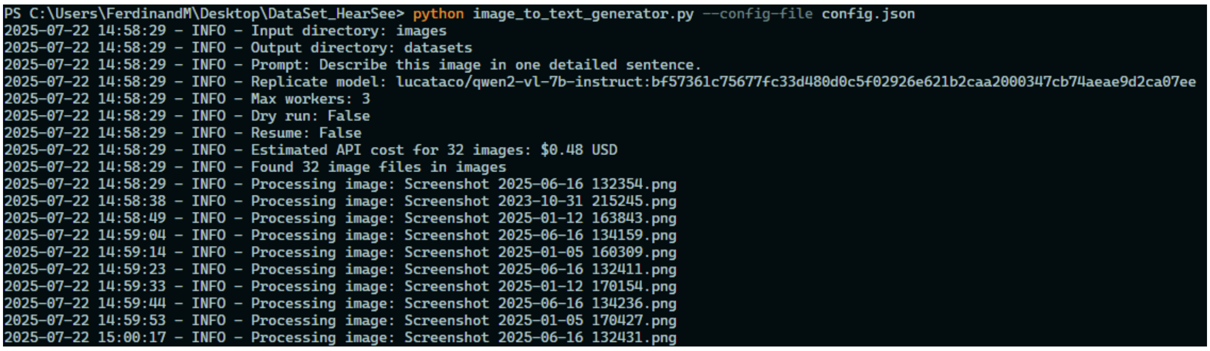


```
PS C:\Users\FerdinandM\Desktop\HearSeePrototypeFYP(77988)> pytest
====================================================================================================== test session starts =========
platform win32 -- Python 3.12.1, pytest-8.3.5, pluggy-1.5.0 -- C:\Users\FerdinandM\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\FerdinandM\Desktop\HearSeePrototypeFYP(77988)
configfile: pytest.ini
testpaths: tests
plugins: anyio-4.8.0, cov-6.1.1
collected 139 items
```

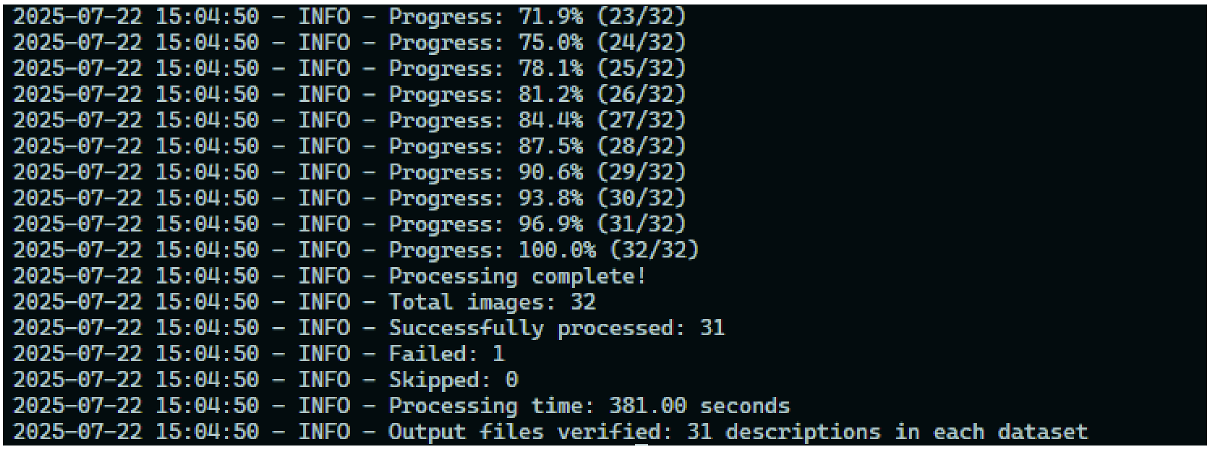


```
tests\unit\services\test_replicate_service.py            35      0   100%
tests\unit\services\test_tts_service.py                  86      0   100%
tests\unit\ui\__init__.py                                 0      0   100%
tests\unit\ui\test_chat_interface.py                     60      1    98%
tests\unit\ui\test_components.py                         78      0   100%
tests\unit\ui\test_guide_interface.py                    54      0   100%
tests\unit\utils\__init__.py                              0      0   100%
tests\unit\utils\test_image_utils.py                     87      0   100%
tests\unit\utils\test_validators.py                      61      0   100%
ui\__init__.py                                           35     20    43%
ui\chat_interface.py                                     34      0   100%
ui\components.py                                         12      0   100%
ui\guide_interface.py                                     7      0   100%
utils\__init__.py                                        43     35    19%
utils\image_utils.py                                     97      1    99%
utils\logger.py                                          63     45    29%
utils\validators.py                                      62      0   100%
------------------------------------------------------------------------
TOTAL                                                  1779    257    86%
Coverage HTML written to dir htmlcov
================================================================================================ 139 passed, 13 warnings in 10.31s =
```

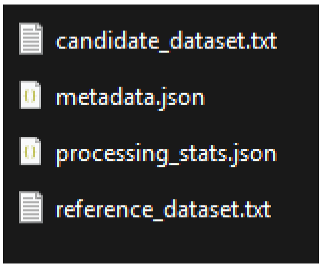
Dataset Generator Tool Results Image

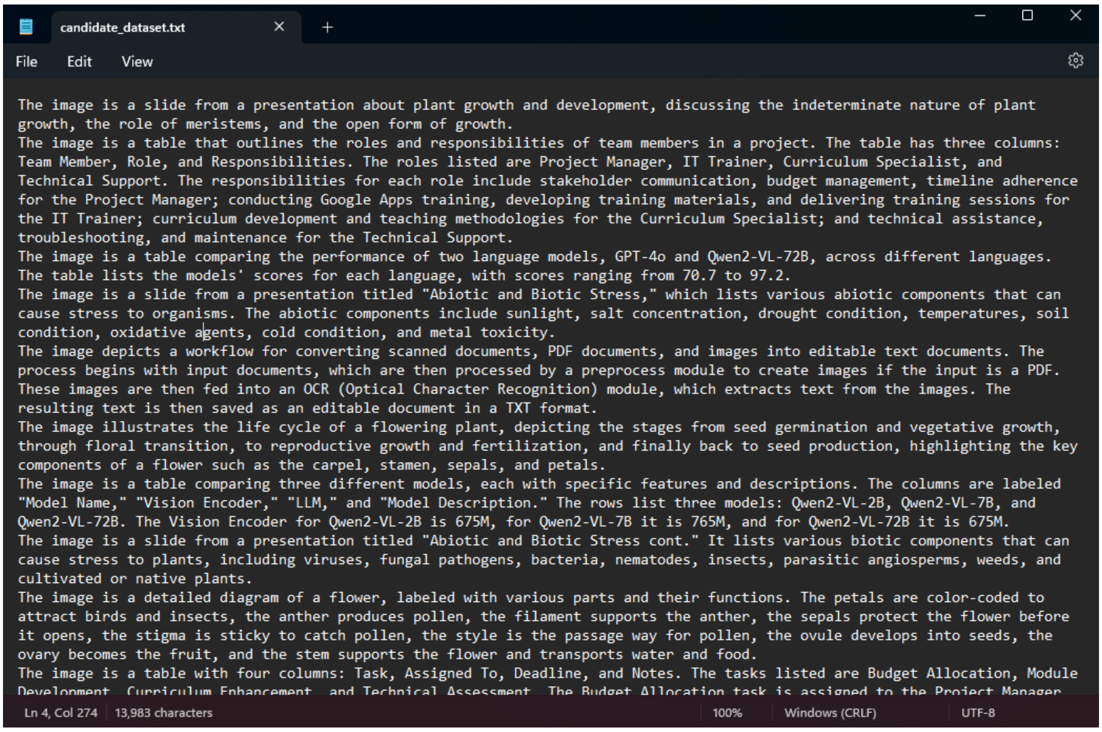


```
PS C:\Users\FerdinandM\Desktop\DataSet_HearSee> python image_to_text_generator.py --config-file config.json
2025-07-22 14:58:29 - INFO - Input directory: images
2025-07-22 14:58:29 - INFO - Output directory: datasets
2025-07-22 14:58:29 - INFO - Prompt: Describe this image in one detailed sentence.
2025-07-22 14:58:29 - INFO - Replicate model: lucataco/qwen2-vl-7b-instruct:bf57361c75677fc33d480d0c5f02926e621b2caa2000347cb74aeae9d2ca07ee
2025-07-22 14:58:29 - INFO - Max workers: 3
2025-07-22 14:58:29 - INFO - Dry run: False
2025-07-22 14:58:29 - INFO - Resume: False
2025-07-22 14:58:29 - INFO - Estimated API cost for 32 images: $0.48 USD
2025-07-22 14:58:29 - INFO - Found 32 image files in images
2025-07-22 14:58:29 - INFO - Processing image: Screenshot 2025-06-16 132354.png
2025-07-22 14:58:38 - INFO - Processing image: Screenshot 2023-10-31 215245.png
2025-07-22 14:58:49 - INFO - Processing image: Screenshot 2025-01-12 163843.png
2025-07-22 14:59:04 - INFO - Processing image: Screenshot 2025-06-16 134159.png
2025-07-22 14:59:14 - INFO - Processing image: Screenshot 2025-01-05 160309.png
2025-07-22 14:59:23 - INFO - Processing image: Screenshot 2025-06-16 132411.png
2025-07-22 14:59:33 - INFO - Processing image: Screenshot 2025-01-12 170154.png
2025-07-22 14:59:44 - INFO - Processing image: Screenshot 2025-06-16 134236.png
2025-07-22 14:59:53 - INFO - Processing image: Screenshot 2025-01-05 170427.png
2025-07-22 15:00:17 - INFO - Processing image: Screenshot 2025-06-16 132431.png
```

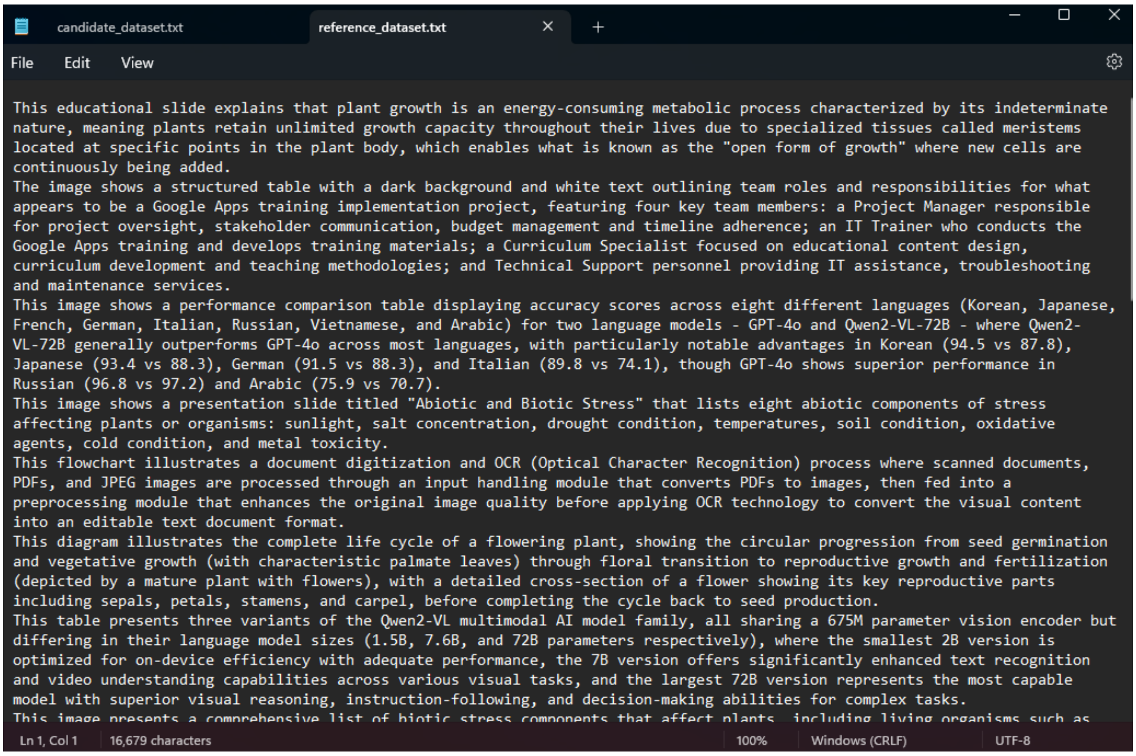


```
2025-07-22 15:04:50 - INFO - Progress: 71.9% (23/32)
2025-07-22 15:04:50 - INFO - Progress: 75.0% (24/32)
2025-07-22 15:04:50 - INFO - Progress: 78.1% (25/32)
2025-07-22 15:04:50 - INFO - Progress: 81.2% (26/32)
2025-07-22 15:04:50 - INFO - Progress: 84.4% (27/32)
2025-07-22 15:04:50 - INFO - Progress: 87.5% (28/32)
2025-07-22 15:04:50 - INFO - Progress: 90.6% (29/32)
2025-07-22 15:04:50 - INFO - Progress: 93.8% (30/32)
2025-07-22 15:04:50 - INFO - Progress: 96.9% (31/32)
2025-07-22 15:04:50 - INFO - Progress: 100.0% (32/32)
2025-07-22 15:04:50 - INFO - Processing complete!
2025-07-22 15:04:50 - INFO - Total images: 32
2025-07-22 15:04:50 - INFO - Successfully processed: 31
2025-07-22 15:04:50 - INFO - Failed: 1
2025-07-22 15:04:50 - INFO - Skipped: 0
2025-07-22 15:04:50 - INFO - Processing time: 381.00 seconds
2025-07-22 15:04:50 - INFO - Output files verified: 31 descriptions in each dataset
```



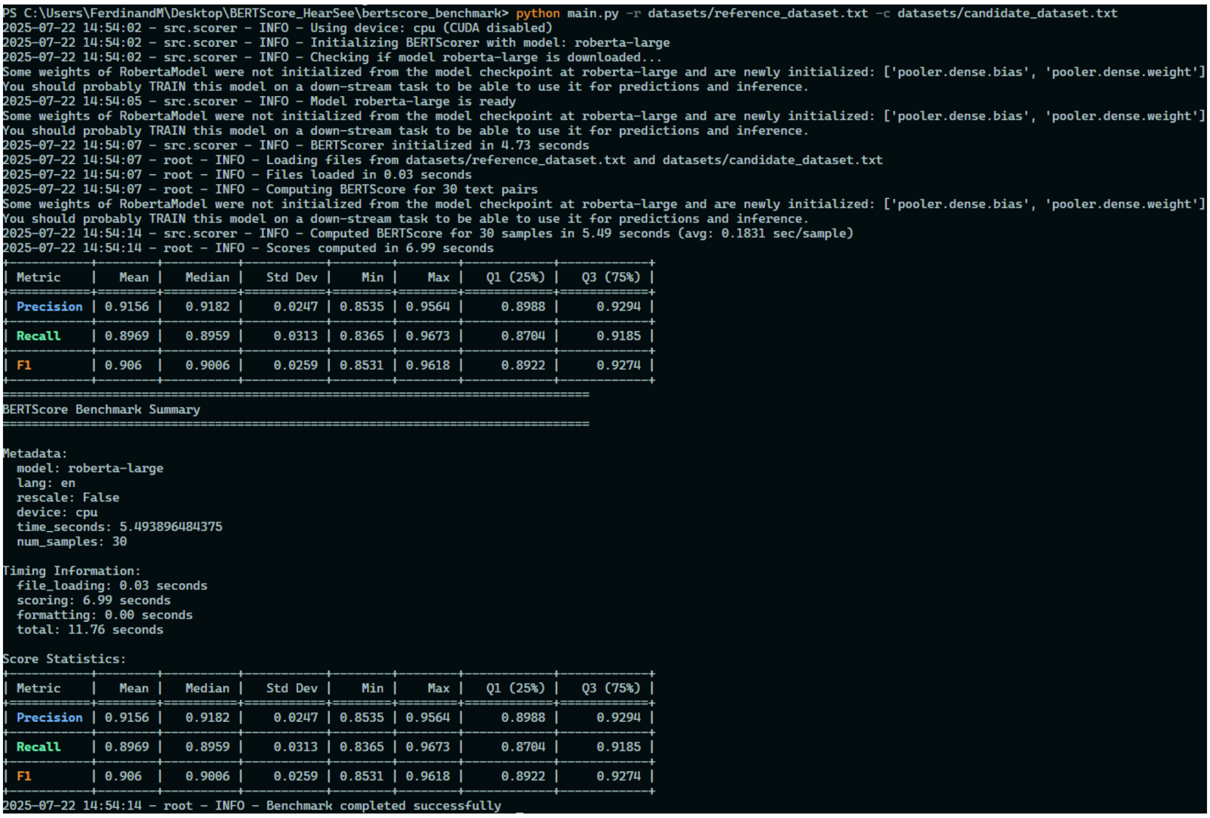


```
The image is a slide from a presentation about plant growth and development, discussing the indeterminate nature of plant
growth, the role of meristems, and the open form of growth.
The image is a table that outlines the roles and responsibilities of team members in a project. The table has three columns:
Team Member, Role, and Responsibilities. The roles listed are Project Manager, IT Trainer, Curriculum Specialist, and
Technical Support. The responsibilities for each role include stakeholder communication, budget management, timeline adherence
for the Project Manager; conducting Google Apps training, developing training materials, and delivering training sessions for
the IT Trainer; curriculum development and teaching methodologies for the Curriculum Specialist; and technical assistance,
troubleshooting, and maintenance for the Technical Support.
The image is a table comparing the performance of two language models, GPT-4o and Qwen2-VL-72B, across different languages.
The table lists the models' scores for each language, with scores ranging from 70.7 to 97.2.
The image is a slide from a presentation titled "Abiotic and Biotic Stress," which lists various abiotic components that can
cause stress to organisms. The abiotic components include sunlight, salt concentration, drought condition, temperatures, soil
condition, oxidative agents, cold condition, and metal toxicity.
The image depicts a workflow for converting scanned documents, PDF documents, and images into editable text documents. The
process begins with input documents, which are then processed by a preprocess module to create images if the input is a PDF.
These images are then fed into an OCR (Optical Character Recognition) module, which extracts text from the images. The
resulting text is then saved as an editable document in a TXT format.
The image illustrates the life cycle of a flowering plant, depicting the stages from seed germination and vegetative growth,
through floral transition, to reproductive growth and fertilization, and finally back to seed production, highlighting the key
components of a flower such as the carpel, stamen, sepals, and petals.
The image is a table comparing three different models, each with specific features and descriptions. The columns are labeled
"Model Name," "Vision Encoder," "LLM," and "Model Description." The rows list three models: Qwen2-VL-2B, Qwen2-VL-7B, and
Qwen2-VL-72B. The Vision Encoder for Qwen2-VL-2B is 675M, for Qwen2-VL-7B it is 765M, and for Qwen2-VL-72B it is 675M.
The image is a slide from a presentation titled "Abiotic and Biotic Stress cont." It lists various biotic components that can
cause stress to plants, including viruses, fungal pathogens, bacteria, nematodes, insects, parasitic angiosperms, weeds, and
cultivated or native plants.
The image is a detailed diagram of a flower, labeled with various parts and their functions. The petals are color-coded to
attract birds and insects, the anther produces pollen, the filament supports the anther, the sepals protect the flower before
it opens, the stigma is sticky to catch pollen, the style is the passage way for pollen, the ovule develops into seeds, the
ovary becomes the fruit, and the stem supports the flower and transports water and food.
The image is a table with four columns: Task, Assigned To, Deadline, and Notes. The tasks listed are Budget Allocation, Module
Development, Curriculum Enhancement, and Technical Assessment. The Budget Allocation task is assigned to the Project Manager
```



```
This educational slide explains that plant growth is an energy-consuming metabolic process characterized by its indeterminate
nature, meaning plants retain unlimited growth capacity throughout their lives due to specialized tissues called meristems
located at specific points in the plant body, which enables what is known as the "open form of growth" where new cells are
continuously being added.
The image shows a structured table with a dark background and white text outlining team roles and responsibilities for what
appears to be a Google Apps training implementation project, featuring four key team members: a Project Manager responsible
for project oversight, stakeholder communication, budget management and timeline adherence; an IT Trainer who conducts the
Google Apps training and develops training materials; a Curriculum Specialist focused on educational content design,
curriculum development and teaching methodologies; and Technical Support personnel providing IT assistance, troubleshooting
and maintenance services.
This image shows a performance comparison table displaying accuracy scores across eight different languages (Korean, Japanese,
French, German, Italian, Russian, Vietnamese, and Arabic) for two language models - GPT-4o and Qwen2-VL-72B - where Qwen2-
VL-72B generally outperforms GPT-4o across most languages, with particularly notable advantages in Korean (94.5 vs 87.8),
Japanese (93.4 vs 88.3), German (91.5 vs 88.3), and Italian (89.8 vs 74.1), though GPT-4o shows superior performance in
Russian (96.8 vs 97.2) and Arabic (75.9 vs 70.7).
This image shows a presentation slide titled "Abiotic and Biotic Stress" that lists eight abiotic components of stress
affecting plants or organisms: sunlight, salt concentration, drought condition, temperatures, soil condition, oxidative
agents, cold condition, and metal toxicity.
This flowchart illustrates a document digitization and OCR (Optical Character Recognition) process where scanned documents,
PDFs, and JPEG images are processed through an input handling module that converts PDFs to images, then fed into a
preprocessing module that enhances the original image quality before applying OCR technology to convert the visual content
into an editable text document format.
This diagram illustrates the complete life cycle of a flowering plant, showing the circular progression from seed germination
and vegetative growth (with characteristic palmate leaves) through floral transition to reproductive growth and fertilization
(depicted by a mature plant with flowers), with a detailed cross-section of a flower showing its key reproductive parts
including sepals, petals, stamens, and carpel, before completing the cycle back to seed production.
This table presents three variants of the Qwen2-VL multimodal AI model family, all sharing a 675M parameter vision encoder but
differing in their language model sizes (1.5B, 7.6B, and 72B parameters respectively), where the smallest 2B version is
optimized for on-device efficiency with adequate performance, the 7B version offers significantly enhanced text recognition
and video understanding capabilities across various visual tasks, and the largest 72B version represents the most capable
model with superior visual reasoning, instruction-following, and decision-making abilities for complex tasks.
This image presents a comprehensive list of biotic stress components that affect plants, including living organisms such as
```

BERTScore Tool Results Image



```
PS C:\Users\FerdinandM\Desktop\BERTScore_HearSee\bertscore_benchmark> python main.py -r datasets/reference_dataset.txt -c datasets/candidate_dataset.txt
2025-07-22 14:54:02 - src.scorer - INFO - Using device: cpu (CUDA disabled)
2025-07-22 14:54:02 - src.scorer - INFO - Initializing BERTScorer with model: roberta-large
2025-07-22 14:54:02 - src.scorer - INFO - Checking if model roberta-large is downloaded...
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
2025-07-22 14:54:05 - src.scorer - INFO - Model roberta-large is ready
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
2025-07-22 14:54:07 - src.scorer - INFO - BERTScorer initialized in 4.73 seconds
2025-07-22 14:54:07 - root - INFO - Loading files from datasets/reference_dataset.txt and datasets/candidate_dataset.txt
2025-07-22 14:54:07 - root - INFO - Files loaded in 0.03 seconds
2025-07-22 14:54:07 - root - INFO - Computing BERTScore for 30 text pairs
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
2025-07-22 14:54:14 - src.scorer - INFO - Computed BERTScore for 30 samples in 5.49 seconds (avg: 0.1831 sec/sample)
2025-07-22 14:54:14 - root - INFO - Scores computed in 6.99 seconds
+-----------+--------+----------+-----------+--------+--------+------------+------------+
| Metric    |   Mean |   Median |   Std Dev |    Min |    Max |   Q1 (25%) |   Q3 (75%) |
+===========+========+==========+===========+========+========+============+============+
| Precision | 0.9156 |   0.9182 |    0.0247 | 0.8535 | 0.9564 |     0.8988 |     0.9294 |
+-----------+--------+----------+-----------+--------+--------+------------+------------+
| Recall    | 0.8969 |   0.8959 |    0.0313 | 0.8365 | 0.9673 |     0.8704 |     0.9185 |
+-----------+--------+----------+-----------+--------+--------+------------+------------+
| F1        | 0.906  |   0.9006 |    0.0259 | 0.8531 | 0.9618 |     0.8922 |     0.9274 |
+-----------+--------+----------+-----------+--------+--------+------------+------------+
================================================================================
BERTScore Benchmark Summary
================================================================================

Metadata:
  model: roberta-large
  lang: en
  rescale: False
  device: cpu
  time_seconds: 5.493896484375
  num_samples: 30

Timing Information:
  file_loading: 0.03 seconds
  scoring: 6.99 seconds
  formatting: 0.00 seconds
  total: 11.76 seconds

Score Statistics:
+-----------+--------+----------+-----------+--------+--------+------------+------------+
| Metric    |   Mean |   Median |   Std Dev |    Min |    Max |   Q1 (25%) |   Q3 (75%) |
+===========+========+==========+===========+========+========+============+============+
| Precision | 0.9156 |   0.9182 |    0.0247 | 0.8535 | 0.9564 |     0.8988 |     0.9294 |
+-----------+--------+----------+-----------+--------+--------+------------+------------+
| Recall    | 0.8969 |   0.8959 |    0.0313 | 0.8365 | 0.9673 |     0.8704 |     0.9185 |
+-----------+--------+----------+-----------+--------+--------+------------+------------+
| F1        | 0.906  |   0.9006 |    0.0259 | 0.8531 | 0.9618 |     0.8922 |     0.9274 |
+-----------+--------+----------+-----------+--------+--------+------------+------------+
2025-07-22 14:54:14 - root - INFO - Benchmark completed successfully
```