



VQAsk: a multimodal Android application to help blind users visualize pictures

Clizia Giorgia Manganaro^a Chiara Giacanelli^b Davide Santoro^c Alessio Palma^d

^amanganaro.2017897@studenti.uniroma1.it ^bgiacanelli.1801145@studenti.uniroma1.it

^csantoro.1843664@studenti.uniroma1.it ^dpalma.1837493@studenti.uniroma1.it

Computer Science Departement, Sapienza University of Rome
Multimodal Interaction Project

Abstract

We developed VQAsk with a strong emphasis on improving accessibility for visually impaired individuals. Our project enables users to interact with their photographs through a dynamic question-and-answer interface. VQAsk is an Android Application, and its core approach involves seamlessly integrating three key modalities: **Speech Interaction:** users can communicate with the application using their voices, asking questions and receiving spoken responses. This natural and intuitive voice interaction ensures effortless engagement for visually impaired users; **Usage of Sight:** while primarily designed for visually impaired users, the application is designed to accommodate users with varying levels of visual ability. Inclusive design elements make it accessible to those with residual vision, allowing them to benefit from the app's features; **Haptic Feedback:** our mobile application provides haptic responses that facilitate navigation and interaction within the application. This enriches the user experience and ensures usability for all, regardless of visual ability.

In addition to these modalities, VQAsk incorporates advanced NLP and Computer Vision techniques to enhance image processing and also to answer all user questions about the image, which is the main goal of this project. Notably, a background removal feature utilizes segmentation models to identify important elements from photographs. This background removal feature enables users to focus on key details and gain a deeper understanding of image content.

We eventually conducted a testing phase in order to collect responses from ten different users' perceptions on the application, which confirmed the importance of this Android Application as a first attempt of using multimodality to enhance visually impaired users' experience.

Contents

1	Introduction	3
2	Tools and Technologies Used	3
2.1	Flutter Packages	4
2.1.1	Image-related Packages:	4
2.1.2	Service-related Packages:	4
2.1.3	Audio-related Packages:	4
2.1.4	UI and User Experience Packages:	4
2.1.5	Miscellaneous Packages:	5
3	Implementation	5
3.1	Requirements Analysis	5
3.2	UML diagrams	6
3.3	Building the Application	10
3.4	Designing Multimodal Interaction	16
3.5	Machine Learning Models Used	17
3.5.1	MiniGPT-4	17
3.5.2	Remove.bg	18
3.5.3	Flutter_tts	18
4	Testing and Evaluation	20
5	Conclusions	24

1 Introduction

As the years go by, multimodal research is advancing exponentially, making many important tasks emerge, such as **Visual Question Answering (VQA)**. VQA is a computer vision task where a system is given a text-based question about an image, and it must infer the answer. The broader idea of this problem is to design systems that can understand the contents of an image similar to how humans do, and communicate effectively about that image in natural language. This is a challenging task since it requires image-based models and natural language models to interact with and complement each other [9]. In fact, while compared to other vision-language tasks such as image captioning, text-to-image retrieval, VQA is more challenging because: (1) the questions are not predetermined. In other tasks, the question to be answered is fixed and so the operations required to answer it only the image changes. (2) The supporting visual information is very high dimensional. (3) VQA necessitates solving many computer vision sub tasks (such as object detection, activity recognition and scene classification) [6]. On the other hand, Visual Question Answering can be included in a vast amount of potential applications, such as helping visually impaired persons through their searches online and communicating by pictures, improving experiences at museums by allowing observers to directly ask questions they interested in, or in what's have been called "visual dialogue", that aims to give natural language to robots.

With our project, our aim was to implement a system to help visually impaired (and possibly also blind) people visualize pictures through an application that uses **speech interaction** through **trigger words** and **spontaneous speech** for the questions, and **haptic feedbacks** (as well as the normal touch-screen to type and observe). We built an Android Application with the following characteristics: it can be used entirely through speech and the users can input images from their gallery or can capture pictures through the app. Then, they can ask whatever question about the uploaded picture and the application will respond. Users can also decide to crop their image manually or to adopt Machine Learning in order to automatically detect important features of the image.

2 Tools and Technologies Used

This project is built using Flutter, an open-source framework developed by Google. Flutter is an excellent choice for creating mobile applications that need to run seamlessly on both Android and iOS platforms. It offers a wide range of tools and capabilities that make it a preferred framework for cross-platform development.

In the project's configuration, the chosen environment specifies that it should be compatible with SDK versions greater than or equal to 3.0.6 but less than 4.0.0. This ensures that the project remains compatible with the latest stable releases of Flutter.

For Android development, this project targets Android API version 34, which allows it to take advantage of the latest Android features and optimizations.



Figure 1: Flutter Icon



Figure 2: Android Icon

2.1 Flutter Packages

Now, let's delve into the dependencies of this project, which provide various functionalities and features:

2.1.1 Image-related Packages:

- **cupertino_icons** (version 1.0.2): This package includes icons with a Cupertino-style design, perfect for achieving a consistent and native look on iOS devices.
- **image_picker** (version 1.0.1): This package simplifies the process of selecting images from the device's gallery.
- **image_cropper** (version 4.0.1): It allows users to crop images, enhancing image editing capabilities within our app.
- **remove_bg** (version 0.0.3): This package is employed to eliminate the background from an image. Its purpose is to isolate the object within the image.

2.1.2 Service-related Packages:

- **replicate** (version 1.0.0): The "Replicate Dart Client" is a Dart package for interacting with Replicate.com's APIs. It offers prediction management, real-time updates, dynamic input support, and easy configuration. Users can set their API key for authentication. In our case, it was used to leverage the machine learning model MiniGPT-4 employed for visual question answering.
- **http** (version 0.13.6): It's utilized for making HTTP requests, allowing our app to interact with web services and APIs.
- **path_provider** (version 2.0.15): This package offers a platform-independent way to access the device's file system, enabling our app to manage and interact with files.

2.1.3 Audio-related Packages:

- **porcupine_flutter** (version 2.2.1): This package is most likely utilized for voice or audio processing within our application. Within our app, it plays a crucial role in recognizing various wake words, contributing to making the app more inclusive for users with disabilities.
- **flutter_tts** (version 3.7.0): It adds text-to-speech capabilities to our app, allowing it to convert text into spoken words, which can be beneficial for accessibility or interactive features.
- **speech_to_text** (version 6.3.0): It facilitates the conversion of spoken words into text, enabling speech recognition features in the app.
- **audioplayers** (version 5.1.0): This package provides functionality for playing audio file.

2.1.4 UI and User Experience Packages:

- **sn_progress_dialog** (version 1.1.3): It provides a progress dialog that can be used to inform users about loading or processing tasks, enhancing the user experience.
- **dotted_border** (version 2.0.0+3): This package offers a widget for creating dotted borders, which can be used for visual design and customization.
- **flutter_vibrate** (version 1.3.0): It enables the app to trigger device vibrations, which can be useful for notifications or haptic feedback.

2.1.5 Miscellaneous Packages:

- **flutter_dotenv** (version 5.1.0): This package is used for managing environment variables and configuration settings, which is crucial for maintaining security and flexibility.
- **camera** (version 0.10.5+4): This package grants access to the device’s camera, facilitating functionalities such as capturing photos or scanning QR codes.
- **shared_preferences** (version 2.2.1): It allows for simple data persistence, making it easier to store and retrieve user preferences and app settings across sessions.

In summary, our project’s overarching objective is to create a versatile mobile application that encompasses a diverse array of functionalities, such as networking, image and audio processing, user interface design, and device interaction, all aimed at achieving a specific goal or outcome.

3 Implementation

3.1 Requirements Analysis

We started our work by collecting and specifying the requisites of our application. Requirements are split into two types: **Functional** and **Non-functional** requirements.

Functional Requirements These are the requirements that the end user specifically demands as basic facilities that the system should offer. They basically specify “*What should the software system do?*”. They are divided into user-functional requirements and system-functional requirements. Below is the description of the functional requirements, defined through the MoSCoW table¹, which provides “Must have” and “Should have” voices for each of the requirement.

User-Functional Requirements		
ID	DESCRIPTION	MoSCoW
0	The user must be able to load pictures either from the gallery or the smartphone’s camera	Must have
1	The user must be able to activate the system with his voice	Must have
2	The user must be able to ask questions about his pictures using the touch-screen keyboard or by voice	Must have
3	The user must be able to read and to listen the answer given by the system	Must have
4	The user must be able to receive instructions about the usage of the system	Should have
5	The user must be able to erase his questions and change it	Must have
6	The user must be able to listen to the inserted question and the given answer whenever he needs	Should have
7	The user must be able to edit his pictures to highlight important portions of them	Should have
8	The user must be able to edit his pictures also by voice	Should have

¹The **MoSCoW** method is a prioritization technique used in management, business analysis, project management, and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement; it is also known as MoSCoW prioritization or MoSCoW analysis [12].

System-Functional Requirements		
ID	DESCRIPTION	MoSCoW
0	The system must provide touch-screen interaction, voice interaction and haptic feedbacks	Must have
1	The system must show clearly all the elements that allow the user to listen and type questions and answers	Must have
2	The system must allow the user to select pictures from the gallery and to shoot them from the camera	Must have
3	The sytem must show if there are errors in the typing of the question	Should have
4	The system must include an accurate speech recognition module to convert spoken language into text and should support various accents, dialects, and languages for robust speech processing	Must have
5	The system should be able to associate spoken or written questions with the relevant features extracted from the images	Must have
6	The system should allow the user to edit the images	Should have

Non-functional requirements These are the quality constraints that the system must satisfy according to the project contract. These are also called *non-behavioral requirements* and usually deal with issues like portability, scalability, security, performance, maintenance and reusability. Below is the MoSCoW table which describes them in our application.

Non-Functional Requirements		
ID	DESCRIPTION	MoSCoW
0	The application has to work in Android v.11 systems (or higher)	Must have
1	The images have to be processed securely	Must have
2	The application must be able to process .jpg and .PNG formats	Must have
3	The application must be able to ask the permission to the user to activate the microphone and the camera	Should have
4	The application must quit listening if the user stops pronouncing words	Should have
5	The application must provide an answer in a reasonable time	Must have

3.2 UML diagrams

In order to better understand and document the informations about the system we made three UML diagrams: one Use Case Diagram and two Sequence Diagrams. A Use Case Diagram describes the high-level functions and scope of a system and it also identifies the interactions between the system and its actors. It consists of use cases and actors that describe what the system does and how the actors use it.

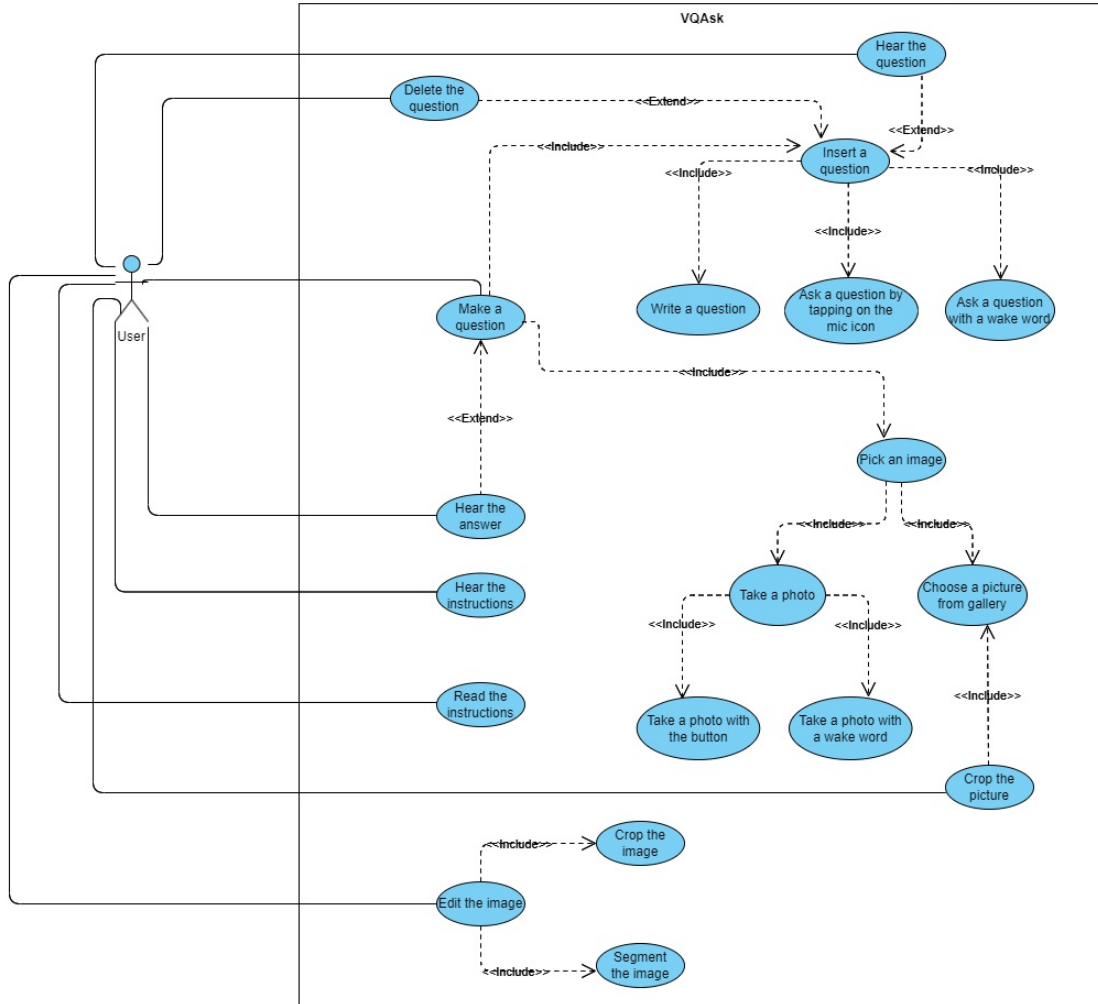


Figure 3: Use Case Diagram

In Figure 3 we can see that the user, which is the main actor, is connected with the ellipses that represent the use cases of the system. A generic use case may "extend" the behaviour of another use case adding optional actions that can be performed or may "include" other use cases during the execution to embed their functionalities.

The Use Case Diagram is very useful but it doesn't describe how the system operates internally. For this reason, in order to document the inner workings of the system, we decided to also represent the Sequence Diagrams. A Sequence Diagram is a diagram used to represent a scenario, i.e. a specific description of a behaviour or sequence of actions occurring in a system. Scenarios are an important part of the design and documentation of systems and software, as they help to understand how the system works in specific situations. The arrows and lines in the Sequence Diagram represent method calls, messages and interactions between objects in the system during the execution of the scenario. Since it is possible to interact with the system in different ways, we decided to represent the "standard" interaction mode and the "alternative" interaction mode in two different Sequence Diagrams. The standard mode of interaction with the application is mainly based on visual interaction, while the alternative mode is based on voice interaction, which allows the use of the application also by visually impaired people. Despite this logical division, which is useful to distinguish and better understand the different modes of interaction with the system, it is important to emphasise that it is possible to complete each subtask of the application in an alternative way, depending on the preferences or needs of the users. We decided to model the actors external to the system as services to which our application

connects via the Internet through tokens provided by the services themselves. We also included other application functionalities such as Text to Speech or Speech to Text within the VQAsk actor, since the use of these services is permitted by locally installed libraries that work even without Internet access.

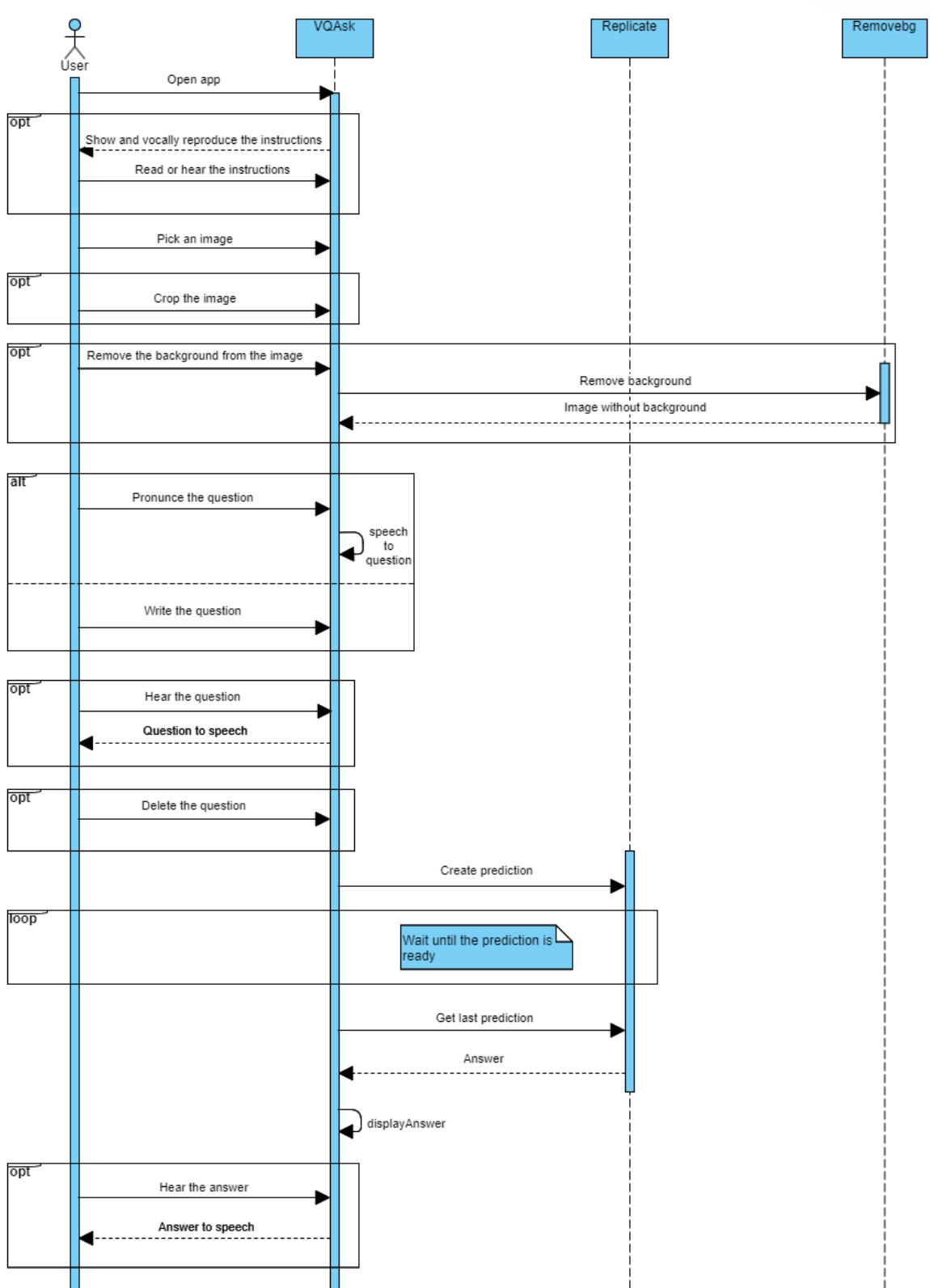


Figure 4: Sequence Diagram of standard interaction

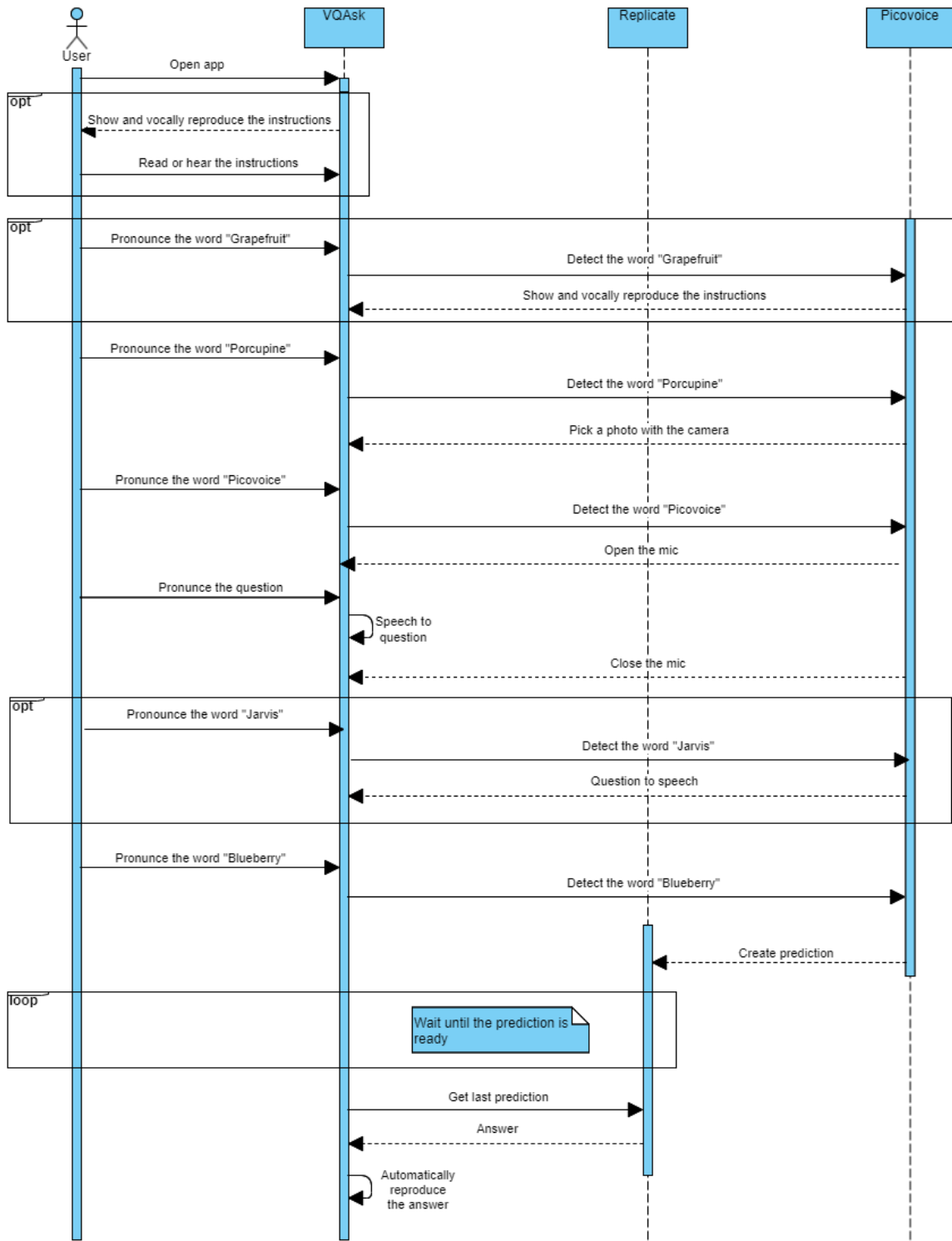


Figure 5: Sequence Diagram of alternative interaction

The Figure 4 shows the Sequence Diagram that represents the "standard" sequence of actions that occur in the system. We have four main actors: the User that interacts with the application, VQAsk that is the application itself, Replicate, an external service that allows to use the Minigpt model through API in order to get the answers to the questions we submit and Removebg, an online tool accessible via API to remove the background from an image.

Regarding the "alternative" usage scenario shown in Figure 5, we introduced another actor that

is Picovoice. It's a platform that, among the many services it offers, allows to make the Wake Word Detection through which specific spoken word can be associated to specific commands within the application. Like other services such as Replicate and Removebg, it allows to access to its functionality via a token.

3.3 Building the Application

The Android Application is composed of two main screens: the *"Homepage"* and the *"Editable photograph"* section. The Homepage is minimal and the elements are positioned in a gerarchical way: first of all, there's the AppBar, which contains the name of the Application on the left and two icons on the right: the "info" icon describes how the application works (Figure 7), and the "crop" is the link to the other page (which will be described later in this section).

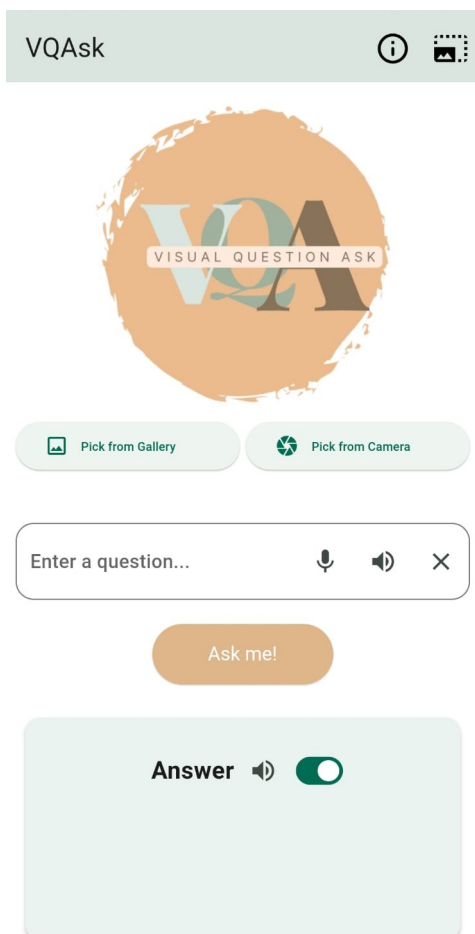


Figure 6: Homepage of the Application

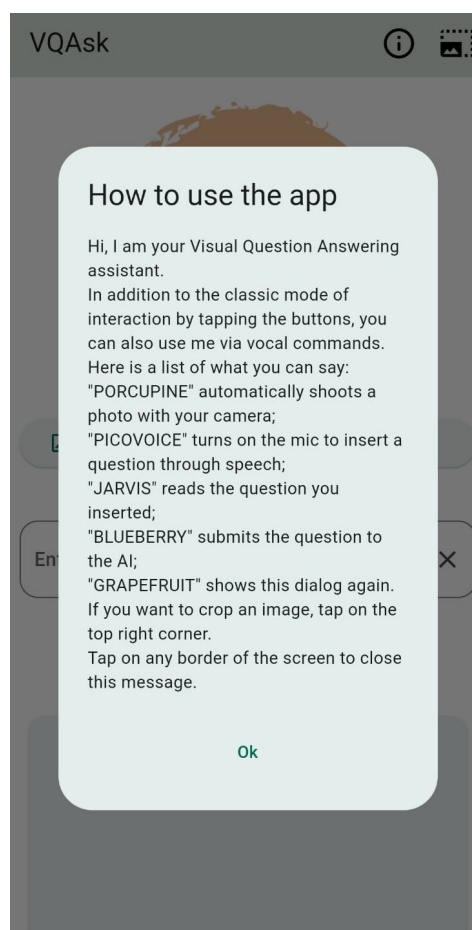


Figure 7: Info Section

The Info Section is also synthesized to speech in order to allow visually impaired users to gain better knowledge about the different usages of the application. As we can read from the image, the app can be entirely used by exploiting the following **semaphoric words**:

- **PORCUPINE**: if the user pronounces this word, the application automatically captures an image from the external camera of the mobile phone, and loads it in order to be used by the app;
- **PICOVOICE**: if the user pronounces this word, the following sentences will be recorded in order to be used as the question related to the image. The user's speech can be **continuous** and the speaking style **spontaneous**;
- **JARVIS**: if the user pronounces this word, the app will output the inserted question by voice;
- **BLUEBERRY**: if the user pronounces this word, the question will be submitted in order to be answered;
- **GRAPEFRUIT**: if the user pronounces this word, the Info Section will be opened and read again by the Vocal Assistant:

Being it a not so trivial task, we didn't implement a trigger word that allows the user also to select an image from the gallery, and it would be an interesting task to deepen in future developments.

A user can select an image from the gallery by touching the appropriate button on the left, or capture a photograph instantly through the internal and external cameras of the phone.

Once an image has been loaded, the following task is to ask the system a question about it. In particular, we managed three error situations in which the app can't proceed with the answer: when the user clicks on the "Ask me!" button or pronounces **BLUEBERRY** without typing anything (Fig. 8), when the cursor is at the beginning and the question isn't already typed on the input form (Fig. 9), or when the question is too short (Fig. 10). When handling the *empty question error*, we decided to insert an **haptic feedback** along with the simple visible modal to enhance multimodality: when the error pops up, the phone vibrates, indicating that something went wrong. An user can also insert

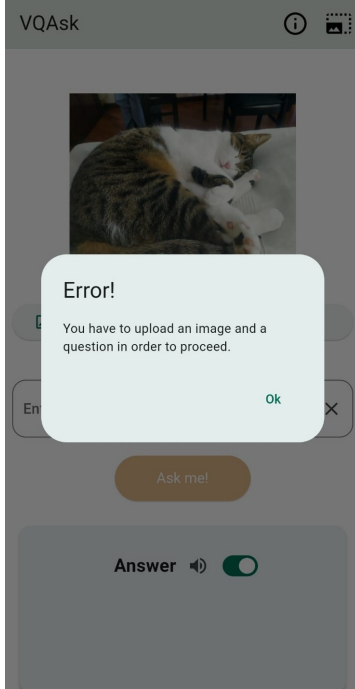


Figure 8: First Error

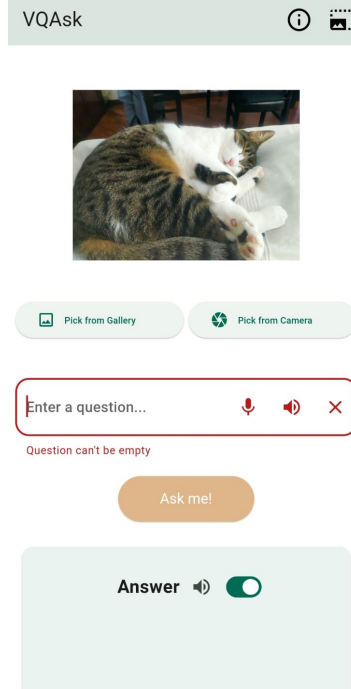


Figure 9: Second Error

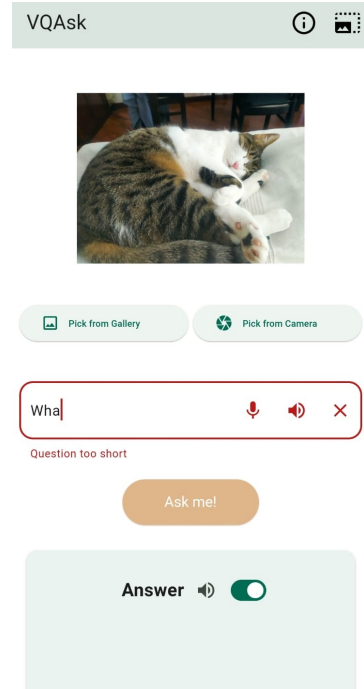


Figure 10: Third Error

a question by recording it clicking to the *microphone* icon, positioned in the input form on the right. They can also decide to erase the question by clicking to the *X* icon. After a proper question has been inserted, the user can proceed with asking the answer to the app. Before doing it, they can decide to

listen to it by clicking to the appropriate icon (the *volume-up* icon).

When a user wants to receive the answer to the question, they can click on the "Ask Me!" button, or pronounce the proper semaphoric word. When submitted, a question is given as input along with the image to the **MiniGPT-4 Model** and the app stands on the "Thinking stage" (Fig. 11), in which a circle progress bar is shown and the Vocal Assistant outputs the sentence "I'm thinking" to the user. The answer is displayed in a green Card, in which there are two important buttons: a *volume-up* button to listen to the answer again whenever needed, and a *switch button*. When the switch button is enabled (as in 12) the app automatically reads the answer out loud when the model outputs it.

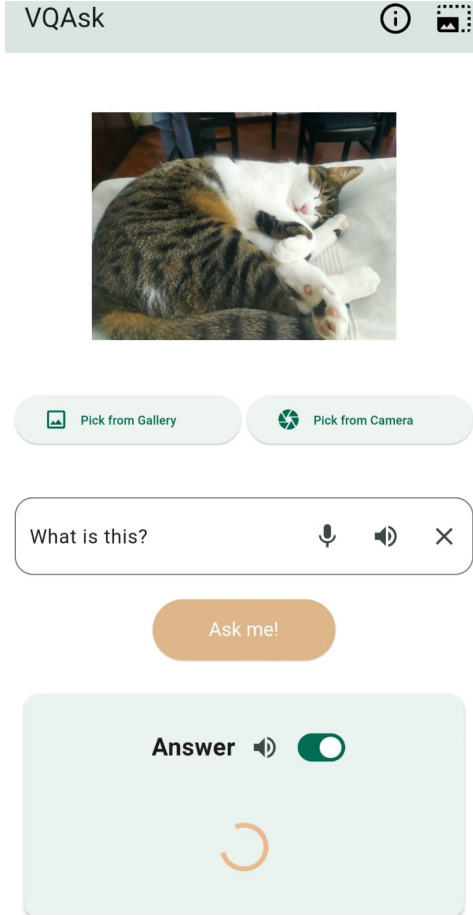


Figure 11: Thinking Stage

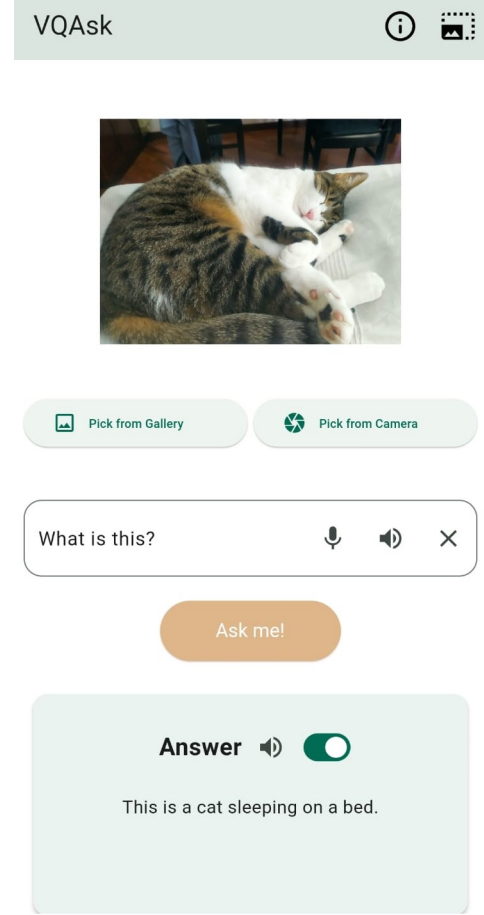


Figure 12: Answer given

The other section is called *Editable photograph*. Through this page, users can crop their images however they want, and can also segment them automatically in order to find the most important elements in a picture. We called the latter "magic cropping", and allows visually impaired users to locate the essential objects within a photograph and ask specific questions about them. Even in this page, we inserted an information icon, which explains the function of the TERMINATOR semaphoric word, that triggers the activation of the segmentation procedure within the image (Fig. 14). The modal will be automatically read by the Visual Assistant to gain immediate understanding of how the page works.

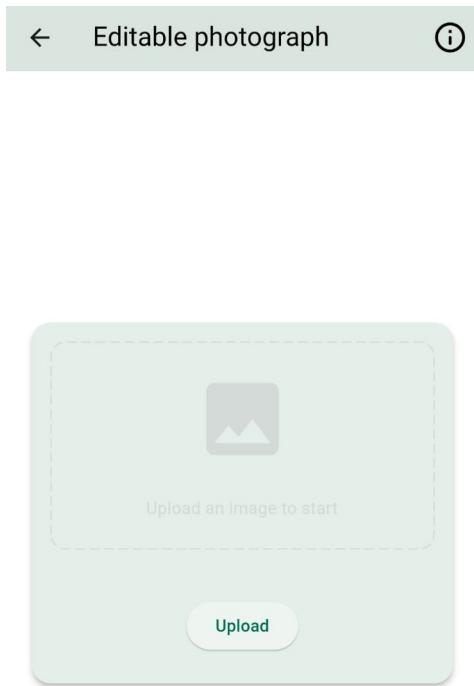


Figure 13: Editable Photograph Page

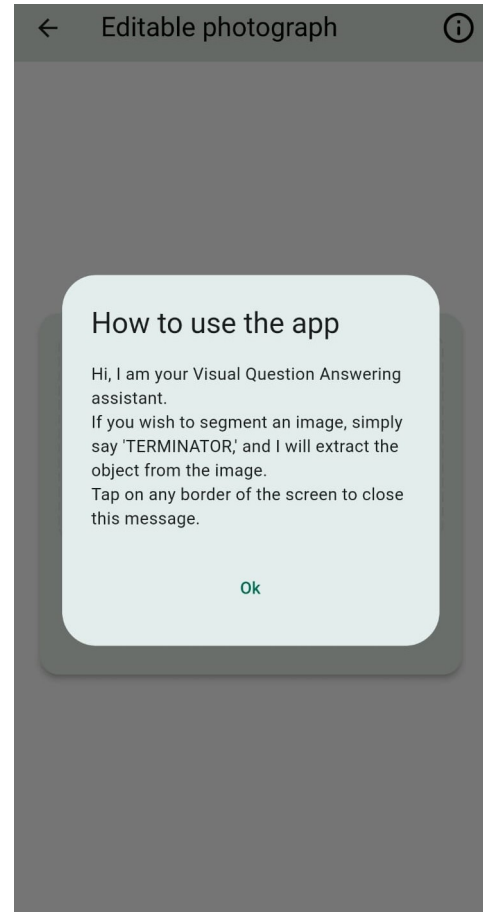


Figure 14: Info Modal for the Editable Page

Users can select photographs from their gallery, and after the image is loaded, four buttons will be available under it (Figure 18) that respectively (from left to right) allow the user to:

- **Apply changes** and load the image into the main page in order to ask questions and receive answers about it;
- **Trash** the image and return to the home of Editable Photograph as in 13;
- **Crop** the image. Once the Crop button is pressed, users will be able to crop, rotate and scale the choosen image. When they are done, they can apply the changes by pressing the confirm button at the top right corner (Figure 20) and the user will be re-directed to the Editable Photograph page with the edited image (Figure 17); We have to specify that, for a visually impaired user, this section wouldn't be so feasible, because it needs the attention of the user who has to select the right portion of the image;

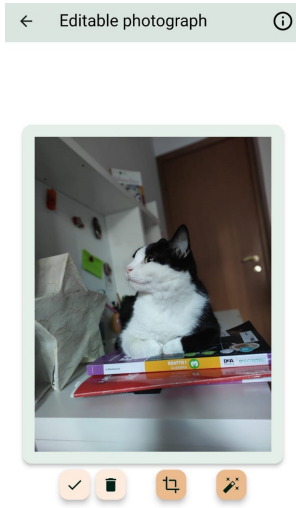


Figure 15: Image Insertion

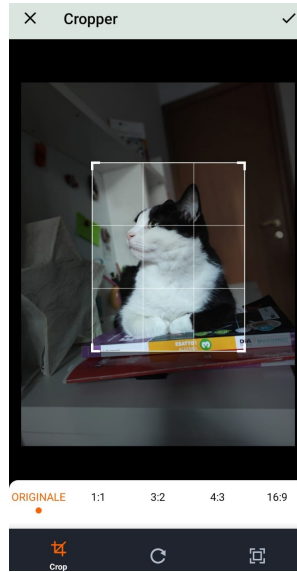


Figure 16: Manual Cropping

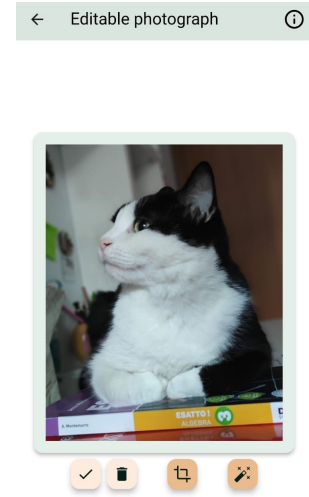


Figure 17: Cropped Image

- **Apply segmentation** to the image. Segmentation can only be applied once per editing. We introduced this feature to break the image down into meaningful components, and automatically select the most important one. Through this functionality, visually-impaired users have the possibility to identify specific objects within their photographs, and ask peculiar questions about them. We integrated **speech recognition** with the trigger word *TERMINATOR*, that automatically applies segmentation to the selected image.



Figure 18: Applied Segmentation

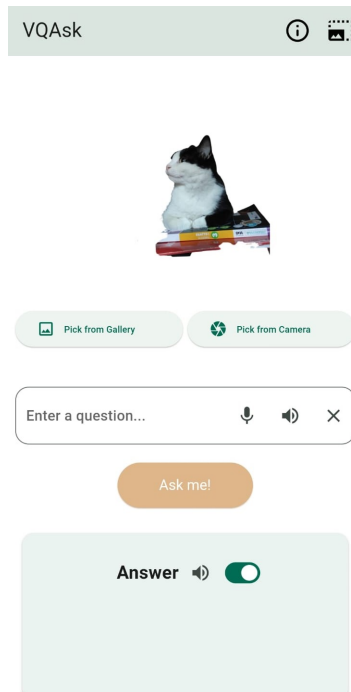


Figure 19: Segmented Input Image to VQAsk

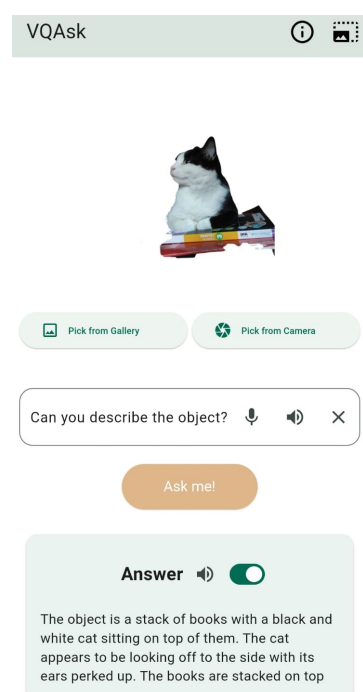














Figure 20: Answer to the segmented image

Below is the table of the icons with their functionalities used in the Homepage and the Editable Photograph Page.

HOMEPAGE FLUTTER ICONS TABLE

	ICON	FUNCTION
Icons.info_outline_rounded		To show the Info Section
Icons.image_outlined		To pick Image from Gallery
Icons.camera		To capture Image from Camera
Icons.mic		To record the question with the mic
Icons.volume_up		To listen to the question and the answer
Icons.clear		To erase the question inserted
Switch Object		To automatically listen to the answer

EDITABLE PHOTOGRAPH FLUTTER ICONS TABLE

	ICON	FUNCTION
Icons.info_outline_rounded		To show the Info Section
Icons.check		To bring the cropped Image to the Homepage
Icons.delete		To remove the inserted image
Icons.crop		To open the Crop Section
Icons.auto_fix_high		To apply the automatic segmentation

3.4 Designing Multimodal Interaction

Our VQask application was created to provide a variety of interaction modalities that can accommodate users with diverse abilities and preferences. These interaction methods are designed to provide an inclusive, accessible and multimodal user experience. In this section we will delve deeper into each interaction modalities, outlining their descriptions, functions and purposes.

1. Voice Interaction:

Description: Users can easily communicate with the application by using their voice to activate features and ask questions about images. Our Voice Interaction system is a **speaker independent system**, in which the user can perform a **continuous speech** when asking a question; the speech can be **spontaneous**, after the pronunciation of the **semaphoric words**.

Functionality: Users have the ability to ask questions verbally and receive vocal responses from the application thanks to VQA’s Machine Learning model that allows it to process the question and provide answers.

Purpose: This natural and intuitive voice interaction is particularly beneficial for visually impaired users who may experience difficulty using touchscreens or reading text. It ensures that everyone can interact with the application without any problems.

2. Haptic Feedback:

Description: Haptic Feedback plays a fundamental role in our multimodal approach designed to provide feedback on how the app is functioning.

Functionality: : The application uses signals and haptic responses via device vibrations when an user mistakenly asks for an answer without a question, simplifying interaction within the application.

Purpose: Haptic feedback enriches the user experience and guarantees usability for all users, irrespective of their visual ability.

3. Visual Interaction:

Description: The application has been carefully designed to accommodate users with varying levels of visual ability, with a focus on a minimal interface and a hierarchy of buttons.

Functionality: Users can leverage remaining vision to navigate the app’s interface and access its feature set.

Purpose: Through the incorporation of inclusive design elements, the application becomes accessible to those with residual vision, allowing them to enjoy its features alongside visually impaired users.

Furthermore, to offer users the freedom to choose whether they want to receive responses through vocal interaction or not, we have incorporated a toggle switch within the interface. This switch empowers users to decide whether they prefer to listen to the app’s responses or not. These designed interaction modalities collectively cultivate a user-centric and inclusive environment within the VQAsk Android application. By doing so, we ensure that users of all abilities can engage with its features effectively and comfortably. This unified approach to interaction design underscores our commitment to inclusivity and equitable access for all users.

3.5 Machine Learning Models Used

3.5.1 MiniGPT-4

The model used to answer our visual questions is MiniGPT-4 [14], a large language model. In the realm of artificial intelligence and natural language processing, the emergence of Large Language Models (LLMs) represents a paradigm shift. LLMs are, at their core, deep neural network architectures with billions of parameters, mainly based on Transformer architectures [11] and meticulously designed to process and generate human language with unparalleled competence. These models leverage colossal computational resources and extensive textual corpora to learn and represent the intricate nuances of language. The scale of LLMs not only challenges the limits of available hardware but also redefines the boundaries of what AI can achieve in the domain of NLP. They exhibit an extraordinary proficiency in a diverse range of linguistic tasks, encompassing machine translation, text generation, question-answering, sentiment analysis, and much more [8, 13, 1]. These models are not confined to traditional monomodal text processing; they have evolved into multimodal powerhouses, fusing text with other data modalities, such as images or audio. This convergence has given rise to Vision-Language Models (VLMs), which have the ability to comprehend, generate, and manipulate both textual and visual information seamlessly, introducing multimodality inside neural networks! VLMs offer a holistic approach to understanding and generating content in the context of the visual world. These models excel at tasks that demand cross-modal comprehension, such as image captioning, object recognition, content generation from visual inputs and even more complex challenges like visual question answering, and so they foster applications that span a multitude of domains [4]. Furthermore, VLMs are instrumental in addressing real-world problems where a fusion of linguistic and visual understanding is essential. They can diagnose issues depicted in images and offer corresponding solutions, craft narratives or creative content inspired by visual stimuli, and even extract structured information from images to fuel knowledge bases and recommendation systems.

The very recent GPT-4 [7], a large-scale multimodal model, has been introduced with demonstrating many impressive capabilities. For example, it can produce very detailed and accurate image descriptions, explain unusual visual phenomena and even construct websites based on handwritten text instructions. Although GPT-4 has exhibited remarkable capabilities the methods behind its exceptional abilities are still a mystery, since it is closed-source, and moreover its APIs are fee-based, so using it for our project wasn't a bearable choice. For this reasons, we decided to use MiniGPT-4, which is a pretrained open-source model that utilizes Vicuna [2], which is built upon LLaMA [10], as the language decoder. It's reported to achieve 90% of ChatGPT's quality as per GPT-4's evaluation metrics. In terms of visual perception, they employ the same pretrained vision component of BLIP-2 [5] that consists of a ViT from [3] and a Q-Former. MiniGPT-4 adds a single projection layer to align the encoded visual features with the Vicuna language model and freezes all the other vision and language components, as shown in image 21.

Even if MiniGPT-4 processes numerous advanced vision-language capabilities, it currently still faces several limitations like all models today. The main ones are:

- **Language hallucination:** as MiniGPT-4 is built upon LLMs, it inherits LLM's limitations like unreliable reasoning ability and hallucinating nonexistent knowledge. This issue might be alleviated by training the model with more high-quality, aligned image-text pairs, or aligning with more advanced LLMs in the future;
- **Inadequate perception capacities:** MiniGPT-4's visual perception remains limited. It may struggle to recognize detailed textual information from images, and differentiate spatial localization. This limitation may stem from several factors like a lack of sufficient aligned image-text data containing adequate information (such as spatial localization and optical character annotations), the fact that the pretrained visual encoder may lose some essential features, the fact that training only one projection layer might not provide enough capacity to learn extensive visual-text alignment.

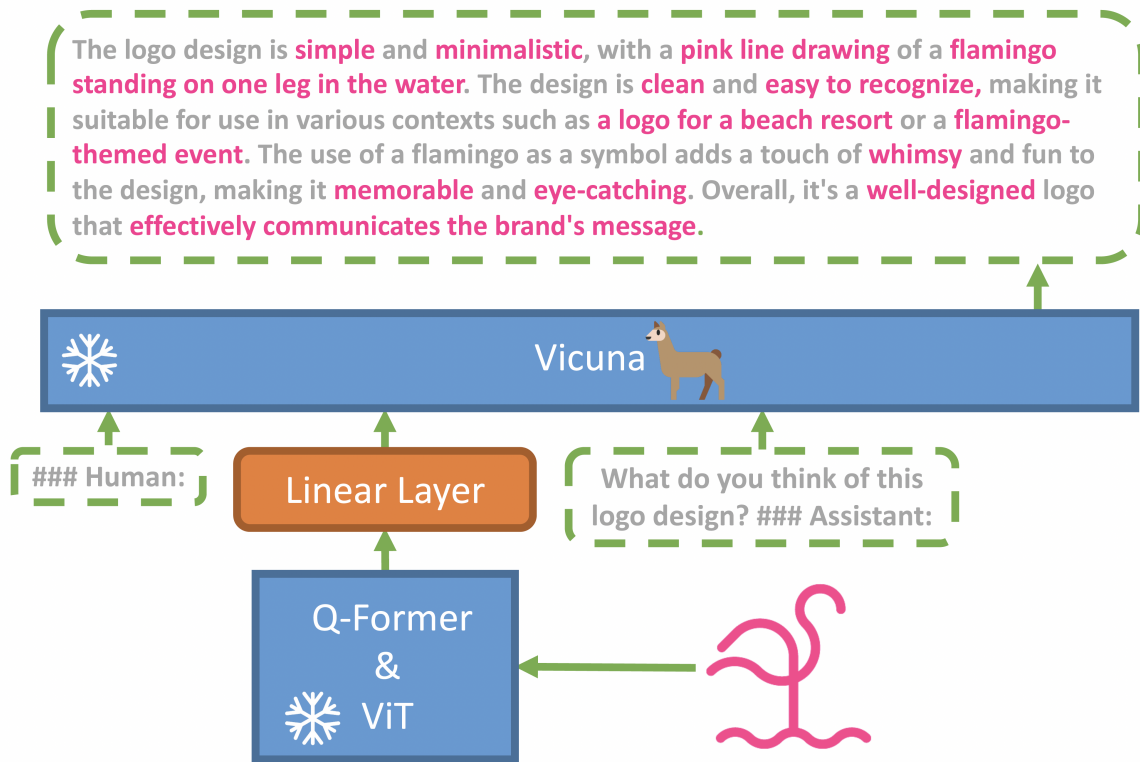


Figure 21: The architecture of MiniGPT-4, which only requires training the linear projection layer to align the visual features with the Vicuna model.

3.5.2 Remove.bg

In order to segment objects within images and enhance the application's performance, we have utilized *remove.bg* APIs². Remove.bg is a web-based service designed to eliminate the background from any photo. It employs closed-source advanced AI technology to identify foreground layers and separate them from the background. To enhance the outcomes, they have implemented various additional algorithms, such as improving fine details and preventing color contamination.

We have ensured the security of the service, as images are securely transferred using SSL/TLS encryption. They are used exclusively for background removal and subsequently offered for download. No sharing with third parties, publication, or any other use is conducted with these images.

3.5.3 Flutter_tts

*flutter_tts*³ is a popular Flutter plugin that allows developers to integrate Text-to-Speech (TTS) functionality into their Flutter applications. This plugin provides an easy-to-use interface for synthesizing speech from text and is supported on iOS, Android, Web, and macOS.

We used this library to synthesize speech from text in our Application whenever the Vocal Assistant is involved. Some of the features provided by *flutter_tts* are:

- `speak`: this function takes a string as parameter, and returns a vocal reproduction of it;
- `stop`: it instructs the text-to-speech engine to cease generating speech and to clear any buffers or resources associated with the synthesis;
- `setLanguage`: this function is used specify the language in which the text should be synthesized into speech;
- `setSpeechRate`: it's used to control the rate of speech, allowing for faster or slower speech output;

²<https://www.remove.bg/it/tools-api>

³https://pub.dev/packages/flutter_tts

- `setVolume`: this function is used to adjust the volume of the speech output;
- `setPitch`: this function is used to control the tonal quality of the speech output.

```

1 import 'package:flutter_tts/flutter_tts.dart';
2
3 FlutterTts fluttertts = FlutterTts();
4
5 void textToSpeech(String text) async {
6   await fluttertts.setLanguage("en-US");
7   await fluttertts.setVolume(1);
8   await fluttertts.setSpeechRate(0.5);
9   await fluttertts.setPitch(1);
10  await fluttertts.speak(text);
11 }

```

Above you can see our `fluttertts` configurations for the application.

The *flutter_tts* plugin itself is primarily a wrapper around the native text-to-speech engines provided by the Android Operating System (in our case). These engines utilize various algorithms and techniques for text-to-speech synthesis, and even AI to process and generate audio (that's the reason why we found it appropriate to insert this in the section of the Machine Learning Models Used). The most common workflow of such algorithms is the following:

1. **Text Analysis**: this involves tokenization and part-of-speech tagging. Tokenization consists in breaking the input text into smaller units, such as words or phonemes, for further processing, and the latter identifies the grammatical parts of speech (e.g., nouns, verbs, adjectives), which helps to improve prosody and intonation in speech synthesis;
2. **Phonetic Analysis**: to identify phonemes in a word. This is crucial to determine the accurate pronunciation.
3. **Prosody Generation**: to determine the pitch (intonation) and duration of each phoneme, word, or phrase to give natural-sounding speech, along with stress and intonation patterns.
4. **Waveform Synthesis**: this step is made of three substeps: concatenative synthesis, which is the pre-recording of a large database of natural speech then concatenated into small units (e.g., phonemes or diphones) to form new utterance; formant synthesis, which is the modeling of the resonance frequencies of the vocal tract to control audio signal generation; finally there's the articulatory synthesis, which tries to model human speech production in full 3D (simulating the movement of the articulators (e.g., tongue, lips));
5. **Linguistic Rules and Context Processing**: accounting for how speech sounds change based on their neighboring sounds and the rules of the language (to handle special cases in pronunciation);
6. **Voice Modelling**: capturing characteristics like pitch, tone, and quality that define a particular speaker's voice.
7. **Pitch, rate and Volume Control**: adjusting the pitch, speed and loudness of the synthesized speech to match the desired tone;
8. **Error handling and Recovery**: implementing mechanisms to handle cases where the synthesis engine encounters challenges or is unable to accurately render certain text.

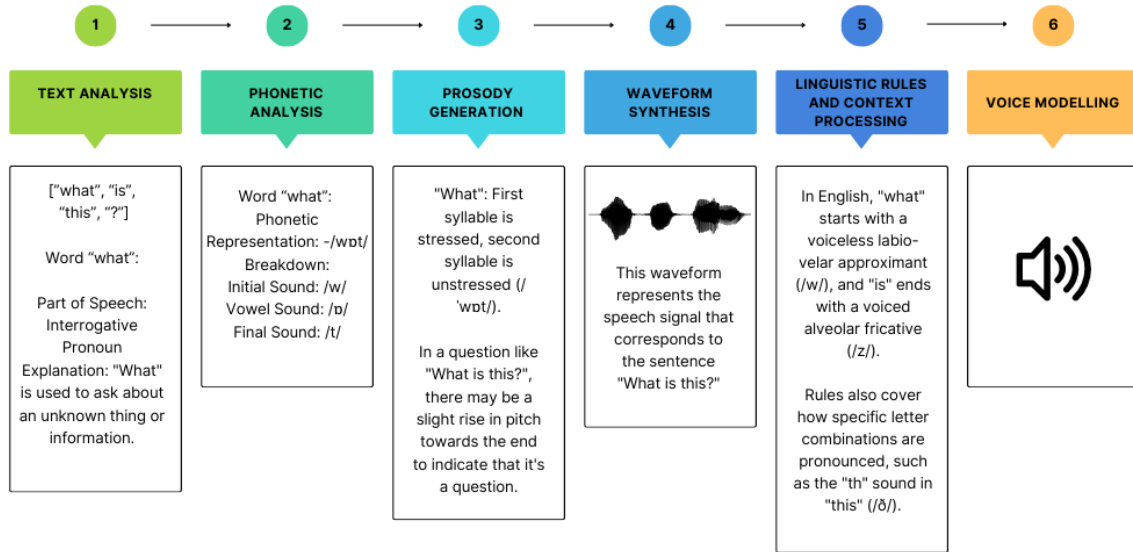


Figure 22: Example Flowchart of the TTS framework on the question "What is this?"

4 Testing and Evaluation

In order to evaluate the usability of our application, we decided to perform some tests. The tests were carried out on a sample of 10 users whom we asked to do a series of pre-established tasks. We performed the tests with a *Think Aloud*⁴ methodology: during the tests execution we were careful to observe whether the actions we asked the users to perform were intuitive and whether they encountered any bugs or difficulties in the general use of the app.

We did not have the opportunity to test the application on visually impaired people, however we asked users to interact via all possible modes so that we could simulate different use cases of the app.

At the end of the test, we asked users to fill in a short questionnaire to analyse the interaction modes they preferred, to evaluate their experience and to communicate any suggestions or improvements. It is available at [this](#) link. Below we can see some results of the questionnaire:

⁴In a **thinking aloud** test, you ask test participants to use the system while continuously thinking out loud — that is, simply verbalizing their thoughts as they move through the user interface.

Per inserire la domanda hai preferito:

10 risposte

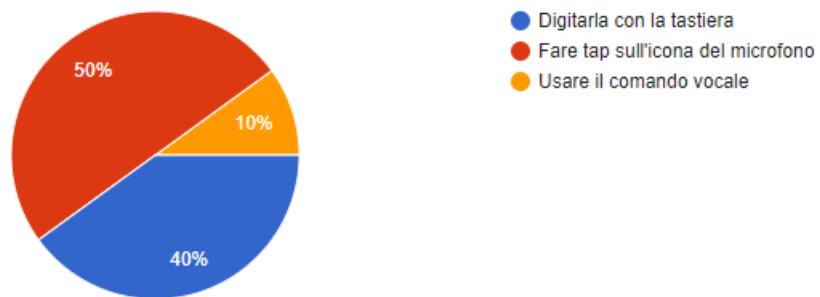


Figure 23: Users preferences in entering the question

Per inviare la domanda al sistema hai preferito:

10 risposte

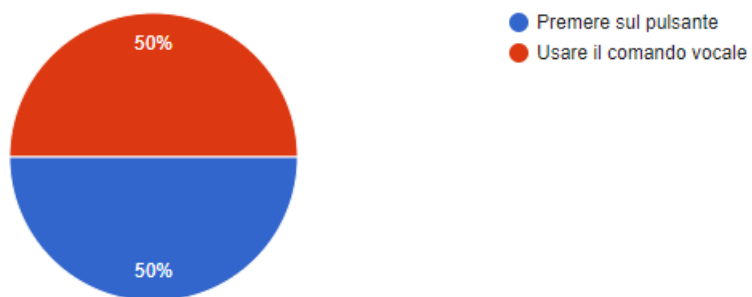


Figure 24: Users preferences in submitting the question

Per ascoltare la risposta hai preferito:

10 risposte

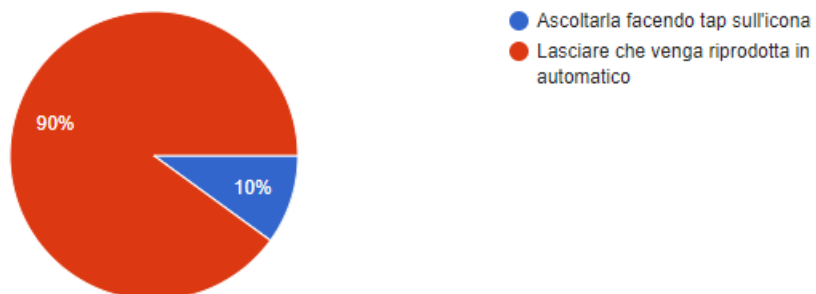


Figure 25: Users preferences in hearing the answer

Pensi che saresti in grado di usare l'applicazione se fossi ipovedente (quindi usando solo l'interazione vocale)?

10 risposte

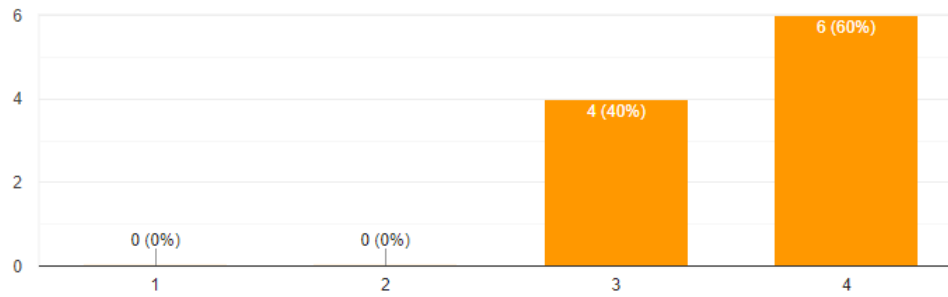


Figure 26: Rating of possible app usage by impaired people

Come valuti l'esperienza generale di utilizzo dell'app:

10 risposte

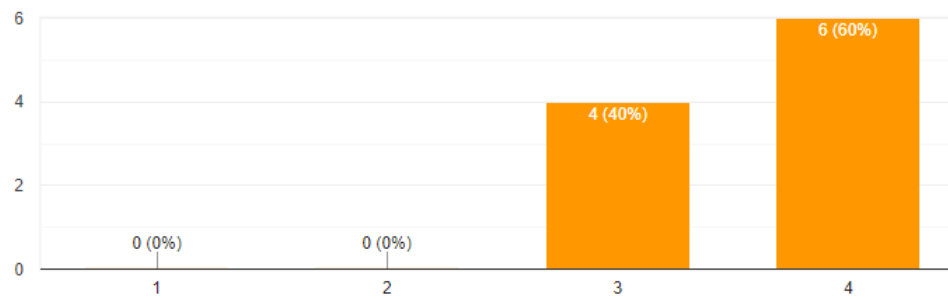


Figure 27: Rating of general app usage experience by the users

Quanto pensi che questa applicazione possa essere utile in contesti reali?

10 risposte

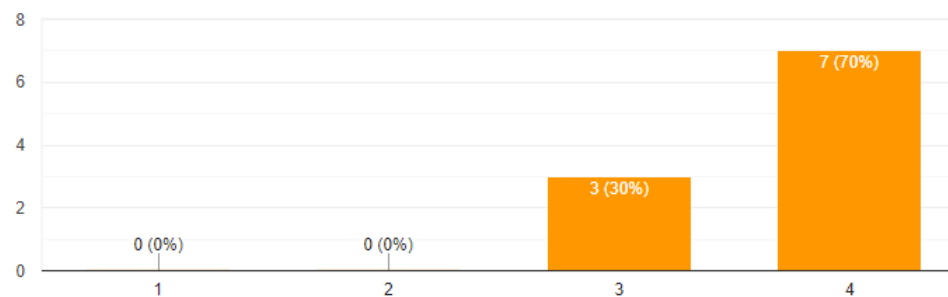


Figure 28: Rating of app usage in real contexts

The results of the questionnaire showed that more than half of the people preferred to interact vocally with the application to enter the question rather than write it down (Figure 23). Moreover, as shown in Figure 24, half of the people prefer to submit the question via vocal command and almost all people prefer to let the answer play automatically, rather than tap on the icon to achieve the same result (Figure 25). These results may demonstrate how in some situations voice interaction may be more comfortable for the user due to its ease of use, the naturalness of human communication and the possibility of being able to interact with the app without having to touch the screen or the keyboard. The effectiveness of the use of voice commands is also proven by the results shown in Figure 26, which on a scale of 1 to 4 shows a distribution of ratings formed by 40% by the value 3 and 60% by the value 4. People therefore, identifying themselves in the context where voice interaction is the only possible mode of interaction, think that they would be quite or perfectly able to use the application. We also obtained good results regarding the overall experience of using the app and its actual usefulness for the real contexts, as shown in Figures 27 and 28, and users didn't encounter particular difficulties in using the app except for the the pronunciation of some wake words in such a way that the system could detect them.

5 Conclusions

In conclusion, our application presents itself as a solution aimed at improving accessibility for people with visual disabilities. VQAask seamlessly integrates three crucial modalities: voice interaction, visual interaction, and haptic feedback. These components collectively enable users with varying degrees of visual impairment to interact with the application effectively and independently.

One of VQask’s standout features is its voice interaction functionality, which allows users to easily communicate with the application through voice commands, ask questions, and receive spoken responses. This intuitive approach to interaction makes the app extremely easy to use and accessible, which was one of the primary goals during the initial development of the app.

The application’s vision integration is another noteworthy feature, allowing individuals with varying visual abilities to benefit from its features. This inclusive approach ensures that a wide range of users can access and utilize the app’s features, regardless of their specific visual impairments.

Haptic feedback, a core component of VQask, plays a critical role in facilitating navigation and interaction within the application. The haptic cues and responses provided by haptic feedback not only enhance the user experience but also significantly contribute to the app’s accessibility.

The application’s functionality is based on machine learning models, enabling the app to effectively respond to user questions about images and allowing for editing and processing of images to perform cropping, subject segmentation and provide more precise answers. This aspect aligns with the project’s primary objective of providing users with comprehensive information about the content of images.

VQask was developed using Flutter, an open-source framework from Google, and integrates various packages to provide networking functionality, image and audio processing, intuitive interface design, and seamless interaction with devices.

Project requirements were analyzed using a MoSCoW table, which encompassed both functional and non-functional aspects. This approach to requirements analysis laid the foundation for a robust, user-centric application.

To evaluate the effectiveness of the application, a sample group of 10 users took part in testing, and the results, as indicated by questionnaire responses, demonstrated that voice interaction was the preferred method for inputting questions. Additionally, the majority of users preferred automatic playback of responses over manual tapping, highlighting the importance of voice commands and the user-centric design philosophy that emphasizes multimodal accessibility.

In summary, the VQask Android application’s successful integration of voice interaction, vision utilization, and haptic feedback, coupled with NLP and Computer Vision techniques, positions it as a valuable solution for improving accessibility and enhancing the overall user experience for individuals with visual impairments. The project’s accomplishment in combining these elements underscores its potential to enhance user accessibility.

Some of the possible improvements that could be added in the future might include:

1. introducing additional features that extend the application’s utility across various domains, further enhancing its versatility and usefulness for users;
2. incorporating acoustic interactions, such as beeps or other auditory cues, can provide additional feedback and guidance for users, enhancing their overall experience;
3. exploring the adaptation of this application onto specialized hardware designed to assist individuals with visual difficulties in their everyday lives, creating a dedicated device.

References

- [1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [2] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.
- [3] Y. Fang, W. Wang, B. Xie, Q. Sun, L. Wu, X. Wang, T. Huang, X. Wang, and Y. Cao. Eva: Exploring the limits of masked visual representation learning at scale. *arXiv preprint arXiv:2211.07636*, 2022.
- [4] S. Huang, L. Dong, W. Wang, Y. Hao, S. Singhal, S. Ma, T. Lv, L. Cui, O. Mohammed, Q. Liu, K. Aggarwal, Z. Chi, J. Bjorck, V. Chaudhary, S. Som, X. Song, and F. Wei. Language is not all you need: Aligning perception with language models, 02 2023.
- [5] J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 01 2023.
- [6] S. Manmadhan and B. C. Kooor. Visual question answering: a state-of-the-art review. *Artificial Intelligence Review (2020)*, 2020.
- [7] OpenAI. Gpt-4 technical report, 2023.
- [8] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. E. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. J. Lowe. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155, 2022.
- [9] A. Sable. Visual question answering: a survey. <https://blog.paperspace.com/introduction-to-visual-questionanswering/>, 2020.
- [10] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.
- [11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [12] Wikipedia. https://en.wikipedia.org/wiki/moscow_method.
- [13] B. Workshop, T. L. Scao, A. Fan, and C. A. et al. Bloom: A 176b-parameter open-access multilingual language model, 2023.
- [14] D. Zhu, J. Chen, X. Shen, X. Li, and M. Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.