SOFTWARE REQUIREMENTS SPECIFICATION

Assistive Vision

Version 3.0

Written by:

Venkat Yenduri, Pierre Tawfik, Sukrut Nadigotti, Sharefa Alshaary

0. Version History

Date	Description	Author	Comments
2/10/2025	Version 1.0	Venkat Yenduri Sukrut Nadigotti Pierre Tawfik Sharefa Alshaary	Initial Draft
2/15/2025	Version 2.0	Venkat Yenduri Sukrut Nadigotti Pierre Tawfik Sharefa Alshaary	Edits after receiving feedback
03/31/2025	Version 3.0	Venkat Yenduri Sukrut Nadigotti Pierre Tawfik Sharefa Alshaary	Final version after prototype 3 feedback.

1. Introduction

1.1 Purpose

The purpose of this Software Requirement Specification (SRS) is to provide a detailed description of the requirements for the Assistive Vision Project. This document outlines the application's constraints, interfaces, and capabilities. It also explains the functional and non-functional requirements, offering an in-depth guide for anyone looking to understand the application thoroughly. This SRS will serve as a reference for both the development team and the client throughout the development process.

1.2 Scope

The Assistive Vision iOS application is designed to help blind and visually impaired individuals by providing real-time object recognition with audio feedback. The app is built to be easy to use, requiring little effort from the user. Interaction is done through simple touch gestures or voice commands, with audio guidance helping users navigate. Once activated, the app uses the iPhone's camera to continuously or on-demand scan the surroundings. It then detects objects in real time using a custom YOLOv11m model and describes them with clear audio feedback, making it easier for users to understand their environment.

Although the Assistive Vision iOS application delivers fast, real-time object recognition and audio feedback, it has some limitations. The app does not provide full spatial awareness beyond basic directional feedback, it cannot run in the background due to iOS restrictions, and detection history is not permanently stored. Additionally, all processing occurs locally without the use of cloud-based AI, and the system does not offer direct haptic feedback beyond standard system vibrations.

The app includes customizable settings that allow users to adjust audio feedback, prioritize certain objects for detection, and enable alerts for important objects, such as obstacles or hazards. Users can choose between Continuous and On-Demand scanning modes, allowing them to decide when the app should detect objects. A detection history feature is also included, letting users review recently recognized objects. The app is designed to work smoothly with iOS accessibility features, such as VoiceOver and Siri Shortcuts, ensuring it integrates well with existing tools for visually impaired users.

Although the Assistive Vision app provides real-time object detection and audio feedback, it has some limitations. The app must stay open to work and cannot run in the background due to iOS system restrictions. All object detection is done directly on the device, which protects user privacy but means the app does not use cloud-based AI for additional processing. While the app can tell users where objects are, it does not provide full spatial awareness beyond basic directions like "left," "right," or "ahead." Additionally, the app does not permanently store detection history, so recent object logs are deleted after a short time and are not saved across multiple devices. Because continuous scanning uses a lot of battery, users may need to enable Battery Saver mode for longer use.

By outlining these capabilities and limitations, the Assistive Vision iOS application gives users a clear understanding of what it can and cannot do. The app remains focused on delivering

fast and accurate object recognition with clear audio feedback, helping visually impaired individuals navigate their surroundings with confidence.

1.3 Intended Audience and Definitions

Term	Definition
AVP	The abbreviation for the application name, Assistive Vision Project.
User	An individual who uses the Assistive Vision application, typically a visually impaired person.
Role	The type of user that can interact with the Assistive Vision application, such as administrator or end-user.
Administrator	A user of the Assistive Vision application who has full access to application settings and management features.
End-User	A visually impaired individual who uses the Assistive Vision application for real-time object recognition and audio feedback.
FR	The abbreviation for the term Functional Requirements.
NFR	The abbreviation for the term Non-Functional Requirements.
Object Detection	The process by which the Assistive Vision application identifies and classifies objects in real-time using a camera and computer vision algorithms.
Audio Feedback	The application's spoken output provides users with information about their environment based on object detection results.
Environment	The surrounding area where the Assistive Vision application operates, including indoor and outdoor spaces.

Device	The hardware setup used for the Assistive Vision application is an iPhone with a camera and a microphone.

1.4 References

[1] Ultralytics, “YOLOv11 Documentation,” https://docs.ultralytics.com

[2] Apple Developer Documentation, “Core ML Framework,” https://developer.apple.com/documentation/coreml

[3] Apple Developer Documentation, “AVFoundation Framework,” https://developer.apple.com/documentation/avfoundation

[4] OpenCV Documentation, “Introduction to OpenCV,” https://docs.opencv.org/4.x/d9/df8/tutorial_root.html

[5] TensorFlow Lite Documentation, “TensorFlow Lite for Mobile and Embedded Devices,” https://www.tensorflow.org/lite

1.5 Overview

Several components are contained in the remaining sections of this SRS document. The second section will discuss the general description of the Assistive Vision application. To elaborate further, this section of the document will explain the software’s interactions, including how it operates with hardware components like cameras and audio output devices. The third section will provide an in-depth explanation of the software requirements for the application. This section covers functional and non-functional requirements, external interface requirements, design constraints, and logical data processing requirements. The final section of the document will present the data flow diagrams to illustrate how data moves through the application

2. General Description

2.1 Product Perspective

The Assistive Vision Project is an iOS-based application designed exclusively to assist visually impaired individuals. It functions as a standalone mobile app, leveraging the iPhone’s built-in camera for real-time object detection and audio output for feedback. The app uses a custom YOLOv11m model, an advanced object detection algorithm known for its speed and accuracy, to recognize objects in the user’s environment. Once an object is detected, the app provides immediate audio feedback, describing the object clearly to the user. This application operates independently on the iOS platform without the need for external hardware or application administrators, making it highly portable and user-friendly.

2.2 Product Functions

2.2.1 General Features

2.2.1.1 Application Activation

- End-users can activate the Assistive Vision iOS application through simple voice commands or by tapping the app icon on their device.
- The application automatically initializes the iPhone's camera and audio feedback components upon activation, requiring no additional setup from the user.

2.2.1.2 Application Deactivation

- Users can easily deactivate the application using voice commands such as "Stop" or "Exit Assistive Vision."
- Alternatively, the application can be closed manually through standard iOS gestures.

2.2.1.3 Navigation Menu

- The application features an audio-based navigation system designed for visually impaired users.
- Voice-guided prompts assist users in navigating key functions such as starting or stopping object detection, adjusting audio feedback settings, switching scanning modes, and accessing help features.
- Users can issue simple commands like "Start Detection," "Adjust Volume," or "Help" to interact with the app efficiently.

2.2.2 End-User Features

2.2.2.1 Real-Time Object Detection

- The Assistive Vision app uses a custom YOLOv11m model to detect objects in real time through the iPhone's camera.
- Detected objects are described audibly to the user with concise, contextually relevant information to support situational awareness.

2.2.2.2 Customizable Audio Feedback

- Users can adjust audio feedback settings, including volume, speech speed, and language preferences, through voice commands like "Increase volume" or "Change speech speed."
- The app allows users to prioritize specific object categories (e.g., obstacles, doorways, or people) for detection based on personal preferences.

2.2.2.3 Environment Scanning Modes

- The application offers two primary scanning modes:
    - Continuous Scanning: The app continuously detects and describes objects as the user moves through their environment.
    - On-Demand Detection: The app performs object detection only when the user issues a command like “Scan now.”
- Users can switch between modes using simple voice commands for greater control.

2.2.2.4 Object Recognition History

- The app maintains a brief history of recently detected objects, which users can access by asking, “What did you detect recently?”
- This feature helps users recall important objects they may have missed during active scanning.

2.2.2.5 Accessibility Features Integration

- The application is fully compatible with iOS accessibility tools such as VoiceOver and Siri.
- Users can launch the app and control basic functions using Siri shortcuts, enabling hands-free operation for enhanced convenience.

2.3 User Classes and Characteristics

This iOS application is designed exclusively for visually impaired individuals who rely on assistive technology to navigate their environments more confidently and independently. The users of this product will be English speakers with varying levels of familiarity with mobile devices and technology. Therefore, the application focuses on simplicity, accessibility, and intuitive voice-guided interactions to ensure ease of use for everyone, regardless of their technical background.

The features included are designed to support users in their daily activities, providing real-time object recognition with clear audio feedback. The app leverages advanced computer vision technology while maintaining a user-friendly interface that minimizes complexity. Every feature is tailored to enhance the user’s ability to identify objects in their surroundings effortlessly, promoting greater independence and situational awareness.

2.4 General Constraints

Due to the project needing to be completed within the timeframe of a college semester, time is a critical limiting factor that must be considered when deciding on the number of features to implement in the application. Decisions regarding which features to include will be based on what can realistically be developed within the given timeframe, balancing both the team’s goals

and the needs of the intended users. Each feature must be carefully evaluated to ensure it aligns with the project's objectives while remaining feasible within the semester's constraints.

Additionally, the development process must take into account the team's current skill set. Since the team is new to iOS app development and working with technologies such as Swift and CoreML, there may be a learning curve that affects productivity. This limitation requires the team to prioritize features that are within their current technical capabilities, focusing on core functionalities like real-time object detection and audio feedback, while postponing more complex features for potential future iterations.

The project utilizes a custom YOLOv11m model for object detection, which introduces its own set of technical considerations, including model optimization for real-time performance on iOS devices. Data management will rely on local storage and in-app databases optimized for lightweight operations, eliminating the need for complex server-side databases. This approach helps minimize development time while ensuring the application remains responsive and efficient for end-users.

2.5 Assumptions and Dependencies

The team assumes that users will consist solely of visually impaired individuals seeking assistance with real-time object recognition to enhance their independence and daily navigation. The application is designed to be intuitive and accessible, requiring no prior technical knowledge or familiarity with complex software systems. It is intended exclusively for personal use and will not be available to the general public through commercial distribution channels beyond standard app platforms like the Apple App Store. The app is primarily for visually impaired individuals, but can be used by anyone needing object recognition. While the iPhone is the initial target, the app is designed to be cross-platform in future versions. The app does not require internet connectivity for core features, and no user data is transmitted to external servers to protect privacy.

Since the application is developed as an iOS app, users will need access to an iPhone with a functioning camera and audio output capabilities. The app is optimized to work seamlessly with iOS accessibility features such as VoiceOver, ensuring compatibility with devices running the latest versions of iOS. No additional hardware or external devices are required, and the application does not rely on internet connectivity for its core object detection functionalities, providing a reliable and self-contained user experience.

3. Specific Requirements

3.1 External Interface Requirements

The Assistive Vision iOS application features a clean, intuitive interface designed with accessibility at its core. The user interface prioritizes simplicity to support visually impaired

individuals, offering both voice-guided commands and tactile screen interactions. When the user opens the app, they are greeted with the Assistive Vision logo, as shown in Figure 1 below. This splash screen indicates that the application is loading and initializing core services like camera access and object detection modules.

3.1.1 Assistive Vision Launch Screen

Figure 1: Assistive Vision Splash Screen

3.1.2 Home Screen

Figure 2: Object Detection Module with Adjustable Parameters

When switching modes, the interface dynamically adjusts to display detection results in real time, highlighting detected objects with bounding boxes and confidence scores. Figure 3 illustrates the object detection interface while operating in "Accurate" mode, showcasing high-confidence detections with a clean, minimal layout.

3.1.3 Recent Detection History

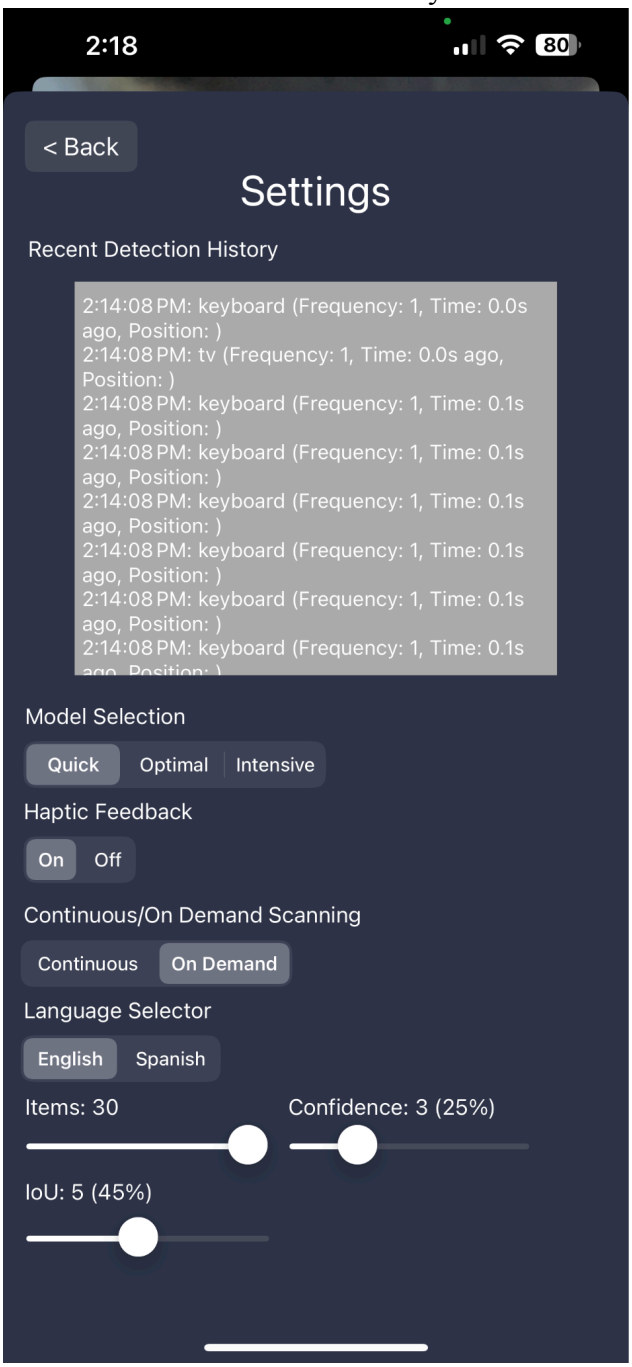


Figure 3: Recent Detection History Screen

Users can access the detection history through the "History" tab, which logs recent detections along with timestamps, as depicted in Figure 4. This feature allows users to review previously detected objects, providing a comprehensive history for reference.

3.1.4 Settings Screen

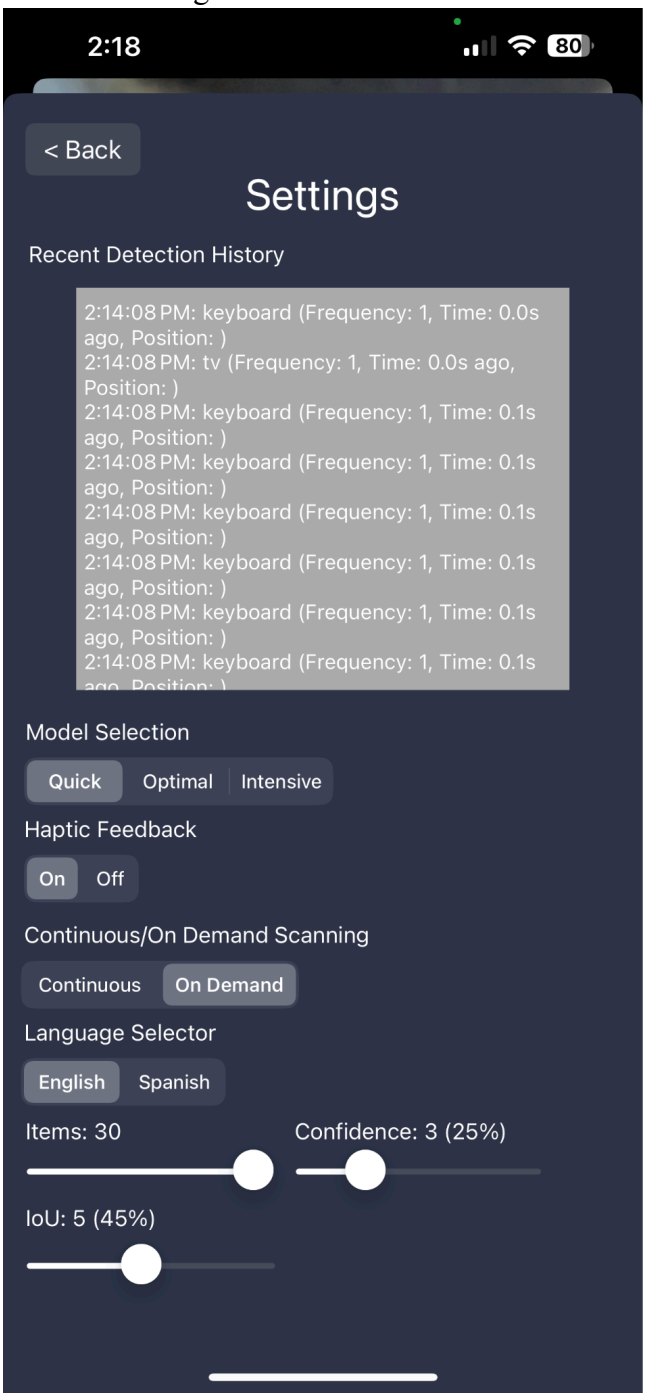


Figure 4: Settings Screen

The "Settings" tab provides users with the ability to customize various aspects of the application. As shown in Figure 5, users can adjust scan modes, enable or disable audio feedback and haptic alerts, control speech speed, and specify critical objects for prioritized detection.

3.1.5 General Settings UI

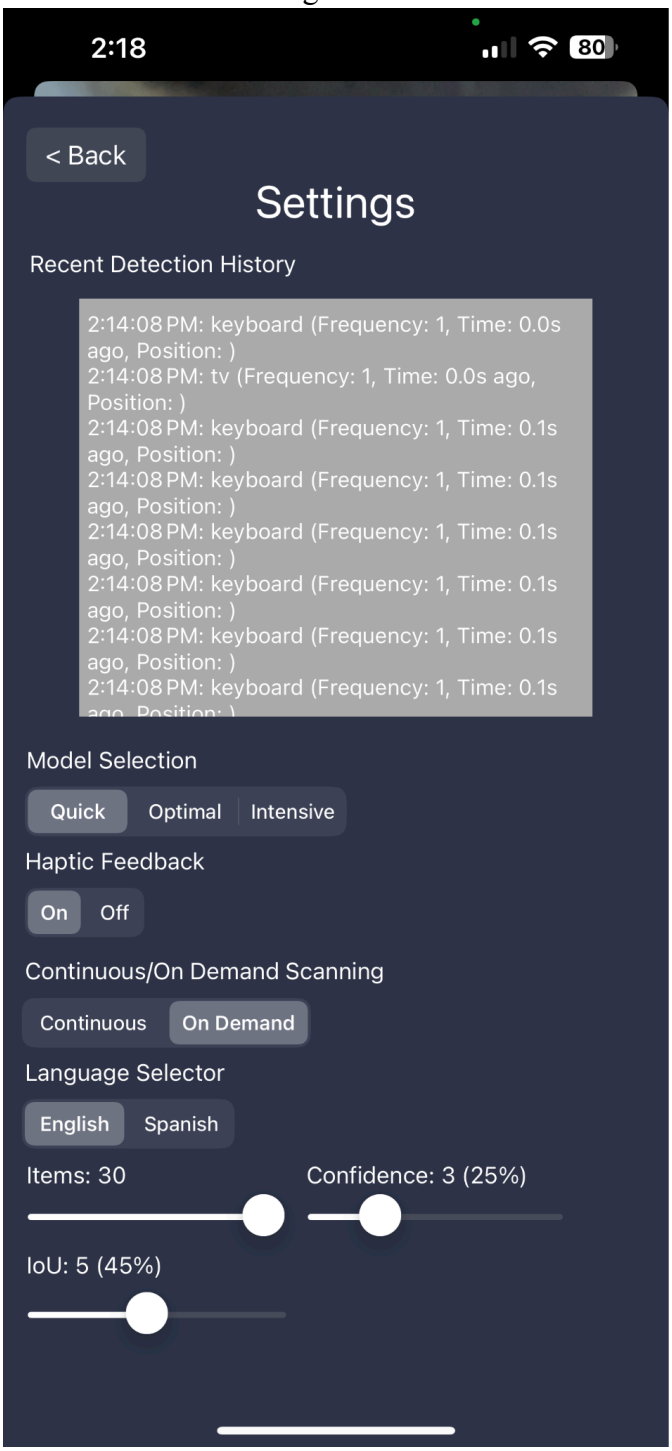


Figure 5: General Settings Interface

Users can switch between "Continuous" and "On-Demand" scanning modes via a dropdown menu, as shown in Figure 6. This flexibility allows users to choose between constant object detection and manual activation based on their preferences and battery-saving needs

3.1.7 Haptic Alerts Selection Interface

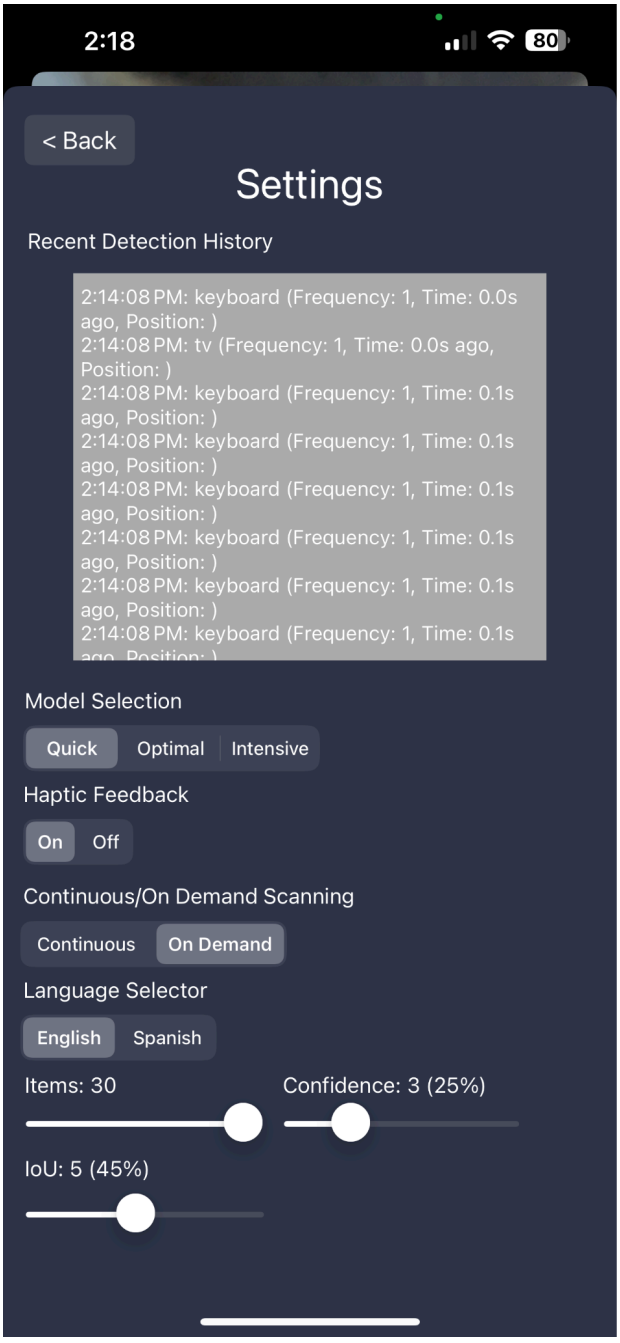


Figure 7: Haptic Alerts Selection Interface

Users can toggle the Haptic Alerts setting between enabled and disabled. When enabled, this feature provides gentle vibrations to alert users of potential hazards, warnings, or nearby objects. This tactile feedback enhances situational awareness, offering an additional layer of support alongside audio cues.

3.1.8 Assistive Vision Help Screen

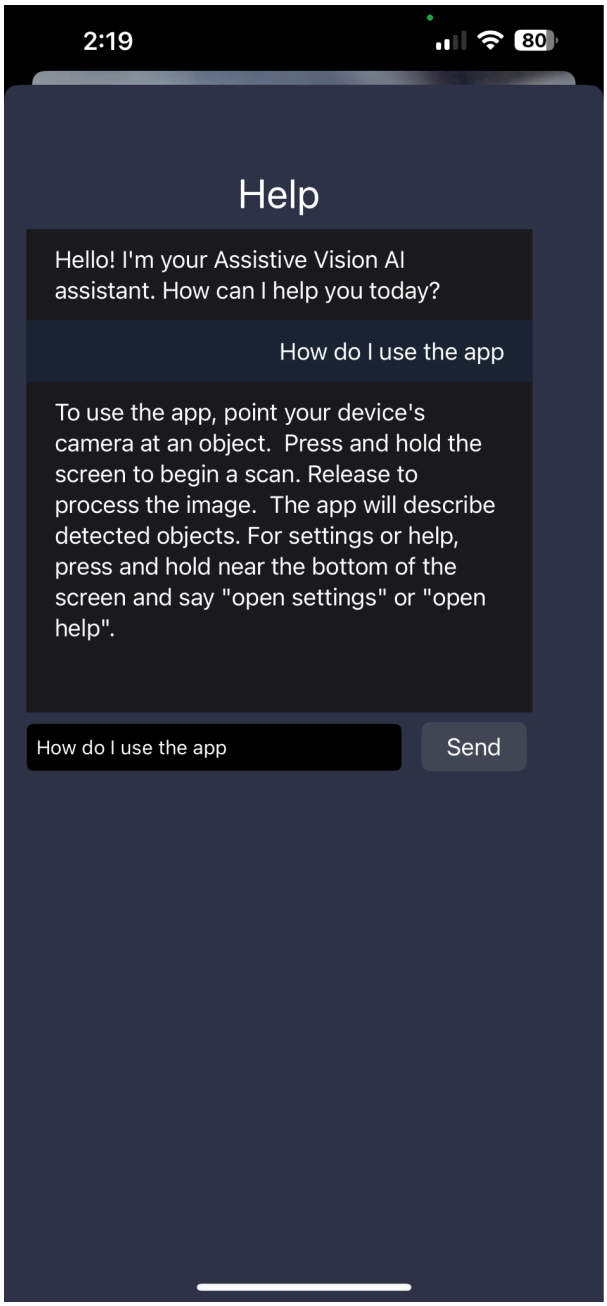


Figure 8: Help Screen

Users can open the help screen UI by pressing and holding down on the main screen and saying “open help”. The help screen will then open to which the user will then be able to press and hold the screen to ask their question verbally or type their question in and hit send to get a response.

3.2 Functional Requirements

ID	**FR1**
Title	Launch Application
Description	When the application is launched—either by a voice command (e.g., "Hey Siri open Assistive Vision") or by tapping the app icon—the system initializes the camera, object detection model, and audio engine. An audio message ("Assistive Vision activated") is played and the app becomes fully operational within 2 seconds.
Inputs	Voice command or tap input; current camera and microphone permissions status
Processing	Check permissions; initialize the camera using the VideoCapture module, object detection model, and audio engine; play the activation audio message.
Outputs	Audio confirmation "Assistive Vision activated" and the app ready for object detection and audio feedback.
Error Handling	If camera or microphone access is denied, inform the user and provide instructions to enable permissions.
Dependencies	Backward: None; Forward: FR2
Priority	High

ID	**FR2**
Title	Camera Initialization

Description	Upon launch, the system automatically initializes the iPhone camera using the VideoCapture module. It checks and requests camera permissions if needed, configures settings (e.g., 1080p resolution, 30 FPS, auto-focus, exposure control), and readies the camera stream for object detection.
Inputs	iPhone camera hardware; user launch action; permission status from iOS
Processing	1. Check and request camera permissions if needed. 2. Configure camera parameters (resolution, frame rate, focus, exposure). 3. Initialize the camera stream using VideoCapture. 4. Hand over the live feed to the object detection module.
Outputs	Audio confirmation "Camera ready object detection enabled" or appropriate error messages.
Error Handling	If permissions are denied, prompt the user to enable them; if the camera is unavailable or already in use, instruct the user accordingly.
Dependencies	Backward: FR1; Forward: FR3
Priority	High

ID	**FR3**
Title	Real Time Object Detection (Continuous Mode)

Description	The app continuously processes live video from the rear camera using a custom-trained YOLOv11m model. It extracts object labels and confidence scores, applies user-defined filters, and maintains a detection history with timestamps.
Inputs	Live camera feed; YOLOv11m model; user detection settings (confidence threshold, prioritized categories).
Processing	1. Capture and preprocess video frames. 2. Perform inference using the YOLOv11m model. 3. Apply filters and update the detection history. 4. Trigger audio feedback based on detections.
Outputs	Internal log of detected objects and real-time audio announcements (e.g., "Person detected at 92 percent confidence").
Error Handling	If the model fails to load or camera access is lost, announce "Detection model unavailable please restart" or prompt the user to re-enable permissions.
Dependencies	Backward: FR2; Forward: FR4
Priority	High

ID	**FR4**
Title	Object Detection (On Demand Mode)
Description	When the user issues a scan command (e.g., "Scan now"), the app captures a single video frame, processes it for object detection, and provides immediate audio feedback before returning to an idle state.

Inputs	Single video frame captured on-demand; user scan command
Processing	1. Capture a single frame upon command. 2. Preprocess the frame and run object detection. 3. Generate and deliver audio feedback describing detected objects and their confidence levels.
Outputs	Audio announcement detailing detected objects or a message if no objects are found.
Error Handling	If detection fails (e.g., due to camera issues), suggest rescanning or checking lighting conditions.
Dependencies	Backward: FR2, FR3; Forward: FR5
Priority	High

ID	**FR5**
Title	Provide Audio Feedback
Description	The app converts object detection outputs into natural language using the AVFoundation text-to-speech engine. It announces detected objects with contextual details such as confidence level and estimated distance, adapting the feedback based on scanning mode and user settings.
Inputs	Detection data (object labels, confidence scores, optional distance estimates); user TTS configuration settings.
Processing	1. Receive detection output from FR3 or FR4. 2. Filter and format results based on user settings. 3. Convert text to speech and deliver audio feedback.

Outputs	Spoken descriptions (e.g., "Chair detected 87 percent confidence, 5 feet ahead").
Error Handling	If the TTS engine fails or playback is interrupted, restart audio services or provide a fallback alert.
Dependencies	Backward: FR3, FR4; Forward: FR6
Priority	High

ID	**FR6**
Title	Voice Command Interface
Description	The app supports comprehensive voice commands using SwiftSpeech and Siri Shortcuts. Users can control functions such as starting/stopping detection, adjusting settings, and accessing help or tutorials. Commands are interpreted using fuzzy matching and confirmed via audio feedback.
Inputs	User spoken commands via the microphone; predefined command list.
Processing	1. Activate voice recognition. 2. Convert speech to text and match against predefined commands. 3. Execute the corresponding function and provide audio confirmation.
Outputs	Spoken confirmations (e.g., "Detection started") or error messages for unrecognized commands.
Error Handling	If voice recognition fails or background noise interferes, prompt the user to repeat the command.
Dependencies	Backward: FR5; Forward: FR7, FR11

Priority	High

ID	**FR7**
Title	Customizable Detection Settings
Description	Users can customize object detection settings by selecting which object categories to prioritize or ignore and by setting a confidence threshold. These settings can be adjusted via voice commands or through the settings screen and persist across app restarts.
Inputs	User voice commands or manual selections; predefined object categories; numeric threshold values.
Processing	1. Validate and update detection filters based on user input. 2. Apply changes immediately to the detection module. 3. Save settings persistently.
Outputs	Audio and visual confirmations (e.g., "Chairs will no longer be detected").
Error Handling	If an unsupported category or invalid threshold is provided, notify the user and suggest alternatives.
Dependencies	Backward: FR3, FR6; Forward: FR8
Priority	Medium

ID	**FR8**
Title	Audio and Haptic Feedback Configuration

Description	Users can adjust text-to-speech settings (such as speech speed, volume, pitch, language) and configure haptic feedback intensity. Changes are applied immediately and persist across sessions.
Inputs	Voice commands (e.g., "Increase volume", "Change language to Spanish") or manual adjustments via the settings screen.
Processing	1. Interpret and validate user input. 2. Update the TTS engine and haptic settings accordingly. 3. Save configuration persistently.
Outputs	Audio confirmation (e.g., "Speech speed set to fast") and corresponding visual feedback in the settings interface.
Error Handling	If an unsupported value is provided, revert to default settings and alert the user.
Dependencies	Backward: FR5, FR7; Forward: FR9
Priority	Medium

ID	**FR9**
Title	Continuous versus On Demand Detection Modes
Description	The app allows users to toggle between continuous detection (where the camera is always processing frames) and on-demand detection (triggered by a scan command). The selected mode is saved persistently across app restarts.
Inputs	Voice commands (e.g., "Enable continuous mode" or "Switch to on demand detection") or manual selection via the settings screen.

Processing	1. Detects and verifies the mode selection. 2. Activate continuous detection or wait for a manual trigger accordingly. 3. Save the mode selection persistently.
Outputs	Audio confirmation (e.g., "Continuous scanning enabled") and visual indication in the settings screen.
Error Handling	If system resources are low, prompt the user to switch to on-demand mode.
Dependencies	Backward: FR3, FR6; Forward: FR10
Priority	Medium

ID	**FR10**
Title	Detection History
Description	The app maintains a temporary log of the last 10 object detections (including object labels, confidence scores, and timestamps) that can be retrieved via voice command or through the settings screen.
Inputs	Continuous detection data; voice command "What did you detect recently?"
Processing	1. Store the last 10 detections in a local log. 2. Retrieve and format the log upon request. 3. Optionally clear the log after a defined period.
Outputs	Audio playback of recent detections or a message stating "No recent detections available" if the log is empty.
Error Handling	If no detections are logged, announce "No recent detections available."
Dependencies	Backward: FR3; Forward: FR11
Priority	Low

ID	**FR11**
Title	Siri Integration and Shortcuts
Description	The app integrates with Siri Shortcuts to enable hands-free activation of key features such as launching the app and toggling detection modes. Users can configure these shortcuts via the settings screen.
Inputs	Siri voice commands; iOS SiriKit integration.
Processing	1. Detect and validate Siri commands. 2. Route the command to the appropriate function. 3. Execute the action and provide audio confirmation.
Outputs	Audio confirmation (e.g., "Assistive Vision started detection") and visual feedback for shortcut configuration in settings.
Error Handling	If Siri Shortcuts are misconfigured or permissions are missing, notify the user to adjust settings.
Dependencies	Backward: FR1, FR6; Forward: FR12
Priority	Medium

ID	**FR12**
Title	User Help and Tutorial
Description	The app provides an interactive, voice-guided tutorial powered by Gemini AI. When the user requests help (via "Help" or "Tutorial"), the system delivers personalized guidance on app usage and navigation through audio (and optionally

	visual cues).
Inputs	Voice command "Help" or "Tutorial"; manual selection from the settings screen.
Processing	1. Detect the help request. 2. Activate Gemini AI to generate context-aware guidance. 3. Deliver audio and visual tutorial instructions.
Outputs	Spoken tutorial instructions (e.g., "To start detection, say 'Start detection'").
Error Handling	If the AI assistant fails, provide a fallback message ("Tutorial not available at this time").
Dependencies	Backward: FR6; Forward: FR13
Priority	Low

ID	**FR13**
Title	Battery Optimization Mode
Description	The app monitors battery levels and enables Battery Saver Mode automatically when the battery falls below 15 percent or via a voice command. In this mode, the camera frame rate and detection frequency are reduced (e.g., from 30 FPS to 15 FPS) to conserve energy while maintaining core functionality.
Inputs	Battery status updates; voice command "Enable Battery Saver" or manual toggle in the settings.
Processing	1. Detects low battery or user activation. 2. Adjust camera frame rate and detection frequency. 3. Save power-saving settings persistently until disabled.

Outputs	Audio confirmation "Battery Saver mode enabled" and a visual indicator in the settings screen.
Error Handling	If Battery Saver cannot be activated due to system constraints, notify the user appropriately.
Dependencies	Backward: FR10; Forward: FR14
Priority	Medium

ID	**FR14**
Title	Offline Operation
Description	All object detection and processing are performed locally on the device using CoreML. The app functions fully without an internet connection, ensuring continuous detection and audio feedback even when offline.
Inputs	Live camera feed; preloaded YOLOv11m model stored locally.
Processing	1. Capture video frames. 2. Perform on-device inference using the local ML model. 3. Generate audio feedback based on detection results.
Outputs	Real-time object detection and audio feedback without reliance on a network.
Error Handling	No network-dependent error handling required.
Dependencies	Backward: FR3, FR5; Forward: FR15
Priority	High

ID	**FR15**
Title	Hazard Alert Customization
Description	Users can customize hazard alerts by specifying critical objects (e.g., stairs, vehicles, obstacles) for enhanced warnings. When such objects are detected, the app provides a distinct audio alert and triggers haptic feedback if enabled.
Inputs	Voice commands (e.g., "Enable stairs alert") or manual configuration via the settings screen.
Processing	1. Update the hazard alert list based on user input. 2. Monitor detections and compare with the hazard list. 3. Trigger enhanced audio and haptic alerts when a match occurs.
Outputs	Spoken hazard warnings (e.g., "Caution: Stairs detected ahead") and distinct alert tones.
Error Handling	If an unsupported hazard type is requested or settings fail to update, notify the user with an appropriate message.
Dependencies	Backward: FR3, FR5, FR7; Forward: FR16
Priority	Medium

ID	**FR16**
Title	Scene Exploration Mode
Description	The user can pause real-time scanning and request a detailed scene analysis. The app captures a high-resolution snapshot, detects at least five distinct objects with directional cues (left, right, ahead), and delivers a structured spoken report within 2 seconds.

Inputs	Voice command "Explore scene"; live camera snapshot.
Processing	1. Capture a high-resolution snapshot. 2. Process the image using the detection model. 3. Determine distances and directions (±15° accuracy). 4. Generate a spoken report summarizing object positions.
Outputs	Spoken scene summary (e.g., "Chair 1.5 meters to your left, table 2 meters ahead").
Error Handling	If the snapshot is unclear, announce "Scene unclear please try again" and retry up to three times before suggesting adjustments.
Dependencies	Backward: FR3, FR5, FR10; Forward: FR17
Priority	Medium

ID	**FR17**
Title	Local Usage Analytics
Description	The app tracks local usage statistics such as detection session duration, object count, and battery consumption. When requested, it provides a structured audio report of these statistics.
Inputs	Automatic logging of session data; voice command "Show my usage stats".
Processing	1. Log session duration, object count, and battery usage. 2. Retrieve and format the logged data upon request. 3. Generate a structured spoken report.
Outputs	Audio report (e.g., "You have used continuous detection for 45 minutes and detected 52 objects today") or a message if

	no data is available.
Error Handling	If logging fails or data is missing, announce "No usage stats found" or "Unable to retrieve statistics".
Dependencies	Backward: FR2, FR3; Forward: FR18
Priority	Low

ID	**FR18**
Title	Haptic Feedback for Close Objects
Description	The app provides haptic feedback to alert the user when obstacles are detected within a predefined proximity. If haptic feedback is enabled and the device supports vibration, a short vibration is triggered; otherwise, an audio alert is used.
Inputs	Live camera feed; user preference for haptic alerts; device capability for vibration.
Processing	1. Detects obstacles from the camera feed. 2. Check if an obstacle is within the threshold distance. 3. Trigger haptic feedback if enabled and supported; otherwise, use an audio alert.
Outputs	A brief vibration or a spoken alert (e.g., "Obstacle close by").
Error Handling	If haptic feedback is not supported or an error occurs, announce "Haptic feedback not available on this device" and default to audio feedback.
Dependencies	Backward: FR3, FR5; Forward: None
Priority	Medium

3.3 Non-Functional Requirements

ID	**NFR1**
Title	Performance
Priority	High
Description	The application must process object detection and provide audio feedback within 1 second to ensure real-time responsiveness on supported iOS devices. Resource usage must be optimized to prevent excessive CPU or battery consumption.

ID	**NFR2**
Title	Reliability
Priority	High
Description	The application must function consistently without unexpected failures. Object detection accuracy should remain above 90%, exceptions must be handled gracefully, and errors logged for debugging.

ID	**NFR3**
Title	Availability
Priority	High
Description	The application must be available 99.9% of the time. Core features such as object detection and audio feedback must operate offline to ensure seamless usability even without an internet connection.

ID	**NFR4**

Title	Security
Priority	High
Description	User data, including detection history and settings, must be securely stored using encryption. The app must request explicit permission for camera and microphone access and comply with iOS privacy policies.

ID	**NFR5**
Title	Maintainability
Priority	High
Description	The application must be developed using modular design principles with thorough documentation. At least 80% of new features and bug fixes should require minimal code changes, and updates must be deployable with less than one day of downtime.

ID	**NFR6**
Title	Portability
Priority	High
Description	The app must support iOS 14 and later, ensuring compatibility with at least 90% of active iPhone models. The design should allow potential expansion to other platforms with at least 80% code reusability.

3.4 Design Constraints

The following technologies will be used to build the Assistive Vision iOS application: Swift, CoreML, TensorFlow Lite, AVFoundation (for Text-to-Speech), and OpenCV. The application is designed to run exclusively on iOS devices, leveraging native iOS frameworks to ensure optimal performance, security, and compatibility with accessibility features.

3.4.1 Model and Processing Constraints

3.4.1.1 Model Size

The YOLOv11m model used for object detection has size constraints due to mobile device memory limitations. The model must be optimized to ensure efficient performance without exceeding the device's available RAM, which can vary depending on the iPhone model. The optimization process includes reducing model complexity and compressing model weights to ensure smooth execution on devices with limited memory.

3.4.1.2 Processing Power

Since the application performs real-time object detection, processing power is a key constraint. The app must balance detection speed and accuracy to ensure smooth performance on devices with different CPU and GPU capabilities. Optimizations such as hardware acceleration using Apple's Neural Engine and efficient multi-threading will be implemented to enhance performance without overloading the processor.

3.4.1.3 Battery Consumption

Continuous camera usage and real-time processing can significantly impact battery life. The application is designed to minimize battery consumption through efficient use of device resources and optimized background processing. Power-saving modes, such as reducing the frame capture rate during continuous scanning, will be incorporated to extend battery life during prolonged use.

3.4.1.4 Storage Limitations

The application stores temporary data, such as user preferences and recent detection history, locally on the device. iOS imposes limitations on app data storage, and the app must ensure efficient data management to avoid excessive storage use. Data retention policies will be applied to clear old detection logs automatically, and user preferences will be stored in lightweight, encrypted formats.

3.4.2 iOS-Specific Constraints

3.4.2.1 App Store Guidelines

The application must comply with Apple's App Store Review Guidelines, which include strict requirements related to privacy, data security, and performance. This includes obtaining user consent for camera and microphone access, ensuring data encryption, and adhering to Apple's policies on data collection and sharing.

3.4.2.2 iOS Version Compatibility

The app will support iOS 14 and above, limiting compatibility with older devices that do not meet the minimum system requirements for running CoreML and AVFoundation efficiently. Compatibility testing will be conducted on various devices to ensure stability and performance across supported iPhone models.

3.4.2.3 Background Processing Restrictions

iOS restricts background processing to conserve battery life and system resources. As a result, the app cannot perform continuous object detection in the background and requires active user engagement. To address this, the app will provide quick resume capabilities, allowing it to restart detection seamlessly when brought back to the foreground.

These constraints are considered throughout the development process to ensure the Assistive Vision iOS application delivers a high-quality, secure, and efficient experience for visually impaired users.

3.5 Logical Database Requirements

The Assistive Vision iOS application is designed exclusively to assist visually impaired individuals by providing real-time object recognition with audio feedback. Since the application is intended for personal use without the need for administrative roles or multiple user hierarchies, data security and integrity are managed through device-based protocols that ensure both privacy and reliability. The application processes all data locally on the user's device, eliminating the need for external servers or cloud databases, which significantly reduces potential security risks. User preferences, such as audio settings and object detection configurations, are stored securely within the device's local storage using encrypted formats provided by iOS. The object detection data is processed in real-time, and temporary detection logs are stored only for short periods to maintain performance and privacy.

The application architecture relies on structured data storage optimized for mobile devices, utilizing key-value pairs and lightweight databases like Core Data or SQLite for managing session data and user settings. This ensures data consistency, fast access, and ease of maintenance. Data integrity is maintained through strict validation protocols. When users input data, such as adjusting settings, the application enforces constraints like character limits, data type restrictions, and format checks to prevent errors or inconsistencies. In cases where invalid data is entered, the application generates error messages to alert users and prevent the storage of faulty information.

Data security is enhanced through the use of iOS's built-in security features, including biometric authentication like Face ID or Touch ID, and system-level encryption for both stored data and data in transit within the device. The application also ensures that temporary data, such as recent object detection history, is regularly cleared to prevent unnecessary data accumulation and potential privacy risks. Since no data is transmitted to external servers, the app avoids common vulnerabilities associated with network-based attacks.

The application allows users to sort, filter, and search through recent detection logs to enhance usability. For example, users can request information about recently detected objects or filter detections based on categories like obstacles or frequently encountered items. While the app does not require database updates like traditional systems, it ensures data consistency through automated background processes that maintain system integrity without user intervention. Backup and data recovery are managed through iOS's native iCloud services, allowing users to restore their preferences and app data if needed. This approach ensures a secure, efficient, and privacy-focused experience tailored to the needs of visually impaired individuals.

4. Analysis Models

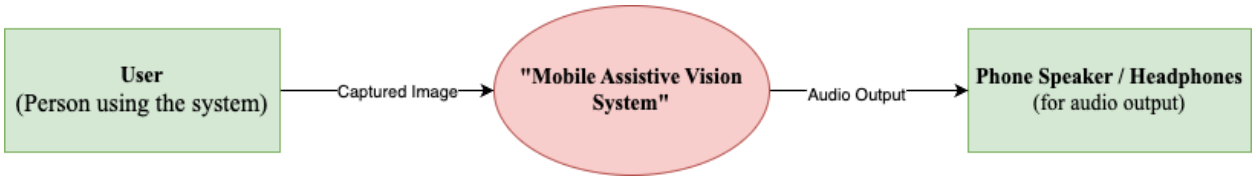
4.1 Data Flow Diagrams (DFD)

The data flow diagram is structured into three levels, starting from Level 0, which provides a high-level overview of the Assistive Vision iOS application, and progressing to Level 2, which offers a more detailed breakdown of system processes. Level 0 outlines the core interactions between the user and the application, while Level 1 expands on how data flows through the primary modules. Level 2 delves into specific sub-processes, giving a comprehensive view of how image data is captured, processed, and converted into audio feedback. These diagrams illustrate the complete flow of information, from image capture using the iPhone's camera to object detection and real-time spoken feedback.

This data flow assumes that all processing occurs locally on the iPhone, ensuring maximum speed, security, and reliability without reliance on external servers.

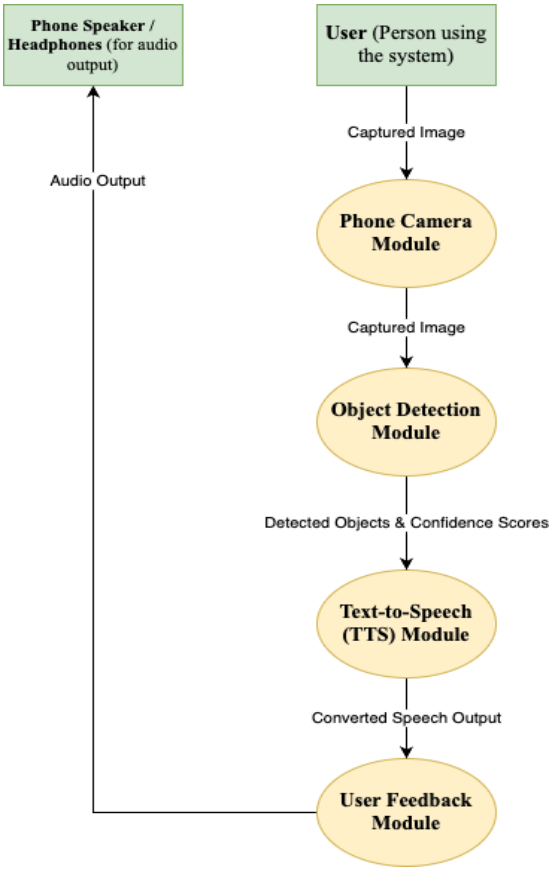
1. Context Diagram (Level 0)

The user interacts with the Assistive Vision iOS Application by capturing images with the iPhone's camera. The application processes these images to detect objects and converts the results into spoken feedback. The detected objects are then described through the phone's speaker or connected Bluetooth headphones, forming an updated data flow where the user captures an image, the Assistive Vision Application processes it, and the audio output is sent to the phone speaker or headphones.



2. Level 1 DFD (High-Level Breakdown)

The Level 1 DFD provides a high-level breakdown of the application into four core modules responsible for processing images and delivering audio feedback. The Phone Camera Module captures images using the iPhone's camera, while the Object Detection Module leverages CoreML and TensorFlow Lite to identify objects in real-time and outputs object names with confidence scores. The Text-to-Speech (TTS) Module then converts these detected object names into speech using Apple's AVFoundation framework, and finally, the User Feedback Module outputs the generated speech through the phone's speaker or Bluetooth headphones.



Data Flow:

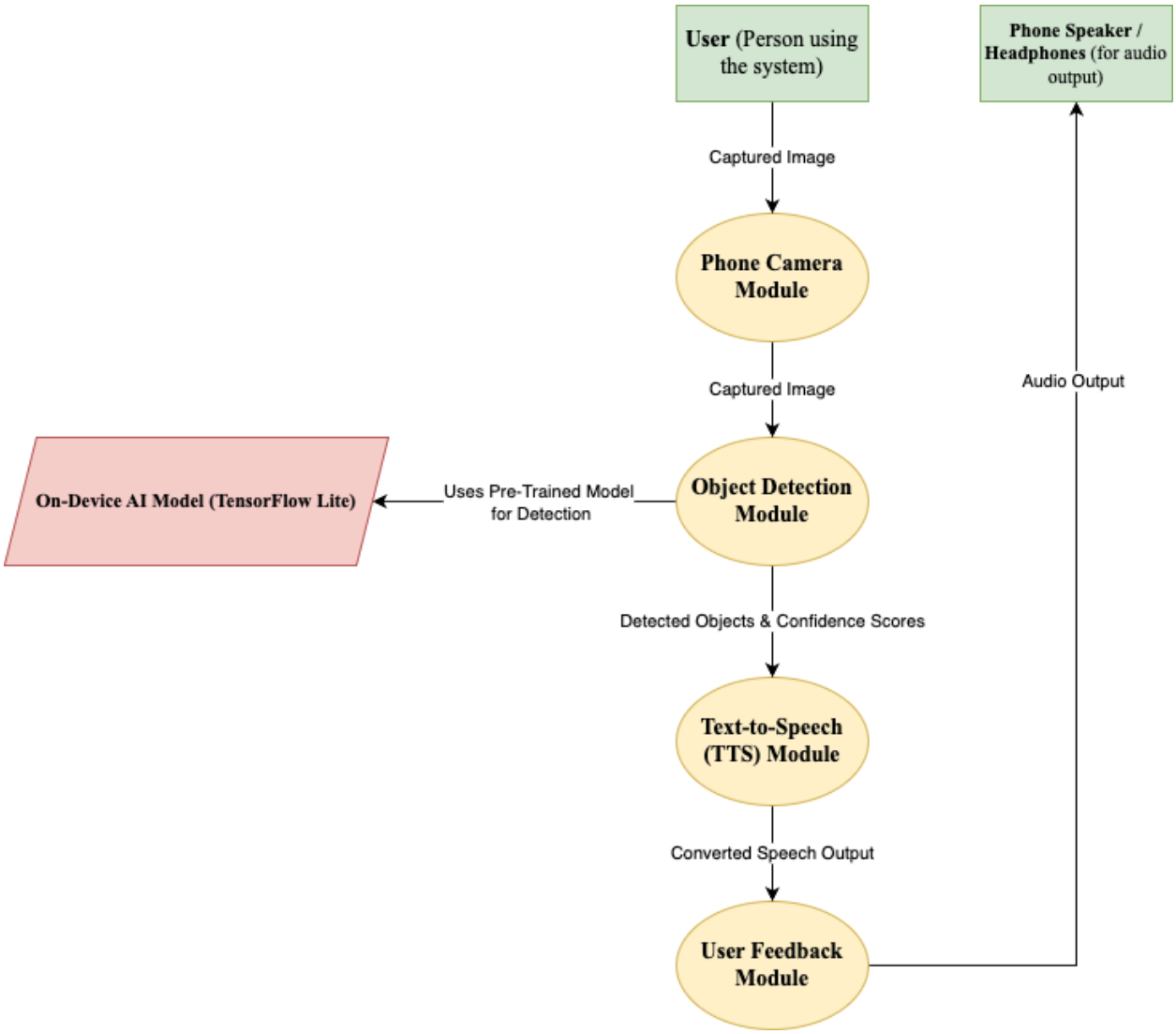
The data flow begins when the user captures an image with the phone camera, which is then sent to the Phone Camera Module. The captured image is forwarded to the Object Detection

Module, where detected objects and their confidence scores are generated. This information is subsequently passed to the Text-to-Speech Module, which converts the detections into spoken audio. Finally, the audio output is delivered to the user via the User Feedback Module, which outputs sound through the phone's speaker or connected headphones.

3. Level 2 DFD (Detailed Breakdown)

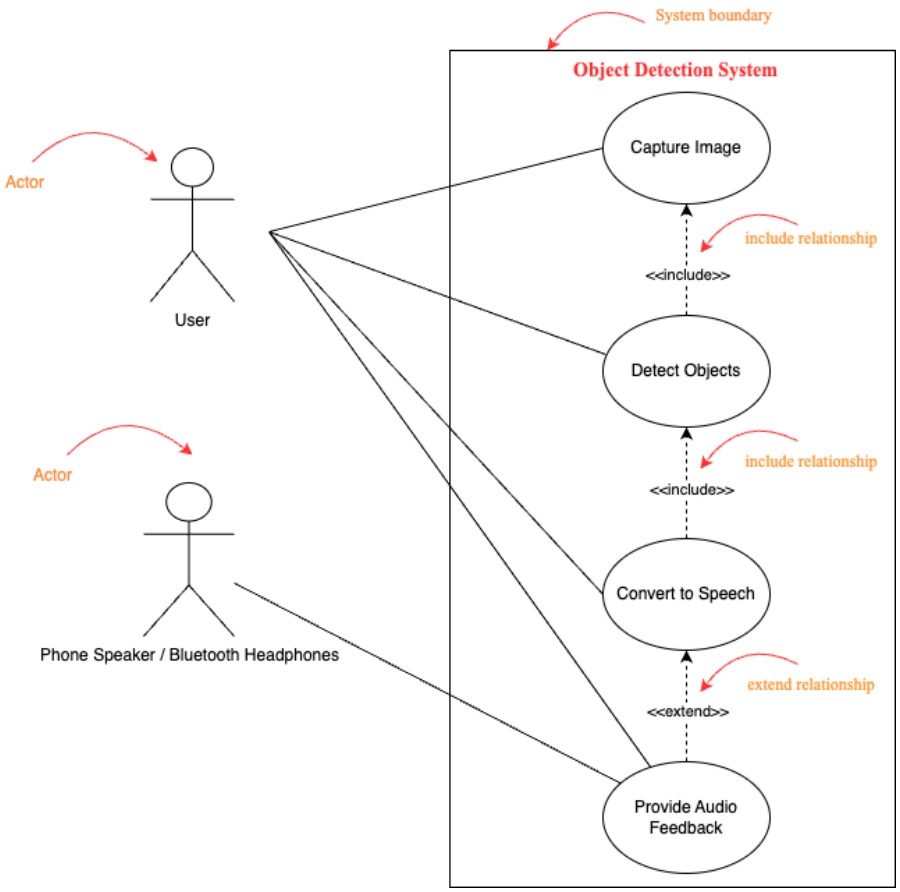
At this level, each module is broken down further to illustrate its internal workings and data handling. The Phone Camera Module uses the iPhone’s built-in camera to capture images when requested by the user and sends the captured images to the Object Detection Module. The Object Detection Module processes these images using the YOLOv11m model via CoreML and TensorFlow Lite, retrieves locally stored object recognition data, and sends the detected object names along with confidence scores to the Text-to-Speech (TTS) Module.

The TTS Module converts the object names into spoken feedback using Apple’s AVFoundation framework and passes the audio output to the User Feedback Module, which then delivers the final output through the phone’s speaker or connected Bluetooth headphones. The data flow progresses as follows: the user captures an image that is sent from the Phone Camera Module to the Object Detection Module; the Object Detection Module leverages a pre-trained model for detection and accesses locally stored data before sending recognized objects and confidence scores to the TTS Module; and finally, the Text-to-Speech Module converts these details into speech, which is output through the User Feedback Module to the phone speaker or headphones.



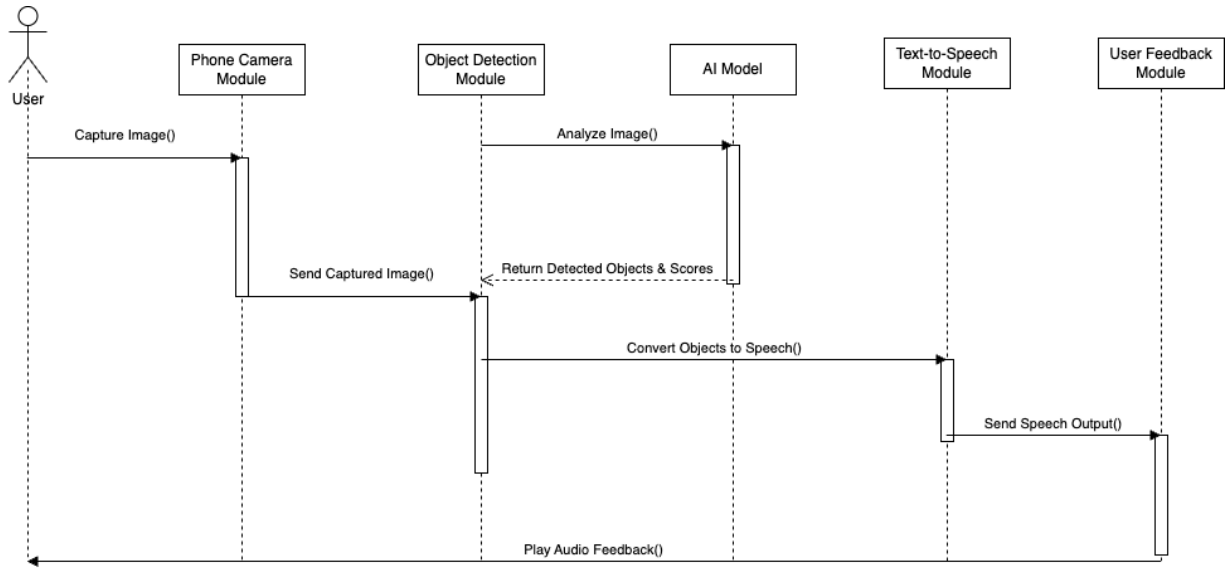
4.2 Use Case Diagram (UCD)

The Use Case Diagram for the Assistive Vision application illustrates how a visually impaired user interacts with the system. The primary interactions involve capturing an image with the iPhone's camera, processing that image to detect objects, converting the recognized objects into spoken feedback, and outputting the audio through the phone's speaker or connected Bluetooth headphones. Optional functionalities include adjusting detection settings for specific object categories and configuring TTS parameters such as speed, volume, and voice preferences.



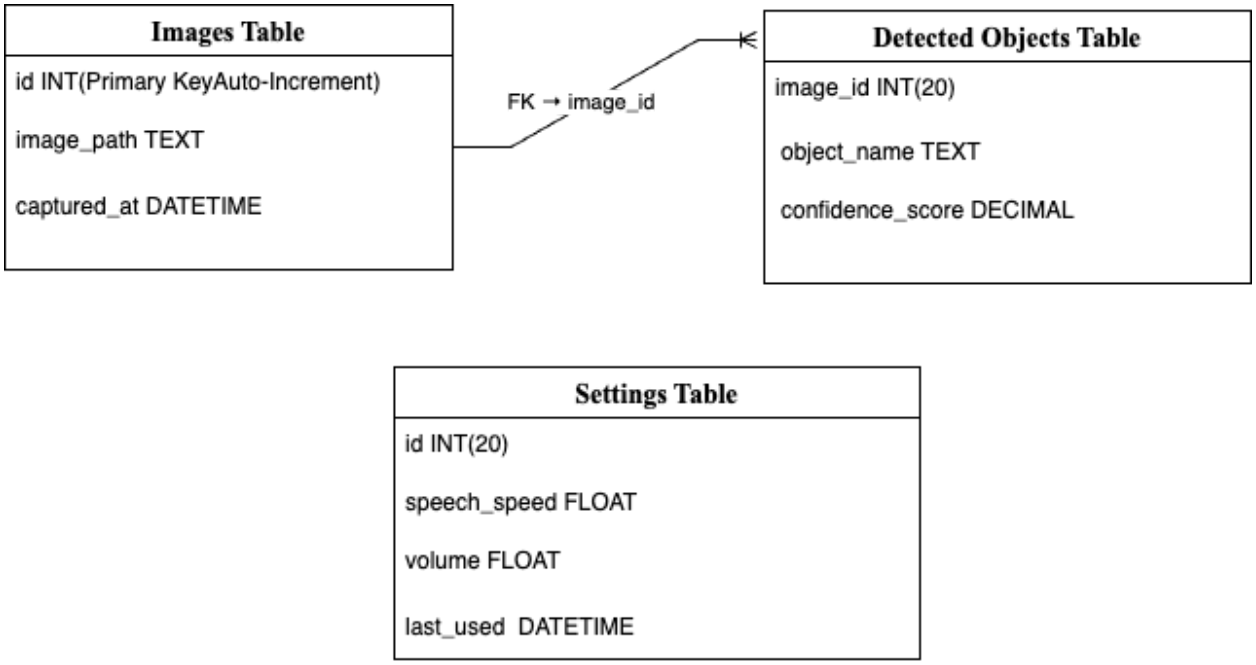
4.3 Sequence Diagram (SD)

The Sequence Diagram details the step-by-step interaction between the user and the application for object detection and audio feedback. It begins when the user captures an image using the phone camera. The image is processed by the Phone Camera Module and forwarded to the Object Detection Module, which analyzes it using a locally stored pre-trained AI model (YOLOv11m via TensorFlow Lite) that detects objects and returns confidence scores. The detected objects are then converted into speech by the Text-to-Speech Module, and the resulting audio feedback is delivered to the user by the User Feedback Module through the phone's speaker or connected headphones.



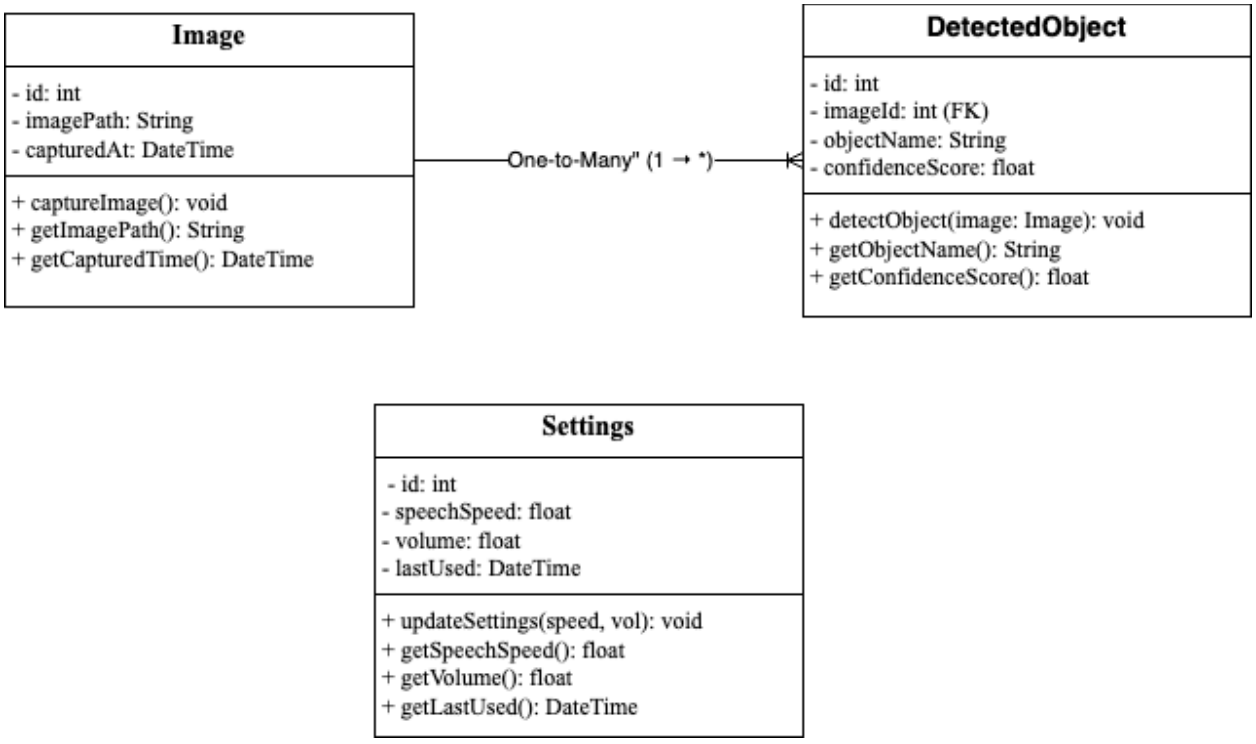
4.4 Database Diagram (DD)

The Database Diagram (DD) represents the structure of the Assistive Vision system by outlining its key entities, their attributes, and their relationships. The system comprises an Images Table that stores captured images along with metadata, a Detected Objects Table that stores the detected objects and links them to the images, and a Settings Table that stores user preferences such as speech speed and volume. The diagram illustrates a one-to-many relationship, where one image can be associated with multiple detected objects, while the Settings Table exists independently without referencing other tables.



4.5 Class Diagram (CD)

The Class Diagram provides an overview of the system's structure, including its classes, attributes, and methods. Classes are depicted as rectangular boxes that list attributes and methods, while associations illustrate relationships between classes, such as inheritance or composition. Additionally, multiplicity indicators (e.g., 1, 0..1) specify the number of instances that can be related between classes.



5. Hardware, Software, and Communication Interfaces

The application interacts with the iPhone's camera for object detection and uses the internal speaker or connected headphones for audio feedback. It integrates with iOS Accessibility APIs, such as VoiceOver and Siri Shortcuts, to facilitate user interaction, and leverages CoreML and TensorFlow Lite for on-device object detection while using AVFoundation to convert detections into spoken feedback. Additionally, the app operates fully offline, though it offers optional iCloud integration for backing up user preferences.