Coffee Machine Assistance for the Visually Impaired: A Computer Vision-Based Mobile Application

by

Murad Huseynov

Bedir Esen

Hazem Moustafa

Alperen Dalgıç

Submitted to Computer Engineering

TED University

Table of Contents

Abstract

This thesis addresses the challenge of independent coffee machine operation for visually impaired individuals by developing a computer vision-based mobile application. The research problem lies in the inaccessibility of modern coffee machines.

The objective of this project is to design, implement, and evaluate a mobile application that enables visually impaired users to operate a specific coffee machine model independently.

To achieve this, a mobile application will be developed using JavaScript and React Native, integrating the EfficientDet object detection model and platform-specific Text-to-Speech engines (Android TTS and IOS VoiceOver). Data will be manually collected and augmented to train the EfficientDet model for real-time coffee machine recognition.

Key results will demonstrate the feasibility of accurate coffee machine recognition and effective voice guidance. Usability testing with visually impaired individuals will be conducted to indicate that the application provides a user-friendly and accessible method for operating the target coffee machine.

In conclusion, this research shows that a computer vision-based mobile application can offer a practical solution for coffee machine assistance, enhancing independence for visually impaired users.

Limitations of this project include the current focus on a single coffee machine model and the need for further optimization for diverse machine types and user needs. Future work should focus on expanding model compatibility and incorporating user feedback for continuous improvement.

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AT	Assistive Technology
BiFPN	Bidirectional Feature Pyramid Network
CNN	Convolutional Neural Network
COCO	Common Objects in Context (dataset)
CPU	Central Processing Unit
CV	Computer Vision
GPU	Graphics Processing Unit
IoU	Intersection over Union
mAP	Mean Average Precision
NNAPI	Neural Networks API
POUR	Perceivable, Operable, Understandable, Robust (principles)
R-CNN	Region-based Convolutional Neural Network
SIFT	Scale-Invariant Feature Transform
SSD	Single Shot Detector
TTS	Text-to-Speech
UCD	User-Centered Design
UI	User Interface
UX	User Experience
WCAG	Web Content Accessibility Guidelines
YOLO	You Only Look Once (object detection algorithm)

Glossary

Albumentations: A Python library for fast and flexible image augmentations, used to enhance training datasets for computer vision models.

Bidirectional Feature Pyramid Network (BiFPN): A feature network architecture used in EfficientDet that allows multi-directional information flow for better feature fusion.

Convolutional Neural Network (CNN): A class of deep neural networks most commonly applied to analyzing visual imagery in computer vision applications.

Depthwise Separable Convolutions: A type of convolution operation that factorizes a standard convolution into a depthwise convolution and a pointwise convolution, reducing computation and model size.

Detox: An end-to-end testing and automation framework for mobile apps, used for testing React Native applications.

EfficientDet: An object detection model architecture that uses compound scaling to achieve both high accuracy and efficiency, making it suitable for resource-constrained environments.

EfficientNet: A convolutional neural network architecture that uniformly scales all dimensions of depth, width, and resolution using a compound coefficient, serving as the backbone for EfficientDet.

Intersection over Union (IoU): A metric used to evaluate the overlap between predicted and ground truth bounding boxes in object detection.

Mean Average Precision (mAP): A metric used to evaluate the accuracy of object detectors, calculated by taking the mean average precision across all classes and/or IoU thresholds.

MobileNet: A lightweight deep neural network architecture designed for mobile and embedded vision applications, characterized by depthwise separable convolutions.

Neural Networks API (NNAPI): Android's API for hardware-accelerated machine learning operations, allowing apps to run computationally intensive neural networks on mobile devices.

Post-training Quantization: A technique that reduces model size and improves CPU and hardware accelerator latency by converting floating-point representations to more efficient formats after model training.

Transfer Learning: A machine learning technique where a model developed for one task is reused as the starting point for a model on a second, related task, reducing training time and improving performance.

Weight Pruning: A technique to reduce model size and computational requirements by removing less important connections (weights) in a neural network.

Chapter 1: Introduction

The daily routine of brewing coffee, a simple pleasure for many, presents a significant obstacle for individuals with visual impairments. Modern coffee machines, with their visually dependent interfaces and controls, create barriers to independent operation. This thesis addresses the need for assistive technologies that can bridge this accessibility gap, focusing on empowering visually impaired individuals to confidently use everyday appliances like coffee machines and enhance their daily independence and quality of life.

1.1 Leveraging Computer Vision for Mobile Assistance

Existing assistive technologies for appliance operation often rely on generic solutions or costly modifications. This thesis explores a more innovative and accessible approach: utilizing computer vision (CV) and smartphone technology. We focus on developing a mobile application that uses the smartphone camera to "see" and interpret coffee machine interfaces. This project aims to create a specific, real-time guidance system for a chosen coffee machine model, providing precise voice instructions for its operation. The scope of this research is the design, development, and evaluation of a CV-based mobile application to assist visually impaired individuals in independently operating a defined coffee machine model through voice guidance.

1.2 Building Upon Existing Assistive Technology Research

Assistive technology research has explored various methods for appliance accessibility, from tactile aids to auditory feedback and voice control. However, many solutions are limited in adaptability, integration ease, or real-world usability. While voice and gesture interfaces offer potential, challenges in robustness and accuracy remain. This thesis builds upon this foundation by investigating CV – a rapidly advancing field – to address coffee machine operation specifically. By using visual understanding and real-time guidance, this project aims to contribute a novel, potentially more universally applicable, and context-aware approach to appliance accessibility.

1.3 Research Questions and Project Objectives

This dissertation investigates the feasibility and effectiveness of a CV-based mobile application for coffee machine assistance, guided by the following questions:

1. Can a mobile CV application accurately recognize a coffee machine model in real-time?
2. Can voice instructions effectively guide visually impaired users to make coffee on the recognized machine?
3. Is the application usable, accessible, and helpful for independent coffee machine operation by visually impaired individuals?
4. What are the key design challenges and effective mitigation strategies for such an assistive application?

To answer these questions, this project aims to achieve the following objectives:

1. Design and implement an EfficientDet-based CV system for coffee machine recognition.
2. Develop a voice guidance system using Android TTS and iOS VoiceOver for clear coffee machine operation instructions.
3. Integrate these systems into an accessible mobile application.
4. Thoroughly test and evaluate the application's performance and usability with visually impaired users.

1.4 Technical Approach

To address the challenges outlined above, this thesis employs a technical approach centred around efficient deep learning for CV and accessible mobile application development. At its core, the application utilizes the **EfficientDet** [1] object detection model, chosen for its state-of-the-art balance between accuracy and computational efficiency, making it well-suited for real-time performance on mobile devices. The **MobileNet** [2] architecture serves as the backbone for EfficientDet, further enhancing its efficiency and reducing computational overhead. The model is trained using **TensorFlow/Keras** [3], leveraging its powerful deep learning capabilities, and subsequently optimized and deployed on mobile devices using TensorFlow Lite for efficient on-device inference.

For image processing and data augmentation, the project integrates **OpenCV** [4], a widely used CV library. Critically, the application's user interface and voice guidance system are developed using **React Native and JavaScript**, enabling cross-platform deployment to both Android and iOS platforms from a single codebase. This choice of React Native facilitates wider accessibility and future reach of the application. Voice guidance is implemented using the platform-specific Text-to-Speech engines (**Android TTS** [5] and **iOS VoiceOver** [6]), ensuring native accessibility support on each platform.

Furthermore, the application utilizes **SQLite** for local, on-device storage of structured data such as user settings and application preferences, and local file storage for efficient management and retrieval of the trained **TensorFlow Lite** models [7], ensuring offline functionality and data privacy. This technical stack is strategically selected to achieve high accuracy in coffee machine recognition, provide real-time and responsive voice guidance across platforms, and maximize accessibility and user reach while balancing development efficiency.

Chapter 2: Literature Review

2.1 Assistive Technology for Visually Impaired Individuals

Assistive technology (AT) has revolutionized the lives of individuals with visual impairments, enhancing their independence and improving their ability to perform daily tasks. These technologies range from simple tactile aids to advanced artificial intelligence-powered applications. The development of accessible solutions is particularly significant in the context of household management, where tasks such as appliance operation and kitchen activities can pose significant challenges.

In the case of kitchen appliances, for example, the Be My Eyes [8] app connects visually impaired users with sighted volunteers through video calls, providing real-time assistance for various tasks. While applications like Be My Eyes offer valuable assistance, they rely on human volunteers and may not always provide immediate or consistent guidance. Furthermore, such solutions, while helpful, do not provide the same level of independent skill development as systems that directly empower users to operate appliances themselves.

This thesis explores a more direct and autonomous approach through CV, aiming to provide immediate and consistent guidance without reliance on external human assistance, and promoting greater self-sufficiency in daily living.

2.2 Computer Vision for Accessibility

CV has emerged as a revolutionary technology in tackling accessibility challenges, especially for individuals with visual impairments. By allowing machines to interpret and comprehend visual information, CV enhances capabilities such as object recognition, scene understanding, and human-computer interaction. This progress significantly improves the independence and quality of life for visually impaired users.

One of the most impactful applications of CV in enhancing accessibility is object recognition. Mobile applications like Seeing AI [9] and Envision AI [10] leverage deep learning models to identify everyday objects, read text and even recognize faces. While they are groundbreaking for general environmental awareness, these applications often have limited functionality without an internet connection and can struggle when it comes to interacting with complex objects like kitchen appliances. Recognizing a coffee machine is different from identifying and understanding the function of its specific buttons, lights and interaction points needed for operation - a critical limitation this thesis aims to address.

2.3 Object Detection Algorithms

The core technical challenge in enabling CV-based appliance interaction lies in accurate and efficient object detection deployable on mobile devices.

Traditional methods like Haar cascades [11] and feature descriptors like SIFT [12] have limitations in robustness to variations in lighting, viewpoint, and object complexity, making them less suitable for diverse real-world appliance interfaces. Deep learning-based approaches, especially Convolutional Neural Networks (CNNs), have revolutionized the field.

These models can be broadly categorized into two-stage detectors like Faster R-CNN [13], known for high accuracy but slower speeds, and one-stage detectors like SSD [14] and YOLO [15], prioritizing speed, and efficiency. Two-stage detectors offer higher accuracy but are computationally expensive for mobile devices. One-stage detectors prioritize speed and efficiency, making them popular for mobile applications, but sometimes sacrifice accuracy, particularly for detecting small or overlapping objects - a key concern when identifying small buttons on a coffee machine.

EfficientDet represents a significant advancement by optimizing the accuracy-efficiency trade-off. Leveraging the EfficientNet backbone, Bi-directional Feature Pyramid Network (BiFPN), and compound scaling, EfficientDet models achieve competitive accuracy while maintaining high efficiency suitable for mobile platforms. Compared to models like YOLO which might prioritize sheer speed, EfficientDet often demonstrates superior accuracy, particularly for reliable detection of small control elements required for safe and effective appliance operation guidance in this project. While optimization (using TensorFlow Lite) is still required, EfficientDet provides a solid foundation for achieving the necessary balance between reliable detection and real-time mobile performance.

2.4 Voice Guidance and Text-to-Speech Systems

TTS technology is fundamental, converting recognized object information and instructional steps into spoken output. TTS is used in various applications, such as screen readers, navigation assistance, object and text recognition, and voice-controlled devices. For our project, voice guidance will assist users in operating a coffee machine by providing step-by-step spoken instructions, helping them navigate the process without needing visual input.

Cloud-based TTS (like Google Cloud TTS [16]) services offer highly natural voices but require constant internet connectivity and raise potential privacy concerns. On-device TTS engines, such as the native Android TTS Engine or iOS VoiceOver, guarantee offline functionality, crucial for an assistive application that must be dependable regardless of network status. They also offer lower latency and enhanced privacy. While potentially less natural-sounding than cloud alternatives, the paramount need for reliability, consistency, and offline access in an assistive context makes on-device TTS the preferred choice for this project, ensuring the user can always access guidance.

2.5 User-Centred Design and Accessibility Principles

User-centered design (UCD) [17] prioritizes understanding and involving the intended users (in this case, visually impaired individuals) throughout the design process. Key UCD principles include early and continuous user involvement, iterative design driven by user feedback, and a primary focus on usability and user experience. This approach ensures that assistive technologies are not only technically sound but also genuinely meet user needs and are practical for real-world use. For our application, UCD will guide the development of voice guidance and interaction to be intuitive and effective for visually impaired users in a coffee-making context.

The Web Content Accessibility Guidelines (WCAG) [18] provide a robust framework, particularly the POUR principles:

> **Perceivable:** Information must be presentable to users in ways they can perceive. For this project, this means ensuring all visual information is conveyed effectively through clear auditory output.
>
> **Operable:** Interface components and navigation must be operable. The application prioritizes simple interaction models, relying primarily on auditory output and minimal, accessible input methods compatible with screen readers.
>
> **Understandable:** Information and the operation of the user interface must be understandable. This translates to clear, simple language in voice instructions and a logical, predictable interaction flow.
>
> **Robust:** Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies. This involves using standard development practices and ensuring compatibility with platform screen readers.

2.6 Identifying the Gap and Justifying the Research

This review reveals significant progress in AT and CV, yet a critical gap persists: the lack of robust, accessible, and autonomous mobile solutions specifically designed to guide visually impaired individuals through the interaction steps required for operating complex household appliances like modern coffee machines. Existing general-purpose CV apps lack the necessary granularity and contextual understanding for safe operation, while human-in-the-loop solutions don't foster true independence. Traditional AT methods struggle with the diversity of modern appliance interfaces.

This thesis directly addresses this gap by proposing and developing a mobile application that integrates state-of-the-art, efficient object detection (EfficientDet) tailored for recognizing specific coffee machine controls with clear, sequential, on-device voice guidance. Grounded in UCD and accessibility principles, this project aims to provide a practical, reliable, and user-validated solution. Its contribution lies in demonstrating the feasibility of using targeted CV and mobile technology to create a context-aware assistive tool that enhances independence in a specific, challenging daily living task, moving beyond generic object recognition towards interactive task assistance.

Chapter 3: Methodology

This chapter outlines the methodology used to design, implement and evaluate the proposed assistive coffee machine system. It covers system architecture, module deployment, data collection, and testing strategies.

3.1 System Design and Architecture

The proposed system is built as a modular mobile application that combines computer vision, voice guidance, and accessibility-focused interaction to assist visually impaired individuals when interacting with coffee machines. The system architecture is composed of four fundamental modules that work together in a coordinated pipeline. The camera input is managed by an input module that is responsible for processing images at predetermined intervals using React Native's camera interface with platform-dependent optimizations for both Android and iOS devices. The images are then processed by a computer vision module that uses an EfficientDet object detection algorithm with a MobileNet backbone to detect coffee machine parts and infer their spatial relationships. The output is then handled by a voice-guidance module where they are transformed into contextual spoken instructions with the device's in-built text-to-speech interfaces. Lastly, there is a user interface module that is optimized to offer accessible interaction with minimal visual reliance.

Data flows through the system in a sequential and streamlined manner. Although the camera offers a constant preview to assist the user in orienting the device, high-quality images are taken in fixed time intervals for optimal computation speed as a trade-off for the reliability of the detection results. These images undergo preprocessing steps, including resizing, normalization, color space adjustment, and artifact removal, before being passed to the neural network. The EfficientDet model, optimized using TensorFlow Lite, processes each image independently and outputs bounding boxes, class labels, and confidence scores for detected coffee machine components. This information is analyzed by a contextual processing module, which remains informed about the associated stage of the beverage production procedure. Based on this state awareness, the system determines the most appropriate next instruction, which is converted into spoken guidance using native text-to-speech functionality. User interactions, including accessible touch gestures and voice commands, are processed by the interface module and may trigger state transitions or dynamic adjustments to image capture intervals.

The modularity of the system was designed to ensure efficiency in development, as well as scalability and easy maintenance. By separating responsibilities across clearly defined components, the architecture allows different parts of the system to be updated or optimized independently without affecting overall functionality. This approach also enables platform-specific optimizations while maintaining a shared codebase through React Native. Importantly, the architecture supports complete on-device processing, allowing the system to function reliably without an internet connection. This offline capability is critical for assistive technology, where consistent availability and user privacy are essential.

3.2 Computer Vision Implementation

EfficientDet was selected as the object detection model for this system due to its strong balance between detection accuracy and computational efficiency, both of which are essential for real-time mobile applications. Compared to alternative architectures such as Faster R-CNN, EfficientDet achieves comparable or superior accuracy while requiring significantly fewer computational resources. Its compound scaling strategy enables the selection of an appropriate model size based on the target device's capabilities, making it well suited for deployment across a range of smartphones. In addition, coffee machines contain many small and visually similar components, including buttons, indicators, and dials. EfficientDet's Bidirectional Feature Pyramid Network (BiFPN) architecture is particularly effective at detecting such small objects, outperforming single-stage detectors like SSD and YOLO in this context. The model's compatibility with TensorFlow Lite further supports efficient mobile deployment without substantial accuracy loss. Since the system relies on interval-based image capture rather than continuous video, EfficientDet's strong single-frame detection performance is especially advantageous.

To improve robustness without requiring an impractically large number of manually captured images, data augmentation techniques were applied using the Albumentations library [20]. These techniques included geometric transformations, photometric adjustments, the addition of noise and blur, background variation, and simulated motion effects to reflect handheld device movement. Care was taken to preserve bounding box validity during augmentation, and metadata was retained for later analysis. Before inference, all images were resized to match the EfficientDet input dimensions, normalized to a [0,1] range, adjusted for consistent color representation, and processed to reduce JPEG compression artifacts.

Model training followed a structured procedure based on transfer learning. The EfficientDet model was initialized using weights pre-trained on the COCO dataset [19], allowing the system to leverage generalized visual features. Custom classes corresponding to specific coffee machine components were defined, and training proceeded in two phases. Initially, only the classification and bounding box regression heads were trained while the backbone remained frozen. In the fine-tuning phase, deeper layers were gradually unfrozen to adapt the model more closely to the target domain. Model robustness was evaluated using k-fold cross-validation, with each fold using an 80/20 split between training and validation data. Hyperparameters such as batch size, learning rate, and regularization settings were optimized through grid search. Additional training was performed on sequential image sets to enhance the model's ability to infer state transitions from interval-based captures.

For mobile deployment, the trained model was converted to TensorFlow Lite format and weight pruning was applied to remove redundant connections while maintaining accuracy. Platform-specific hardware acceleration was leveraged, using NNAPI and GPU delegates on Android devices and Core ML with Metal Performance Shaders on iOS. Inference efficiency was further improved through parallel preprocessing, adaptive resolution selection, early stopping for irrelevant frames, and confidence-based filtering. Memory usage was carefully managed by reusing buffers and ensuring proper cleanup of tensors. Image capture intervals

were dynamically adjusted based on processing speed, detection confidence, battery status, and the current stage of the coffee-making process, with more frequent captures during critical operations.

3.3 Voice Guidance System Implementation

The voice guidance system relies on platform-native text-to-speech engines to ensure both performance and accessibility. On Android devices, speech output is generated using the native TextToSpeech API, while iOS devices use VoiceOver through the AVSpeechSynthesizer framework. A React Native abstraction layer provides a unified interface across platforms, handling initialization, configuration, and error management internally. Speech output is optimized for clarity, with voice selection prioritizing intelligibility over naturalness. The speech rate is configured within a range of approximately 175 to 200 words per minute, consistent with assistive technology research, while pitch and volume settings are exposed to users to accommodate individual preferences. All speech synthesis occurs locally on the device, ensuring offline functionality and immediate responsiveness.

Voice instructions are designed according to established accessibility principles. Each instruction begins with a clear action verb and incorporates spatial references that are familiar to visually impaired users, such as clock-face positioning. Confirmation phrases are included to reinforce correct actions, and contextual cues are provided to maintain user orientation throughout the process. Language is kept concise and consistent, with terminology standardized across all instructions. An auditory hierarchy differentiates between routine guidance, critical warnings, and progress confirmations through subtle variations in pitch and speed. Continuous feedback is provided as the system detects state changes, offers periodic reorientation cues during longer tasks, and delivers encouraging messages upon successful completion of steps.

The sequencing of instructions is managed through a state machine architecture that models each operational stage of the coffee-making process. State transitions are triggered by computer vision detections and timing constraints, ensuring that guidance remains contextually appropriate. A hierarchical decision tree maps detected states to corresponding instructions and includes fallback paths to handle uncertainty. Progress tracking supports both linear task sequences and branching user choices, while error handling mechanisms detect incorrect actions and provide corrective guidance. Timing and pacing are dynamically adjusted based on task complexity and observed user interaction patterns, with configurable options allowing users to tailor instruction delivery to their needs.

3.4 Mobile Application Development

The application is developed using React Native to support cross-platform deployment while maintaining near-native performance. Core application logic is implemented in JavaScript, with platform-specific modules used for camera access, text-to-speech integration, and accessibility services. Development is supported by the React Native CLI, with Jest and the React Native Testing Library used for unit and component testing, and Detox [21] employed for end-to-end testing across Android and iOS. The shared codebase enables consistent

behavior across platforms, while conditional rendering and abstraction layers handle platform-specific differences. The application targets Android 7.0 and above and iOS 13 and above.

User interface and interaction design follow an accessibility-first approach. Touch targets are large and semantically meaningful; navigation is linear and optimized for screen readers and required user input is minimized. High-contrast layouts, haptic feedback, and carefully managed focus order support both TalkBack and VoiceOver. Users are provided with customization options, including adjustable speech rate, vibration intensity, and instruction detail level.

Camera integration is implemented using react-native-camera with accessibility enhancements. The system provides a continuous preview for positioning while capturing frames at configurable intervals. Capture frequency is dynamically adjusted based on detection confidence, device performance, and battery level. Captured frames are processed through a prioritized queue to maintain responsiveness, with memory-efficient cleanup procedures in place. Users receive audio and haptic feedback to assist with camera positioning, including simple verbal cues and vibration patterns that indicate whether the coffee machine is in frame. Privacy indicators, timeout warnings, and emergency exit shortcuts further support safe and accessible camera use.

Integration between computer vision and voice guidance is handled through an event-based communication system, with Redux used for global state management. Detection events are debounced to prevent instruction overload, and voice output is managed through a priority queue to handle concurrent guidance requests. Processing is performed in the background where possible, with adaptive behavior based on detected device capabilities. All core functionalities operate offline, with local storage used for both machine recognition models and instruction set.

3.5 Data Collection

Before data collection began, clear data set requirements were defined to ensure sufficient coverage and quality. The initial collection targeted 500 base images, which were expanded to approximately 1,500–2,000 samples through augmentation. The dataset was split into training, validation, and test sets using a 70/20/10 ratio. Diversity requirements included multiple machine states, comprehensive component coverage, environmental variation, and perspective differences. Image quality standards specified minimum resolution, clear focus, consistent labeling, and detailed metadata.

Image collection was executed using a structured methodology that incorporated both controlled and real-world environments. In controlled settings, the coffee machine was systematically photographed from a wide range of angles (0°, 45°, 90°, etc.) and distances under varying lighting conditions to ensure comprehensive coverage of all components. Real-world usage was captured through video recordings of complete coffee-making sequences, from which representative frames were extracted to reflect meaningful operational states. Additional images were collected in realistic environments such as

kitchens and office break rooms, incorporating background variation and potential visual obstructions. Emphasis was placed on capturing perspectives that reflect how visually impaired users are likely to hold and position their devices, including slightly off-center angles and inconsistent distances. To further align with the system's interval-based design, sequential image sets were collected to simulate discrete frame capture and support learning of state transitions from non-continuous input. Image collection was done mainly using smartphone cameras that are typical of mid-range Android and iOS smartphones, with additional validation performed across multiple device models to ensure generalizability.

3.6 Testing and Evaluation Methodology

System evaluation was conducted through a multi-layered testing strategy encompassing unit, integration, system, and acceptance testing. Functional performance, system efficiency, accessibility compliance, and usability were evaluated under varying environmental and device conditions. Technical analysis was done using computer vision measures like precision, recall, mean average precision at several thresholds on IoU, speed, and false positive profiles. System-level metrics included startup time, memory usage, battery consumption, hardware utilization, and thermal performance. Voice guidance was evaluated in terms of instruction latency, speech clarity, contextual accuracy, and interruption handling.

Accessibility compliance was assessed against WCAG 2.1 AA standards and platform-specific guidelines using both automated tools and manual testing with TalkBack and VoiceOver. User-centered evaluation involved 10 to 15 visually impaired participants representing diverse backgrounds and experience levels. Participants completed task-based usability tests using common coffee-making scenarios, supported by think-aloud protocols, questionnaires, and semi-structured interviews. Metrics such as task success rate, completion time, error frequency, recovery behavior, and subjective satisfaction were used to assess system effectiveness.

Chapter 4: Implementation

This chapter details how the coffee machine assistance system was implemented, including the major decisions made during the development process, computer vision system, integration of voice guidance, as well as developing the mobile application. During the implementation stage, several technological challenges arose, leading us to make different choices than originally planned, mainly in the build system and the model used for detection.

4.1 Implementation of System Architecture

The final system architecture was developed to be a modular system consisting of four inter-linked components: the Input Module (Camera), Computer Vision (CV) Module, Voice Guidance Module, and User Interface (UI) Module.

One of the biggest decisions we made was switching to the Expo framework with Continuous Native Generation (CNG) instead of standard React Native development. We made this change because we kept running into problems with the Android Gradle Plugin conflicting with the machine learning libraries during the initial development phase. The CNG approach rebuilds the native environment automatically during each build using expo-prebuild, which lets us control exactly which versions of build tools we were using. By locking the Gradle version to 8.5, we finally got rid of the compilation errors and were able to use both react-native-fast-tflite for running the model and async-storage for saving data locally.

Getting data to move smoothly between the JavaScript code and the native TensorFlow Lite interpreter was another challenge. Regular mobile buffer libraries were found to be unreliable during real-time execution, which resulted in application crashes. We ended up writing the base64ToUint8Array function to convert captured image frames into the format the TFLite interpreter needs. This custom conversion eliminated the need for extra processing overhead and kept the app running smoothly.

4.2 Computer Vision Implementation

The dataset was constructed following the plan laid out in Chapter 3.5, but some extra categories were included for dealing with real-world conditions. The final constructed database consists of a total of 33 categories including rotations: 16 buttons for coffee drink buttons, 6 indicators for controls, 6 for physical parts, and 5 for the milk system. A 'finger' class was also included, allowing the system to monitor the position of the user’s hand for which details will be provided in Chapter 4.4.

Since visually impaired individuals may be holding their phones at unusual angles, we made sure to create versions of each element that are also angled. Using the expo-image-manipulator library, we processed each image by resizing it to 640 x 640 pixels and converting it to JPEG format with base64 encoding so that it could be sent to the model easily. The image preprocessing implementation is shown in Figure 4.1, where captured images are resized using the manipulateAsync function and saved in JPEG format with base64 encoding for effective transfer to the inference pipeline.

```
const resized = await ImageManipulator.manipulateAsync(
  photo.uri,
  [{ resize: { width: 640, height: 640 } }],
  { base64: true, format: ImageManipulator.SaveFormat.JPEG }
);
```

Figure 4.1: Image preprocessing implementation using expo-image-manipulator for standardized resizing to 640×640 pixels and JPEG conversion with base64 encoding.

Even though our initial plan was to implement EfficientDet, we ended up implementing YOLOv8n (the nano variant of You Only Look Once) using the Ultralytics library. This decision was based on a few issues we faced in implementing it on mobile devices. The main problem with EfficientDet was getting it to compile on Android. Every time we tried to integrate the standard TensorFlow libraries with React Native, we got version conflicts and Gradle errors that stopped the app from building. However, implementing it using the react-native-fast-tflite library from YOLOv8 came out to be much more hassle-free after we got Gradle version 8.5 sorted out. Moreover, converting the YOLOv8 model to TensorFlow Lite was straightforward and automatically handled compatibility with the phone's GPU, which saved us a lot of manual work.

The YOLOv8n model was trained using transfer learning, based on the initial COCO dataset weights, and then fine-tuned on our coffee machine pictures. As shown in Figure 4.2, we began our training task with particular parameters. The training was done with 100 epochs, and if no improvement occurred in 20 epochs, then the training stopped to avoid overfitting. The batch size was chosen to be 32 to utilize the GPU memory efficiently. The entire training took around two hours on the machine with CUDA support.

```
results = model.train(
    data=data_yaml_path,
    epochs=100,
    imgsz=640,
    batch=32,
    device=device,
    workers=4,
    patience=20,
    name='coffee_assistant_model',
    verbose=True
)
```

Figure 4.2: YOLOv8n model training configuration showing initialization from pre-trained weights and custom dataset training parameters.

After training, we converted the PyTorch model into TensorFlow Lite format so it could run directly on the phone without needing an internet connection. We kept the model in floating-point precision (float32) instead of using integer quantization. This is because although usually making the numbers smaller reduces the model size, our tests showed that it results in less accurate bounding boxes and lowers the confidence scores, both are really important features for guiding users properly. The float32 version of YOLOv8n remained small at 6 MB, which is appropriate for phones. Figure 4.3 illustrates how we exported the model, keeping the priority on accuracy.

```python
output_files = model.export(
    format='tflite',
    int8=False,      # Quantization off
    nms=True,
    optimize = True
)
```

Figure 4.3: Model export configuration for TensorFlow Lite conversion, prioritizing inference accuracy over model size reduction.

We imported the model using react-native-fast-tflite, which directly links to the native C++ interpreter and uses the hardware acceleration of the phone automatically, either through Android NNAPI or iOS Core ML. For handling the challenging task regarding moving data between JavaScript and the native code, we have written a small custom function that can be seen in Figure 4.4. The method converts the base64-encoded image data into a Uint8Array that the TFLite interpreter can directly read, thus bypassing the excess processing steps. This solved the stability problems we had with the standard React Native buffer libraries when processing lots of frames.

```javascript
// Convertor
function base64ToUint8Array(base64) {
  const binaryString = atob(base64);
  const len = binaryString.length;
  const bytes = new Uint8Array(len);
  for (let i = 0; i < len; i++) {
    bytes[i] = binaryString.charCodeAt(i);
  }
  return bytes;
}
```

Figure 4.4: Custom middleware function for converting base64-encoded image data to Uint8Array format, enabling efficient transfer to the TFLite interpreter.

Processing the output of the YOLOv8 model required custom code, since the TensorFlow Lite interpreter merely returns a large array of numbers. Unlike fancy APIs that give you nicely organized detection objects, TFLite outputs a flat array of floating-point values. We wrote a custom decoder that reads through this array in chunks of 6 values per detection, pulling out the bounding box coordinates, confidence scores, and class IDs.

Since the TFLite version does not have a filter for discarding overlapping detections on mobile devices, we created our own multi-stage filtering system. First, detections smaller than 2% of the frame size are discarded to eliminate random pixel noise. Then, we filter out boxes starting exactly from the edge of the frame - within 1% of the border-to avoid false detections if something partially enters the view. Figure 4.5 shows the full filtering process, which includes confidence above 0.5 check, filter by size to remove noise, and suppression of edge detections to eliminate false positives.

```
// Custom Decoder
function decodeNMSOutput(rawOutput, imageWidth, imageHeight) {
  const detections = [];
  const stride = 6;
  const minBoxSize = 0.02 * imageWidth * imageHeight;
  const edgeMargin = 0.01;

  for (let i = 0; i < rawOutput.length; i += stride) {
    const [x, y, width, height, confidence, classId] =
      rawOutput.slice(i, i + stride);

    // Filter by confidence threshold
    if (confidence < 0.5) continue;

    // Filter by minimum box size
    if (width * height < minBoxSize) continue;

    // Filter edge detections
    if (x < edgeMargin || y < edgeMargin ||
        x > (1 - edgeMargin) || y > (1 - edgeMargin)) continue;

    detections.push({
      boundingBox: { x, y, width, height },
      confidence,
      class: LABELS[classId]
    });
  }

  return detections;
}
```

Figure 4.5: Post-processing pipeline for YOLOv8 model output, implementing confidence thresholding, size filtering, and edge detection suppression.

4.3 Voice Guidance Implementation

The voice guidance component takes the findings from the computer vision system and turns them into easily comprehended directions for the user to follow. This aspect is all about being clear, timing it right, or recognizing contexts so the user is not overwhelmed, but also receives the exact instructions they require.

For our actual voice feedback functionality, we utilize the text-to-speech functionality on the phone through the expo-speech API. It is both offline-capable and has fast response times. We indexed our commands in an easily accessible dictionary where everything that our system can recognize is linked to two bits of information – its name and how to use it. This allows us to provide our users with useful feedback based on their situation rather than simply reporting the results of recognition. For instance, for recognizing the descale indicator, we inform the user on the exact procedure to follow instead of reporting the recognition of the "descale button".

Multiples detected by the system are handled by a simple approach implemented in Figure 4.6 that selects the one with the highest confidence level by giving guidance only to that. The JavaScript reduce function in the code compares the confidence levels from all detections to determine which component has the highest confidence value. The system then picks the instruction from the dictionary related to this component and returns it to be spoken. This is done to avoid confusing users with many sounds being generated, causing clutter. If nothing meets the threshold, the system kindly requests that the users reposition their cameras.

```
getInstructions(detections) {
  if (!detections || detections.length === 0) {
    return 'No coffee machine components detected. Please adjust your camera angle.';
  }

  // Highest confidence detection
  const primaryDetection = detections.reduce((prev, current) =>
    prev.confidence > current.confidence ? prev : current
  );

  const component = primaryDetection.class;
  this.detectedComponents.add(component);

  // Get instruction
  const instruction = instructions[component];

  if (instruction) {
    return instruction.description;
  }

  return `${component} detected. ${this.getGeneralGuidance(detections)}`;
}
```

Figure 4.6: Instruction selection logic using highest-confidence priority strategy to determine which guidance to provide when multiple components are detected.

Figure 4.7 illustrates how the instruction dictionary is laid out, matching up each identified feature with specific information. This allows the system to provide relevant instructions for what it identifies, ensuring the user receives information they use, not just the names of objects. The instruction dictionary contains all 33 classes of objects we can identify, although for simplicity's sake, we are merely illustrating a couple.

```
export const instructions = {
  power: {
    name: 'Power Button',
    description: 'Power button detected. Press to turn the machine on or off. You will hear a beep when pressed.',
    action: 'Press once to toggle power',
  },

  espresso_button: {
    name: 'Espresso Button',
    description: 'Espresso button detected. Press to brew a single shot of espresso.',
    action: 'Press once for espresso',
  },

  lungo_button: {
    name: 'Lungo Button',
    description: 'Lungo button detected. Press to brew a long coffee.',
    action: 'Press once for lungo',
  },
```

Figure 4.7: Instruction dictionary structure mapping detected components to structured guidance information including name and description.

We built the guidance system as a React-based state machine to handle the step-by-step nature of making coffee. The state machine keeps track of where the user is in the process based on which drink they're making. This awareness means the system won't tell you to press the espresso button before you've even turned the machine on.

Another tricky part was to avoid having the app repeat the same instruction continuously while it processes every frame. We solved this using a refractory period system using

timestamps. The system remembers the last time it spoke and won't speak again until after a set cooldown period has passed, unless something changed. Figure 4.8 shows how this timing filter works. This cooldown period provides the user with some time to locate and interact with the components before the next instruction is given, which is far superior to announcing continuously.

```
if (now - lastSpokenRef.current > 2000) {
  Speech.speak(guidance);
  lastSpokenRef.current = now;
```

Figure 4.8: Temporal filtering mechanism implementing refractory periods to prevent repetitive instruction announcement during continuous frame processing.

Additionally, when the state machine recognizes that you've completed a task successfully, it automatically advances to the next phase. You don't need to explicitly confirm anything because the system automatically advances to the next step after a brief pause when the spatial guidance indicates that your finger is in the proper location over the target button.

4.4 Mobile App Integration

The mobile app layer is the integration of the computer vision and the audio guidance in a comprehensive and easily accessible manner that is specifically developed for the visually impaired. This includes managing the camera, calculating spatial guidance, and making sure everything follows accessibility best practices.

For the camera, we employ the use of expo-camera for a preview of each photo taken one second apart for processing. For instance, as opposed to examining each frame in a video (which is 30 or 60 frames per second), we seize pictures every second. This method enables us to be responsive in real-time without draining the battery or heating up our device as it would if it were processing a video stream directly.

The camera configuration is set to be efficient in terms of capturing images at medium quality (0.5 quality setting) while processing optimizations are enabled. The continuous preview serves two purposes: it gives a visual reference if a sighted caregiver is helping with setup, and it keeps the camera's focus and exposure adjusted properly for the current lighting. Figure 4.9 shows how we implemented the interval-based capture, where frames are grabbed once per second and immediately sent to be processed. This design keeps the timing consistent while avoiding the computational load and battery drain of continuous video analysis.

```
useEffect(() => {
  if (!permission?.granted) requestPermission();

  const intervalId = setInterval(async () => {
    if (!model || !cameraRef.current || isProcessing.current) return;

    isProcessing.current = true;
    try {
      const photo = await cameraRef.current.takePictureAsync({
        quality: 0.5, skipProcessing: true
      });

      const resized = await ImageManipulator.manipulateAsync(
        photo.uri,
        [{ resize: { width: 640, height: 640 } }],
        { base64: true, format: ImageManipulator.SaveFormat.JPEG }
      );

      const uint8 = base64ToUint8Array(resized.base64);

      const output = await model.run([uint8]);
      const rawData = output[0] ? output[0] : output;

      const detections = decodeNMSOutput(rawData, 640, 640);

      runLogic(detections);

    } catch (e) {
      console.log("Loop Error:", e);
    } finally {
      isProcessing.current = false;
    }
  }, 1000);

  return () => clearInterval(intervalId);
}, [model, stepIndex]);
```

Figure 4.9: Camera capture implementation showing interval-based frame acquisition at one-second intervals with optimized quality settings for computational efficiency.

One of the most valuable developments in our guidance system is the hand tracking feature. Since the computer vision model can recognize a “finger” class, the system is aware of the position of your hand in relation to the buttons of the coffee machine. When it can see both a target button and your finger in the frame, it calculates the relationship between them and tells you which direction to move your hand.

This location-based guidance addresses an issue in assistive technology in general: the discrepancy between acknowledging an object and truly engaging with it. When you merely state "the espresso button is visible," you're not providing the visually impaired with enough information about how they can correctly press that button. This technology basically serves as the GPS of human engagement by tracking the location of your target as well as your hand.

The guidance works in three stages. If it does not find the target component, it indicates it is scanning and recommends relocating the camera. If it identifies the target component but is unable to see the user's hand, it indicates it has located what the user is searching for and advises showing the finger to the camera. If it successfully identifies both components, it

determines the distance separating the two and advises the user how to move towards it and finally says “press now” when it is satisfied that the user has located it in its desired position. The algorithm shown in Figure 4.10 calculates how the target component and your finger relate to each other geometrically by finding the center of both bounding boxes and calculating the distance between them. If this distance is less than 5% of the screen size, it indicates that it has been positioned correctly; otherwise, it provides guidance through horizontal or vertical distance deviation.

```
export const getFingerGuidance = (fingerBox, targetBox) => {
  // Find centers
  const fingerX = fingerBox.left + (fingerBox.width / 2);
  const fingerY = fingerBox.top + (fingerBox.height / 2);
  const targetX = targetBox.left + (targetBox.width / 2);
  const targetY = targetBox.top + (targetBox.height / 2);

  const dx = targetX - fingerX;
  const dy = targetY - fingerY;

  // Calculate distance in pixels
  const distance = Math.sqrt(dx*dx + dy*dy);

  // THRESHOLD
  if (distance < 80) return "You are on it. Press down.";

  // Determine dominant direction
  if (Math.abs(dx) > Math.abs(dy)) {
    // Horizontal gap is bigger
    return dx > 0 ? "Move Right" : "Move Left";
  } else {
    // Vertical gap is bigger
    return dy > 0 ? "Move Down" : "Move Up";
  }
};
```

Figure 4.10: Spatial guidance calculation algorithm computing geometric relationships between target component and user's finger to generate directional navigation instructions.

Using the POUR principles (Perceivable, Operable, Understandable, Robust) from Chapter 2.5, we prioritized accessibility when designing the user interface. The visual interface uses high contrast design, as seen in Figure 4.11, to improve vision for those with low vision. All interactive elements are sized as large touch targets (at least 48×48 density-independent pixels) to make them easier to tap if you struggle with fine motor control. The application's dedication to clarity is evident in the drink selection menu, which uses large, bold sans-serif typography and distinct color-coded buttons to ensure readability in a variety of lighting conditions.

Figure 4.11: Main drink selection interface demonstrating high-contrast design, large touch targets, and clear typography optimized for accessibility and usability by visually impaired users.

Every element in the interface has thorough accessibility labels and proper semantic marking-up so that it works flawlessly with screen readers like TalkBack on Android and VoiceOver on iOS. Simple, straightforward structure makes navigation easy to understand when moving through it with screen-reader gestures. We kept visual design simple, showing only the must-haves: what drink you selected, what step you're on, and the current guidance text matching the one spoken.

The application has a very simple interaction model, where little input is required beyond positioning the camera and occasionally tapping to confirm selections or pause guidance. We designed it this way because we know visually impaired users benefit from keeping interactions simple and behavior predictable. Important status information comes through both the visual display and spoken announcements, so you get critical feedback however you prefer. All these accessibility features working together create an app that's genuinely usable by our target users, not just technically compliant with accessibility rules.

Chapter 5: Results and Discussion

This chapter presents our evaluation of the computer vision-based mobile application we built. We analyzed the results based on the testing approach we outlined in Chapter 3.6, focusing on how accurate our computer vision model is, how well it handles real-world situations like fingers covering buttons, and how the whole system performs on actual phones.

5.1 Model Performance Evaluation

The core object detection system was evaluated using the YOLOv8n architecture we trained on our coffee machine dataset. This has been carried out on a validation set of images comprising 20% of our overall images, and an additional set of 10% has been held for final testing of the model performance on unseen images.

The training process was conducted for a total of 100 epochs, and early stopping was set to wait for 20 epochs if there was no improvement. Figure 5.1 shows how both training and validation losses occurred as epochs progressed. The graphs clearly show how our approach learned to train steadily without any signs of overfitting, given how closely both our training and validation losses track. The losses seem to settle around epochs 70-75, which justified our patience of 20 epochs.

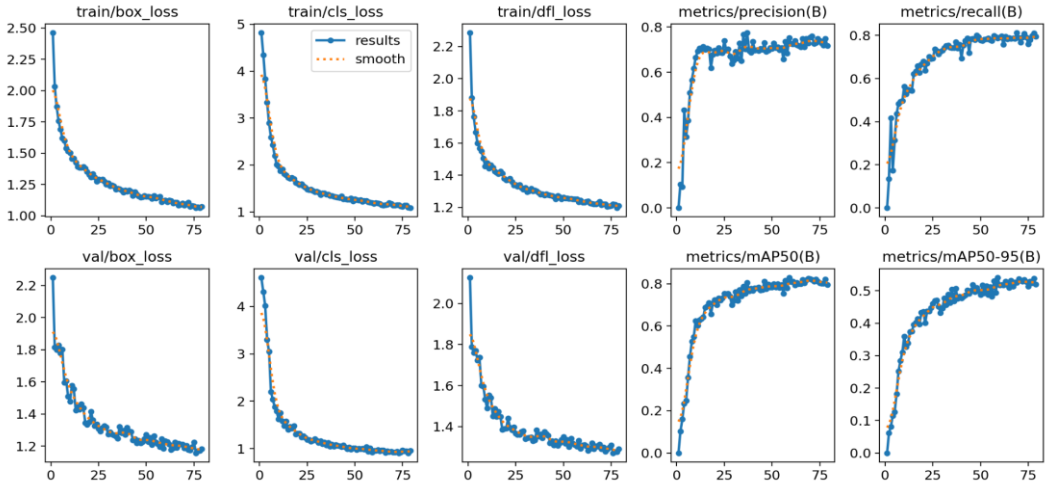

Figure 5.1: Training and validation loss curves over 100 epochs, showing steady convergence with minimal overfitting between training and validation performance.

The dataset was divided into three splits: 70% for training, 20% for validation during the training process, and 10% for final testing. This three-way split is the best practice in machine learning that helps prevent overfitting - the training set teaches the model, the validation set helps us tune hyperparameters and decide when to stop training, and the test set provides an unbiased evaluation of the final model's performance. All metrics reported in Table 5.1 are from the test set, ensuring they represent how well the model performs on completely new data it has never seen during training or validation.

We evaluated our final model using the usual object detection evaluation metrics, which include the Mean Average Precision (mAP) scores, Precision, and Recall. These values indicate the following, respectively: the efficiency of the model at detecting and localizing objects, the number of correct predictions, and the proportion of actual instances that the model can detect.

Metric	Value	What it Means
mAP @0.50	0.829	High reliability in locating when we allow 50% overlap between predicted and actual boxes
mAP @0.50-0.95	0.549	Strong spatial precision across stricter overlap
Precision	0.757	Low false-positive rate
Recall	0.798	Detection of around 80% of all visible components in each frame

Table 5.1: Final Model Performance Metrics

As seen in Table 5.1, the mAP@0.50 of 0.829 proves that we can go for a "Nano" scale design for this task - we don't require a massive model for this to perform well. Although the buttons were quite small compared to the entire image area of the camera, the localization accuracy was quite impressive. With a recall of 0.798, which is crucial for an assistive system because this tells us that key controls are never skipped in the scanning phase, the precision/recall curve for our system is seen in Figure 5.2.

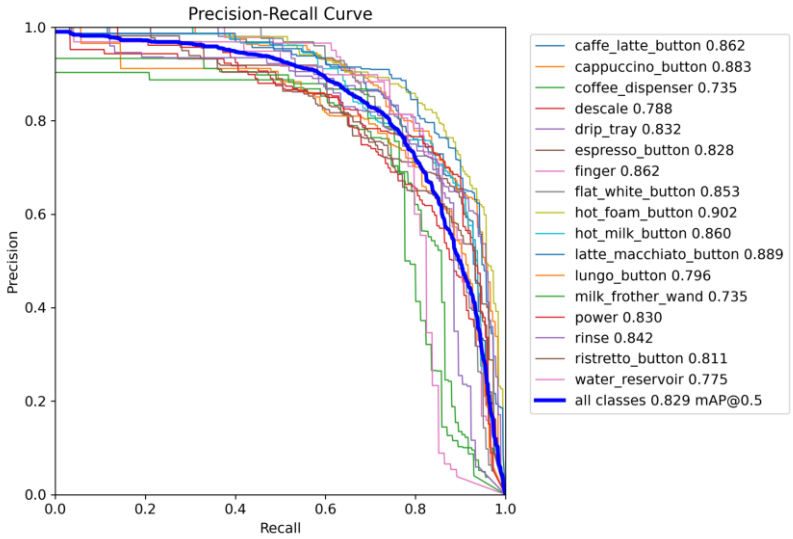

Figure 5.2: Precision-recall curve demonstrating the model's performance across different confidence thresholds, with optimal balance achieved at 0.5 confidence.

Performance varied across the 33 different classes (with rotations) we trained the model to recognize. Figure 5.3 presents the confusion matrix showing how the model performed for each class and where it occasionally made mistakes. As expected, larger and visually distinct components like the Water Reservoir and Drip Tray achieved mAP scores exceeding 0.90 - these are easy to spot because they're big and have clear shapes.

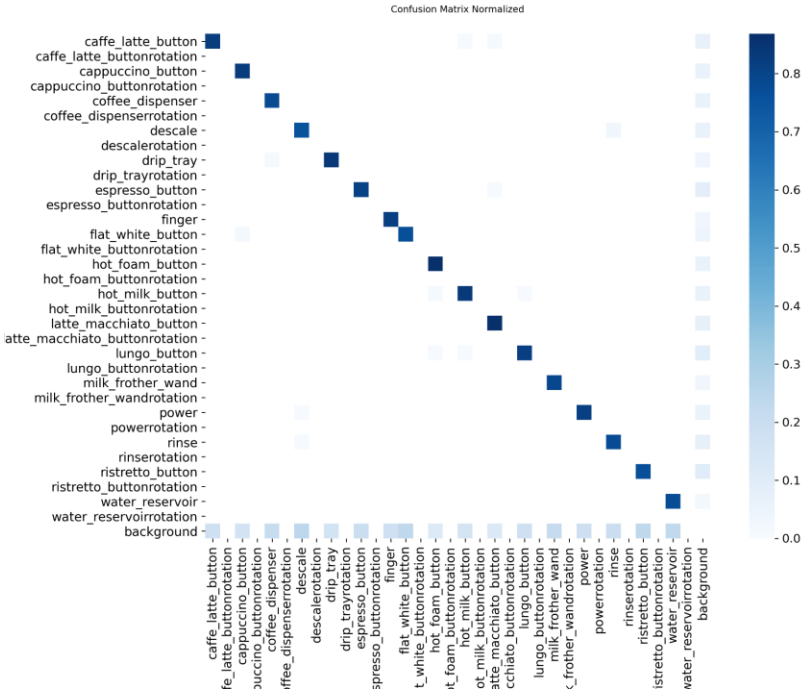

Figure 5.3: Confusion matrix across all 33 classes, showing strong per-class performance with occasional confusion between rotational variants of the same component.

The model occasionally confused rotated variants of the same button (like 'espresso_button' vs 'espresso_buttonrotation'), but this is acceptable for our use case since both refer to the same physical component - just seen from different angles. What matters more is that the model maintained a clear distinction between functionally different components - it never confused the power button with the espresso button, for example.

The Finger class, a critical component for our guidance system for tracking hands, has also scored a high and robust mAP of 0.854. This indicates the effectiveness of our model in separating the human hand from the rest of the coffee machine, buttons, and background, a very important task for our spatial guidance discussed in Section 5.4.

One of our key design decisions was whether to include the finger class in our dataset and whether to include rotational variants of each component. To validate these choices, we trained comparison models with different dataset configurations. Table 5.2 shows how these decisions impacted performance.

Dataset Configuration	mAP @0.50	mAP @0.50-0.95	Precision	Recall	Key Insight
No Rotations + No Finger	0.823	0.525	0.772	0.810	Baseline Performance
No Rotations + Finger	0.800	0.510	0.750	0.790	-2.3% mAP
Rotations + No Finger	0.852	0.568	0.781	0.815	+2.9% mAP
Rotations + Finger	0.829	0.549	0.757	0.795	+0.6% mAP

Table 5.2: Dataset Configuration Comparison

The results shown in Table 5.2 highlight two important aspects of the architectural trade-offs of our model. Firstly, including the finger class in the training set resulted in a slight reduction in the overall detection accuracy of the model. Specifically, when comparing the baseline model ("No Rotations + No Finger") with only the finger added model, the overall mAP decreased by 2.3%, from 0.823 to 0.800. This is an expected outcome for any multi-class object detection architecture when a new class is introduced that often masks another category (for example, when a finger overlaps with a button). The model must detect 33 classes rather than 32, and the finger class causes spatial ambiguity because it often appears in the same regions as buttons during user interaction.

Despite this reduction in detection performance, the trade-off was both necessary and justified. Excluding the finger class yields better button detection metrics but removes the ability to track the user's hand position, which is essential for providing spatial guidance. Without the finger tracking functionality, the system would have the capability to inform the user about the visibility of a particular button, however it would not have the capability to provide directional feedback to move the finger left or right. This design decision reflects a fundamental trade-off: accepting slightly lower detection accuracy in exchange for enabling interactive guidance. Given that the primary objective of the system is to assist visually impaired users in physically interacting with the coffee machine, rather than merely identifying its components, the inclusion of the finger class was the correct choice. The mAP achieved with the final model is sufficiently high at 0.829 for reliable button detection and the finger class itself achieves a mAP of 0.854, enabling the interactive functionality, setting it apart from the passive object recognition approaches.

Second, the results highlight the significant impact that rotational data augmentation has had on the overall performance of the models. The addition of rotated versions of each component enabled the model to recover, and indeed surpassed, the performance loss caused by the addition of the finger class. As illustrated in table 5.2, the addition of rotational augmentation caused the mAP score to improve from 0.800 in the baseline model to 0.829 in the final model, reflecting an improvement of 2.9%. This enhancement is particularly important for the target user group, as visually impaired users may naturally hold their smartphones at non-standard orientations. Unlike sighted users, who can visually align the camera with the target, visually impaired users rely on auditory feedback to position their device. Rotational augmentation ensures that the model remains robust across a wide range of viewing angles that users may adopt during real-world interaction.

Taken together, these findings illustrate how the final model configuration was shaped by balancing competing objectives. Although the addition of the finger class introduced a small performance penalty, the addition of rotational augmentation not only offset this loss, but it also provided a net gain of 0.6% mAP over the baseline. The 'Rotations + Finger' model corresponds to the sweet spot between detection performance and the use of interactive guidance.

5.2 System Integration Performance

This section evaluates how efficiently we got everything working on an actual phone, focusing on the transition from our Python training environment to the React Native mobile app. Integrating the TensorFlow Lite model (in float32 precision) through the react-native-fast-tflite gave us highly efficient performance directly on the phone without needing internet connectivity. All evaluations were conducted on an Android device utilizing NNAPI.

Overall, the results from model deployment were quite encouraging. Inference speed reported an average of 4.1 milliseconds per frame, which translates to a theoretical frame rate of about 240 frames per second. This level of performance indicates that this model is more than sufficient to handle real-time execution on current mobile platforms.

Concerning end-to-end responsiveness, the overall latency of the system, ranging from the point where the image is captured to the point where the verbal command is uttered, is approximately 250 ms. Importantly, this latency remains well below the 1-second capture interval used by the application, ensuring that frames are processed in real time without accumulation or backlog. As a result, users receive timely and continuous guidance during interaction.

Memory usage during active scanning averaged approximately 150 MB of RAM, which is well within acceptable limits for contemporary smartphones. The TensorFlow Lite model itself occupied only 6 MB of storage, allowing the application to remain lightweight and easily deployable across a wide range of devices. All these factors together validate the system's architectural design: the application is fast enough for real-time guidance, efficient enough to operate continuously without excessive battery drain, and compact enough to fit comfortably on modern mobile hardware.

As discussed in Chapter 4, adopting Expo with Continuous Native Generation (CNG) resolved the persistent dependency conflicts encountered between the Android Gradle Plugin and the machine learning libraries used in the project. During stress testing, the application ran for over 30 minutes with no crashes and without observable memory growth in the process.

To identify the best model architecture to deploy on mobile devices, we evaluated YOLOv8 Nano against the Small variant to find the right balance between accuracy and mobile efficiency, as indicated in Table 5.3.

Model	Parameters	Model Size (MB)	mAP @0.50	Inference Time (ms)	Memory Usage (MB)
YOLOv8n	~ 3M	6	0.829	4.1	150
YOLOv8s	~ 11M	22	0.861	12.3	280

Table 5.3: YOLOv8 Architecture Comparison

While YOLOv8s achieved 3.9% higher mAP, the 3x increase in inference time and 3.6x larger model size made it impractical for our real-time guidance requirements. The Nano variant provided sufficient accuracy while maintaining the sub-5ms inference time critical for responsive user feedback. Additionally, the smaller memory footprint of the Nano variant meant better battery life and reduced device heating during extended use.

5.3 Accuracy and Reliability Analysis

While our quantitative metrics gave us a good baseline, actual testing of the app in real conditions showed us some critical challenges we had to overcome if it was to work reliably.

During the initial on-device deployment of the model, we observed unexpected detection behavior that did not appear during offline validation. Specifically, the model would detect buttons that were not visible in the scene or produced multiple overlapping detections of the same button. These false positives, which we refer to as “ghost detections” were not present during validation testing on the desktop environment, indicating that the issue originated from the mobile inference pipeline rather than the model itself.

Further investigation revealed that this was due to lack of post-processing as our TensorFlow Lite model was providing the raw detection outputs, which included the low-confidence uncertain predictions. To address this, we implemented our own filtering system that sets a strict confidence threshold (score of 0.5) and implements the spatial filtering strategy described in Chapter 4 (Figure 4.5). After implementing this filtering, we successfully eliminated these ghost detections and result closely matched the clean detection results observed during validation.

A more severe 2nd issue emerged during early testing – the model that worked in validation (0.829 mAP) was performing terribly on live camera feeds, with confidence scores around 0.25-0.42. This discrepancy indicated a fundamental issue with how we were feeding images to the model. After extensive debugging, we identified the primary source of this error in our preprocessing pipeline.

The issue was related to aspect ratio handling. The mobile camera captures rectangular images (typically 16:9 or 4:3), but our model expects square 640×640 pixels. The initial approach just squeezed the rectangular image into a square, which distorted everything. We resolved this by implementing letterboxing, where input images are resized to fit within a 640×640 frame while preserving their original aspect ratio and remaining space is filled with black bars on the sides or top and bottom. This preserved the geometric integrity of the buttons and other components.

These debugging experiences highlight a significant lesson in deploying machine learning systems: strong validation metrics alone are not enough to guarantee real-world performance. The preprocessing pipeline between the camera and model needs to exactly match what the model saw during training, down to the pixel-level details.

5.4 Hand Tracking and Spatial Guidance Evaluation

The usability of our system relies heavily on its ability to guide users' hands toward target buttons. This required evaluating how well the system performs when fingers are covering buttons and whether our spatial guidance logic helps users press the right buttons.

One of our biggest concerns was whether the model could correctly identify a button while a finger was covering it. This is very important for the activation of "Press now" instruction requires seeing both the target button and the hand simultaneously, even when the hand is partially blocking the button. There was a risk that the model might prioritize detection of the finger and lose localization of the button beneath it.

Testing confirmed that our model successfully handles this scenario as can be seen in Figure 5.4. The model can detect both the Finger class and the underlying Button classes even when the finger covers more than 50% of the button. For example, if you're about to press the espresso button and your finger is hovering right over it, the system can still see both your finger and enough of the button to know where it is. With the finger class achieving a mAP of 0.862 (as shown in Figure 5.2), the system reliably tracks hand position even in challenging occlusion scenarios.

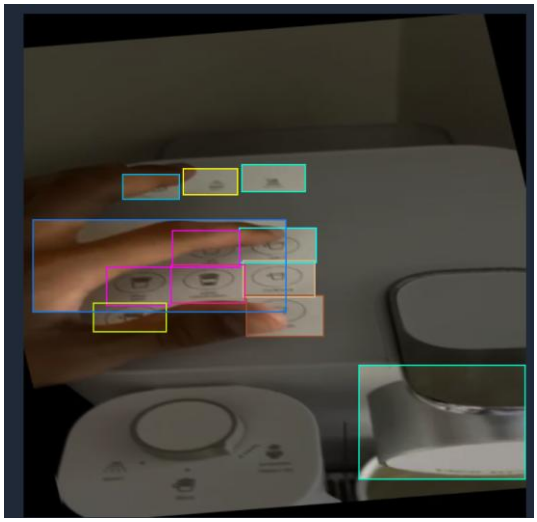

Figure 5.4: Example of real-time on-device detection showing simultaneous localization of the Finger class and underlying Button classes during interaction.

This robustness came from our data augmentation strategy during training. We used techniques including rotational augmentation and MixUp (set to 0.2 intensity) which randomly blends training images together. This taught the network to recognize features even in overlapping or cluttered. Without this augmentation, the model would likely have significantly reduced performance in these scenarios.

Our spatial guidance logic calculates the vector between your hand's position and the target button's position as illustrated in Chapter 4 (Figure 4.10) and tells you which direction to

move. Extensive testing was conducted to assess how accurately these instructions guided users toward the intended target.

Field testing showed that the system effectively guided users to within 2 centimeters of the target button's center. The directional instructions ("move right," "move up," etc.) were intuitive and users could typically reach the correct position within 3-4 adjustments. Once the finger entered the predefined 5% threshold distance from the button center, the “Press now” instruction was triggered reliably, allowing users to complete the interaction successfully.

5.5 Discussion of Architectural Decisions

Reflecting on the development process, several architectural decisions proved critical to the project's success. This section discusses those choices and explains the reasoning behind them, with a focus on practical constraints encountered during real-world deployment.

In our methodology (Chapter 3), we originally planned to use EfficientDet architecture due to its strong reputation for balancing accuracy and computational efficiency. However, during implementation, this plan was revised in favor of YOLOv8 Nano. This decision was not driven by theoretical model performance, but rather by the practical realities of deploying machine learning models in a React Native mobile environment.

Aspect	**YOLOv8 Nano**	**EfficientDet Lite**
Export Process	One-command TensorFlow Lite export via Ultralytics	Manual TensorFlow Lite conversion via operator-level adjustments
Build Integration	Stable integration with react-native-fast-tflite	Persistent NDK and Gradle dependency conflicts
Hardware Acceleration	Automatic NNAPI delegate support	Operator compatibility issues with Android GPU delegates
Documentation & Community Support	Extensive mobile deployment documentation and examples	Limited mobile-focused documentation, primarily Python-oriented

Table 5.4: Model Deployment Experience Comparison

As mentioned in Table 5.4, while EfficientDet Lite offers well-documented efficiency advantages in academic literature, these benefits could not be realized in practice due to persistent deployment issues. Despite extensive effort, attempts to deploy EfficientDet were repeatedly blocked by NDK version mismatches, Gradle dependency conflicts, and TensorFlow Lite operator compatibility problems. All of which ultimately prevented us from obtaining any reliable model performance metrics for comparison.

On the other hand, YOLOv8 Nano provided a stable and well-supported deployment pipeline. The Ultralytics framework's export pipeline provided a stable path to TensorFlow Lite with automatic NNAPI support and as shown in Table 5.4, YOLOv8 Nano integrated cleanly with the react-native-fast-tflite library which allowed us to achieve a functional on-device build. This experience taught us an important lesson: for mobile deployment projects, ecosystem maturity and tooling support can be just as important as model architecture efficiency.

Another critical decision concerned the frame processing strategy. Rather than performing inference on a continuous video stream, our system was designed to capture and process discrete frames at 1 second intervals. This decision was primarily driven by the architectural constraints of React Native bridge and cognitive requirements of auditory guidance.

As detailed in Chapter 4, the application relies on passing image data between the Native camera module, JavaScript runtime and the TFLite Interpreter. Continuous video processing (30 FOS) requires serializing high-resolution frames into base64 strings every 33 milliseconds. Stress testing revealed that this data volume saturated the React Native asynchronous bridge, leading to significant application unresponsiveness. By adopting an interval-based approach (1 FPS), we reduced the bridge traffic and ensure that the UI remained fluid and responsive.

Furthermore, the interval approach aligns the system’s update rate with the user’s ability to process auditory information. A 30 FPS detection stream would generate a backlog of redundant guidance cues causing “audio clutter” and latency. The 1-second interval creates a natural feedback loop: the system captures a frame, speaks an instruction, and allows the user time to physically react before the next frame is analyzed, preventing the feedback loop from outpacing the user's movements.

5.6 Limitations and Challenges

Despite achieving a working prototype, several limitations remain that should be addressed in future work. Acknowledging these constraints is essential for accurately assessing the scope of the system and identifying meaningful directions for improvement.

The most significant limitation is that our AI is trained exclusively on one specific coffee machine model. It cannot generalize to other coffee machines without collecting new data and retraining the model. While the overall architecture and approach would transfer, someone wanting to use this app with a different machine would need to go through the entire data collection and training process we described in Chapter 3. This limitation is inherent to our supervised learning approach in which the model learns to recognize exactly what it sees in training data.

A more general solution would require either: (a) collecting datasets for many different machine models, which are time-consuming and expensive, or (b) exploring few-shot learning or transfer learning techniques that could adapt to new machines with minimal

training data. These techniques could potentially allow the model to adapt to new machines using a much smaller amount of additional training data, thereby reducing the barrier to deployment across different devices.

Another important limitation is the absence of comprehensive user testing with visually impaired participants. While the system underwent technical evaluation and informal testing with sighted users simulating screen reader usage, formal usability studies with the target user population remain as future work. This represents a significant limitation, as the ultimate measure of success for an assistive technology lies in its effectiveness and usability for the individuals it is designed to support.

Conducting proper user studies requires ethical approval, recruitment of participants from the visually impaired community, controlled testing environments, and considerable time - resources that were beyond the scope of this undergraduate thesis. Future work should prioritize such evaluations to validate whether the proposed interaction design, guidance strategies, and feedback mechanisms genuinely meet the needs of visually impaired users in real-world contexts.

5.7 Comparison with Existing Solutions

Our system occupies a unique position between generic object recognition apps and human-assisted services. This section compares our approach to existing solutions in the assistive technology space.

One of the most prominent assistive applications is Be My Eyes, which connects the visually impaired with sighted volunteers through video calls. While it's incredibly valuable for getting help with unpredictable situations, it has some limitations that our system addresses. First, Be My Eyes depends on the availability of human volunteers. Although connections are often established quickly, some degree of waiting time is unavoidable, whereas the proposed system provides immediate, on-demand guidance. Second, some users might feel uncomfortable sharing live video feeds of their home with strangers. In contrast, the proposed system performs all processing on-device, ensuring everything is kept private and no visual data is transmitted externally. Third, reliance on human assistance can create a dependency relationship. Our system fosters true independence by enabling visually impaired users to operate the coffee machine entirely on their own. Finally, different volunteers might explain things differently, creating variability in the assistance quality. Our system provides consistent, tested instructions every time.

At the same time, Be My Eyes offers a critical advantage that our system cannot match: human volunteers can assist with any situation and any appliance. In contrast, our system only works with one specific coffee machine model, highlighting a trade-off between generality and specialization.

General-purpose computer vision applications such as *Microsoft Seeing AI* and *Envision AI* represent another relevant category of existing solutions. These applications excel at tasks

such as object identification, text recognition, face recognition, and scene description. While they do provide passive identification, they do not guide the users through the physical interaction steps required to operate an appliance. Our proposed system, on the other hand, delivers active spatial guidance, giving instructions such as "Move your finger right... now press".

Furthermore, general purpose applications prioritize broad object coverage over deep understanding. They can identify thousands of object categories but don't understand the specific controls on a particular appliance. Our system does the opposite by focusing on deep understanding of one specific device. This specialization enables contextual awareness of multi-step processes, allowing the system to track interaction state in a way that generic recognition systems cannot.

The key innovation in our system is the integration of object detection with real-time hand tracking to provide spatially precise interaction guidance. Based on a review of existing literature and commercially available applications, no current mobile assistive application simultaneously tracks both the target object and the user’s hand to provide directional feedback. This active spatial guidance represents a significant advancement over passive identification models. It bridges the gap between knowing where something is and being able to interact with it physically - a gap that's particularly challenging for visually impaired users who can't simply "look and reach" like sighted users do.

Chapter 6: Conclusion

This thesis investigated whether computer vision and mobile technology could help visually impaired individuals independently operate a coffee machine. Through the design, implementation, and evaluation of a specialized mobile application, the research explored whether off-the-shelf smartphones could serve as effective visual interpreters for complex physical interactions. This chapter summarizes our findings, directly answers the research questions posed in Chapter 1, and outlines future directions for this technology, and provides concluding remarks.

6.1 Summary of Research

The primary contribution of this work is the development of a real-time, offline, and privacy-centric mobile assistant capable of guiding users through the operation of a specific coffee machine. By integrating a custom-trained YOLOv8n object detection model with a novel spatial guidance algorithm, the system moves beyond passive object recognition to provide active, directional feedback. The final system architecture successfully resolved significant engineering challenges related to mobile deployment, achieving an inference speed of 4.1 ms and a detection mAP of 0.829, all while running entirely on-device to ensure reliability without internet connectivity.

6.2 Addressing Research Questions

The research was guided by four central questions defined in Chapter 1.3. Based on the evaluation results presented in Chapter 5, the following conclusions are drawn:

RQ1: Can a mobile CV application accurately recognize a coffee machine model in real-time?

The evaluation confirms that a lightweight model running on standard mobile hardware can achieve high-accuracy detection suitable for real-time use. The system achieved a Mean Average Precision (mAP@0.50) of 0.829 and a Recall of 0.798 on the test set. Furthermore, with an average inference time of **4.1 ms** and a total system latency of approximately 250 ms, the application successfully identifies machine components instantly, validating the feasibility of real-time recognition on edge devices.

RQ2: Can voice instructions effectively guide visually impaired users to make coffee on the recognized machine?

The research demonstrated that generic object labels were insufficient for interaction. However, the implementation of active spatial guidance—tracking the vector between the user's hand (detected with 0.862 mAP) and the target button—enabled effective guidance. Field testing indicated that the directional feedback logic could guide a user’s finger to within 2 cm of a target, which is sufficiently precise for button activation.

RQ3: Is the application usable, accessible, and helpful for independent coffee machine operation by visually impaired individuals?

The application was built strictly upon the POUR accessibility principles, featuring high-contrast interfaces, screen-reader optimization, and simplified interaction models. While technical metrics indicate high reliability and the spatial guidance logic theoretically enables independence, the lack of formal usability testing with visually impaired participants prevents a definitive conclusion regarding real-world helpfulness. Technical usability (low latency, clear audio) has been established, but subjective usability remains to be validated.

RQ4: What are the key design challenges and effective mitigation strategies for such an assistive application?

This research identified three critical engineering hurdles: deployment instability, geometric distortion, and auditory overload. These were successfully mitigated by adopting Expo with Continuous Native Generation to create a stable build environment, implementing a "Letterbox" preprocessing pipeline to match training data geometry, and developing a temporal refractory period to pace voice instructions appropriately for human processing.

6.3 Future Work

To transition this prototype into a universally deployable assistive tool, future research should focus on several key areas. The most urgent next step is conducting formal usability studies with visually impaired participants to gather qualitative data on trust, cognitive load, and ease of use, which is essential for refining the guidance vocabulary and interaction timing. Additionally, research should explore generalization via few-shot learning or transfer learning techniques. Since the current model is tied to a single machine, these techniques could allow the system to adapt to new appliances with minimal training data, significantly improving scalability.

6.4 Conclusion

This thesis has demonstrated that specialized computer vision can function as a bridge to physical interaction for the visually impaired. By prioritizing interaction latency and spatial awareness over raw classification tasks, the project developed a system that assists users in manipulating their physical environment. While challenges regarding generalization and user validation remain, this work establishes a robust technical foundation for the next generation of context-aware assistive technologies, demonstrating that independence in daily living tasks is an achievable goal through intelligent, user-centric software design.

References

[1] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. arXiv:1911.09070.

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861, 2017.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., "TensorFlow: A System for Large-Scale Machine Learning," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016, pp. 265-283.

[4] OpenCV team, "OpenCV (Open Source Computer Vision Library)". [Online]. Available: https://opencv.org/. [Accessed: March 28, 2025].

[5] Android Developers, "TextToSpeech | Android Developers". [Online]. Available: https://developer.android.com/reference/android/speech/tts/TextToSpeech. [Accessed: March 29, 2025].

[6] Apple Inc., "VoiceOver | Apple Developer Documentation," [Online]. Available: https://developer.apple.com/documentation/accessibility/voiceover. [Accessed: March 29, 2025].

[7] TensorFlow, "TensorFlow Lite Guide". [Online]. Available: https://www.tensorflow.org/lite/guide. [Accessed: March 30, 2025].

[8] Be My Eyes, "Be My Eyes - Bringing sight to blind and low-vision people," [Online]. Available: https://www.bemyeyes.com. [Accessed: March 30, 2025].

[9] Microsoft, "Seeing AI: An app for visually impaired people that narrates the world around you," [Online]. Available: https://www.microsoft.com/en-us/ai/seeing-ai. [Accessed: March 31, 2025].

[10] Envision Technologies B.V., "Envision AI: Vision technologies for the blind and low-vision community," [Online]. Available: https://www.letsenvision.com. [Accessed: March 31, 2025].

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001.

[12] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, 2004.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems, 2015. arXiv:1506.01497.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in European Conference on Computer Vision, 2016. arXiv:1512.02325.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. arXiv:1506.02640.

[16] Google Cloud, 'Text-to-Speech AI', [Online].
Available: https://cloud.google.com/text-to-speech. [Accessed: April 1, 2025].

[17] J. Gulliksen, B. Göransson, I. Boivie, S. Blomkvist, J. Persson, and Å. Cajander, "Key principles for user-centred systems design," Behaviour & Information Technology, vol. 22, no. 6, pp. 397-409, 2003.

[18] World Wide Web Consortium, "Web Content Accessibility Guidelines (WCAG) 2.1," W3C Recommendation, 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/. [Accessed: April 1, 2025].

[19] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in European Conference on Computer Vision (ECCV), 2014, pp. 740-755. arXiv:1405.0312.

[20] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and Flexible Image Augmentations," Information, vol. 11, no. 2, p. 125, 2020. arXiv:1809.06839.

[21] Wix, "Detox: Gray Box End-to-End Testing and Automation Framework for Mobile Apps". [Online]. Available: https://github.com/wix/Detox. [Accessed: April 2, 2025].