DATA SCIENCE & AI LAB (BSCSS3001)

MILESTONE 3: Model Design, Architecture, and Pipeline Documentation

GROUP NO. 2

PRASHASTI SARRAF **(21f1001153)**
TANUJA NAIR **(21f1000660)**
BALASURYA K **(22f3002744)**
KARAN PATIL **(22f2001061)**
JIVRAJ SINGH SHEKHAWAT **(22f3002542)**

IITM BS Degree Program
Indian Institute of Technology,
Madras, Chennai,
Tamil Nadu, India, 600036

Vision Assist: Real-Time Navigation Support for the Visually Impaired

Model Design, Architecture, and Pipeline Documentation

1. Project Overview

Objective:

Develop a computer vision-based assistive system that interprets a real-world scene and provides context-aware navigational instructions (e.g., *"A car is moving towards you from the right"* or *"There's a static bench ahead, slightly to your left"*).

Key Capabilities:

- Detect objects in real time using an object detection model (trained/fine-tuned on COCO + custom data).
- Determine motion (static vs. moving objects) using temporal frame analysis.
- Estimate distance and direction of objects relative to the user.
- Generate context-driven spoken feedback via a text-to-speech (TTS) module.
- Cloud-based deployment for inference scalability and model experimentation.

The system integrates **YOLO-World** for open-vocabulary object detection, **ByteTracker/DeepSORT** for motion tracking, **MiDaS** for depth estimation, and **Coqui TTS (Tacotron2 + HiFi-GAN) + LLM or Rule Based Templates** for speech-based output. Together, these components form an intelligent assistive pipeline that detects, tracks, measures, and narrates environmental context to the user for safe and autonomous navigation.

The data used for training and fine-tuning the models is sourced from the **COCO dataset**, which provides labeled everyday objects, and **frames extracted from YouTube videos**, which capture real-world, dynamic scenarios. This combination ensures that the system can generalize well to diverse and unpredictable environments encountered in daily life.

Deployment Approach

The system will be hosted on the cloud (using platforms like AWS or GCP) for scalable processing. This eliminates hardware constraints during the prototype phase and allows smooth model experimentation and updates.

Future Possibilities (Refer Annexure A)

2. Dataset Preparation and Readiness

2.1 Dataset Overview

The project uses the MS COCO dataset (2017 release) as the primary source of annotated visual data. This dataset provides images with detailed object annotations (bounding boxes, segmentation masks, and class labels) across 80 categories. A smaller subset of this dataset has been sampled and curated for early experimentation and validation of the end-to-end pipeline.

Additional custom samples may be introduced in later stages to capture more real-world navigation contexts such as indoor corridors, sidewalks, or crosswalks (refer Annexure A for future dataset integration plans)

2.2 Data Organization and Splitting

The dataset has been structured to ensure clarity and ease of reproducibility. The following directory structure is maintained for the data used in this project:

- **Raw Data:** Original MS COCO images and JSON annotations.
- **Processed Data:** Resized, normalized, and filtered images ready for model input.
- **Samples Folder:** A small selection of images used for quick visual inspection and EDA.

The dataset split currently follows a ratio of 70% training, 20% validation, and 10% testing, ensuring balanced distribution across key object categories.

```
dataset/
│
├── raw/
│   ├── train2017/
│   ├── val2017/
│   └── annotations/
│
├── processed/
│   ├── train/
│   ├── val/
│   └── test/
│
└── samples/
    ├── input_frames/
    └── processed_frames/
```

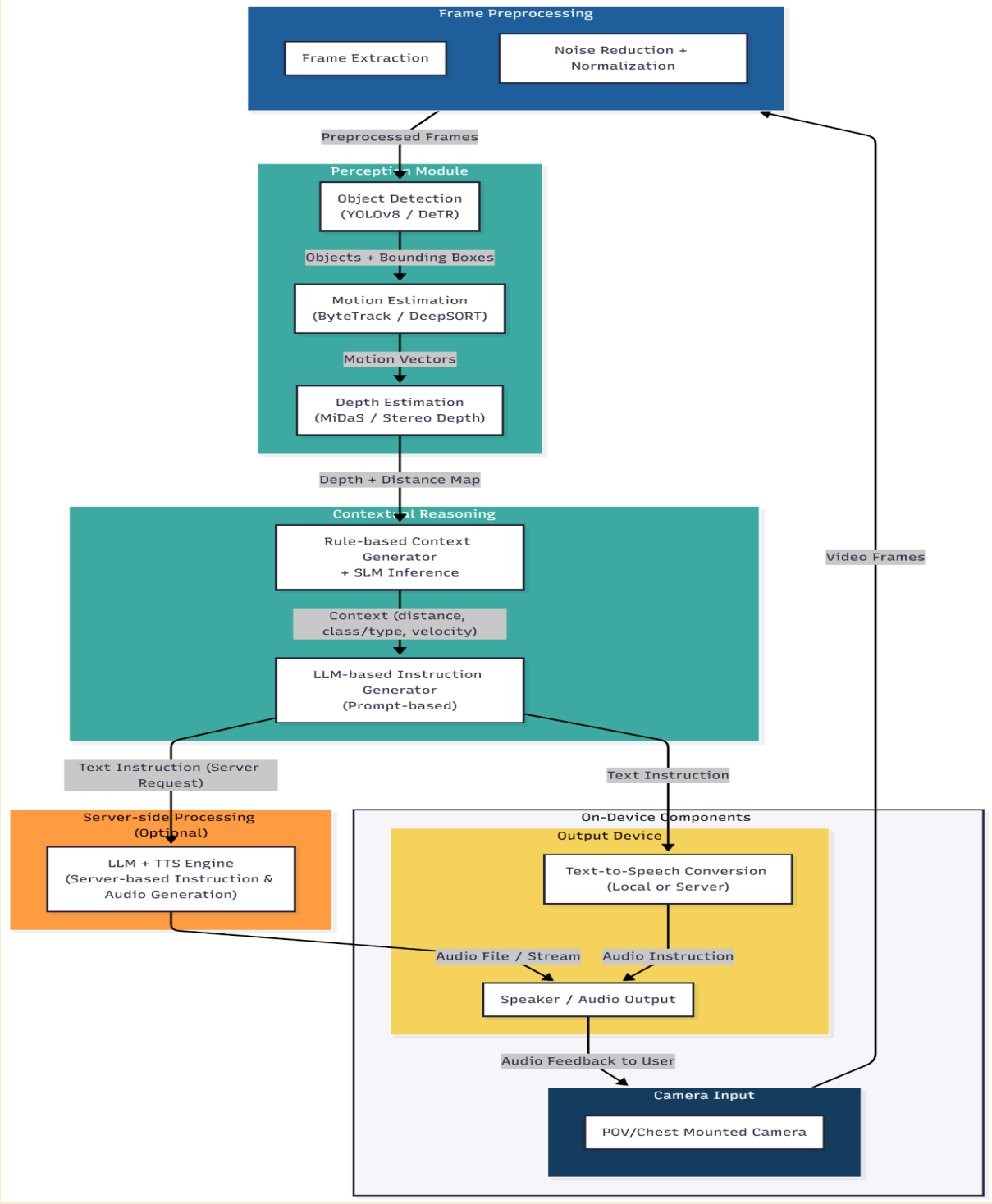
Example Samples ready for training(included only for sample purpose):

Image ID	Category	Bounding Box (x, y, w, h)	Data Type	Example Shape
000001.jpg	person	[120, 85, 60, 170]	image: uint8	(640, 640, 3)
000002.jpg	bicycle	[200, 140, 130, 90]	bbox: float32	(4,)
000003.jpg	traffic_light	[400, 120, 20, 50]	category_id: int	scalar

2. System Overview

Component	Purpose	Selected Model	Output
Input Preprocessing	Pre-process raw input from the camera to get image frames ready to send over the network for inference	Python Libraries and Custom code	Image frames ready to send over the network for inference
Object Detection	Detect and label surrounding objects	YOLO-World/YOLOv8/DeTr	Bounding boxes, object classes and confidence
Object Tracking	Maintain consistent object identity across frames	ByteTracker/DeepSORT	Unique object IDs and movement direction
Depth Estimation	Estimate relative distance of objects from the user	MiDaS	Depth map and distance approximation
Context Generator	Get context about the object based on distance, velocity, class, type for Human understandable context generation out of numbers	Rule Based + LLM Inference	Human Understandable context generated ready to pass on to TTS
Voice Output	Provide natural audio guidance	Coqui TTS (Tacotron2 + HiFi-GAN)	Spoken feedback

3. Model Architecture and Design



Overview

The overall model architecture follows a modular, multi-stage design that transforms raw visual input from wearable or mobile devices into actionable audio instructions. Each stage is optimized for real-time processing while maintaining accuracy and interpretability.

The architecture integrates multiple deep learning components — primarily object detection, motion tracking, depth estimation, and context reasoning — all of which contribute to generating natural language instructions for the user. A schematic view of the complete pipeline is shown in the above diagram.

1. Input and Preprocessing Module

Input Source: Real-time video stream from a chest-mounted or wearable camera.

Processing Steps:

- Frame extraction at 10–15 FPS
- Noise reduction and intensity normalization
- Resizing to standard model input dimensions (640×640)

Purpose: To prepare consistent, high-quality frames for downstream perception tasks.

2. Perception Module

This module is the visual intelligence core of the system. It performs object detection, motion tracking, and depth estimation to extract spatial and semantic features from each frame.

a. Object Detection

Model Used: YOLO(v8 or world) or DeTR (Detection Transformer), depending on deployment mode.

Architecture Highlights:

- Input: (640×640×3) image
- Convolutional backbone: CSPDarknet or ResNet-based encoder
- Feature Pyramid Network (FPN): Multi-scale feature extraction
- Output: Bounding boxes, class labels, and confidence scores

Activation: SiLU (Swish Linear Unit)

Loss Function: Composite of CIoU Loss (for bounding boxes) and BCE Loss (for classification)

b. Motion Estimation

- **Model Used:** ByteTrack / DeepSORT
- **Input:** Consecutive frame detections
- **Output:** Object trajectories and velocity vectors
- **Role:** Maintains temporal consistency and helps identify moving vs static entities.

c. Depth Estimation

- **Model Used:** MiDaS (Monocular Depth Estimation) or stereo-based CNN
- **Output:** Pixel-wise depth map representing relative object distance
- **Use Case:** Essential for understanding proximity and generating spatial context.

3. Contextual Reasoning Module

This stage fuses visual signals (object, motion, and depth) with rule-based and LLM-based reasoning.

a. Context Generator

- Combines structured features: *{object_label, distance, velocity, relative_position}*
- Applies rule-based logic to infer context, e.g.:
  - "Pedestrians approaching from right at 2 meters."
  - "Vehicle crossing ahead, 3 meters away."

b. Instruction Generator

- **Model Used:** Small Language Model (SLM) or an LLM (GPT / LLaMA) depending on compute mode.
- **Input:** Context string (JSON-like structured text)
- **Output:** Short, user-friendly instruction — "Move slightly left" or "Stop ahead"
- **Inference Type:** Prompt-based generation or fine-tuned lightweight LLM
- **Loss Function:** Cross-entropy loss (for text generation fine-tuning)

4. Audio Instruction Module

- **Text-to-Speech Engine:** Converts final text instruction into natural-sounding audio.
- **Deployment Options:**
  - On-Device (Low Latency): Lightweight models like VITS or FastSpeech2
  - Server-Side (High Fidelity): Larger models like Bark or Tacotron2
- **Output:** Speech waveform streamed to the speaker for real-time feedback.

5. Integration and Feedback Loop

All modules are orchestrated in a sequential data flow:

- The camera feed generates live frames.
- Frames are preprocessed and fed to perception modules.
- The contextual reasoning layer generates situational insights.
- LLM-based instruction generation translates insights into language.
- The TTS engine delivers real-time audio guidance.
- The feedback closes the loop, enabling continuous perception and response.

6. Model Training and Evaluation

Component	**Loss Function**	**Metric**	**Remarks**
Object Detection	CIoU + BCE	mAP (mean Average Precision)	Evaluated on COCO subset
Depth Estimation	L1 / Scale-Invariant Loss	RMSE, δ Accuracy	Evaluated on monocular depth data
Instruction Generation	Cross-Entropy Loss	BLEU / ROUGE	Evaluated on synthetic instruction pairs

4. Justification of Chosen Models

The model components selected for this system have been chosen based on a balance of accuracy, computational efficiency, and real-time suitability for assistive scenarios. Each model contributes a unique capability essential for contextual understanding and safe navigation in real-world environments.

Why YOLO-World (Object Detection)?

YOLO-World is selected for its open-vocabulary detection capability and strong generalization to unseen object classes. Unlike traditional YOLO variants trained on fixed label sets, YOLO-World leverages vision-language alignment, allowing it to recognize arbitrary objects described in natural language prompts.

This enables the system to dynamically adapt to user needs — for example, detecting specific items such as “bottle,” “bag,” or “bench” without retraining. The model’s high inference speed and support for on-device deployment make it ideal for real-time use on mobile or wearable hardware.

Key Advantages:

- Open-vocabulary detection using language-conditioned queries
- Real-time inference optimized for edge deployment
- High precision in cluttered scenes and varying lighting conditions
- Scalable for adding new prompts without retraining

Future Scope (Annexure Ref: A1):

If needed, YOLOv8 or Grounding-DINO could be used as alternative baselines to compare trade-offs in latency and accuracy during fine-tuning for domain-specific classes.

Why DeepSORT (Motion Tracking)?

DeepSORT (Simple Online and Realtime Tracking with Deep Association Metrics) ensures stable tracking of detected objects across consecutive frames. This consistency is crucial in a real-world assistive setup, where visual flicker or unstable detections can mislead users.

By associating object identities using appearance embeddings and motion prediction via a Kalman filter, DeepSORT provides reliable temporal coherence — for instance, continuously tracking a “car approaching from the left” or a “person moving ahead.”

Key Advantages:

- Low-latency, real-time tracking suitable for wearable devices
- Handles occlusion and re-identification effectively
- Enhances perception stability for moving environments

Future Scope (Annexure Ref: A2):

Integration of ByteTrack may be explored for enhanced association accuracy in high-motion outdoor scenes with multiple objects.

Why MiDaS (Depth Estimation)?

MiDaS adds **depth awareness** using just a regular camera. It estimates how close or far an object is and gives **distance-based alerts** such as *"obstacle two meters ahead"* or *"wall one meter away."*

This helps users **judge space and distance**, allowing them to navigate safely without colliding with nearby objects.

Real-time suitability: Moderate-High – single RGB input, fast inference (~20–25 FPS on GPU, slower on CPU).

Key Advantages:

- Requires only a single RGB frame (no stereo setup)
- Generalizes well across domains without retraining
- Lightweight versions available for real-time deployment

Future Scope (Annexure Ref: A3):

Depth Anything or FastDepth models could be considered for faster inference or hardware-accelerated optimization during later iterations.

Why Coqui TTS (Text-to-Speech Conversion)?

Coqui TTS is an advanced open-source text-to-speech framework derived from Mozilla's TTS project. It provides state-of-the-art speech synthesis capabilities using deep learning architectures such as **Tacotron 2**, **VITS**, and **Glow-TTS**, combined with high-quality **neural vocoders** like **HiFi-GAN** and **MelGAN**.

It generates clear, natural, and offline-capable voice feedback for detected scene descriptions, and enhances accessibility and usability in real-world assistive settings. Its modular design allows rapid experimentation and deployment, making it ideal for both cloud-based and **on-device (edge)** applications.

Real-time suitability: High – optimized for fast TTS inference on CPU/GPU for edge deployment.

Key Advantages:

- High-quality, natural-sounding voices
- Offline-capable — suitable for on-device TTS
- Fast inference with lightweight neural vocoders (HiFi-GAN, MelGAN)
- Customizable voice cloning or multilingual support

Future Scope (Annexure Ref: A4):

VITS and Bark-based models can be evaluated for enhanced prosody or expressive speech generation when integrated with emotional context reasoning in later phases.

5. Model Comparison: Advantages and Limitations

Task	Selected Model	Alternatives	Advantages of Selected	Limitations / Trade-offs
Object Detection	YOLO-World	DETR / ReDETR, SSD, Faster R-CNN, EfficientDet	• Open-vocabulary detection via CLIP embeddings → detects unseen objects • Real-time speed suitable for edge/mobile devices • High mAP and robust bounding boxes	• Slightly heavier than YOLOv8 • Requires GPU for best performance • Prompt sensitivity
Object Tracking	DeepSORT	SORT, ByteTrack, FairMOT	• Combines motion (Kalman filter) + appearance embeddings for stable tracking • Maintains consistent object IDs even during	• Can lose track under long-term occlusion • Dependent on detection quality

			short occlusions • Lightweight and widely used	
Depth Estimation	MiDaS	Monodepth2, DenseDepth, DepthFormer	• Accurate pixel-wise depth from monocular RGB • Robust to diverse environments (indoor/outdoor) • Lightweight for near real-time inference	• Relative depth only (needs scaling for absolute distance) • Some loss of detail for very small or distant objects
Text-to-Speech	Coqui TTS (Tacotron2 + HiFi-GAN)	Google TTS, Amazon Polly, VITS, Kitten TTS	• Open-source and offline capable → no internet dependency • High-quality, natural-sounding speech • Supports real-time inference on CPU/GPU • Flexible voice customization and multilingual support	• Initial model download size (~1–2 GB) • Slightly higher latency on CPU compared to cloud TTS APIs

Key Takeaways

- **YOLO-World** over DETR/SSD/Faster R-CNN/EfficientDet:
  Chosen for **open-vocabulary support** and **real-time edge inference**, unlike DETR which is slower and SSD/Faster R-CNN which are limited to predefined classes.

- **DeepSORT** over SORT/ByteTrack/FairMOT:
  Offers **appearance-based re-identification**, reducing ID-switches during motion and short-term occlusions, which basic SORT or ByteTrack cannot handle reliably.

- **MiDaS** over Monodepth2/DenseDepth/DepthFormer:
  Produces **high-quality depth maps across diverse scenes** using a single RGB frame, making it practical for indoor/outdoor navigation.

- **Coqui TTS** over VITS/Kitten TTS/cloud APIs:
  Provides **offline, real-time, high-fidelity voice synthesis** while being **open-source** and fully customizable — critical for privacy and assistive applications.

7. Way Forward

1. Quantize YOLO-World and MiDaS for **mobile deployment** (Jetson, Android).
2. Incorporate **spatialized audio cues** for directional awareness.
3. Add **speech recognition** for user commands ("find door", "avoid obstacle").
4. Integrate **Coqui TTS** for ultra-fast real-time speech synthesis in edge devices.

8. Conclusion

The final architecture — **YOLO-World + DeepSORT + MiDaS + Coqui TTS** — provides a robust, multimodal perception and guidance system for visually impaired navigation.

This setup balances accuracy, adaptability, and accessibility, achieving real-time environmental understanding, intuitive speech-based assistance, and enhancing independence for visually impaired users.

Annexure

Quick Links

Github Repository

Drive Data Storage Link (dsai project)

Annexure A

The architecture is designed to remain flexible. Future improvements may include:

- Using advanced models like DETR for better contextual detection or different models from the YOLO family as per the constraints and availability
- Refining instruction generation through multimodal or LLM-based reasoning.

- **Object Detection: YOLO-World, YOLOv8, and DeTr**

  Our architecture lists multiple object detection models to mitigate risks and align with the project's evolving needs.

- **Primary Choice (YOLO-World):** Our primary research target is YOLO-World due to its "open-vocabulary" detection capability. For our end-user (a visually impaired person), the ability to detect objects based on text prompts (e.g., "find a 'mailbox'" or "is there a 'pothole'?") without retraining the model is a significant, high-value feature.
- **High-Performance Baseline (YOLOv8):** If YOLO-World proves to be too computationally expensive or slow for real-time video processing on our target hardware, **YOLOv8** will be our primary alternative. YOLOv8 offers an exceptional balance of high-speed inference and state-of-the-art accuracy on standard object classes (like 'car', 'person', 'bench'). It serves as a robust and reliable baseline to ensure the core functionality of the system is met.
- **Alternative Architecture (DeTr):** The DEtection TRansformer (DeTr) is listed as a fundamentally different architectural approach. If we find that both YOLO models struggle with accuracy in extremely crowded or complex scenes with many overlapping objects, we will evaluate DeTr, as its transformer-based attention mechanism can be more effective in such scenarios.

- **Motion Estimation (Tracking): ByteTrack vs. DeepSORT**

  The ability to distinguish static from dynamic objects is critical. Both ByteTrack and DeepSORT achieve this, but they have different strengths.

- **Primary Choice (ByteTrack):** We plan to begin with **ByteTrack**. It is a more recent, state-of-the-art tracker that is often faster than DeepSORT. It excels at handling occlusions (when an object temporarily goes behind another) by retaining low-confidence detections, which is ideal for a busy, real-world video feed.

- **Alternative (DeepSORT):** If, during testing, we find that ByteTrack struggles with *persistent re-identification* (e.g., correctly identifying the *same car* after it has been hidden for several seconds), we will implement **DeepSORT**. DeepSORT's core strength is its use of a deep-learning-based appearance model for re-identification. While this can be slower, it may be necessary to provide a more stable and less "jumpy" tracking experience for the end-user, which is crucial for building trust in the system.

- **Text-to-Speech (TTS): Coqui TTS vs. Cloud APIs and Other Models**

  The choice of a TTS engine is critical for delivering clear, responsive, and reliable audio instructions to the user.

- **Primary Choice (Coqui TTS):** Our primary selection is **Coqui TTS**, specifically using a pipeline like **Tacotron2 (for spectrogram generation) + HiFi-GAN (for vocoding)**.
    - **Rationale:** The single most important factor for our project is **offline capability**. A navigation tool for the visually impaired cannot fail due to a lost internet connection. Coqui TTS is an open-source framework that runs entirely on-device (offline), eliminating this critical point of failure.
    - Furthermore, it provides high-quality, natural-sounding speech, which is essential for user comfort and comprehension. Its open-source nature gives us full control over the model, including voice customization and multilingual support, without any ongoing API costs or data privacy concerns.
- **Alternatives (Cloud-based: Google TTS, Amazon Polly):** These services offer state-of-the-art, extremely low-latency, and natural-sounding speech. However, they are **not a viable primary choice** for our core functionality because they are **entirely dependent on an internet connection**. We may consider them as an optional "high-quality online mode," but the system's fundamental operation must be offline.
- **Alternatives (Other Open-Source: VITS, etc.): VITS** is another strong open-source, end-to-end model that we will consider as a high-priority alternative. It is newer and can sometimes produce even more natural-sounding speech in a single, more compact model. We will benchmark Coqui's pipeline against VITS during implementation to select the best-performing model that meets our on-device latency requirements.

- **Accepted Trade-offs:** By choosing an offline model like Coqui TTS, we are knowingly accepting two trade-offs:
    - **Initial Size:** The models require an initial download (as listed, ~1-2 GB).
    - **CPU Latency:** Inference on a CPU will likely be slightly slower (higher latency) than a highly optimized cloud API call. We will need to manage this during implementation to ensure the audio feedback is delivered to the user without a noticeable or unsafe delay.