

# Describe My Environment: A Dual-Loop Vision-Language Accessibility System for Low-Vision Users

Akhil Nishad

*Master of Data Science*

*University of Michigan*

*Ann Arbor, MI, USA*

UMID: 38818750, Unique Name: akhlnis

**Abstract**—This paper presents ‘Describe My Environment,’ an end-to-end accessibility system that helps blind and low-vision users understand their surroundings through real-time object detection, scene captioning, and audio narration. The system combines a dual-loop computer vision pipeline with multiple user-facing entry points: a command-line application, a Jupyter notebook demo, and a full-stack web application built with FastAPI and Next.js for interactive testing. Users can run the system locally to stream camera input, receive hazard warnings at 30 frames per second, and trigger on-demand scene descriptions at approximately 2 frames per second on Apple Silicon hardware. Experiments on the COCO dataset and live camera feeds show that a YOLOv11n detector and BLIP captioning model meet a 500 ms latency target under local deployment, achieving a median end-to-end latency of 194 ms per frame while maintaining interactive performance. The report discusses design choices, performance trade-offs between the reflex and cognitive loops, practical limitations of browser-based deployment on constrained backends, and the lessons learned in exposing the pipeline through multiple interfaces to make the system easy to evaluate and extend.

**Index Terms**—accessibility, computer vision, object detection, vision-language models, dual-loop architecture, web application, blind and low-vision users.

## I. INTRODUCTION

### A. Background

Millions of blind and low-vision (BLV) users face significant barriers to independently understanding their visual surroundings in real-time. While traditional accessibility tools such as magnification software and screen readers exist, they do not address the core challenge: providing continuous, contextual descriptions of dynamic, egocentric environments. Recent commercial applications such as Microsoft Seeing AI and Envision demonstrate growing demand for camera-based scene understanding, yet these systems typically require users to capture static photos or manually select specific modes (text reading, object recognition) rather than providing continuous, interactive feedback during navigation.

### B. Motivation

The advent of real-time object detection (e.g., the YOLO family) and vision-language models (e.g., BLIP, LLaVA) now

makes it feasible to build integrated systems that continuously perceive and narrate dynamic scenes. Surveys on assistive systems for visually impaired individuals emphasize that integrating AI, embedded hardware, and machine learning is transforming navigation and scene-understanding capabilities, but also highlight that deployment in real-world, latency-constrained, egocentric conditions remains a significant challenge. Recent YOLO-based obstacle detection systems and YOLOv8-based assistive platforms show that detection-driven navigation aids can effectively identify hazards in cluttered outdoor and indoor scenes while running in real time on consumer hardware. In parallel, emerging LVLM-based systems and human-centered studies on scene description use cases show that BLV users benefit from both quick, safety-critical alerts and richer contextual descriptions, motivating architectures that separate fast “reflex” processing from slower “cognitive” reasoning.

### C. Project Goal

This project addresses the gap in open-source, end-to-end solutions by building “Describe My Environment”—a real-time accessibility assistant that detects objects, summarizes scenes, and narrates potential hazards through audio feedback. The system is designed to be easy to run and evaluate in multiple ways: as a command-line application, as a Jupyter notebook demo, and as a full-stack web application that lets users test the pipeline from a browser while models run locally. The design targets a latency budget of 500 ms per frame, a minimum throughput of 2 frames per second (FPS) for narrated descriptions, and 30 FPS for a safety-critical hazard loop, so that hazard warnings remain responsive while richer narrations can be triggered on demand.

### D. Literature Review

1) *Commercial and Research Precedents*: Microsoft Seeing AI (launched in 2017) is a foundational tool that uses computer vision to support BLV users with scene descriptions, document reading, currency identification, and face recognition, delivered through multiple task-specific modes on mobile devices. Similarly, the Envision app combines OCR,

object detection, and product recognition in a subscription-based model, offering users multiple targeted modes such as instant text, document scanning, and object finding. These systems demonstrate strong demand for AI-powered visual assistance but primarily rely on photo-style or mode-based interactions rather than continuous streaming and unified scene understanding.

Recent research prototypes extend this work toward continuous, real-time perception. WorldScribe, a system developed at the University of Michigan, focuses on live, adaptive scene descriptions for BLV users by integrating YOLO World for flexible object detection with GPT-4 to generate narrative descriptions whose verbosity can be adjusted. Other work uses large vision-language models (LVLMs) coupled with segmentation and depth estimation to provide both global scene summaries and local object information, augmented with depth-aware audio or haptic feedback through wearable devices. These approaches motivate the use of structured visual inputs, such as segmentation or detection outputs, to reduce hallucinations in language model responses and improve safety.

*2) Object Detection for Navigation and Safety:* From its inception in YOLOv1 through to modern variants such as YOLOv8 and YOLOv11, the YOLO series has become a de facto standard for real-time object detection in embedded and mobile systems. Specialized variants designed for accessibility, such as YOLO-based obstacle detection models, introduce architectural improvements for small-obstacle detection in cluttered navigation scenes, explicitly targeting visually impaired users. Other studies on YOLOv8-based assistive systems for visually impaired individuals validate that YOLO-family detectors can achieve acceptable real-time performance (often exceeding 15 FPS) while maintaining sufficient accuracy for navigation warnings. Together, these works justify selecting YOLO11n as a modern, lightweight variant within a well-established and thoroughly tested family of detectors.

*3) Vision-Language Models and Scene Captioning:* Vision-language models such as BLIP and LLaVA have emerged as practical choices for image captioning and visual question answering. BLIP achieves low-latency captioning on modern hardware while maintaining strong caption quality, making it suitable for real-time or near-real-time applications. LLaVA and related LVLMs offer richer scene understanding and conversational capabilities, but their larger memory footprint and higher inference cost make them challenging to deploy on local, real-time assistive devices without substantial GPU resources. In this project, BLIP is used as the primary captioning engine, while LLaVA-style models are reserved for future cloud-augmented or wearable-optimized architectures.

*4) Human-Centered Design for Accessibility:* // Human-centered studies of AI-powered scene description tools report that BLV users want both quick, task-oriented feedback (for example, “Is there an obstacle in front of me?”) and richer contextual narratives (for example, “Describe the room and what people are doing”), with preferences varying by context. This empirical evidence directly motivates the dual-

loop design in this work: a fast reflex loop that prioritizes time-critical hazard alerts and a slower cognitive loop that delivers on-demand, higher-level descriptions. Such designs align with broader principles in assistive technology, where safety-critical and exploratory feedback are often separated to satisfy competing constraints on latency, information density, and cognitive load.

## II. METHODS

### A. Problem Formulation

The system is framed as a real-time perception and narration task. Given a continuous video stream from a user’s camera (for example, handheld, mounted on a wearable, or accessed via browser WebRTC), the model must:

- 1) detect objects and track them frame-to-frame;
- 2) identify immediate hazards and generate time-critical warnings; and
- 3) on demand, produce natural language descriptions of the scene that are converted to audio for the user.

The input consists of RGB video frames at 1280×720 resolution, captured at up to 30 frames per second (FPS) from a webcam. The core constraints are:

- latency budget of 500 ms for end-to-end processing;
- reflex loop throughput of approximately 30 FPS for safety-critical detection;
- cognitive loop throughput of at least 2 FPS for on-demand narration.

The outputs are:

- continuous bounding boxes and class labels for detected objects (reflex loop);
- immediate audio warnings when hazards are detected;
- structured scene summaries that include detected objects, spatial relationships, and inferred motion (cognitive loop); and
- spoken narration synthesized from those descriptions via text-to-speech.

### B. Dataset Description

The COCO (Common Objects in Context) dataset is used as the primary source of object categories, visual diversity, and detection benchmarks. It contains roughly 330,000 images across 80 object classes (for example, person, car, chair, dog), with bounding-box annotations and a strong emphasis on everyday scenes with clutter and occlusion. This makes COCO well suited for accessibility applications, where users often navigate crowded indoor and outdoor environments.

For caption quality assessment and to understand linguistic variability, the Flickr30K Captions and COCO Captions datasets are examined qualitatively. Both provide multiple human-written captions per image, highlighting the subjectivity in how people describe scenes and informing the design of robust narration prompts.

Figure 1 illustrates the distribution of the top 20 most common object classes in COCO and the train/validation/test split (approximately 72.1% / 3.0% / 24.8%). The heavy

concentration of “person” and vehicle categories aligns with accessibility priorities such as navigating around people and avoiding traffic hazards, but also reveals a long tail of rare object types. As a result, the system is expected to perform best on frequent classes and less reliably on specialized or infrequent categories, motivating future fine-tuning or domain adaptation.

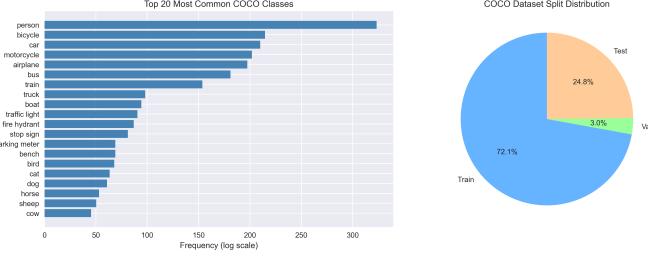


Fig. 1. Example COCO statistics: class distribution and dataset split.

### C. Models Used

**Object Detection (YOLO11n):** YOLO11n, the nano variant of the latest YOLO architecture, is used as the primary detector. It is pretrained on COCO and selected over larger YOLO variants and older versions (such as YOLOv8) due to its parameter efficiency, modern design, and strong real-time performance on consumer hardware. Prior work on YOLO-based assistive systems and specialized variants for obstacle detection supports YOLO’s suitability for navigation aids in visually impaired contexts.

**Scene Captioning (BLIP):** Scene understanding is performed using BLIP (Bootstrapping Language-Image Pre-training) from the Hugging Face Transformers library. BLIP provides a good balance between caption quality and inference speed; on the target Apple Silicon hardware it achieves latency on the order of 160 ms per image without relying on cloud infrastructure. Larger vision-language models such as LLaVA offer richer reasoning but have significantly higher compute and memory requirements and are therefore reserved for future cloud-augmented or wearable-focused versions.

**Narration LLM (Llama 3.2 via Ollama):** To turn raw captions and detection metadata into more natural, user-friendly descriptions, the system uses a lightweight Llama 3.2 3B language model served locally through Ollama. For each narration request, the model receives BLIP’s caption together with structured context from YOLO11n (object classes, locations, and motion summaries) and generates a more comprehensive description of the surroundings. This separation allows YOLO and BLIP to focus on perception while the LLM handles linguistic fluency and prioritization of information.

**Text-to-Speech (pyttsx3):** Narration is delivered through audio using pyttsx3, a Python wrapper around system text-to-speech engines. It offers sub-millisecond overhead, offline operation, and cross-platform compatibility. Although more advanced TTS services are available, pyttsx3 is chosen to keep the system fully local, privacy-preserving, and easy to deploy without external APIs.

### D. Pipeline Overview

The system employs a dual-loop architecture to satisfy competing constraints: safety-critical hazard detection demands low latency and high throughput, while rich scene narration can tolerate higher latency but requires more computation.

**Reflex Loop (Safety-Critical,  $\approx 30$  FPS):** The reflex loop runs continuously on each incoming frame. YOLO11n detects objects and produces bounding boxes and class labels. A simple tracker associates detections across frames, and a physics-based safety monitor estimates object velocity and apparent size change to identify potential collisions or rapidly approaching obstacles. When a hazard is detected in the user’s path, the loop triggers a high-priority audio warning (beep or short phrase). This loop is designed to maintain roughly 30 FPS so that warnings feel instantaneous.

**Cognitive Loop (On-Demand Narration,  $\approx 2$  FPS):** The cognitive loop is activated only when the user requests a description (for example, via the SPACE key in CLI mode or a button in the web interface). It takes a snapshot of recent frames from a history buffer that stores detections and tracking data, runs BLIP on the latest frame to obtain a scene caption, and uses trajectory analysis to summarize object movements and interactions. These elements are fused into a structured prompt that is passed to a lightweight language model (Llama 3.2 via Ollama), which generates a coherent narrative. The narration text is then converted to speech with normal priority, allowing reflex warnings to preempt it if necessary.

**Threading and Coordination:** A central “Dual Loop System” orchestrates the two loops using separate threads, thread-safe queues, and locks:

- The main thread captures frames at the target frame rate, pushes them into the reflex queue, and optionally displays a visualization window.
- The reflex thread consumes frames, performs YOLO detection and safety checks, updates tracking and history, and queues hazard alerts.

The cognitive thread waits for triggers, runs the BLIP + LLM narration pipeline, and queues narration audio.

An audio worker thread manages a priority queue of speech requests, ensuring that high-priority warnings are spoken before lower-priority narrations.

This design ensures that the safety-critical reflex loop is never blocked by the slower cognitive loop and that users receive hazard alerts as soon as possible.

**Web Integration:** To make the system easier to test and share, a full-stack web application wraps the pipeline:

The FastAPI backend exposes a WebSocket endpoint (`/ws/camera`) where clients stream base64-encoded frames and receive annotated frames, detections, and hazard alerts, as well as REST endpoints for system status and narration triggers.

The Next.js 16 frontend provides several pages: a landing page, challenges page, about and roadmap pages, a demos page, and an interactive test page with WebRTC camera

access, bounding-box overlays, hazard banners, narration controls, and a status panel.

In local deployments, the frontend and backend communicate over WebSocket and HTTP on localhost, allowing users to experience the full pipeline from a browser while all heavy computation remains on their own machine.

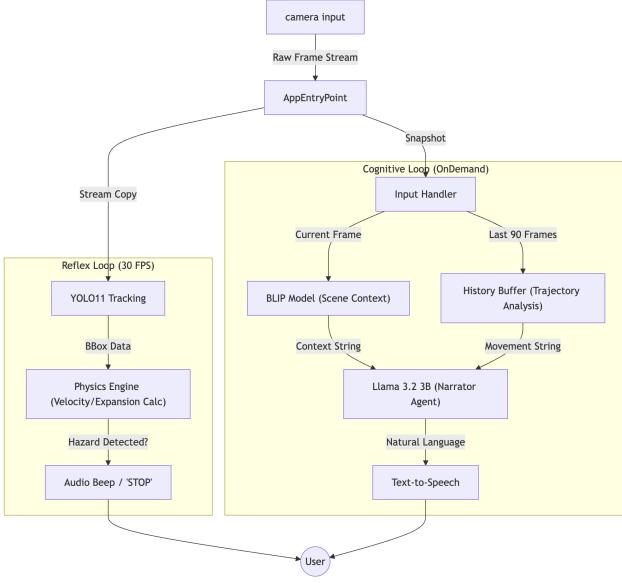


Fig. 2. High-level dual-loop system architecture showing the reflex loop, cognitive loop and audio worker.

### III. RESULTS

#### A. Detection Performance

Benchmark experiments were conducted on a 14-inch MacBook Pro (M4 Pro, 24 GB unified memory) using PyTorch 2.9.1 with the MPS backend. YOLO11n achieved a mean per-frame latency of  $34.7 \pm 2.6$  ms, corresponding to a throughput of about 28.8 FPS, which is close to the 30 FPS target for the reflex loop. Memory usage during detection was approximately 2 GB for the model and inference activations. Comparison with YOLOv8n showed negligible differences in per-frame latency (both around 30–35 ms), supporting the choice of YOLO11n for its more modern architecture and anticipated long-term support.

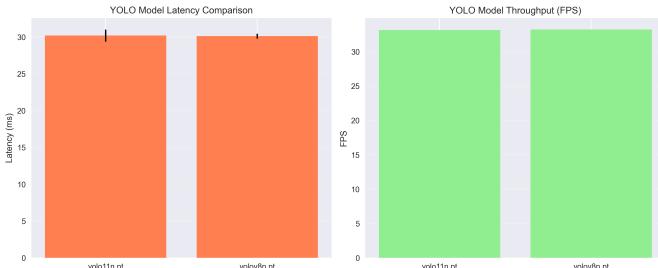


Fig. 3. Latency and fps comparison of the two yolo models.

#### B. Captioning Performance

BLIP image captioning exhibited a mean latency of  $158.7 \pm 2.6$  ms per image and an effective throughput of 6.3 FPS, limited by model size and attention computation. Peak memory usage during captioning was roughly 1.5 GB. BLIP's latency is approximately twice as fast as some reported values on older hardware, likely due to Apple Silicon GPU acceleration, but it still dominates the overall pipeline runtime. In practice, BLIP is the primary bottleneck and constrains the cognitive loop to roughly 2 FPS when full captioning and narration are requested.

#### C. End-to-End Performance

The combined pipeline (YOLO detection, BLIP captioning, and TTS) achieved an end-to-end median latency of about 194 ms per frame under local deployment, comfortably below the 500 ms target set at the beginning of the project. The reflex loop alone (detection plus hazard check) operates at roughly 35 ms per frame, easily sustaining 30 FPS for safety-critical warnings, while the full cognitive loop (detection, captioning, LLM narration, and TTS) operates at around 2 FPS.

A summary of the latency breakdown is shown conceptually in Table I.

TABLE I  
LATENCY BREAKDOWN (LOCAL DEPLOYMENT)

| Component       | Latency (ms)  | Notes                   |
|-----------------|---------------|-------------------------|
| YOLO Detection  | $\approx 35$  | Reflex loop core        |
| BLIP Captioning | $\approx 159$ | Dominant bottleneck     |
| TTS (pyttsx3)   | $\approx 1$   | Negligible overhead     |
| End-to-End      | $\approx 194$ | Reflex + cognitive path |

Overall throughput is therefore:

- Reflex loop (detection only): 28–30 FPS.
- Cognitive loop (detection + captioning + LLM narration):  $\approx 2$  FPS.

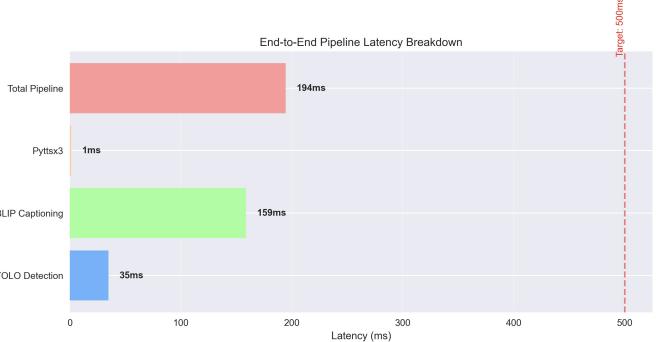


Fig. 4. Latency of pipeline.

#### D. Web Application Behavior

The full-stack application was evaluated locally using live camera input accessed through the browser. The frontend successfully displays YOLO detections in real time, with

bounding boxes and class labels rendered on a canvas overlay; the additional latency from WebSocket transmission and frame encoding is approximately 2–3 frames, which is acceptable for interactive use. Hazard alerts, such as “STOP! Person in front of you,” typically appear within 50–100 ms of an object entering the critical zone, indicating that the reflex loop’s responsiveness is preserved even when accessed via the web interface. On-demand narrations are generated within 200–800 ms end-to-end (depending on LLM response time), with the text displayed and audio playback remaining smooth. During these tests, the backend process used roughly 2–3 GB of RAM and 20–40% CPU, leaving sufficient headroom for the frontend and other applications.

#### E. Comparative Context

The measured 194 ms end-to-end latency and 30 FPS hazard loop compare favorably with reported performance from related systems. YOLO-based assistive navigation systems in the literature often target 15–20 FPS for obstacle detection, whereas the reflex loop here reaches 28–30 FPS, providing faster updates for collision avoidance. Recent YOLOv8-based assistive platforms report similar per-frame detection latencies (on the order of 35–50 ms), which validates the choice of a YOLO-family model and suggests that YOLO11n is competitive with state-of-the-art detectors in this domain. LVLM-based environment perception systems and cloud-backed tools such as WorldScribe demonstrate rich scene descriptions but incur additional network and model latency; by contrast, this project shows that a purely local pipeline can meet strict latency targets while still enabling both safety-oriented detection and higher-level narration.

## IV. DISCUSSION

### A. Technical Achievements

The project shows that a multi-model vision–language pipeline can satisfy a strict  $\approx 200$  ms latency budget on consumer hardware while maintaining separate safety-critical and exploratory feedback loops. The dual-loop design lets the reflex loop prioritize immediate hazard alerts without being blocked by slower scene narration, and wrapping the system in a web application, CLI, and notebook interface makes it easier for others to run, inspect, and extend the pipeline.

### B. Limitations and Challenges

**Latency bottleneck:** BLIP captioning accounts for most of the end-to-end latency and limits the cognitive loop to around 2 FPS, so rapidly changing scenes may be described after they have evolved. Lighter captioning models, distillation, or caching strategies are natural next steps.

**Depth and motion ambiguity:** Because the system uses a single RGB camera, distance and relative motion must be inferred from bounding-box heuristics, which can be fooled by scale and perspective. Adding stereo cameras, depth sensors, or IMU data would improve distance estimates and “who is moving toward whom” reasoning.

**Model hallucinations:** BLIP occasionally produces spurious descriptions such as treating the camera holder as a person in a mirror; prompt engineering and simple post-processing reduce but do not eliminate this behavior. LVLM-based assistive systems suggest that combining segmentation or detection outputs with LVLMs is a promising direction to reduce hallucinations.

**Web architecture overhead:** The browser-based prototype introduces WebSocket round-trip and frame-encoding costs, which are acceptable for local demos but not ideal for safety-critical wearables. A future production system would likely run entirely on edge hardware (smartphone or embedded device) to avoid network latency and dependence on a desktop browser.

**Class imbalance:** COCO’s long-tailed distribution means common objects (people, vehicles) are detected reliably, while rare categories are less robust. For real-world accessibility, fine-tuning on blind-user-relevant scenarios such as curbs, steps, and doorways would be important.

### C. Alignment with Accessibility Needs

Human-centered studies of AI scene-description tools report that BLV users want both fast, task-oriented feedback and richer contextual narratives. The reflex–cognitive split in this system matches that pattern by delivering high-frequency hazard alerts alongside on-demand, slower narrations, and the same principle could inform other real-time assistive AI designs where safety and context must be balanced.

## V. FUTURE WORK

**Immediate extensions:** Sensor fusion is a priority: integrating IMU data (accelerometer/gyroscope) would distinguish user motion from object motion and improve “approaching user” versus “user approaching” hazard classification. Depth sensing via LiDAR on supported phones or stereo cameras would replace bounding-box heuristics with true distance estimates in meters. Another direction is structured hallucination reduction: combining BLIP captions with YOLO detections and, optionally, segmentation masks to feed the language model high-confidence, structured facts instead of raw captions.

**Medium-term (wearable and mobile):** On the modeling side, quantizing YOLO and BLIP to INT8 or FP16 and targeting runtimes such as TensorRT or CoreML would enable deployment on smartphones and embedded wearables. On the interaction side, planned features include a “find my object” mode (e.g., “Where are my keys?” with spatial-audio or haptic guidance), voice control via speech-to-text (Whisper) instead of keyboard or button triggers, and haptic feedback on phones or watches for silent, directional warnings.

**Long-term vision:** Longer term, the system is intended to evolve from a web-based prototype into a true wearable or smartphone-linked platform (for example, smart glasses or a chest-mounted camera) that runs the dual-loop pipeline entirely on edge hardware. This would preserve the current safety–context split while optimizing for latency, battery life, and privacy.

## VI. CONCLUSION

This project demonstrates that a dual-loop pipeline combining YOLO11n detection, BLIP captioning, a lightweight local language model, and text-to-speech can deliver real-time environmental descriptions for low-vision users within an end-to-end latency of roughly 200 ms on consumer hardware. By exposing the pipeline through a command-line interface, a Jupyter notebook demo, and a full-stack FastAPI + Next.js web application, the system moves beyond a standalone script and becomes a reproducible, interactive tool suitable for classroom use, user testing, and further research. The results validate the feasibility of multi-model vision–language systems for accessibility while outlining clear future directions in sensor fusion, model optimization, richer interaction modes, and wearable, edge-native deployment.

## REFERENCES

- [1] 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 *Proc. 13th Eur. Conf. Comput. Vis. (ECCV)*, Zurich, Switzerland, 2014, pp. 740–755.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788.
- [3] J. Li, D. Li, C. Xiong, and S. Hoi, “BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation,” in *Proc. 39th International Conference on Machine Learning (ICML)*, Baltimore, MD, USA, 2022, pp. 12888–12900.
- [4] H. Liu, C. Li, Q. Wu, and Y. J. Lee, “Visual Instruction Tuning,” *arXiv preprint arXiv:2304.08485*, 2023.
- [5] Z. Chen, Z. Liu, K. Wang, K. Wang, S. Lian, “A Large Vision-Language Model based Environment Perception System for Visually Impaired People,” in *Proc. 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 221-228, 2024.
- [6] W. Wang, B. Jing, X. Yu, Y. Sun, L. Yang, C. Wang, “YOLO-OD: Obstacle Detection for Visually Impaired Navigation Assistance,” *Sensors*, vol. 24, no. 23, pp. 7621, 2024.
- [7] S. More, N. Patil, V. Lobo, N. Shet, D. Goswami, P. Rane, “Empowering the Visually Impaired: YOLOv8-based Object Detection in Android Applications,” *Procedia Computer Science*, vol. 252, pp. 457-469, 2025.
- [8] A. R. K. Kumar, P. V. A. S., “Assistive Technologies for Blind and Visually Impaired Individuals – A Short Review,” in *Proc. 2025 3rd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAEECA)*, pp. 1-6, 2025.
- [9] P. Kathiria, S. Mankad, J. Patel, M. Kapadia, N. Lakdawala, “Assistive systems for visually impaired people: A survey on current requirements and advancements,” *Neurocomputing*, vol. 606, pp. 128284, 2024.
- [10] R. Gonzalez Penuela, J. Collins, C. Bennett, S. Azenkot, “Investigating Use Cases of AI-Powered Scene Description Applications for Blind and Low Vision People,” in *Proc. Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1-21, 2024.
- [11] Ultralytics, “YOLO11 Documentation,” 2024. [Online]. Available: <https://docs.ultralytics.com>
- [12] Hugging Face, “Transformers: Vision-Language Models.” [Online]. Available: <https://huggingface.co/docs/transformers/tasks/image-captioning>
- [13] Microsoft, “Seeing AI: New technology research to support the blind and low vision community,” Microsoft Accessibility Blog, Aug. 2019. [Online]. Available: <https://blogs.microsoft.com/accessibility/seeing-ai/>
- [14] Microsoft Garage, “Seeing AI,” 2025. [Online]. Available: <https://www.microsoft.com/en-us/garage/wall-of-fame/seeing-ai/>
- [15] Envision, “Assistive technology for blind and low vision.” [Online]. Available: <https://www.letsenvision.com>
- [16] Perkins School for the Blind, “Using the Envision app with low vision,” Dec. 2023. [Online]. Available: <https://www.perkins.org/resource-using-the-envision-app-with-low-vision/>
- [17] University of Michigan News, “Real-time descriptions of surroundings for people who are blind,” Oct. 2024. [Online]. Available: <https://news.umich.edu/real-time-descriptions-of-surroundings-for-people-who-are-blind/>