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: akhilnis

***Abstract*—This paper introduces 'Describe My Environment,' an accessibility system for blind and low-vision users that provides real-time object detection, scene captioning, and audio narration. The dual-loop computer vision system features several interfaces: a command-line app, a Jupyter demo, and a web application built with FastAPI and Next.js. Users can locally stream camera input, get hazard warnings at 30 FPS, and trigger on-demand scene descriptions at about 2 FPS on Apple Silicon. Experiments with the COCO dataset and live feeds show a YOLO11n detector and BLIP captioning model achieve a 500 ms latency target, with a median end-to-end latency of 194 ms per frame and interactive performance. The report covers design decisions, performance trade-offs between reflex and cognitive loops, browser-based deployment limitations, and lessons learned from making the system accessible across multiple interfaces.**

***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 struggle to independently understand their surroundings in real time. Traditional tools like magnifiers and screen readers lack continuous, contextual descriptions of dynamic environments. Commercial apps such as Microsoft Seeing AI and Envision show demand for camera-based scene understanding but usually require static photos or manual mode selection rather than offering continuous, interactive feedback during navigation.

B. Motivation

Recent advances in real-time object detection (YOLO family) and vision-language models (BLIP, LLaVA) make it possible to build systems that continuously perceive and narrate scenes. Surveyed assistive systems show AI and embedded hardware are transforming navigation, but real-world, low-latency deployment is still a challenge. YOLO-based aids can identify hazards in real time, and LVLM-based systems show BLV users value both quick alerts and richer descriptions, motivating architectures with fast "reflex" and slower "cognitive" loops.

C. Project Goal

This project fills the gap in open-source, end-to-end tools by building "Describe My Environment"—a real-time assistant that detects objects, summarizes scenes, and narrates hazards via audio. The system runs as a command-line app, Jupyter demo, or web application where models execute locally. The design targets a 500 ms latency per frame, at least 2 FPS for narrations, and 30 FPS for the safety-critical hazard loop, ensuring responsive warnings and on-demand narrations.

D. Literature Review

1) Commercial and Research Precedents: Microsoft Seeing AI supports BLV users with computer vision for scene descriptions, document reading, currency identification, and face recognition, delivered through several mobile modes. The Envision app combines OCR, object detection, and product recognition, also using targeted modes. Both show demand for AI-powered visual assistance but mostly rely on photos or mode-based use, not continuous streaming or unified scene understanding.

Recent prototypes move toward continuous, real-time perception. WorldScribe uses YOLO World with GPT-4 for adaptable scene descriptions. Other work applies large vision-language models (LVLMs) with segmentation and depth estimation for both global and local information, often adding audio or haptic feedback. Structured visual inputs like segmentation or detection help reduce hallucinations and improve safety.

2) Object Detection for Navigation and Safety: The YOLO series, from v1 to v11, is standard for real-time object detection in embedded and mobile systems. Variants for accessibility improve small-obstacle detection for visually impaired users. YOLO-based systems achieve real-time performance (often over 15 FPS) with enough accuracy for navigation warnings. This supports using YOLO11n as a lightweight, modern detector.

3) Vision-Language Models and Scene Captioning: Vision-language models like BLIP and LLaVA are practical for image captioning and visual QA. BLIP delivers low-latency, high-quality captions, suitable for real-time use. LLaVA and similar

LVLMs offer richer understanding but require more memory and compute, making local deployment challenging. Here, BLIP is the main captioning model; LLaVA-like models are for future cloud or wearable use.

4) Human-Centered Design for Accessibility: Studies show BLV users want both quick, task-focused feedback and richer contextual narratives, depending on context. This motivates the dual-loop design: a fast reflex loop for immediate hazard alerts and a slower cognitive loop for on-demand, detailed descriptions—aligning with assistive tech principles of separating safety-critical and exploratory feedback.

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 provides the main object categories, diversity, and detection benchmarks. With about 330,000 images across 80 classes and bounding-box annotations, COCO's focus on everyday scenes with clutter and occlusion makes it well suited for accessibility applications, where users navigate crowded spaces.

To assess caption quality and linguistic variety, the Flickr30K Captions and COCO Captions datasets are qualitatively reviewed. Both offer multiple human-written captions per image, revealing subjectivity in scene descriptions and helping design 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.

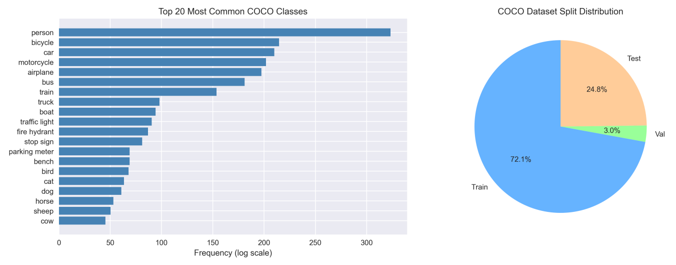

Fig. 1. Example COCO statistics: class distribution and dataset split.

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 uses BLIP from Hugging Face, balancing caption quality and inference speed. On Apple Silicon, BLIP achieves about 160 ms latency per image locally. Larger models like LLaVA offer richer output but require more resources and are reserved for future versions.

Narration LLM (Llama 3.2 via Ollama): A lightweight Llama 3.2 3B model via Ollama converts raw captions and detection metadata into natural, user-friendly descriptions. The LLM receives BLIP's caption and structured context from YOLO11n to generate a comprehensive scene summary, letting YOLO and BLIP focus on perception while the LLM handles language.

Text-to-Speech (pyttsx3): Narration uses pyttsx3, a Python text-to-speech wrapper with low overhead, offline operation, and cross-platform support. It is chosen for privacy and ease of deployment over more advanced online TTS services.

D. Pipeline Overview

The system uses a dual-loop architecture: a fast hazard detection loop for low latency and high throughput, and a richer scene narration loop that tolerates higher latency but needs more computation.

Reflex Loop (Safety-Critical, $\approx$30 FPS): The reflex loop runs on each frame. YOLO11n detects objects and outputs bounding boxes and class labels. A tracker and safety monitor estimate object velocity and size change to spot possible collisions. Hazards trigger high-priority audio warnings. The loop maintains about 30 FPS for instant feedback.

Cognitive Loop (On-Demand Narration, ≈2 FPS): The cognitive loop activates when the user requests a description. It gathers recent detection and tracking data, runs BLIP to generate a caption, analyzes object motions, and combines the results into a prompt for the Llama 3.2 LLM, producing a narrative. Narration converts to speech at normal priority, and reflex warnings can preempt it.

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 BLIP + LLM narration, and queues audio. An audio thread manages speech requests, ensuring high-priority warnings interrupt narrations. This design keeps reflex alerts fast and unblocked by narration.

Web Integration: A full-stack web app wraps the pipeline. The FastAPI backend provides a WebSocket endpoint for streaming frames, sending annotations, detections, and hazard alerts, plus REST endpoints. The Next.js frontend includes pages for testing, demos, and status displays, with camera access and overlays. Locally, frontend and backend use WebSocket and HTTP on localhost, keeping all computation on the user's machine.

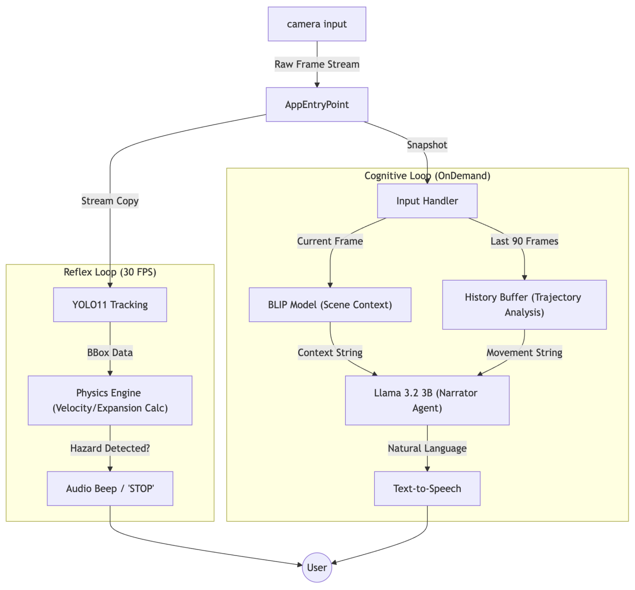

Fig. 2. High-level dual-loop system architecture showing the reflex loop, cognitive loop and audio worker.

III. Results

A. Detection Performance

Benchmarks on a 14-inch MacBook Pro (M4 Pro) using PyTorch 2.9.1 showed YOLO11n's mean per-frame latency was 34.7 ms (±2.6), yielding 28.8 FPS—close to the 30 FPS reflex loop target. Detection used 2 GB RAM. YOLOv8n gave similar latency (30–35 ms), confirming YOLO11n's suitability for real-time use and modern architecture.

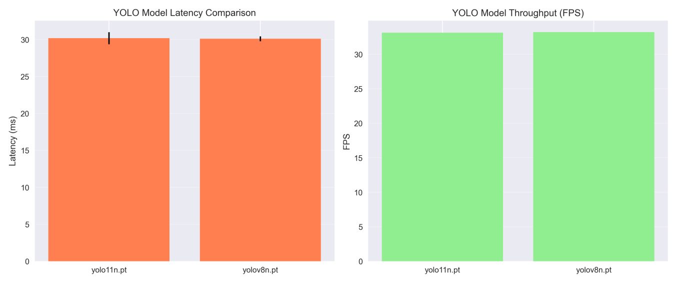

Fig. 3. Latency and fps comparison of the two yolo models.

B. Captioning Performance

BLIP captioning averaged 158.7 ms per image (±2.6), with 6.3 FPS throughput and 1.5 GB RAM use. Apple Silicon GPU acceleration improves speed, but BLIP remains the pipeline's main bottleneck, limiting the cognitive loop to about 2 FPS.

C. End-to-End Performance

The full pipeline (YOLO, BLIP, TTS) reaches a median end-to-end latency of 194 ms per frame, well under the 500 ms target. The reflex loop alone runs at 35 ms per frame (30 FPS), and the cognitive loop (with narration) at 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	≈ 35	Reflex loop core
BLIP Captioning	≈ 159	Dominant bottleneck
TTS (pyttsx3)	≈ 1	Negligible overhead
End-to-End	≈ 194	Reflex + cognitive path

Overall throughput is therefore:

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

D. Web Application Behavior

The full-stack app was tested locally with live camera input in the browser. The frontend displays YOLO detections in real time; extra WebSocket and encoding latency is 2–3 frames, which is acceptable. Hazard alerts appear within 50–100 ms, preserving reflex loop responsiveness. On-demand narrations generate in 200–800 ms. Backend used 2–3 GB RAM and 20–40

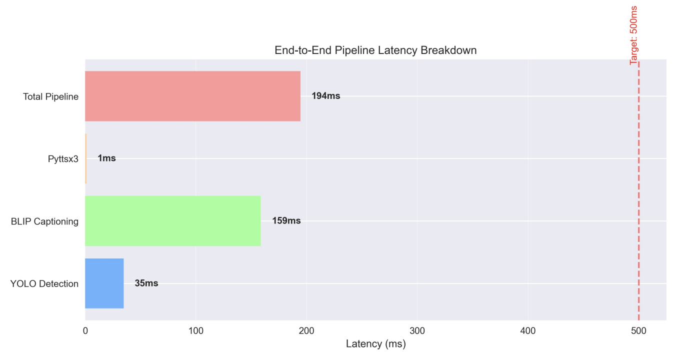

Fig. 4. Latency of pipeline.

E. Comparative Context

The 194 ms end-to-end latency and 30 FPS hazard loop compare well with related systems. Literature often targets 15–20 FPS for obstacle detection; this reflex loop achieves 28–30 FPS. YOLO-family models remain competitive. Unlike cloud-based tools, this project's local pipeline meets latency goals while enabling both detection and narration.

IV. Discussion

A. Technical Achievements

This project demonstrates a multi-model vision–language pipeline can meet a 200 ms latency budget on consumer hardware while maintaining separate safety and exploratory feedback. The dual-loop design keeps hazard alerts fast, and the system's web, CLI, and notebook interfaces make it easy to run and extend.

B. Limitations and Challenges

Latency bottleneck: BLIP captioning is the main latency source, capping the cognitive loop at 2 FPS; lighter models or caching could help.

Depth/motion ambiguity: With only an RGB camera, distance and motion are estimated heuristically. Adding stereo/depth sensors or IMU would improve estimates.

Model hallucinations: BLIP may produce spurious descriptions; prompt engineering and combining detection outputs help but don't fully solve this.

Web overhead: Browser-based prototypes add WebSocket and encoding delays, workable for demos but not ideal for wearables; edge deployment is preferred.

Class imbalance: COCO's long tail means common objects are detected well, but rare types aren't; fine-tuning for accessibility use cases is needed.

C. Alignment with Accessibility Needs

Studies show BLV users want fast, task-focused feedback and richer narratives. This system's reflex–cognitive split matches that need, balancing frequent hazard alerts with on-demand, slower narrations—an approach valuable for other assistive AI designs.

V. Future Work

Immediate extensions: Sensor fusion is key: adding IMU data (accelerometer/gyroscope) would distinguish user versus object motion for better hazard classification. Depth sensing (LiDAR or stereo cameras) would improve distance accuracy. For hallucination reduction, combining BLIP captions with YOLO detections and segmentation masks would provide the language model with more reliable facts.

Medium-term (wearable and mobile): For modeling, quantizing YOLO and BLIP and using runtimes like TensorRT or CoreML will enable deployment on smartphones and wearables. Planned features include "find my object" mode with spatial-audio or haptics, voice control (Whisper), and haptic feedback for silent, directional alerts.

Long-term vision: Long term, the system aims to become a wearable or smartphone-linked platform (e.g., smart glasses or chest camera) running the dual-loop pipeline entirely on edge hardware, preserving the safety–context split and optimizing for latency, battery, and privacy.

VI. Conclusion

This project shows a dual-loop pipeline (YOLO11n, BLIP, local LLM, TTS) can deliver real-time environmental descriptions for low-vision users within 200 ms latency on consumer hardware. Making the pipeline accessible through CLI, Jupyter, and web apps turns it into a reproducible, interactive tool for research and testing. Results validate multi-model vision–language systems for accessibility and highlight future work in sensor fusion, model optimization, richer interactions, and edge-native wearables.

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 (ICAECA)*, 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/