

EECE 490: Introduction to Machine Learning

Guided Vision - Project Report

Batoul Hashem

Joud Senan

Aya El Hajj

1 Problem Definition and Real-World Impact

Every day, visually impaired individuals face life-threatening hazards that may be trivial for people with normal vision. Examples of these dangers include sharp edges, exposed cables, fire sources, open pits, steps, obstacles and so on. Traditional assistive tools (like canes) offer limited spatial awareness and cannot interpret complex scenes or describe dangers with contextual meaning.

Guided Vision addresses this critical accessibility gap by introducing a real-time, AI-powered hazard-awareness assistant that sees through a camera (laptop webcam or Raspberry Pi module), understands the environment using a Vision-Language Model (VLM), and provides instant voice alerts that describe both the danger and its relative direction.

2 Timeline and Iterative Refinement

We began by defining and narrowing the problem into four concrete hazard categories:

- fire,
- exposed cables,
- mechanical tools,
- knives

We collected labeled datasets for each class from Roboflow, then expanded them using data augmentation techniques including horizontal flipping, rotations, brightness adjustments, and perspective changes, to reduce overfitting.

Our first approach involved training four separate YOLO models, each dedicated to a single hazard type. We then moved to a second approach where we combined all datasets and trained a single multiclass YOLO detector capable of recognizing the four categories.

While this second solution was effective, it revealed important limitations, most notably the retraining overhead required whenever new danger classes are added which makes it difficult to generalize to different hazards.

After evaluating these constraints, we transitioned to a Vision–Language Model (VLM) architecture. Unlike fixed-class object detectors, the VLM allows us to modify, extend, or remove hazard categories at any time without retraining, enabling far greater adaptability for real-world assistive scenarios.

3 Technical Implementation

As mentioned above, we decided to adopt a VLM-based implementation.

Guided Vision uses a ML pipeline centered around SmolVLM-256M-Instruct, a compact Vision–Language Model capable of generating concise and context-aware descriptions of the scene.

To elaborate, we describe the two deployment modes of Guided Vision: the Raspberry Pi implementation and the laptop webcam implementation.

3.1 Raspberry Pi implementation

To operate this system in real time, we designed a client–server architecture:

- The client (on either a laptop or a Raspberry Pi) continuously captures frames from its camera and sends them to the server.
- The server, built with FastAPI, receives these images, runs the VLM to describe the scene, and checks whether any danger is present.
- The server sends back a short message containing the caption and a danger warning when needed.
- The client then outputs the result to the user by speaking it aloud through TTS (Text-to-Speech).

This pipeline keeps all the heavy AI processing on the server, allowing the client devices to remain lightweight, fast, and responsive.

3.2 Laptop webcam implementation

For the computer version, we included a simple web frontend, which directly takes the role of the client. Instead of using a separate Python script, the browser itself becomes the component that captures the webcam and communicates with the server:

- The frontend file uses the laptop’s webcam to capture frames.
- These frames are sent to the FastAPI server for analysis.
- The server processes the image with the VLM, generates the caption, checks for danger, and sends the response back.
- The frontend immediately updates the dashboard by showing the live video, displaying the caption, and speaking danger alerts.

4 Submission and Deployment

For the final submission, Guided Vision was packaged and delivered using Docker and GitHub, making the system easy to run (for the computer version only).

With Docker, each component (server and frontend) includes its own Dockerfile, and the entire system can be launched using a single docker compose command.

As for GitHub, all project files for both implementations (server code, client code, frontend dashboard) and the Dockerfiles were submitted through a GitHub repository.

GitHub repository: https://github.com/Joud158/Guided_Vision

For more technical details or set-up steps, please refer to the README inside the repository.