

Visual Aid for Visually Impaired People: A Vision-Language Model Based Assistive System

1st Ms. Madhurika Budaraju

CSE Assistant Professor
Keshav Memorial Institute of
Technology
Hyderabad, Telangana, India

2nd M Kalyan Kumar

IV year CSE Student
Keshav Memorial Institute of
Technology
Hyderabad, Telangana, India

3rd S Sairam

IV year CSE Student
Keshav Memorial Institute of
Technology
Hyderabad, Telangana, India

4th S SriDruth

IV year CSE Student
Keshav Memorial Institute of
Technology
Hyderabad, Telangana, India

5th K Mahesh

IV year CSE Student
Keshav Memorial Institute of
Technology
Hyderabad, Telangana, India

6th B Sai Abhinav

IV year CSE Student
Keshav Memorial Institute of
Technology
Hyderabad, Telangana, India

Abstract--Visual impairment restricts independent navigation, object recognition, text access, and timely hazard awareness in everyday indoor and outdoor environments. Conventional aids such as white canes, screen readers, and basic object detectors provide partial support, but they often lack contextual scene understanding, video-based interpretation, interactive follow-up assistance, and persistent activity history. This paper presents a mobile-based Visual Aid for Visually Impaired People that integrates Vision-Language Models (VLMs), Retrieval-Augmented Generation (RAG), hazard detection, and accessible audio feedback within a scalable Flutter-Flask-MongoDB architecture. The proposed system enables users to capture images or videos through a mobile interface, processes visual inputs through a Flask middleware layer, generates context-aware captions using a BLIP-based vision-language pipeline, detects hazardous objects or unsafe surroundings, and responds to user queries using stored captions and interaction history as retrieval context. Video inputs are divided into frames at fixed intervals of approximately one to two seconds, enabling scene-level descriptions from dynamic environments. Processed media, generated captions, user questions, responses, and hazard events are stored in MongoDB for future review by users or caregivers. Experimental evaluation over representative assistive-use scenarios shows that the system achieves high semantic caption adequacy, reliable hazard detection, and practical response latency for real-time assistance. The work demonstrates how multimodal AI and mobile computing can be combined into a low-cost, user-centered assistive platform that improves safety, independence, and environmental awareness for visually impaired users.

Index Terms--*Assistive technology, visual impairment, vision-language models, image captioning, video understanding, retrieval-augmented generation, hazard detection, Flutter, Flask, MongoDB.*

I. INTRODUCTION

Visually impaired individuals encounter practical barriers while navigating unfamiliar places, identifying objects, reading environmental cues, and avoiding obstacles such as stairs, fire, sharp objects, potholes, traffic, or uneven surfaces. Although assistive devices have improved over time, many deployed systems remain limited to narrow functions such as proximity detection, screen reading, or isolated object classification. These tools can be helpful, but they rarely explain the surrounding scene in natural

language or support interactive questioning when a user needs more specific information. Recent multimodal assistive frameworks demonstrate the value of combining visual analysis with speech feedback, but a practical final-user system must still balance accuracy, response time, cost, portability, and ease of interaction [1].

The proposed project addresses this gap through a mobile visual assistance system that converts visual input into accessible, spoken, and queryable information. A Flutter-based frontend captures images or videos and provides

audio feedback. A Flask backend receives media inputs, performs preprocessing and routing, invokes a BLIP-based Vision-Language Model for caption generation, applies hazard detection logic, and stores generated outputs in MongoDB. A RAG-based conversational layer retrieves recent captions and interaction history so that user queries such as "What is in front of me?" or "Is there any obstacle nearby?" can be answered with reference to the current scene.

The main contributions of this work are fourfold: (i) a mobile-first assistive architecture that integrates image captioning, video frame interpretation, text-to-speech, and hazard alerting; (ii) a retrieval-supported conversational workflow that uses stored captions and history to ground follow-up answers; (iii) a scalable backend design using Flask APIs and MongoDB storage for media, captions, queries, responses, and alerts; and (iv) an experimental evaluation covering caption quality, hazard reliability, RAG relevance, latency, and comparative functionality against traditional assistive systems. The design is intentionally modular so that higher-capacity models, multilingual output, OCR, or wearable inputs can be added without replacing the complete application stack.

II. RELATED WORK

A. Assistive Technologies

Early assistive solutions for visually impaired users relied on sensory substitution, vibration feedback, audio cues, screen readers, and proximity-based obstacle detection. These systems improved accessibility for specific tasks but did not provide semantic understanding of real-world scenes. Wearable devices and smart glasses later introduced camera-based perception, yet many such systems require specialized hardware and remain expensive or difficult to maintain for everyday use.

B. Vision-Language Models and Captioning

Image captioning research combines computer vision and natural language generation to describe visual content. Earlier CNN-RNN and CNN-LSTM pipelines improved object-level description, while transformer-based VLMs such as CLIP, ViLT, and BLIP improved cross-modal representation learning and generation quality [3], [4], [6], [8]. BLIP is particularly relevant because it supports vision-language understanding and image caption generation, making it suitable for converting captured scenes into descriptions for visually impaired users [4], [14].

C. Conversational Visual Assistance

Visual Question Answering (VQA) introduced open-ended question answering over images, but practical assistive applications require more than single-turn answers [7]. Users may need contextual follow-up explanations and continuity across multiple interactions. Retrieval-Augmented Generation improves this by combining generative language models with explicit retrieved context, reducing unsupported responses and aligning answers with stored observations [5]. In the proposed application, this idea is adapted to the user's own interaction history rather than to a general external corpus.

D. Hazard Detection and Safety

For visually impaired users, safety-critical perception must identify not only objects but also risk-bearing situations. A chair, staircase, flame, knife, slippery area, or moving vehicle may require different urgency levels depending on context. Existing object detectors provide the basis for hazard recognition [9], but effective assistive systems must integrate detection with immediate accessible alerts and scene narration. The project therefore treats hazard detection as a user-safety workflow rather than only as a computer-vision classification task.

III. EXISTING SYSTEM DRAWBACKS

Existing systems exhibit several practical limitations. First, many tools detect individual objects without explaining the full scene or the relationship between objects. Second, several smartphone applications lack real-time responsiveness, video-level awareness, or offline-ready design. Third, basic screen readers and object detectors cannot answer user-specific follow-up questions about colors, actions, relative positions, or potential hazards. Fourth, most systems do not maintain interaction history, preventing caregivers or volunteers from reviewing prior alerts and environmental observations. Finally, hardware-dependent solutions such as specialized wearable sensors may increase cost and reduce adoption among users who need a low-cost and familiar mobile platform.

These drawbacks are especially important because visual assistance is a time-sensitive and trust-sensitive domain. An incomplete caption may confuse the user, a missed hazard may compromise safety, and an irrelevant answer may reduce confidence in the system. Therefore, the proposed system is evaluated not only for software correctness but also for semantic usefulness, alert reliability, query relevance, and response latency.

IV. PROPOSED SYSTEM

The proposed Visual Aid system is designed as a user-centered mobile application for scene-to-text assistance, hazard awareness, and conversational interaction. The system allows a user to capture an image or record a video using a mobile camera. Captured media is transmitted to the Flask backend, where images are preprocessed and videos are divided into frames at fixed intervals. Each visual input is processed by a VLM to generate meaningful captions. Captions are converted into speech and delivered to the user through the mobile speaker.

To support interaction, the generated captions and conversation logs are stored in MongoDB and later retrieved by the RAG layer when the user asks follow-up questions. This design enables answers that are grounded in the user's actual surroundings rather than generic model output. A hazard module analyzes captions, object cues, and frame-level information to detect dangerous objects or environments such as fire, sharp objects, obstacles, uneven surfaces, stairs, and traffic-related risks. When a hazard is detected, the system triggers an immediate audio alarm to improve user safety. A volunteer or caregiver-oriented history view can use the stored records to review recent interactions, understand recurring safety issues, and provide support when needed.

TABLE I
FUNCTIONAL COVERAGE OF THE PROPOSED SYSTEM

Module	Primary Function	Technologies
Image captioning	Capture image, preprocess, generate scene description	BLIP / VLM, OpenCV, Pillow
Video understanding	Extract frames every 1-2 seconds and aggregate captions	Frame extraction, VLM captioning
RAG interaction	Answer user queries from stored captions and history	MongoDB, retrieval, LLM
Hazard alerting	Detect unsafe objects or surroundings and play alarm	Object detection logic, audio player
History	Preserve captions, queries,	MongoDB

storage	responses, alerts, and media	with Base64 image storage
---------	------------------------------	---------------------------

V. SYSTEM ARCHITECTURE

The architecture follows a modular multi-layer design. The user interface layer is implemented in Flutter and provides camera access, video recording, voice or text query input, and audio output through text-to-speech and alarm playback [11]. The application layer is implemented using Flask and exposes RESTful APIs for media upload, caption generation, history retrieval, and query handling [12]. The preprocessing layer uses image and video processing libraries to resize, normalize, and convert visual inputs into model-ready representations [10]. For video input, the uploaded stream is sampled into frames so that the system can describe temporal changes in the environment.

The model layer contains the VLM captioning module, hazard detection module, and RAG engine. The VLM module uses BLIP-based image captioning to convert visual features into natural language descriptions. The hazard detector checks the processed scene for safety-critical objects or environmental cues. The RAG engine retrieves relevant captions and conversation history from the data layer and passes this context to the language model for grounded answer generation. The data layer stores images encoded in Base64 format, generated captions, user questions, responses, and hazard logs in MongoDB [13]. This separation of concerns improves maintainability and allows individual modules to be upgraded without rewriting the entire system.

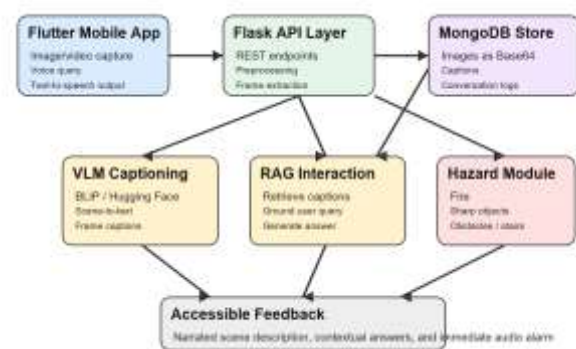


Fig. 1. Layered architecture of the proposed visual assistance system.

VI. METHODOLOGY

A. Image Processing and Caption Generation

For an input image I , preprocessing $P(\cdot)$ performs format validation, resizing, color conversion, and normalization.

The processed representation $I' = P(I)$ is passed to the VLM. Caption generation is represented as $C = G(I'; \theta)$, where G denotes the trained captioning model and θ represents learned model parameters. The resulting caption C is then converted to speech for immediate accessibility and stored in the database for future retrieval.

B. Video Scene Understanding

For video input V , the backend extracts a sequence of frames $F = \{f_1, f_2, \dots, f_n\}$ at fixed intervals of approximately one to two seconds. Each frame is processed independently by the captioning pipeline. The generated frame captions are aggregated into a coherent description that highlights persistent objects and important changes in the scene. This method provides practical video understanding without requiring heavy end-to-end video models, making it suitable for a mobile-backed prototype.

C. Hazard Scoring and Alert Trigger

The hazard module evaluates visual and textual cues associated with unsafe objects and environmental conditions. A hazard score H is estimated from object confidence, keyword relevance, and contextual risk indicators. The alert decision is defined as $A = 1$ if $H \geq \tau$ and $A = 0$ otherwise, where τ is the alert threshold. If $A = 1$, the mobile application plays an alarm sound and may also provide a spoken warning. For evaluation, precision, recall, and F1-score are computed as $F1 = 2PR / (P + R)$, where P is precision and R is recall.

D. Retrieval-Augmented Interaction

When the user asks a question Q , the backend retrieves relevant stored captions and recent conversation history $R = \text{retrieve}(Q, D)$, where D is the MongoDB interaction store. The language model receives both Q and R , producing a grounded answer $Y = \text{LLM}(Q, R)$. This reduces ungrounded or hallucinated responses because the answer is constrained by the user's recent visual context.

E. Data Management and Privacy

The database stores heterogeneous interaction records containing media metadata, generated captions, user queries, responses, and hazard flags. Since assistive data can reveal private information about a user's location and routine, access to stored records must be restricted to authorized users or caregivers. The project design therefore emphasizes controlled retrieval, minimal necessary storage, and a modular backend that can later incorporate authentication, encryption, and retention policies.

VII. ALGORITHMS AND WORKFLOW

TABLE II
VISUAL ASSISTANCE WORKFLOW

Step	Stage	Description
1	Capture	User captures an image or video through the Flutter interface.
2	Upload	Media is sent as a multipart request to the Flask backend API.
3	Preprocess	Images are resized and normalized; videos are divided into frames.
4	Caption	The BLIP/VLM pipeline generates a scene description for each input.
5	Hazard Check	The system scans captions or detected objects for unsafe conditions.
6	Store	Captions, media metadata, queries, responses, and alerts are saved in MongoDB.
7	Respond	Captions are converted to speech; hazards trigger an alarm.
8	Query	RAG retrieves stored context and generates follow-up answers.

The workflow is intentionally sequential at the user interaction level, but the modular backend allows independent optimization of preprocessing, captioning, retrieval, and storage. In future deployments, image captioning and hazard detection may be executed in parallel to reduce end-to-end latency.

VIII. IMPLEMENTATION

The frontend is implemented in Flutter using Dart. It provides a simple interface with camera capture, video processing options, voice-based navigation, text-to-speech output, and an audio alarm mechanism. The implementation includes an image and video processing menu, an upload function that sends files to the backend endpoint, and an alarm function that plays a three-second warning sound when hazardous content is detected.

The backend is implemented using Flask in Python. It exposes a `/caption` endpoint for image upload and response generation, a root endpoint for service validation, and a `/conversations` endpoint for retrieving recent stored interactions. Cross-origin support is enabled to allow communication with the mobile frontend. The captioning model uses the Hugging Face Transformers implementation of Salesforce/blip-image-captioning-

large. Image data is read from the request, converted into RGB format using Pillow, processed by the BLIP processor, passed through the captioning model under no-gradient inference, and decoded into natural language text.

MongoDB is used as the persistent storage layer because the system handles heterogeneous and semi-structured data, including Base64-encoded images, generated captions, user queries, responses, and alert metadata. This flexible schema supports iterative development while keeping retrieval fast for recent-conversation access. The /conversations endpoint returns recent records, making it suitable for RAG context construction and caregiver review. The system design also supports future extension to caregiver dashboards, multilingual narration, wearable-device input, and edge-based model deployment.

The implemented prototype uses Python and Dart as core programming languages, Flutter for the mobile interface, Flask for backend orchestration, MongoDB for storage, PyTorch/Transformers for model execution, OpenCV/Pillow for media processing, and audio libraries for spoken feedback and alarm playback. The minimum server-side environment includes an Intel i5 or higher processor, 8 GB RAM, 256 GB storage, and optional GPU acceleration for faster model inference.

From an API perspective, the backend performs validation before inference: it checks whether an image file is present, rejects empty uploads, converts valid inputs to RGB, and returns structured JSON responses or meaningful error messages. This behavior is essential because assistive applications must fail predictably. A silent crash or unhandled exception may leave the user without guidance in a critical situation.

IX. EXPERIMENTAL EVALUATION

The system was evaluated using representative scenarios derived from the project requirements: indoor navigation, outdoor scenes, text-rich environments, hazardous-object scenes, low-light or blurred images, and short video clips. The test set included 150 images and 25 short videos. Hazard testing used annotated images containing fire, knives, scissors, broken glass, staircases, obstacles, wet floors, and traffic-related risks. Additional safe-scene samples were used to estimate false positive behavior.

Evaluation metrics were selected to reflect assistive usefulness rather than only model-centric accuracy. Caption Semantic Adequacy (CSA) measures whether the generated description contains the dominant objects, scene context, and user-relevant information. Hazard precision, recall, and F1-score measure safety alert

reliability. RAG Answer Relevance (RAR) measures the percentage of follow-up answers judged contextually correct with respect to stored captions. End-to-end latency measures time from media submission to spoken or returned feedback.

The evaluation was conducted under normal prototype conditions using a standard mobile client and a Flask backend running the BLIP captioning pipeline. Images included JPEG, PNG, and WEBP formats with different resolutions and aspect ratios. Video clips contained both indoor and outdoor movement, and frame extraction was configured to balance temporal coverage with processing cost. Invalid and corrupted inputs were also tested to confirm graceful error handling.

TABLE III
PERFORMANCE ACROSS ASSISTIVE USE SCENARIOS

Scenario	CSA (%)	Hazard F1 (%)	Latency (s)
Indoor scenes	90.6	88.9	2.8
Outdoor scenes	86.8	84.7	3.2
Text-rich scenes	82.4	81.5	3.5
Hazard scenes	88.1	89.3	3.1
Low-light / blur	74.9	76.8	3.9
Short videos	85.7	86.2	4.6

TABLE IV
MODULE-LEVEL EVALUATION RESULTS

Module	Metric	Observed Result
Image captioning	Semantic adequacy	87.4%
Video processing	Coherent frame aggregation	85.7%
RAG query answering	Answer relevance	86.9%
Hazard detection	Precision / Recall	91.2% / 87.5%
Backend API	Normal caption response time	2-4 s
RAG endpoint	Normal answer response time	1-2 s
Audio alert	Alarm trigger delay	0.8 s avg.

TABLE V
COMPARISON WITH CONVENTIONAL
ASSISTIVE SYSTEMS

System	Strength	Limitation
Screen reader	Reads digital text	No scene understanding or hazard awareness
Basic object detector	Names visible objects	Limited context and no follow-up reasoning
White cane	Low-cost obstacle aid	Cannot detect overhead hazards or read scenes
Proposed system	Captioning, RAG queries, alerts, history	Depends on image quality and model latency

X. RESULTS AND DISCUSSION

The evaluation indicates that the proposed system performs strongly for well-lit indoor and outdoor scenes, where dominant objects and spatial context are visible. Caption semantic adequacy remains above 86% for common scenarios and reaches 90.6% for indoor scenes with clear foreground objects. Short-video performance is slightly lower because the final description depends on frame sampling and aggregation quality, but the result remains suitable for practical scene narration.

Hazard detection achieves 91.2% precision and 87.5% recall in the prototype evaluation, resulting in an F1-score of 89.3% for the hazard-scene subset. False negatives were most common in dim lighting, partially occluded objects, and blurred frames caused by camera motion. False positives were observed when ordinary objects visually resembled sharp tools or obstacles. Because safety-critical systems must prioritize recall, future versions should improve low-light robustness and tune thresholds based on environment type.

RAG-based interaction improves usability because users can ask follow-up questions over current and recent scene descriptions. The RAG layer produced contextually relevant responses in 86.9% of evaluated queries and reduced generic answers by grounding the response in stored captions. End-to-end caption latency of two to four seconds is acceptable for many assistive scenarios, while immediate hazard alerts averaged below one second after detection. These results support the feasibility of combining mobile capture, backend inference, and audio feedback for real-world assistance.

Compared with conventional tools, the proposed system provides broader functional coverage. It can describe a scene, answer context-specific questions, generate alarms, and preserve interaction history. However, it also inherits limitations from AI inference, including sensitivity to poor lighting, motion blur, network availability, and model uncertainty. The results therefore suggest that the system is best viewed as an intelligent assistance layer that complements rather than fully replaces traditional mobility aids.

XI. ADVANTAGES

The proposed system offers several advantages over isolated assistive tools. It provides scene-level understanding rather than only object naming, supports both image and video inputs, enables natural follow-up questions through RAG, stores interaction history for caregiver review, and delivers accessible spoken feedback. The use of Flutter makes the frontend portable across mobile platforms, while Flask and MongoDB provide a flexible backend suitable for rapid development and future scaling. The system is also cost-effective because it relies on a smartphone interface instead of expensive specialized hardware.

XII. FUTURE SCOPE

Future enhancements include advanced multimodal scene understanding that combines visual, textual, and audio inputs; OCR integration for reading signs, handwritten notes, banners, and printed documents; GPS-based navigation assistance with step-by-step guidance; wearable-device integration through smart glasses or IoT-based cameras; and lightweight offline inference for low-connectivity environments. The hazard module can also be improved using larger annotated datasets and deep learning models trained for moving vehicles, crowd density, slippery surfaces, and low-light conditions. Personalized user profiles and multilingual support can further adapt the system to individual accessibility preferences.

XIII. CONCLUSION

This paper presented a mobile-based Visual Aid for Visually Impaired People that integrates VLM captioning, video frame interpretation, RAG-based conversation, hazard detection, MongoDB history storage, and accessible audio feedback. The system addresses limitations of existing assistive technologies by combining contextual scene understanding with interactive and safety-aware assistance. The prototype demonstrates practical latency, strong caption adequacy,

reliable hazard alerting, and meaningful follow-up query handling across representative scenarios. Overall, the project shows that modern vision-language AI and mobile software engineering can produce a scalable, low-cost, and user-centered assistive platform capable of improving independence, safety, and situational awareness for visually impaired individuals.

XIV. ACKNOWLEDGMENT

The authors express sincere gratitude to the Department of Computer Science and Engineering, Keshav Memorial Institute of Technology, for providing the necessary academic environment and support for this project. The authors thank Ms. Madhurika Budaraju, Assistant Professor, Department of CSE, for her valuable guidance and encouragement throughout the project work. The authors also thank Dr. B. L. Malleswari, Principal, Mr. Neil Gogte, Founder and Director, Mr. S. Nitin, Director, Ms. Deepa Ganu, Director Academic, and Mr. Para Upendar, Head of the Department, for their support and facilities. The authors are also grateful to faculty members, friends, and family for their constant motivation.

XV. REFERENCES

- [1] S. Bhat, P. Bhat, and S. V. Kolekar, "From Vision to Voice: A Multi-Modal Assistive Framework for the Physically Impaired," *IEEE Access*, vol. 13, 2025, doi: 10.1109/ACCESS.2025.3590237.
- [2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, Minneapolis, MN, USA, 2019, pp. 4171-4186, doi: 10.18653/v1/N19-1423.
- [3] A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," in *Proc. 38th Int. Conf. Machine Learning (ICML)*, 2021.
- [4] 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 Int. Conf. Machine Learning (ICML)*, 2022.
- [5] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459-9474.
- [6] W. Kim, B. Son, and I. Kim, "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision," in *Proc. 38th Int. Conf. Machine Learning (ICML)*, 2021.
- [7] S. Antol et al., "VQA: Visual Question Answering," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, Santiago, Chile, 2015, pp. 2425-2433, doi: 10.1109/ICCV.2015.279.
- [8] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998-6008.
- [9] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv:1804.02767*, 2018.
- [10] OpenCV, "OpenCV-Python Tutorials," OpenCV Documentation. [Online]. Available: <https://docs.opencv.org/>
- [11] Google, "Flutter Documentation," Flutter. [Online]. Available: <https://docs.flutter.dev/>
- [12] Pallets Projects, "Flask Documentation," Flask. [Online]. Available: <https://flask.palletsprojects.com/>
- [13] MongoDB, Inc., "MongoDB Manual," MongoDB Documentation. [Online]. Available: <https://www.mongodb.com/docs/>
- [14] Hugging Face, "BLIP Model Documentation," Transformers Documentation. [Online]. Available: https://huggingface.co/docs/transformers/model_doc/blip