



UNIVERSITY OF
THESSALY

UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND BIOMEDICAL
INFORMATICS

Large Language Models (LLMs) to Enhance the Capabilities of Robotic Systems

Chatziefstathiou Vasiliki

Undergraduate Thesis

Supervisors:

Dr. Tasoulis Sotirios, Assistant Professor department of Computer Science
and Biomedical Informatics, university of Thessaly

Dr. Panagiotis Anagnostou, Assistant Professor department of Computer
Science and Biomedical Informatics, university of Thessaly

Lamia, May 2025

Acknowledgments

I would like to thank Dr. Tasoulis Sotirios and Dr. Panagiotis Anagnostou for giving me the opportunity to work on this project.

Contents

List of Figures	4
List of Tables	5
Abstract	6
1 Introduction	7
1.1 Historical Overview and Motivation	7
1.2 Evolution of Object Detection Algorithms	7
1.3 Natural Language and Vision: Toward Multi-modal Understanding . . .	8
1.4 Motivation and Thesis Objective	8
1.5 Structure of the Thesis	9
2 Background	10
2.1 The YOLO Algorithm: Foundations and Advancements	10
2.2 Datasets for Object Detection: COCO vs. Open Images V7	11
2.3 Color Classification Using HSV and K-Nearest Neighbors	12
2.4 Integration of the OpenAI API	14
3 Implementation	15
3.1 System Architecture and Functional Flow	15
3.2 Object Detection and Color Analysis Module	17
3.3 User Interface and Language Interaction Module	19
3.4 Database Schema and Memory Persistence	20
4 Discussion and Conclusions	22
4.1 Primary Failures	22
4.2 Limitations	24
4.3 Practical Implications and Use Cases	24

4.4	Future Work	25
4.5	Conclusion	25
5	Bibliography	26
6	Code	28

List of Figures

2.1	Saved class names in the yaml file	12
2.2	Color Dataset from RGB Values to HSV Values [12]	13
3.1	App Interface	15
3.2	Flowchart of the App	16
3.3	Loading an image	17
3.4	Detection of two different classes of objects	18
3.5	Detection of two different colored objects	18
3.6	Detection of one specific colored object	19
3.7	Detection of all colored Green objects	19
3.8	Tablet of the Detection.db	20
3.9	Tablet of Chat history.db	21
3.10	Recollection of past detections	21
4.1	Failed detection	23
4.2	Confusing color black from grey	23
4.3	Failing detecting the color brown	24

List of Tables

2.1	Comparison of COCO and Open Images Version 7 datasets	11
-----	---	----

Abstract

As personal robots become increasingly integrated into daily life, their ability to perceive and interpret visual scenes and to communicate about them naturally is paramount. We begin by examining the evolution of computer vision and multimodal AI, from early pattern recognition methods to today's deep learning and large language models. Motivated by gaps in existing systems that focus on vision or language, but rarely both in a lightweight pipeline.

This thesis presents a prototype system that fuses the YOLOv8 object detection algorithm pre-trained on Open Image and language interface powered by the OpenAI GPT-4o-mini model. Upon image upload via a Tkinter GUI, YOLOv8 performs single-shot inference to produce bounding boxes, confidence scores, and class labels. Each detected region is converted to HSV space, where k-means clustering isolates dominant shades, these are then translated into human-readable color names using a K-Nearest Neighbors classifier trained on 5,052 labeled HSV samples. All detections including object class, color label, coordinates, and timestamp are logged in an SQLite database, while conversational exchanges and JSON directives drive database queries and UI updates. Experimental evaluation demonstrates robust object-detection performance in static images and reliable color labeling under varied lighting conditions.

Keywords: YOLOv8, color classification, HSV(Hue-Saturation-Value), KNN (K-Nearest Neighbors), OpenAI API, JSON directive

Chapter 1

Introduction

1.1 Historical Overview and Motivation

The ability of machines to "see", that is, to interpret and understand visual information, has long been one of the most challenging and fascinating goals in computer science. The field of computer vision, which underpins this capability, has evolved substantially since its inception in the 1960s. Early systems were limited to basic pattern recognition tasks, such as edge detection or binary shape classification. With the advent of digital computing and the development of more powerful hardware, computer vision expanded its scope from simple object localization to complex scene understanding, semantic segmentation, and real-time object tracking [1].

In parallel, the field of natural language processing (NLP) has matured significantly. From rule-based systems and statistical language models, the introduction of transformer-based architectures like BERT and GPT has led to unprecedented improvements in language understanding and generation [2]. The convergence of vision and language has since become a dynamic research frontier, enabling models to understand and describe visual scenes in human-like language.

The integration of computer vision and NLP holds transformative potential for applications in robotics, autonomous systems, assistive technologies, and intelligent user interfaces. In particular, the ability of a machine to visually identify an object, understand a user's question about that object, and respond meaningfully, all within a natural dialogue, is a core step toward intuitive human-AI interaction.

1.2 Evolution of Object Detection Algorithms

Historically, object detection was approached using handcrafted features like Haar cascades and Histogram of Oriented Gradients (HOG) combined with Support Vector

Machines (SVM) or Random Forest classifiers. These classical pipelines, although innovative for their time, suffered from limitations in generalization, speed, and accuracy under real-world conditions [3].

The deep learning revolution brought forward Convolutional Neural Networks (CNNs), starting with AlexNet in 2012, which outperformed traditional approaches by a significant margin on the ImageNet classification task. This shift led to the development of region-based detectors like R-CNN, Fast R-CNN, and Faster R-CNN, which applied deep features to localized object proposals [4].

To improve real-time performance, YOLO (You Only Look Once) introduced a single stage architecture in 2016 that treats object detection as a regression problem. Unlike its region-based predecessors, YOLO predicts bounding boxes and class probabilities directly from full images in a single evaluation, offering an impressive trade-off between accuracy and speed [5]. Successive versions, including YOLOv4, YOLOv5 (Ultralytics), and the most recent YOLOv8, have further refined this architecture by incorporating anchor-free predictions, decoupled head structures, and transformer-based modules for enhanced contextual reasoning [6].

1.3 Natural Language and Vision: Toward Multimodal Understanding

In the past few years, a shift toward multimodal learning, integrating visual, textual, and even auditory data, has redefined the landscape of artificial intelligence. Landmark models such as CLIP (Contrastive Language–Image Pretraining) [7] and DALL·E have demonstrated the feasibility of aligning visual content with linguistic cues. These advances have fueled the development of interactive AI systems that can both perceive the visual world and communicate about it.

By combining visual recognition systems like YOLO with generative language models (e.g., OpenAI’s GPT), developers can build robust interfaces for querying, understanding, and responding to visual data in natural language. Such capabilities are particularly crucial for robotics, where an AI must reason about its surroundings and relay information to human users, often in complex or ambiguous environments.

1.4 Motivation and Thesis Objective

The goal of this thesis is to design and implement a prototype system that bridges the gap between visual perception and natural language interaction. Specifically, this

system leverages the state-of-the-art YOLOv8 object detection algorithm in conjunction with the OpenAI GPT API to enable users to:

- Perform real-time object and color detection in static images.
- Query previously detected objects and retrieve contextual information via natural language.
- Store visual and conversational data in a structured, queryable format (e.g., SQLite database).

The system architecture includes an image interface, detection module, color classification using HSV space and KNN classifiers, and a conversational chatbot powered by OpenAI. The thesis highlights practical use cases in human-robot communication, accessibility, education, and intelligent surveillance systems.

1.5 Structure of the Thesis

This thesis is organized into six chapters. Chapter 2 provides a detailed overview of the theoretical and technical foundations underlying this work. It discusses the YOLOv8 architecture, the datasets used for training and evaluation, the principles of color classification using HSV space and K-Nearest Neighbors, and the integration of the OpenAI GPT API for natural language processing.

Chapter 3 outlines the implementation of the proposed system. It describes the software design, functional modules, interaction flow, and the interface through which users can upload images and interact with the AI.

Chapter 4 presents a discussion of the observed results, identifies key challenges encountered during development, and suggests potential avenues for future research and system enhancement.

Chapter 5 lists the scientific literature and online resources referenced throughout the thesis, presented in a standardized citation format.

Finally, Chapter 6 contains supplementary code excerpts, configuration settings, and architectural diagrams that support the technical implementation described in earlier sections.

Chapter 2

Background

2.1 The YOLO Algorithm: Foundations and Advancements

Object detection, the task of identifying and localizing multiple objects within an image, has seen a dramatic evolution, particularly with the emergence of deep learning. One of the most prominent families of algorithms in this domain is YOLO (You Only Look Once), first introduced by Redmon et al. in 2016 [5]. Unlike earlier approaches such as R-CNN or Faster R-CNN, which required multiple stages of region proposal and classification, YOLO unified the process into a single neural network, enabling real-time performance with competitive accuracy.

The core concept behind YOLO is to treat object detection as a regression problem. An input image is divided into an $S \times S$ grid, with each cell predicting a fixed number of bounding boxes and corresponding confidence scores. This single-shot design drastically reduces computational complexity and latency.

YOLO has undergone several iterations:

- YOLOv1–v4: Developed by Joseph Redmon and later others, these versions introduced improvements in architecture (e.g., darknet), anchor boxes, and training procedures.
- YOLOv5–v8: Developed and maintained by Ultralytics, these versions transitioned to a PyTorch-native framework. YOLOv8, released in 2023, features anchor-free detection, decoupled head modules for improved performance, and support for classification, segmentation, and tracking tasks [6].

YOLOv8 offers a balance of speed and accuracy, and supports real-time inference on edge devices, making it ideal for mobile robotics and user-facing applications.

2.2 Datasets for Object Detection: COCO vs. Open Images V7

Object detection models rely heavily on annotated datasets for supervised training. Two prominent datasets are MS COCO (Common Objects in Context) and Open Images V7.

COCO is a widely used benchmark dataset that includes over 328,000 images with 2.5 million object instances spanning 80 categories. It also provides instance segmentation masks and keypoints for tasks beyond object detection [8].

In contrast, Open Images V7, maintained by Google AI, is one of the largest publicly available datasets, comprising over 9 million images annotated with more than 600 object categories. It introduces multi-label annotations, visual relationships, point-level labels, and image-level tags, providing a richer and more versatile dataset [9].

Feature	COCO	Open Images V7
Number of Images	328 000	9 million+
Object Classes	80	600+
Bounding-Box Annotations	2.5 million	16 million
Instance Segmentation Masks	330 000	2.8 million (350 classes)
Visual Relationship Annotations	—	3.3 million annotations
Image-level Labels	—	61.4 million
Point-Level / Narrative Annotations	—	66.4 million points

Table 2.1: Comparison of COCO and Open Images Version 7 datasets

In this thesis, the pretrained YOLOv8 model is fine-tuned or directly used on Open Images V7 due to its wider range of object categories and support for real-world applications.

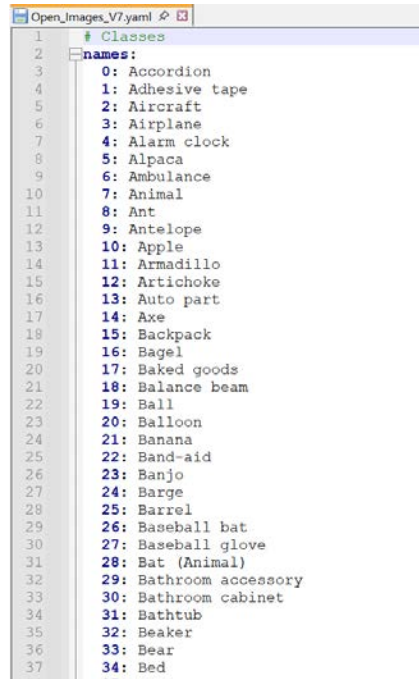


Figure 2.1: Saved class names in the yaml file

2.3 Color Classification Using HSV and K-Nearest Neighbors

While object detection identifies what an object is, it does not inherently classify object color, which is essential for human-centered applications. To address this, color classification is performed using the HSV (Hue–Saturation–Value) color space, which more closely models human perception than the RGB space [10].

- Hue (H): Describes the type of color (e.g., red, green) as an angle from 0° to 360°.
- Saturation (S): Represents color purity from gray (0%) to vivid (100%).
- Value (V): Corresponds to brightness, from black (0%) to full light (100%).

For each detected object, a bounding box is extracted and converted to HSV format. To determine the dominant color, k-means clustering is applied in HSV space (k=4), and the centroid with the highest pixel count is selected.

To assign a human-readable label (e.g., “blue”, “red”), a K-Nearest Neighbors (KNN) classifier is employed:

- A training dataset is built using labeled HSV values from a curated color dataset (5,052 entries, 11 basic colors).

- The classifier computes the nearest neighbors in HSV space to determine the most likely color label.

This approach, inspired by work found on Kaggle and GitHub [11], ensures more robust performance under variable lighting and better aligns with how users describe visual scenes.

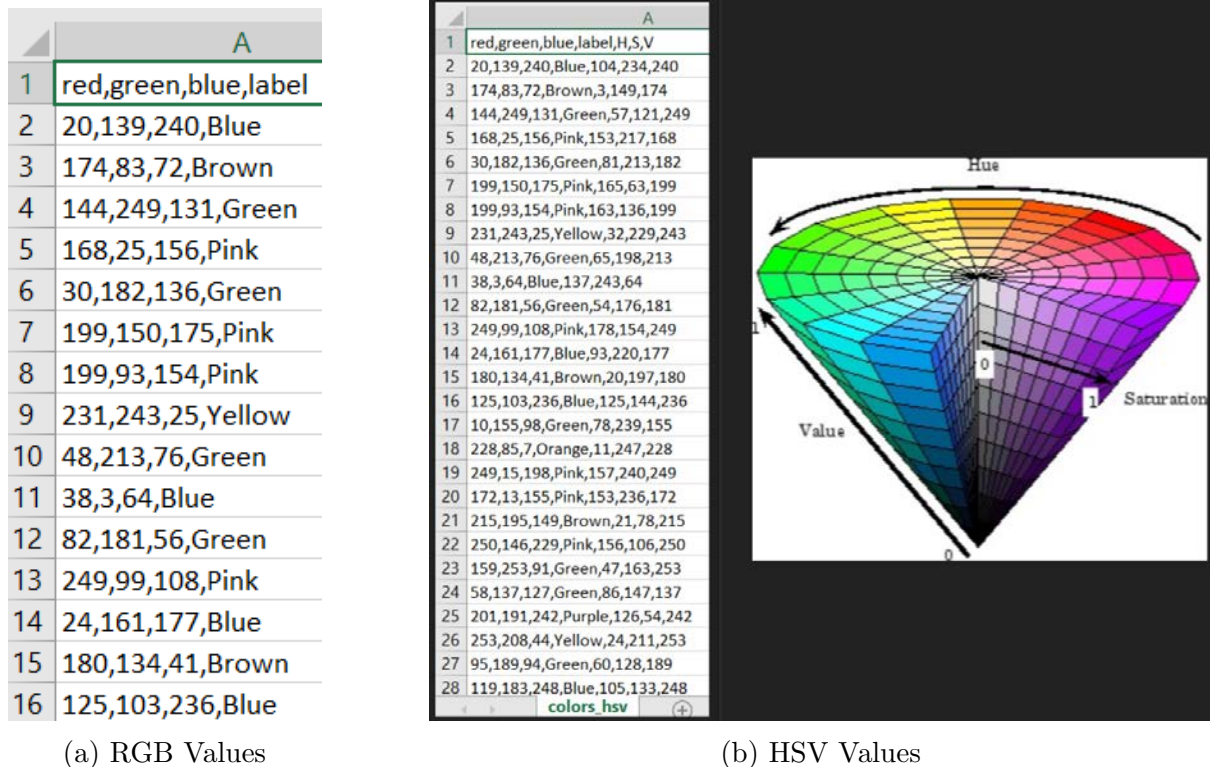


Figure 2.2: Color Dataset from RGB Values to HSV Values [12]

2.4 Integration of the OpenAI API

To enable natural language interaction, the system incorporates the OpenAI GPT API, which provides access to large language models capable of understanding and generating human-like text [13]. Specifically, the gpt-4o-mini model is used due to its efficiency and real-time performance.

Users interact with the system via a text-based interface, issuing queries about object identity, count, color, or historical detection data. These inputs are sent as HTTP requests to the OpenAI API. The response can include:

- A natural language answer (e.g., “The name given to me was Luna. How can I assist you?”).
- A JSON directive, a structured output used internally to trigger system actions such as object re-detection, database updates, or UI refreshes.

All conversations are stored in a chat history database, while detection results are logged in a structured database format, facilitating context-aware interaction and retrieval.

Chapter 3

Implementation

3.1 System Architecture and Functional Flow

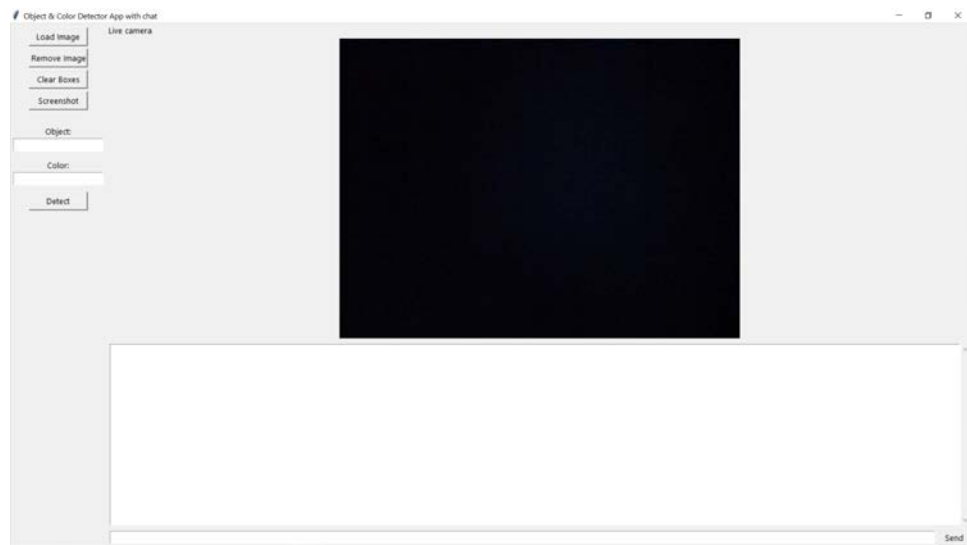


Figure 3.1: App Interface

The proposed system combines visual recognition, color classification, and natural language processing into a unified application capable of operating on static images. The overall architecture comprises several functional modules: an image acquisition and display interface, an object detection engine powered by YOLOv8, a color classification subsystem using HSV space and K-Nearest Neighbors (KNN), an OpenAI-based language interaction module, and a persistent storage system based on SQLite.

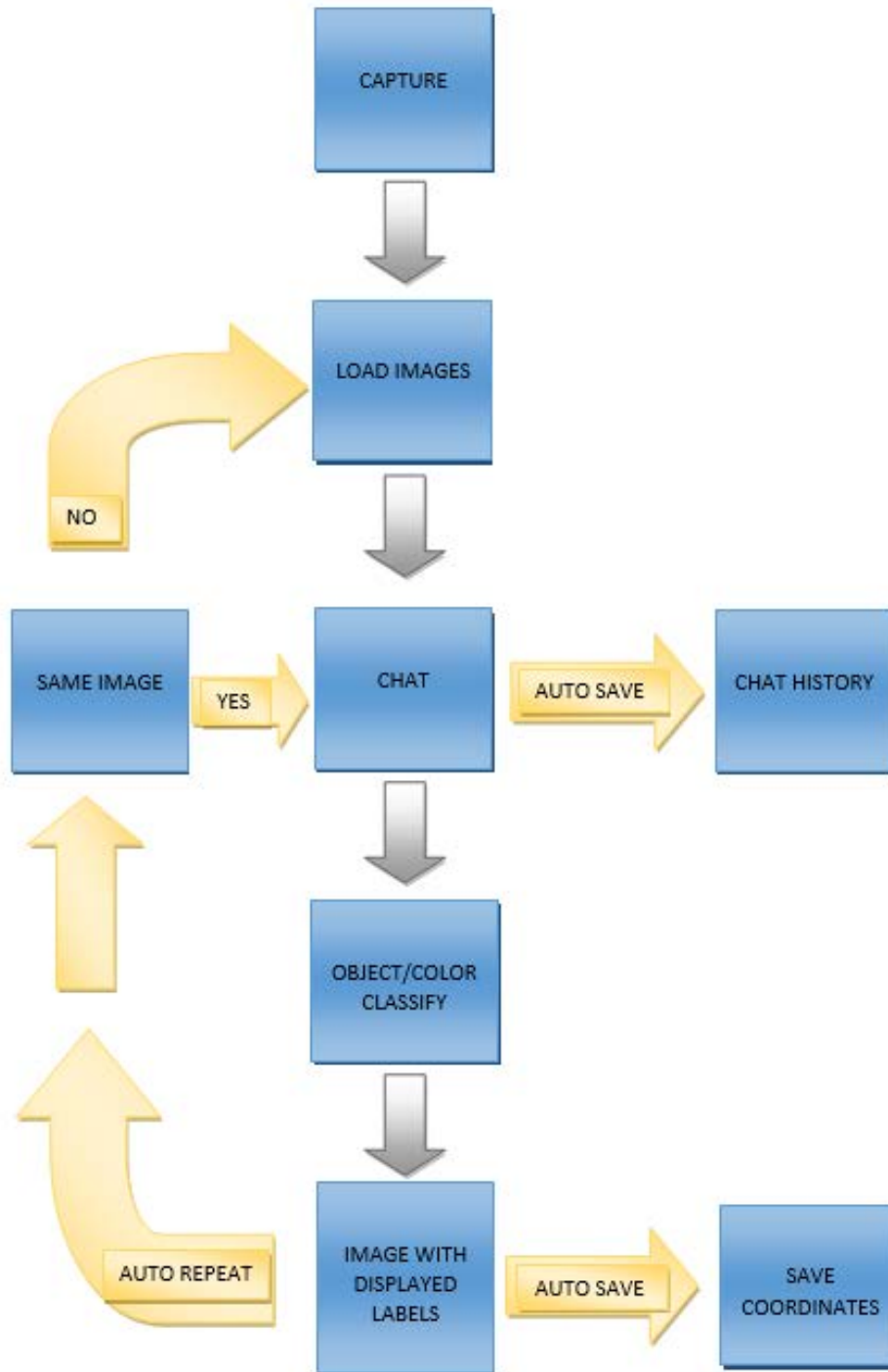


Figure 3.2: Flowchart of the App

The functional flow begins when a user selects or uploads an image via the graphical interface. The image is passed to the object detection module, where the YOLOv8 model performs inference, outputting bounding boxes, confidence scores, and object class predictions. This information is subsequently recorded and stored.

For each bounding box, a color analysis subroutine converts the region of interest to the HSV (Hue–Saturation–Value) color space. K-means clustering (with $k = 4$) is applied to the pixel distribution within this region to identify dominant color clusters. The centroid of the most populous cluster is selected as representative of the object’s color and is passed through a trained KNN classifier to assign a human-readable color label.

Each detection instance — including the object class, bounding box coordinates, dominant color label, and detection timestamp — is stored in a structured SQLite database. Parallel to this, a text-based conversational interface allows users to issue queries or commands in natural language. These interactions are routed to the OpenAI GPT API, which processes the input and returns either free-text answers or structured directives encoded in JSON. This architecture supports both interactive dialogue and automated command execution based on user input.

3.2 Object Detection and Color Analysis Module

The core of the visual analysis pipeline is the YOLOv8 model, a state-of-the-art object detection framework developed by Ultralytics. Implemented in PyTorch, YOLOv8 provides anchor-free object detection with decoupled heads and supports over 600 object categories when pre-trained on datasets such as Open Images V7.

Upon loading an image, the YOLOv8 engine processes it and outputs a list of detected objects, each associated with a bounding box and a confidence score. This information is then parsed and used to extract image regions corresponding to the detected objects.

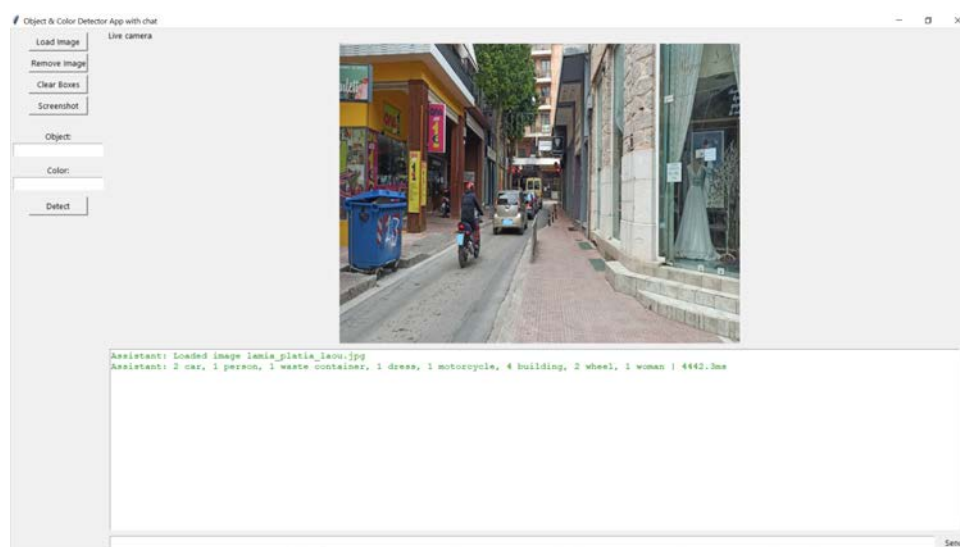


Figure 3.3: Loading an image

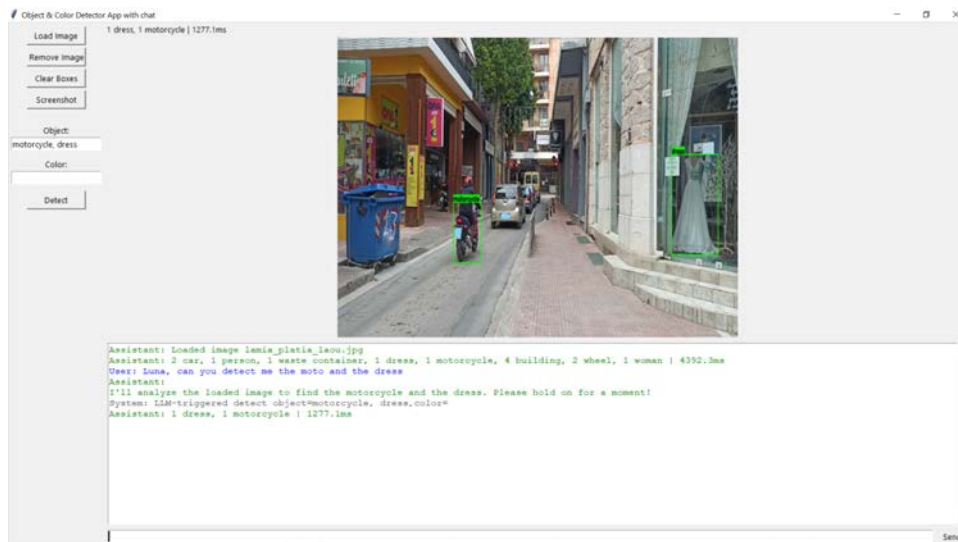


Figure 3.4: Detection of two different classes of objects

For color classification, the extracted regions are converted from RGB to HSV format using OpenCV. HSV is chosen for its better alignment with human color perception, particularly under varying lighting conditions. K-means clustering identifies dominant color regions within each bounding box, and the most representative cluster centroid is selected. This HSV value is then mapped to a semantic color name using a trained KNN classifier that was trained on a manually curated dataset of labeled HSV values corresponding to basic color categories (e.g., red, blue, green, etc.).



Figure 3.5: Detection of two different colored objects



Figure 3.6: Detection of one specific colored object

This two-step process detection followed by color labeling, allows the system to respond accurately to user queries such as “What red objects are present in the image?” or “Show me all green items.”



Figure 3.7: Detection of all colored Green objects

3.3 User Interface and Language Interaction Module

The user interface is implemented using lightweight Python GUI libraries such as Tkinter or PyQt. It supports the following user actions:

- Uploading or selecting an image from local or remote storage.
- Initiating object detection on the selected image.

- Viewing detected objects with visual bounding boxes and class/color labels.
- Communicating with the system via a text input field.

User messages are sent to the language processing module, where they are encoded and transmitted via an HTTP request to the OpenAI GPT-4 API (gpt-4o-mini model). The response from the API may be a plain-language answer or a structured JSON directive, depending on the nature of the user's query. For example, if the user asks "How many objects were detected?", a simple text response is returned. If the user asks "Highlight all red objects," a JSON response may include identifiers that the system uses to update the display and apply visual highlights.

All interactions are recorded in a chat history for contextual tracking, enabling more intelligent, multi-turn conversations.

3.4 Database Schema and Memory Persistence

To enable session persistence and data logging, two local SQLite databases are employed: one for detection metadata and one for chat history.

The detection database (detection.db) records the following fields for each detection event:

- Unique detection ID
- Image name and file path
- Timestamp of detection
- Bounding box coordinates (x1, y1, x2, y2)
- Detected object class label
- Assigned color label

id	image_path	detected_at	x1	y1	x2	y2	object	color
563	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/lamia_platia_1_	2025-05-23 18:23:20	1336	289	1521	536	dress	NULL
564	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/lamia_platia_1_	2025-05-23 18:23:20	466	406	577	555	motorcycle	NULL
565	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/lamia_platia_1_	2025-05-23 18:27:25	615	356	756	473	car	grey
566	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/lamia_platia_1_	2025-05-23 18:27:25	734	362	792	432	car	grey
567	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:28:45	148	55	161	216	NULL	green
568	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:28:45	186	61	201	202	NULL	green
569	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:28:45	167	70	181	200	NULL	green
570	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:29:56	129	54	143	217	office supplies	yellow
571	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:29:56	55	54	68	221	office supplies	red
572	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:30:36	140	9	165	214	pen	green
573	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:30:36	186	61	201	202	pen	green
574	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:30:59	148	55	161	216	office supplies	green
575	C:/Users/xvasi/Desktop/Bathia_uadham_eyyaata/saved_images/pens.jpg	2025-05-23 18:30:59	167	70	181	200	office supplies	green

Figure 3.8: Tablet of the Detection.db

The chat history database (chat_history.db) stores the flow of interaction between the user and the AI system. Each record contains:

- Unique interaction ID
- The role of who is speaking (user/assistant)
- Associated JSON directive (User input string/Model response)
- Timestamp of interaction

id	role	content	timestamp
338	assistant	{"detect": true, "object": "human face", "color": "grey"}\n\nI'll analyze the load.	2025-05-23 17:57:27
339	user	Luna, can you show me the white dress and the motor	2025-05-23 18:20:48
340	assistant	{"detect": true, "object": "dress, motorcycle", "color": "white"}\n\nI'll analyze	2025-05-23 18:20:54
341	user	Luna, can you detect me the moto and the dress	2025-05-23 18:22:59
342	assistant	{"detect": true, "object": "motorcycle, dress", "color": ""}\n\nI'll analyze the l.	2025-05-23 18:23:04
343	user	show me the white dress	2025-05-23 18:26:23
344	assistant	{"detect": true, "object": "dress", "color": "white"}\n\nI'll check the loaded ima.	2025-05-23 18:26:27
345	user	show me a grey car	2025-05-23 18:27:07
346	assistant	{"detect": true, "object": "car", "color": "grey"}\n\nI'll check the loaded image	2025-05-23 18:27:10
347	user	show me all green objects	2025-05-23 18:28:39
348	assistant	{"detect": true, "object": "", "color": "green"}\n\nI'll analyze the loaded image	2025-05-23 18:28:42
349	user	Detect me only the red pen and a yellow office supplies	2025-05-23 18:29:51
350	assistant	{"detect": true, "object": "pen, office supplies", "color": "red,yellow"}\n\nI'll	2025-05-23 18:29:53
351	user	show me the green pen	2025-05-23 18:30:30
352	assistant	{"detect": true, "object": "pen", "color": "green"}\n\nI'll check the loaded image.	2025-05-23 18:30:33
353	user	the green office	2025-05-23 18:30:54
354	assistant	{"detect": true, "object": "office supplies", "color": "green"}\n\nI'll check the	2025-05-23 18:30:56

Figure 3.9: Tablet of Chat history.db

These databases support both the recall of past detection events (e.g., "Can you tell me what objects did you detect today", "Number each of the objects that you detected from 2025-05-12 to 2025-05-23", "what colors you find today") and context maintenance across conversation turns. In addition, they serve as a valuable log for debugging and performance evaluation of the system.



Figure 3.10: Recollection of past detections

Chapter 4

Discussion and Conclusions

The developed system successfully integrates object detection with 75%. The YOLOv8 has demonstrated robust performance in real-time object detection tasks, effectively identifying and localizing objects within images. The incorporation of HSV color space analysis and K-Nearest Neighbors (KNN) classification has enabled the system to assign color labels to detected objects with precision 75%, enhancing the descriptive quality of the outputs.

The natural language interface allows users to interact with the system intuitively, posing queries about object presence, quantity, and attributes. This multi-modal interaction framework aligns with contemporary trends in human-computer interaction, promoting accessibility and user engagement.

4.1 Primary Failures

The significant low accuracy is due to the lighting and the depth of the image have as results that objects collapse into a single bounding box or one object masks another and reducing recall, the cluttering scenes, background patches can trigger high-confidence detections, driving down precision for objects under 32×32 pixels. As an example, in Figure 4.1 we see that the van is not highlighted because the traffic sign is at the front and the blue waste container is not visible because it is too far behind.

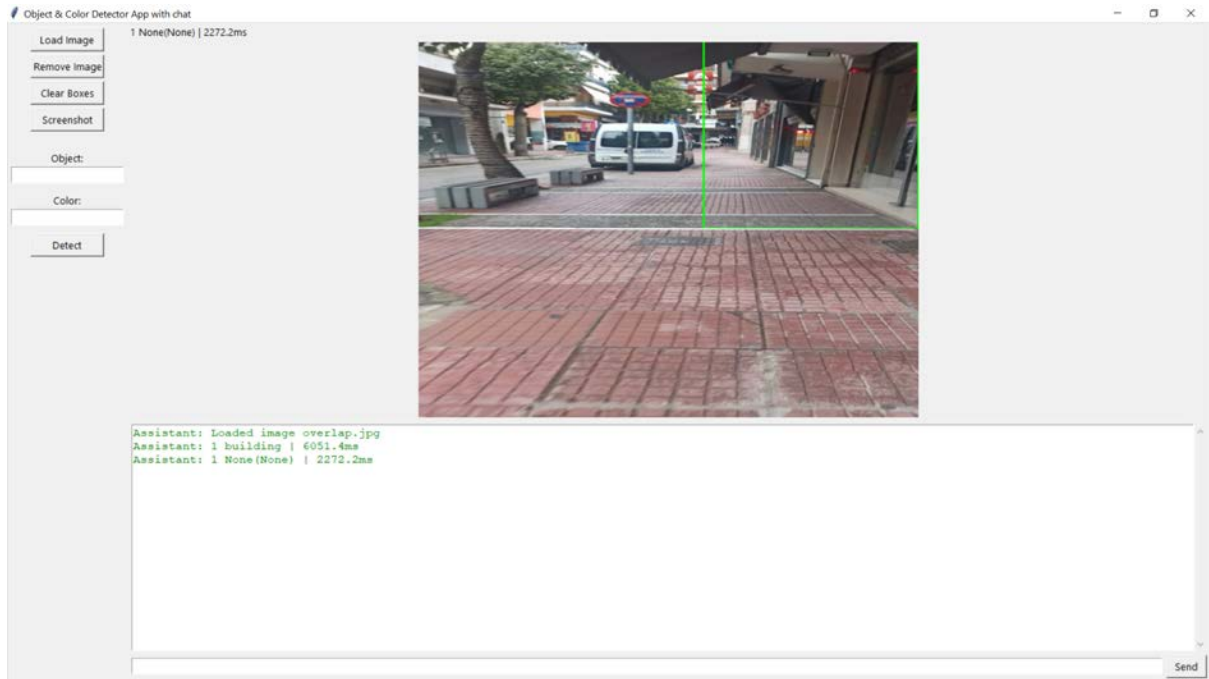


Figure 4.1: Failed detection

The color classification process is particularly sensitive to environmental lighting and intra-class variance in color representation. For instance, the many values for a single labeled color. This leads to challenges such as distinguishing black from grey, or brown from red etc.(refer to Figures 4.2 and 4.3).



Figure 4.2: Confusing color black from grey

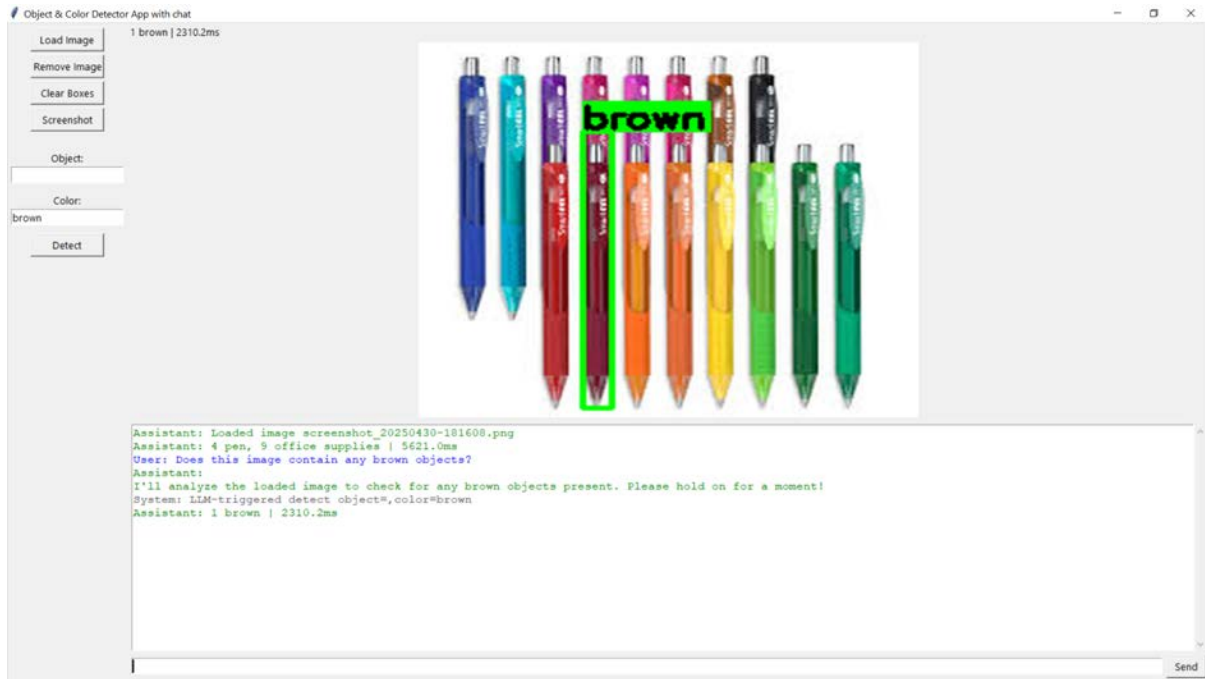


Figure 4.3: Failing detecting the color brown

4.2 Limitations

Despite its overall effectiveness, the current system exhibits several notable limitations. First, it processes only static images and cannot analyze live video feeds, which restricts its applicability in scenarios that require continuous, real-time monitoring. Second, our color-classification module relies on a modestly sized HSV dataset of basic color labels, this narrow scope can lead to misclassifications when objects exhibit subtle shades or unusual lighting conditions. Third, the natural-language interface depends on an external API, making the application vulnerable to network outages or rate limits and unsuitable for offline use. Finally, without any mechanism to organize or deleting older entries, both the detection and chat databases will grow indefinitely, over time this can degrade performance, slow down the detection process, and risk data corruption.

4.3 Practical Implications and Use Cases

By uniting vision and language, the system opens up a range of real-world applications. In assistive technology, it can describe surroundings to users with visual impairments, enhancing autonomy and safety. In education, students can experiment interactively with computer-vision concepts and immediately seeing the results. In security and surveillance contexts, operators can issue natural-language commands, accelerating the analysis of previously captured videos.

4.4 Future Work

Looking ahead, there are several straightforward ways to make the system even more capable. First, adding real-time video processing would let it analyze scenes on the fly, perfect for robots, security cameras and live streams, rather than just static images. Second, training the color classifier on a larger, more diverse palette of samples would help it distinguish colors under any lighting conditions. Third, instead of writing separate code routines for every possible user phrase, we could integrate a small large-language model (LLM) that naturally understands and handles a wide range of commands. We can add a basic language model that runs on the device itself, so the chat still works even when there's no Internet. Finally, implementing a simple data management routine that automatically deletes old or duplicate records and keeps only recent.

4.5 Conclusion

The project demonstrates the feasibility and effectiveness of combining object detection algorithms and color classification with natural language processing to create an interactive system capable of analyzing and describing visual content. While certain limitations exist, the system lays a solid foundation for future developments in multimodal human-computer interaction.

Chapter 5

Bibliography

- [1]. Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer.
- [2] Vaswani, A. et al. (2017). "Attention is All You Need." Advances in Neural Information Processing Systems.
- [3] Dalal, N., and Triggs, B. (2005). "Histograms of Oriented Gradients for Human Detection." CVPR. [4] Ren, S. et al. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." NeurIPS.
- [5] Redmon, J., et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection." CVPR. [6] Ultralytics. (2023). YOLOv8 Documentation. <https://docs.ultralytics.com/>
- [7] Radford, A. et al. (2021). "Learning Transferable Visual Models From Natural Language Supervision." ICML
- [8] Lin, T.Y. et al. (2014). "Microsoft COCO: Common Objects in Context." ECCV.
- [9] Ultralytics YOLO Docs, "The Open Images Dataset V7":<https://docs.ultralytics.com/datasets/detect/open-images-v7/>
- [10] MathWorks. "Understanding Color Spaces and Conversion.":
<https://www.mathworks.com/help/images>
- [11] <https://github.com/AjinkyaChavan9/RGB-Color-Classifer-with-Deep-Learning-using-Keras-and-Tensorflow/tree/master/Dataset>
- [12] MathWorks. Understanding Color Spaces and Color Space Conversion:
<https://www.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html>

[13] OpenAI. “API Reference Documentation.” <https://platform.openai.com/docs/>

Chapter 6

Code

The structure of the implemented code is presented below in Listing 6.1.

Environment and Imports

In the very first lines of our application, we import and configure all of the external libraries that we use throughout the code.(Lines 1-28)

We use Tkinter (with `filedialog`, `scrolledtext`, `Button`, and `Label`) and Pillow (Image/ImageTk) to build a lightweight desktop interface, where the user can select an image file, display it on a canvas, and carry on a text dialogue.

With OpenCV (`cv2`) and NumPy (`np`) we manage all pixel-level operations like converting to HSV color space, cropping detections, and drawing annotated bounding boxes and serve as inputs to our color-analysis.

For object detection, we employ Ultralytics YOLOv8, while Joblib and YAML load any pre-trained clustering or classification models and human-readable configuration files (e.g.the k-means parameters).

Core Python modules (`os`, `time`, `datetime`, and `sqlite3`) handle environment variables such as OPENAI API KEY, time-stamped logging of detections and chat messages to an SQLite database, and filesystem path management. Natural-language capabilities are provided by the OpenAI client (`openai`) and TikToken. We also import `json` and `re` to parse structured JSON directives or extract information from free-text replies, as well as Python utilities like `collections.Counter` is being used for past detections (counting objects/colors) and `difflib.get_close_matches`, for future typo-correction of user inputs.

At startup, the code verifies that the OPENAI API KEY environment variable is set. To minimize usage costs while maintaining sufficient context, we specify the gpt-4o-mini-2024-07-18 model with a maximum of 8,000 tokens in the context window and a 150-token limit on each response.

We create the SQLite files for the databases one for logging every detection and one for recording the entire chat history. (lines 30-58)

It defines `DET_DB = "detections.db"` and opens a connection to that file (creating it if it doesn't already exist), with `check_same_thread=False` to allow access from different parts of the program. It then obtains a cursor and executes a `"CREATE TABLE IF NOT EXISTS"` statement to ensure there is a detections table with columns for a unique id, the image_path used, the timestamp, the bounding box coordinates (x1, y1, x2, y2), and the detected object class and its color label.

The same pattern is repeated for the chat history: `CHAT_DB = "chat_history.db"` opens or creates that database file, sets up a cursor, and ensures that a chat_history table exists with columns for id, the role, the message content, and the timestamp of each turn.

Model Loading and Helper Functions

We load the data and models that form the core of our computer-vision and language model components and define two functions for color extraction and token counting. (lines 59-93) First, we load our object detector and color classifier: YOLOv8 model using the specified weights file (`yolov8x-oiv7.pt`), and `knn = joblib.load()` which stores the pre-trained K-Nearest-Neighbors classifier. The `Open_Images_V7.yaml` file and `yaml.safe_load()` (which define the class names on which YOLOv8 is trained on). We then construct two data structures: a `class_names` dictionary that maps each class ID to its lowercase label, and an `all_classes` list containing every class name for lookup.

The `get_dominant_color` function converts the patch from RGB to HSV color space, flattens its pixels into a 2D array, and applies OpenCV's `kmeans`. By counting which cluster label appears most often, it selects the corresponding HSV centroid as the "dominant" color and returns it as an integer RGB triplet.

The `num_tokens(messages)` function uses `TikToken` to estimate how many tokens a list of chat messages will consume under the gpt-4o-mini token limit. For each message in the conversation history, it adds 4 tokens per message, along with 2 additional tokens for the model's response. The total token count ensures that the message history does not exceed the `MAX_TOKENS_CONTEXT` limit before sending a request to the API.

With `format_date_str(since, until)` we default the time, which we then can use for log summaries.

We define the AI's character, (Lines 112-119), by creating a `chat_history` list. It tells the model to behave like a friendly vision assistant and in any user request for object or color detection it response to in JSON directive, in any other situation reply in plain human language.(Lines 95-111)

The `save_chat(role, content)` and `send_chat_to_gpt(message)` functions ensure that every conversation turn (between the user and the assistant) informs the next and the context never exceeds the model limits.

Graphical User Interface (GUI)

Regarding the UI of the app, (lines 137-187), we create a Tkinter window titled “Object & Color Detector App with chat” and divide it into three main frames: the left control panel for buttons and entry fields, the right display area panel for the video / image canvas, and the bottom panel for chat interface.

In the control panel there are four buttons, for loading an image, clearing the current image, removing any drawn bounding boxes, or taking a screenshot. Below those, are two entry fields (“Object” and “Color”) and a “Detect” button that triggers the detection pipeline with those filters.

On the right, a status label shows whether you're viewing the live camera or a loaded image. Below it, a 640×480 Tkinter canvas displays either the live video feed or the currently loaded image.

Under the canvas sits a chat interface that records the entire conversation, each tagged in a different color for clarity. At the bottom of it there is a single-line entry and a “Send” button for typing a message.

Below is a breakdown of the button functionalities mentioned earlier.(Lines 188-241)

First, when the user clicks “Load Image,” a file-selection dialog prompts them to choose a JPEG or PNG file. Once an image is selected, the application immediately pauses the live-camera feed and renders the static image on the canvas—first converting it to RGB, then resizing it to 640×480. During the same time, the system uses the Ultralytics YOLOv8 model to identify each class of objects present in the image.

The `clear_boxes` method simply redraws the original image without overlays as for the `clear_image` it removes the image entirely and activates back the live camera. With the `save_screenshot` captures and saves the current webcam frame.

Finally, `display_original` centralizes the logic for loading an image from a directory converting it into a Tkinter `PhotoImage` and drawing it on the canvas, used whenever is need to reset the original visual of the image.

Detection Logic

The `detect` method, (lines 242-344), (Lines carries out our end-to-end vision pipeline: when running (either manually via the “Detect” button or automatically), it first ensures that an image has been loaded, otherwise it asks the user to do so. It then reads the image and with OpenCV and calls the YOLOv8 model to perform a object detection pass, timing the operation in milliseconds. For each bounding box returned, it extracts its integer coordinates (x1,y1,x2,y2), look up the object class name, and crop the corresponding region from the image. That cropped patch is converted to HSV and clustered via our `get_dominant_color` routine for the color classification it then passes through the pre-trained KNN classifier to produce the color label.

Once every detection has been paired with an object and color, the ai answers back with a JSON directive, it timestamp the operation and then synchronize these filtered detections with the SQLite “detections” table. If a detection lacks a color or object it the saves them as NULL. After committing to the database, it prepares a visual overlay by converting the original image to RGB, drawing green rectangles box for each match, and rendering the object and/or color label above the box.

The `query_detections` method provides a SQL-based interface for counting how many times a particular object-color combination was detected within an optional date range.

For recalling past detections: when the user types a text into the chat entry and presses “Send,” the `send_chat` method converts the text to lowercase and initializes “since” and “until” timestamps to `None`.(Lines 347-541)

For the ai, we set functions to check for each expression the user may have specified for time like: “from .. to ...”, both “since” and “until”, “yesterday,” or “today”. It also handles phrases such as “x days ago” or “last x weeks,” translating them into appropriate “since” or “until” datetimes by subtracting days, weeks, months/years. By the end of these checks, any failed date parses are silently ignored, leaving the filter unchanged. The `build_where` constructs a SQL-based interface where it filters the object name, color label with the appropriate date range.

After the construction of a SQL-based interface (with the `build_where` function) we performs a series of regular-expression checks but this time is for searching the objects/-colors and the amount of each. It answers questions like: “what objects”, “what colors” and “number of each/count of each object/color”

The function `_append_chat`, keeps the chat display consistent with color-aware, (with the User being color blue, Assistant with color green and System with the dark grey), and protected from accidental modification.

The “if `__name__ == '__main__'`” ensures that `VisionApp()` is only executed when the file is run as a standalone program.(Lines 543-550)

```

1 import tkinter as tk
2 from tkinter import filedialog, scrolledtext, Button, Label
3 from PIL import Image, ImageTk
4 import cv2
5 import numpy as np
6 import os
7 import sqlite3
8 import time
9 import datetime
10 import joblib # type: ignore
11 import yaml
12 import openai
13 import tiktoken
14 import json
15 import re
16 from ultralytics import YOLO # type: ignore
17 from collections import Counter
18 from difflib import get_close_matches # noqa: F401
19
20 #openai_api
21 OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
22 if not OPENAI_API_KEY:
23     raise RuntimeError("OPENAI_API_KEY environment variable is not set!")
24
25 openai.api_key = OPENAI_API_KEY
26 MODEL = 'gpt-4o-mini-2024-07-18'
27
28 MAX_TOKENS_CONTEXT = 8000
29 MAX_TOKENS_RESPONSE = 150
30
31 #create files to save coordinates and chat history
32 DET_DB = "detections.db"
33 det_conn = sqlite3.connect(DET_DB, check_same_thread=False)
34 det_cur = det_conn.cursor()
35 det_cur.execute("""
36 CREATE TABLE IF NOT EXISTS detections (
37     id INTEGER PRIMARY KEY,
38     image_path TEXT,
39     detected_at TEXT,
40     x1 INT, y1 INT, x2 INT, y2 INT,
41     object TEXT,
42     color TEXT
43 )
44 """)
45 det_conn.commit()
46
47 CHAT_DB = "chat_history.db"
48 chat_conn = sqlite3.connect(CHAT_DB, check_same_thread=False)

```

```

48 chat_cur = chat_conn.cursor()
49 chat_cur.execute("""
50 CREATE TABLE IF NOT EXISTS chat_history (
51     id INTEGER PRIMARY KEY,
52     role TEXT,
53     content TEXT,
54     timestamp TEXT
55 )
56 """)
57 chat_conn.commit()
58
59 # Models and Dataset
60 yolo_model = YOLO(r"Path_to_model/yolov8x-oiv7.pt")
61 knn = joblib.load(r"Path_to_model/knn_model.pkl")
62 with open(r"Path_to_model/Open_Images_V7.yaml", encoding="utf-8") as f:
63     data = yaml.safe_load(f)
64 class_names = {int(k): v.lower() for k, v in data['names'].items()}
65 all_classes = list(class_names.values())
66
67 # Get color
68 def get_dominant_color(crop, k=4):
69     hsv = cv2.cvtColor(crop, cv2.COLOR_BGR2HSV)
70     pts = hsv.reshape(-1, 3).astype(np.float32)
71     _, labels, centers = cv2.kmeans(
72         pts, k, None,
73         (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2),
74         10, cv2.KMEANS_RANDOM_CENTERS
75     )
76     most = Counter(labels.flatten()).most_common(1)[0][0]
77     return tuple(map(int, centers[most]))
78
79 def num_tokens(messages):
80     enc = tiktoken.encoding_for_model(MODEL)
81     total = 0
82     for m in messages:
83         total += 4 + len(enc.encode(m['content'])) + 2
84     return total
85
86 def format_date_str(since, until):
87     if since and until:
88         return f"{since.strftime('%d/%m/%Y')} to {until.strftime('%d/%m/%Y')}"
89     elif since:
90         return f"{since.strftime('%d/%m/%Y')}"
91     else:
92         return "all time"
93
94

```

```

95 #Ai role and character
96 chat_history = [{
97     'role': 'system',
98     'content': """
99     You are Luna, a friendly, conversational vision assistant that
100     speaks just like a human.
101     - Whenever the user asks you to detect objects or colors
102     in the current image,
103     you must emit exactly one JSON line at the top:
104     {"detect": true, "object": "<comma-separated filters or empty
105     >", "color": "<comma-separated filters or empty>"}
106     and nothing else on that line.
107     - If the user is simply chatting or asking general questions,
108     reply in plain human language.
109     - If the users object or color terms dont exactly match
110     one of your classes, propose the closest match.
111     """
112 }]
113 chat_cur.execute("SELECT role, content FROM chat_history ORDER BY id")
114 for role, content in chat_cur.fetchall():
115     chat_history.append({'role': role, 'content': content})
116
117 def save_chat(role, content):
118     ts = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
119     chat_cur.execute(
120         "INSERT INTO chat_history(role,content,timestamp) VALUES
121         (?,?/?)",
122         (role, content, ts)
123     )
124     chat_conn.commit()
125
126 def send_chat_to_gpt(message):
127     chat_history.append({'role': 'user', 'content': message})
128     save_chat('user', message)
129     while num_tokens(chat_history) > MAX_TOKENS_CONTEXT and len(
130         chat_history) > 2:
131         chat_history.pop(1)
132     resp = openai.ChatCompletion.create(
133         model=MODEL,
134         messages=chat_history,
135         max_tokens=MAX_TOKENS_RESPONSE,
136         temperature=0.7 # a balance of coherence and creativity
137     )
138     reply = resp.choices[0].message['content'].strip()
139     chat_history.append({'role': 'assistant', 'content': reply})
140     save_chat('assistant', reply)
141     return reply

```

```

136
137 #APP
138 class VisionApp:
139     def __init__(self):
140         self.root = tk.Tk()
141         self.root.title("Object & Color Detector App with chat")
142         self.current_image = None
143         self.last_matches = []
144         self.cap = cv2.VideoCapture(0)
145
146         main = tk.Frame(self.root)
147         main.pack(fill='both', expand=True)
148         ctrl = tk.Frame(main)
149         ctrl.pack(side='left', fill='y', padx=5, pady=5)
150         disp = tk.Frame(main)
151         disp.pack(side='right', fill='both', expand=True)
152
153         for txt, cmd in [
154             ("Load Image", self.load_image),
155             ("Remove Image", self.clear_image),
156             ("Clear Boxes", self.clear_boxes),
157             ("Screenshot", self.save_screenshot)
158         ]:
159             Button(ctrl, text=txt, width=12, command=cmd).pack(pady=2)
160             Label(ctrl, text="Object:").pack(pady=(20,0))
161             self.obj_entry = tk.Entry(ctrl)
162             self.obj_entry.pack()
163             Label(ctrl, text="Color:").pack(pady=(10,0))
164             self.col_entry = tk.Entry(ctrl)
165             self.col_entry.pack()
166             Button(ctrl, text="Detect", width=12, command=lambda: self.
                detect(auto=False)).pack(pady=10)
167
168         self.status = Label(disp, text="Live camera", anchor='w')
169         self.status.pack(fill='x')
170         self.canvas = tk.Canvas(disp, width=640, height=480)
171         self.canvas.pack()
172         cf = tk.Frame(disp)
173         cf.pack(fill='both', expand=True)
174         self.chat_display = scrolledtext.ScrolledText(cf, state='
            disabled', height=12)
175         for tag, col in [("User", "blue"), ("Assistant", "green"), ("System
            ", "#505050")]:
176             self.chat_display.tag_config(tag, foreground=col)
177         self.chat_display.pack(fill='both', expand=True, padx=5, pady
            =5)
178         ef = tk.Frame(cf)
179         ef.pack(fill='x', padx=5, pady=(0,5))

```

```

180     self.chat_entry = tk.Entry(ef)
181     self.chat_entry.pack(side='left', fill='x', expand=True, padx
        =(0,5))
182     Button(ef, text="Send", width=6, command=self.send_chat).pack(
        side='right')
183     self.chat_entry.bind("<Return>", lambda event: self.send_chat()
        )
184
185     self.update_camera()
186     self.root.mainloop()
187
188     def update_camera(self):
189         ret, frame = self.cap.read()
190         if ret and not self.current_image:
191             img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
192             pil = Image.fromarray(img).resize((640,480), Image.
                Resampling.LANCZOS)
193             self.photo = ImageTk.PhotoImage(pil)
194             self.canvas.create_image(0,0,anchor='nw', image=self.photo)
195             self.root.after(30, self.update_camera)
196
197     def load_image(self):
198         path = filedialog.askopenfilename(filetypes=[("Images", "*.jpg
                *.png")])
199         if not path:
200             return
201         self.current_image = path
202         self.last_matches = []
203         self.display_original()
204         start = time.time()
205         results = yolo_model(path)[0]
206         elapsed = (time.time()-start)*1000
207         counts = Counter(class_names[int(b.cls[0])] for b in results.
            boxes)
208         summary = ", ".join(f"{v} {k}" for k,v in counts.items()) + f"
            | {elapsed:.1f}ms"
209         self._append_chat("Assistant", f"Loaded image {os.path.basename
            (path)}")
210         self._append_chat("Assistant", summary)
211
212     def clear_boxes(self):
213         if not self.current_image:
214             return
215         self.last_matches = []
216         self.display_original()
217         self.status.config(text=f"Loaded {os.path.basename(self.
            current_image)} (boxes cleared)")
218         self._append_chat("System", "Cleared bounding boxes")

```

```

219
220 def clear_image(self):
221     self.current_image = None
222     self.last_matches = []
223     self.canvas.delete("all")
224     self.status.config(text="Live camera")
225     self._append_chat("System", "Back to live")
226
227 def save_screenshot(self):
228     ts = time.strftime("%Y%m%d-%H%M%S")
229     frame = cv2.imread(self.current_image) if self.current_image
230         else self.cap.read()[1]
231     fname = f"screenshot_{ts}.png"
232     d = filedialog.askdirectory() or "."
233     cv2.imwrite(os.path.join(d, fname), frame)
234     self._append_chat("System", f"Saved {fname}")
235
236 def display_original(self):
237     if not self.current_image:
238         return
239     pil = Image.open(self.current_image).resize((640,480), Image.
240         Resampling.LANCZOS)
241     self.photo = ImageTk.PhotoImage(pil)
242     self.canvas.create_image(0,0,anchor='nw', image=self.photo)
243
244 #Detection, matching of classes, display of the images with labels
245 def detect(self, auto=False):
246     if not self.current_image:
247         self.status.config(text="Load an image first")
248         return
249     img = cv2.imread(self.current_image)
250     start = time.time()
251     results = yolo_model(self.current_image)[0]
252     elapsed = (time.time()-start)*1000
253
254     allr = []
255     for b in results.bboxes:
256         x1,y1,x2,y2 = map(int, b.xyxy[0])
257         o = class_names[int(b.cls[0])]
258         dom = get_dominant_color(img[y1:y2, x1:x2])
259         c = knn.predict([dom])[0].lower()
260         allr.append((x1,y1,x2,y2,o,c))
261
262     user_terms = [s.strip().lower() for s in self.obj_entry.get().
263         split(',') if s.strip()]
264     user_cols = [s.strip().lower() for s in self.col_entry.get().
265         split(',') if s.strip()]

```

```

263 matches = []
264 for x1,y1,x2,y2,o,c in allr:
265     if (not user_terms or o in user_terms) and (not user_cols
266         or c in user_cols):
267         save_o = o if user_terms else None
268         save_c = c if user_cols else None
269         matches.append((x1,y1,x2,y2,o,c, save_o, save_c))
270
271 self.last_matches = matches
272 ts = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
273
274 for x1,y1,x2,y2,o,c,save_o,save_c in matches:
275     if save_c is not None and save_o is not None:
276         det_cur.execute("""
277             UPDATE detections
278             SET color = ?
279             WHERE image_path = ?
280             AND x1 = ? AND y1 = ? AND x2 = ? AND y2 = ?
281             AND object = ?
282             AND color IS NULL
283             """, (save_c, self.current_image, x1, y1, x2, y2,
284                 save_o))
285         if det_cur.rowcount > 0:
286             continue
287
288         det_cur.execute("""
289             INSERT INTO detections(image_path,detected_at,x1,y1,x2,
290             y2,object,color)
291             VALUES (?, ?, ?, ?, ?, ?, ?, ?)
292             """, (
293                 self.current_image, ts,
294                 x1, y1, x2, y2,
295                 save_o, save_c
296             ))
297
298 det_conn.commit()
299
300 disp = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
301 for x1,y1,x2,y2,o,c,save_o,save_c in matches:
302     cv2.rectangle(disp, (x1,y1), (x2,y2), (0,255,0), 2)
303     label = ""
304     if save_o and save_c:
305         label = f"{save_o}({save_c})"
306     elif save_o:
307         label = save_o
308     elif save_c:
309         label = save_c
310     if label:

```

```

308         (w,h), _ = cv2.getTextSize(label, cv2.
309             FONT_HERSHEY_SIMPLEX, 0.6, 2)
310         cv2.rectangle(dispatch, (x1, y1 - h - 4), (x1 + w, y1),
311             (0,255,0), -1)
312         cv2.putText(dispatch, label, (x1, y1 - 2), cv2.
313             FONT_HERSHEY_SIMPLEX, 0.6, (0,0,0), 2)
314
315     pil = Image.fromarray(dispatch).resize((640,480), Image.Resampling.
316         LANCZOS)
317     self.photo = ImageTk.PhotoImage(pil)
318     self.canvas.create_image(0,0,anchor='nw', image=self.photo)
319
320     cnt = Counter(
321         save_o if save_o and not save_c else
322         save_c if save_c and not save_o else
323         f"{save_o}({save_c})"
324         for *,save_o,save_c in matches
325     )
326     summary = ", ".join(f"{v} {k}" for k,v in cnt.items()) + f" | {
327         elapsed:.1f}ms"
328     self.status.config(text=summary)
329     if not auto:
330         self._append_chat("Assistant", summary)
331
332     def query_detections(self, object=None, color=None, since=None,
333         until=None):
334         clauses, params = [], []
335         if object:
336             clauses.append("object = ?")
337             params.append(object)
338         if color:
339             clauses.append("color = ?")
340             params.append(color)
341         if since:
342             clauses.append("detected_at >= ?")
343             params.append(since.strftime("%Y-%m-%d %H:%M:%S"))
344         if until:
345             clauses.append("detected_at <= ?")
346             params.append(until.strftime("%Y-%m-%d %H:%M:%S"))
347         w = "WHERE " + " AND ".join(clauses) if clauses else ""
348         cur = det_conn.cursor()
349         cur.execute(f"SELECT COUNT(*) FROM detections {w}", params)
350         return cur.fetchone()[0]
351
352     #function for each matching expression
353     def send_chat(self):
354         text = self.chat_entry.get().strip()
355         if not text:

```

```

350         return
351
352     self._append_chat("User", text)
353
354     low = text.lower()
355     now = datetime.datetime.now()
356     since = None
357     until = None
358
359     # Match expressions like "from ... to ..."
360     between = re.search(r"\bfrom\s+(\d{4}-\d{2}-\d{2})\s*(to|-)\s*
361         *(\d{4}-\d{2}-\d{2})", low)
362     if between:
363         try:
364             since = datetime.datetime.strptime(between.group(1), "%
365                 Y-%m-%d")
366             until = datetime.datetime.strptime(between.group(3), "%
367                 Y-%m-%d") + datetime.timedelta(days=1)
368         except ValueError:
369             pass
370
371     # Match 'since <date>'
372     since_m = re.search(r"\bsince\s+(\d{4}-\d{2}-\d{2})", low)
373     if since_m:
374         try:
375             since = datetime.datetime.strptime(since_m.group(1), "%
376                 Y-%m-%d")
377         except ValueError:
378             pass
379
380     # Match 'until <date>'
381     until_m = re.search(r"\buntil\s+(\d{4}-\d{2}-\d{2})", low)
382     if until_m:
383         try:
384             until = datetime.datetime.strptime(until_m.group(1), "%
385                 Y-%m-%d") + datetime.timedelta(days=1)
386         except ValueError:
387             pass
388
389     # Handle standard keywords
390     if re.search(r"\bthe day before yesterday\b", low):
391         target = now.date() - datetime.timedelta(days=2)
392         since = datetime.datetime.combine(target, datetime.time.min)
393         until = datetime.datetime.combine(target, datetime.time.max)
394     elif re.search(r"\byesterday\b", low):
395         target = now.date() - datetime.timedelta(days=1)

```

```

391         since = datetime.datetime.combine(target, datetime.time.min
392         )
393         until = datetime.datetime.combine(target, datetime.time.max
394         )
395     elif re.search(r"\btoday\b", low):
396         target = now.date()
397         since = datetime.datetime.combine(target, datetime.time.min
398         )
399         until = datetime.datetime.combine(target, datetime.time.max
400         )
401
402     # Handle expressions like "x days ago"
403     ago = re.search(r"(\d+)\s*days?\s+ago", low)
404     if ago:
405         n = int(ago.group(1))
406         target = now.date() - datetime.timedelta(days=n)
407         since = datetime.datetime.combine(target, datetime.time.min
408         )
409         until = datetime.datetime.combine(target, datetime.time.max
410         )
411
412     # Handle "last x days", "past x weeks", etc.
413     tm = re.search(r"(last|past)\s*(\d+)?\s*(day|days|week|weeks|
414     month|months|year|years)", low)
415     if tm:
416         num = int(tm.group(2)) if tm.group(2) else 1
417         unit = tm.group(3)
418         if 'day' in unit:
419             since = now - datetime.timedelta(days=num)
420         elif 'week' in unit:
421             since = now - datetime.timedelta(weeks=num)
422         elif 'month' in unit:
423             since = now - datetime.timedelta(days=30*num)
424         elif 'year' in unit:
425             since = now - datetime.timedelta(days=365*num)
426         until = now
427
428     def build_where(since=since, until=until, obj=None, color=None)
429     :
430         clauses, params = [], []
431         if obj:
432             clauses.append("object = ?")
433             params.append(obj)
434         if color:
435             clauses.append("color = ?")
436             params.append(color)
437         if since:
438             clauses.append("detected_at >= ?")

```

```

431         params.append(since.strftime("%Y-%m-%d %H:%M:%S"))
432     if until:
433         clauses.append("detected_at <= ?")
434         params.append(until.strftime("%Y-%m-%d %H:%M:%S"))
435     where = "WHERE " + " AND ".join(clauses) if clauses else ""
436     return where, params
437
438 #Searching the numbers of objects/ colors, how many of each (
439     counter)
440 if re.search(r"what objects", low):
441     where, params = build_where()
442     cur = det_conn.cursor()
443     cur.execute(f"SELECT DISTINCT object FROM detections {where
444         }", params)
445     rows = sorted(r[0] for r in cur.fetchall() if r[0])
446     date_str = format_date_str(since, until)
447     reply = f"On {date_str} I saw: {'', '.join(rows)}." if rows
448         else f"No objects recorded on {date_str}."
449     self._append_chat("Assistant", reply)
450     self.chat_entry.delete(0, 'end')
451     return
452
453 if re.search(r"what colors", low):
454     where, params = build_where()
455     cur = det_conn.cursor()
456     cur.execute(f"SELECT DISTINCT color FROM detections {where}
457         ", params)
458     rows = sorted(r[0] for r in cur.fetchall() if r[0])
459     date_str = format_date_str(since, until)
460     reply = f"On {date_str} I saw colors: {'', '.join(rows)}."
461         if rows else f"No colors recorded on {date_str}."
462     self._append_chat("Assistant", reply)
463     self.chat_entry.delete(0, 'end')
464     return
465
466 if re.search(r"\bhow many of each\b", low):
467     where, params = build_where()
468     cur = det_conn.cursor()
469     cur.execute(f"SELECT color, COUNT(*) FROM detections {where
470         } GROUP BY color", params)
471     rows = cur.fetchall()
472     date_str = format_date_str(since, until)
473     if rows:
474         parts = [f"{cnt} {col}" for col, cnt in rows if col]
475         reply = f"On {date_str} I saw colors: " + ", ".join(
476             parts) + "."
477     else:
478         reply = f"No colors recorded on {date_str}."

```

```

472         self._append_chat("Assistant", reply)
473         self.chat_entry.delete(0, 'end')
474         return
475
476     if re.search(r"(number of each|count of each).*(color|colors)",
477                 low):
478         where, params = build_where()
479         cur = det_conn.cursor()
480         cur.execute(f"SELECT color, COUNT(*) FROM detections {where}
481                     } GROUP BY color", params)
482         rows = cur.fetchall()
483         date_str = format_date_str(since, until)
484         if rows:
485             parts = [f"{cnt} {col}" for col, cnt in rows if col]
486             reply = f"On {date_str} I saw colors: " + ", ".join(
487                 parts) + "."
488         else:
489             reply = f"No colors recorded on {date_str}."
490         self._append_chat("Assistant", reply)
491         self.chat_entry.delete(0, 'end')
492         return
493
494     if re.search(r"(number of each|count of each).*(object|things?)",
495                 low):
496         where, params = build_where()
497         cur = det_conn.cursor()
498         cur.execute(f"SELECT object, COUNT(*) FROM detections {
499                     where} GROUP BY object", params)
500         rows = cur.fetchall()
501         date_str = format_date_str(since, until)
502         if rows:
503             parts = [f"{cnt} {obj}" for obj, cnt in rows if obj]
504             reply = f"On {date_str} I saw objects: " + ", ".join(
505                 parts) + "."
506         else:
507             reply = f"No objects recorded on {date_str}."
508         self._append_chat("Assistant", reply)
509         self.chat_entry.delete(0, 'end')
510         return
511
512     obj_m = re.search(r"how many (\w+)s?\b", low)
513     col_m = re.search(r"color\s+(\w+)", low)
514     if obj_m:
515         obj = obj_m.group(1)
516         cnt = self.query_detections(object=obj, color=None, since=
517                                     since, until=until)
518         since_str = f" since {since.date()}" if since else ""
519         reply = f"I saw {cnt} {obj}s{since_str}."

```

```

513         self._append_chat("Assistant", reply)
514         self.chat_entry.delete(0, 'end')
515         return
516     if col_m:
517         col = col_m.group(1)
518         cnt = self.query_detections(object=None, color=col, since=
            since, until=until)
519         since_str = f" since {since.date()}" if since else ""
520         reply = f"I saw {cnt} {col} things{since_str}."
521         self._append_chat("Assistant", reply)
522         self.chat_entry.delete(0, 'end')
523         return
524
525     reply = send_chat_to_gpt(text)
526     try:
527         first, rest = reply.split('\n', 1)
528         directive = json.loads(first)
529     except (ValueError, json.JSONDecodeError):
530         directive = {'detect': False}
531         rest = reply
532     self._append_chat("Assistant", rest)
533     if directive.get('detect') and self.current_image:
534         obj, col = directive.get('object', ''), directive.get('color
            ', '')
535         self.obj_entry.delete(0, 'end')
536         self.obj_entry.insert(0, obj)
537         self.col_entry.delete(0, 'end')
538         self.col_entry.insert(0, col)
539         self._append_chat("System", f"LLM-triggered detect object={
            obj}, color={col}")
540         self.detect(auto=False)
541     self.chat_entry.delete(0, 'end')
542
543     def _append_chat(self, role, text):
544         self.chat_display.config(state='normal')
545         self.chat_display.insert('end', f"{role}: {text}\n", role)
546         self.chat_display.config(state='disabled')
547         self.chat_display.see('end')
548
549     if __name__ == '__main__':
550         VisionApp()

```

Listing 6.1: Python Code