Forget-Me-Not

Giancarlo Zaniolo, Ethan Muchkin, Swati Anshu

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— Forget-Me-Not is a system capable of tracking and locating commonly misplaced objects such as keys, wallets, and remote controls in indoor environments. Using a camera system coupled with a custom-trained YOLO object detection model, this system identifies and stores the last known location of these items in a database and allows the user to query them. Compared to existing Bluetooth trackers, this solution provides a more scalable, flexible, and hands-off alternative, particularly beneficial for people with dementia or cognitive impairments.**

***Index Terms*— Camera-based tracking, computer vision, object detection, object tracking, Raspberry Pi, real-time location, YOLO, machine learning, indoor environments.**

I. Introduction

The "Forget-Me-Not" system addresses the common problem of misplacing everyday household items such as keys, wallets, and remote controls in indoor environments. The primary use case is for families and students who often lose track of these items due to busy schedules or shared living spaces. Searching for lost objects can be frustrating and time-consuming, particularly when people are in a rush or trying to manage multiple tasks. Forget-Me-Not offers a practical solution by using camera-based object tracking to help users quickly locate these frequently misplaced items.

Existing solutions have several limitations. For example, Bluetooth trackers are often inconvenient because they require physical attachment to each item, and have a difficult time providing exact location data, all while being too expensive to use for all but a few items. Similar complaints can be said about GPS trackers or AirTags, which cost a starting $30 per tag. In contrast, Forget-Me-Not uses machine learning and computer vision to monitor entire rooms and detect as many objects as it can recognize without requiring tags or extra setup. The system also aims to provide two intuitive real-time interfaces (speech and web) to provide a pleasant and simple experience for the end user. Ultimately, the goal is to create a scalable, easy-to-use system that enhances everyday convenience for families and students.

II. Use-Case Requirements

We have determined that Forget-Me-Not must meet several of the following critical requirements to effectively meet the needs of our envisioned customers.

Our first requirement concerns the capabilities that our system should have. Firstly, we would like our system to be able to take photographs of the user's room. This would be necessary as we would believe giving the user the opportunity to see a photo of their room in a state where their object was last seen would be conducive to helping find an object.

Secondly, our system should provide the user the capability to query the system, both through a "voice assistant" frontend and a visual frontend. We believe that both are necessary, as the voice assistant would provide the user a means to query the system if they have lost the device they are looking for, and the visual frontend would assist in the case that the model is not completely accurate, or the object to be found was the second-to-last seen version of said object. We believe both frontends should themselves have their own set of constraints.

For the audio frontend, we think it would be reasonable for it to take 30 seconds between speaking a query and receiving a result. This time was chosen because in most cases, finding an object on your own should take more than 30 seconds. This statement is corroborated with calculations based on data found on the "Lostings Lost and Found Statistics"[1] webpage. This page claims that the average person spends 2.5 days per year searching for lost objects, and that the average person can lose up to 9 items per day. With some calculations, we arrive at the very conservative estimate that

$$2.5\,\frac{days\ searching}{year} * \frac{1}{365\ days} * \frac{1}{24\ hours} * \frac{1}{60 min} * \frac{1}{9} objects \frac{lost}{day}$$

$$= 1.096\ min/object$$

1.096 min/object spent searching on a given day. If we could cut this number in half, it could save the user lots of time in the long run.

For the web frontend, we would like to ensure that the user receives the answer for a query in 10 seconds. This is because according to Uptrends, user attention suffers if a webpage is stuck loading for more than 10 seconds[2].

For both frontends, we would like to ensure that our system returns queries based on data that is at most 30 seconds old. According to the National Library of Medicine, short-term memory lasts for 30 seconds[3].

Regarding our model, we have determined that the system needs to achieve at least 80% accuracy in detecting and identifying predefined objects within a 10x10 ft room under well-lit conditions (≥3000 lumens per square foot). We have chosen these numbers because we believe it is reasonable for our tool to miss 1/5 voice queries and still leave users satisfied, and because we believe the area and lighting constraints provide a reasonable environment for which it would be nontrivial for users to remember which objects were always present.

For our objects, we are choosing to initially support finding phones, wallets, and keys, as they are the most commonly lost objects according to the "Lostings Lost and Found Statistics" webpage[1]. More objects may be added in the future.

Additionally, we have chosen cost constraints which dictate that the hardware setup should not exceed $300, while cloud services for continuous usage should be kept under $40 per user per month. We based these numbers off security.org's SimpliSafe, an existing product which charges between $250 and $730 for the hardware, and $32/month as a monthly subscription. As this is a successful product, we have reason to roughly believe people will buy our product for similar prices[4].

Lastly, privacy is a priority, meaning that access to the system must be restricted to authorized users on authenticated local devices. These requirements are designed to create a system that is both accurate and cost-effective, while maintaining a focus on user privacy and efficiency.

III. Architecture and/or Principle of Operation

To describe the architecture of the system, we begin with an overview of the system's physical layout and operational flow. The system is comprised of three primary components: the Raspberry Pi (with a camera), the, user-facing interface, and our web server.

A. Camera and Raspberry Pi Setup:

The system is centered around one or more Raspberry Pi units, each equipped with a high-resolution camera. These cameras are strategically placed within a room to maximize floor visibility. The Raspberry Pi serves to capture images periodically of its environment and transmit them over HTTP to our server for further processing. Notably, the raspberry pi can be used to run more basic computations. In our overall workload, we expect there to be a large portion of the time where new pictures will never provide new information, such as when the room owner is not present, or at nighttime when not much is happening. During these instances, it would not be useful to be running object detection on, and saving every picture outputted by the camera. As such, we plan on adding certain optimizations to our camera setup to prune such images from those sent to our web server.

B. Query Interfaces:

For our overall system, we plan on supporting two interfaces through which users can query information, a visual website-based interface, and an audio "voice assistant" interface.

In the visual website-based user interface, users will be able to query the last known location of a specific object through a text field. This interface sends requests to the web server via a REST API. After the necessary processing has been done, the web server should return the last time the requested object was seen (updated within the last 30 seconds), within 20 seconds of querying, ensuring real-time usability for users. If the provided image is not to the user's desires, they should have the option to retrieve the previous occurrence of the object in the database. This will be accomplished through another request to the web server's REST API. Also, the HTML for our website will be hosted on the web server too.

The "voice assistant" interface will be structured similarly to the visual website-based interface, only that instead of sending text queries to the server, it should send "spoken text". The raspberry pi will have a microphone that is always recording, and using a ML model to turn any speech it hears into text. Once it deciphers the keyphrase, "Forget-Me-Not, where is my, '__'", it should send the spoken string to the web server, also through a REST API. It should eventually receive a response, which it can synthesize into spoken speech, and play through its speakers.

However, if the users would like all their information to remain local, we are creating an alternative system that allows for all the processing to happen on the edge device and a web server. The system architecture for this version is as follows:

C. Web Server:

For our web server, we plan on supporting two backends, one cloud, and the other running on the user's personal machine. Both of our backends will share some of the same functionality.

Upon receiving an image through its REST API, our web server will run YOLOv11 on the image, and upon detecting one or multiple objects, ensure its last known position is updated and stored in its database, and the image is stored in a filesystem. Additionally, upon receiving text or voice query, the backend will parse the query, retrieve the requested information from its database, and send it back to the requester. In general, the compute also maintains object location histories and handles any necessary computations beyond the capability of the Raspberry Pi, ensuring that processing remains lightweight on the edge device.

The primary differences between our backends lie in their choice of a computing platform and database. Our cloud compute will be hosted on AWS EC2, and store data in Amazon RDS and image storage in S3 buckets, which comes with the challenge of authenticating multiple users, and storing all their data securely. On the other hand, our local platform will simply run the webserver locally, and store data in an SQLite database on the 64GB SD card. To manage storage limitations, old images and location data will periodically be cleared and compressed.

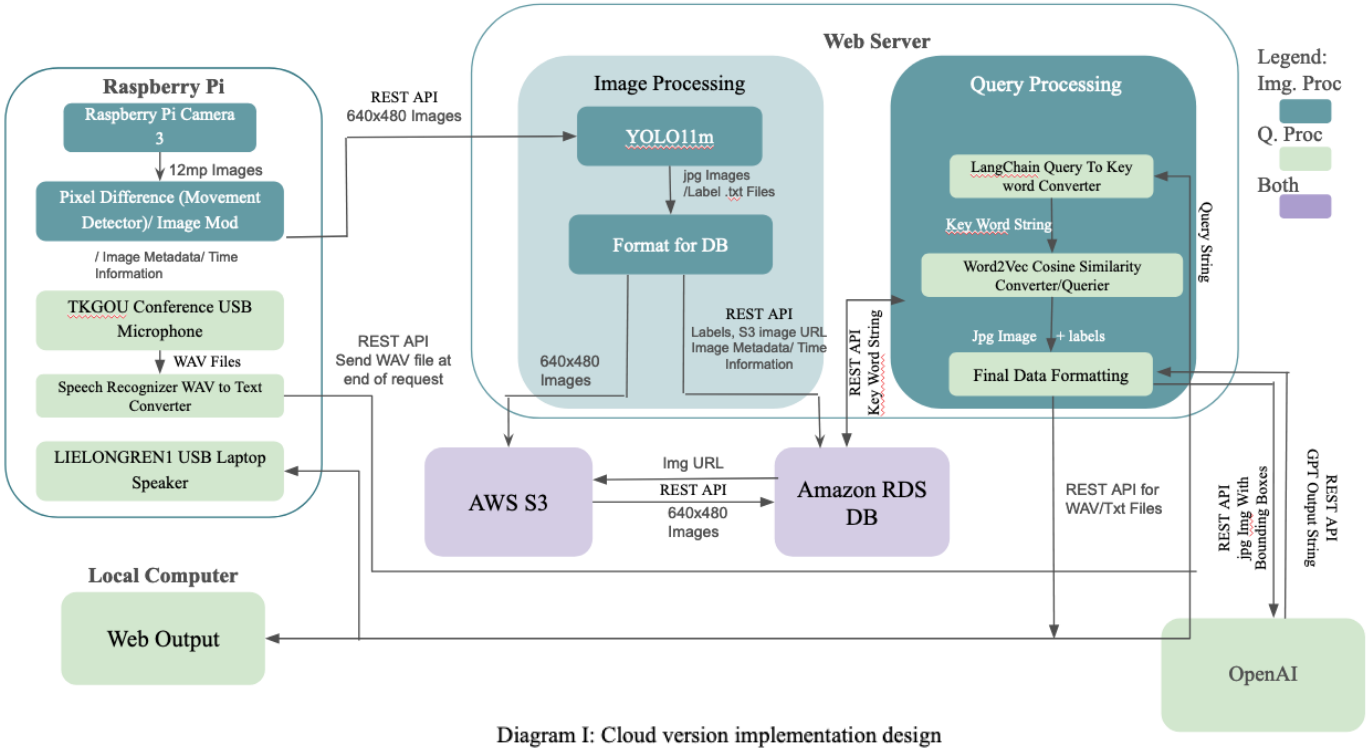

Diagram I: Cloud version implementation design

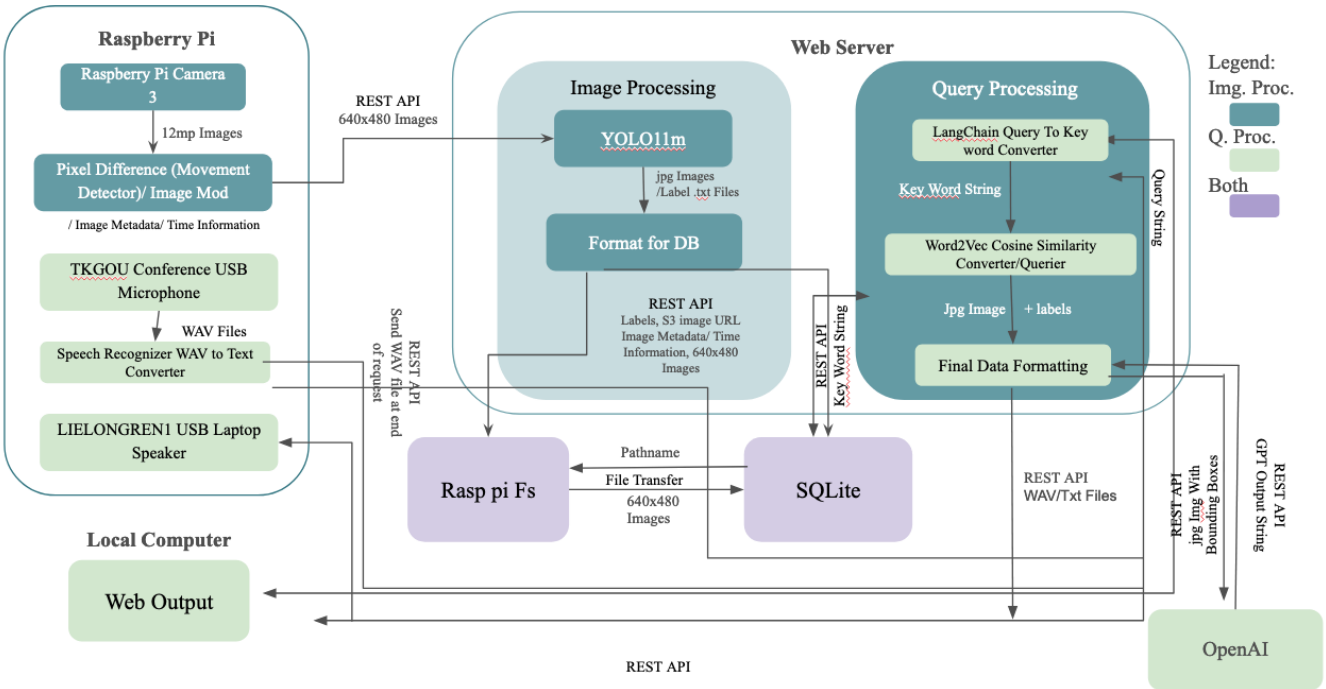

Diagram II: Local Version implementation design

IV. Design Requirements

The system's design requirements are primarily driven by the use case of tracking and identifying misplaced objects in indoor environments, and they can be categorized into hardware, latency, price, and accuracy constraints.

A. Hardware Constraints

The system requires cameras with a resolution of 1080p to ensure that the objects are captured in sufficient detail for accurate detection. This resolution strikes a balance between providing clear images and managing data size for processing. The cameras need to capture images every 5 seconds to maintain up-to-date information on object locations without overburdening the system's processing capabilities. Notably, it took one of our group members 5 seconds to slowly walk across their room, meaning our system should reasonably capture any instance where an object is briefly placed, before being picked up and transported again. Additionally, each camera must have a field of view (FOV) of 40 degrees to ensure adequate coverage of the room. While this FOV might seem narrow, it helps the system focus on specific areas and reduces image distortion, optimizing object detection.

B. Latency Constraints

To ensure real-time operation, the system must meet specific latency targets. For the "Monitor" workflow, which involves capturing and processing images, preprocessing the image data must occur within 1 second, while object detection through machine learning models should take 5 seconds. The webserver "ping" latency should be within 1 second, and the entire process, including database writing and cross-component latencies, should be complete within 13 seconds, ensuring that the system remains responsive.

We chose these values because they seemed like reasonable targets for each of our subsystems. Our YOLO latency was found through personal testing, our preprocessing latency makes sense because a 1080p image has 2 million pixels, and the Raspberry pi has a >200MHz clock, meaning we can run 100 cycles on each pixel and have it done in 1 second. We do not have a great metric for our database latency, as it depends highly on our database choice workload, and database structure, so 5 seconds seemed like a safe settlement. Additionally, by pinging google.com, we can assume that a relatively "normal" ping is somewhere around 20-100ms, which should fit within our 1 second margin unless there are issues with the system.

For the "Query" workflow, which is responsible for providing users with information about the location of objects, speech-to-text processing should take no more than 2 seconds, the, followed by the webserver ping latency, which should take less than 1 second. Next, we list that the database read latency should require less than 7 second, leaving 10 seconds for an additional processing we would like to do using LLMs. The total latency for a query response should not exceed 20 seconds, allowing users to quickly find misplaced items.

C. Price Constraints

Affordability is critical for the target audience, which includes families and students. Therefore, the hardware cost per room is capped at $300, ensuring that the system remains cost-effective. Additionally, the cloud computing and storage costs are set at $40 per user per month. The rationale for these values was previously mentioned in our use-case requirements, with both being based off the price of an existing home security product. With this price point, we hope to balance between maintaining high performance and providing an affordable solution for continuous usage.

D. Privacy

The cloud version of the system will utilize AWS security protocols to safeguard both data at rest and in transit. AWS offers a highly secure infrastructure, which includes encryption services, access management, and monitoring to prevent unauthorized access or data breaches. In addition to the securities provided by AWS, we plan on encrypting all images stored in the S3 buckets filesystem, and storing its keys inside the database, which has encryption gurarantees.

The local version should never make images available to anyone outside of the local network, meaning all data should be secure.

These design requirements ensure that the system operates efficiently, remains affordable, and meets user expectations for responsiveness and reliability in an indoor environment.

V. Design Trade Studies

Design trade-offs play a crucial role in optimizing the performance of the "Forget-Me-Not" system while balancing key factors such as cost, accuracy, latency, and scalability. Each subsystem has been carefully evaluated to identify how different design choices impact the overall system's ability to meet the use-case requirements.

A. Machine Learning Model

Design Specification: Object detection accuracy must meet or exceed 80% in a well-lit 10x10 ft room.

Trade-Off Between Accuracy and Latency: The key trade-off for the YOLOv11 model is between detection accuracy and the latency associated with running the model on edge devices like the Raspberry Pi. YOLO provides high detection accuracy, but as the model complexity increases (e.g., deeper layers for better object recognition), the latency increases due to higher processing time. Conversely, simplifying the model to meet latency requirements may reduce detection accuracy, which is unacceptable for meeting user expectations.

Equation: Latency (L) is proportional to model complexity (C) and inversely proportional to processing power (P):

$$L \propto \frac{C}{P}$$

To balance this trade-off, we will be optimizing the model's size while keeping it within the computational capabilities of the Raspberry Pi, reducing unnecessary complexity to meet both accuracy and speed requirements.

Trade-Off Impacts: Reducing model complexity may result in missed detections, impacting the 80% accuracy target. However, maintaining high complexity could increase detection latency beyond the acceptable threshold of 5 seconds per detection cycle. Therefore, tuning the model architecture (number of layers, input resolution) is key to balancing accuracy and latency for real-time performance.

B. Amazon RDS Database

Design Specification: The database must write new object locations in under 5 seconds and read them in under 7 seconds.

Trade-Off Between Performance and Cost: Amazon RDS offers various instance types that impact both database performance and operational cost. Larger, more powerful instances can process read/write operations quickly, ensuring low latency, but they come at a significantly higher cost. On the other hand, using smaller instances reduces operational costs but may introduce delays in data processing, violating the latency constraints required for real-time object tracking.

Equation: The cost (C) of the database is proportional to the instance size (S), while latency (L) is inversely proportional to the instance size:

$$L \propto \frac{1}{S}, \quad C \propto S$$

This equation highlights the trade-off: to minimize latency, a larger database instance is required, but this increases operational costs.

Trade-Off Impacts: For a low-cost, scalable system, a balance between performance and cost is critical. Choosing an RDS instance size that keeps latency within 5-7 seconds while staying within budget constraints will directly impact user satisfaction and system scalability.

C. Web Server Configurations

Design Specification: The web server must handle user queries and return item locations in under 20 seconds.

Trade-Off Between Scalability and Response Time: The choice of web server components (Nginx, Gunicorn, Flask) involves trade-offs between the server's ability to handle multiple requests simultaneously (scalability) and the time it takes to process each request (response time). Nginx serves as a load balancer to optimize request handling, while Gunicorn acts as the server interface for Flask, which processes API requests. Increasing the number of Gunicorn worker threads improves the server's ability to handle concurrent requests but increases memory usage and potentially adds overhead, increasing response time.

Equation: Response time (T) is inversely related to the number of workers (W) but directly related to memory usage (M):

$$T \propto \frac{1}{W}, \quad M \propto W$$

This trade-off requires optimizing the number of workers to balance memory usage with fast request handling.

Trade-Off Impacts: Overloading the web server with too many worker threads may lead to slower responses due to memory constraints, while too few workers might delay query responses. The system must be tuned to handle multiple users without exceeding the 20-second response time requirement.

VI. System Implementation

A. Machine Learning Model

The first critical subsystem in our system is YOLOv11, which handles object detection. It will be trained using a specialized dataset tailored to help it detect objects in indoor environments, focusing on frequently misplaced items such as keys, wallets, and remotes. Once trained, the model operates by processing incoming image frames captured by the camera and producing bounding boxes around detected objects with labels. The output from YOLOv11 is crucial for object location tracking and is passed to the next subsystem for database storage or user queries.

The primary limitation for this is our ability to create a good dataset with which to train the model on. Dataset engineering requires a significant time investment, particularly if you plan on creating your own training data. Much of our time will be spent trying to find an effective way to train our model within the time bounds allotted by 18-500. However, if we manage to find a successful method, it should be relatively easy for us to add detection classes to the model, and improve its functionality.

Additionally, lighting conditions and object size could also impact detection accuracy, requiring further fine-tuning during training.

B. Amazon RDS/SQLite Database

The database (Amazon RDS for cloud, and SQLite for local) handles data storage for the detected object locations and their associated metadata. This subsystem must meet strict latency constraints, with database writes and reads occurring within 5 and 7 seconds respectively.

Amazon RDS offers a managed relational database, providing scalability, availability, and security, which are critical for a system like ours that handles multiple data transactions per room. Each time YOLOv11 detects an object, the location data, and a link to its corresponding (encrypted) image file is written to the RDS database for subsequent

retrieval during query workflows. Additionally, it integrates seamlessly with AWS security features, ensuring that sensitive data remains protected through encryption and authenticated access.

SQLite is a more lightweight option, but should still provide us all of the necessary guarantees to ensure our system has the necessary functionality while safely executing concurrent transactions.

Our database choices will allow us to maintain an accurate and up-to-date record of item locations, enabling the system to meet the user requirement of delivering the item location within 20 seconds of querying.

C. Web Server

The web server subsystem is responsible for handling the front-end and back-end interactions of the system, connecting the user interface to the machine learning models and database. This subsystem uses a combination of Nginx, Gunicorn, and Flask. Nginx serves as the reverse proxy and static file server, ensuring efficient handling of HTTP requests. Gunicorn is the WSGI (Web Server Gateway Interface) server that bridges Flask (the web framework) with Nginx, facilitating the execution of Python code in response to user queries. Flask determines the REST API calls that trigger the object detection and query workflows. The web server also manages latency between subsystems, ensuring that all interactions meet the required 20-second total query time. By choosing this robust combination, the system can efficiently scale and manage multiple requests in parallel, contributing to its overall performance and user experience.

Depending on whether it is executing the image processing or query processing pipelines, the web server will be running different code paths.

In the case of image processing pipeline, the server was invoked with an image from the Raspberry Pi, and must run YOLOv11 on the image, and store its detected objects in the database, and new image in the file system, after encryption.

In the case of the query processing pipeline, the web server receives text input from the Raspberry pi, uses LangChain to extract the keyword from the query, uses Word2Vec to find out the most similar detected object category to the word, acquires the relevant information from the database (the most recent entry for a requested object), and does some final formatting before returning the result to the user. Please refer to Diagram 3 in the Appendix.

D. Camera Subsystem

The camera subsystem is a fundamental component in the system, responsible for providing the visual data required for object detection. The system uses a 1080p resolution camera, which is connected to the Raspberry Pi through the Camera Serial Interface (CSI) port, ensuring high-speed, low-latency data transmission. The camera is strategically placed to capture approximately 80% of the room's floor area, minimizing blind spots and ensuring that objects like keys, wallets, and remotes are within its field of view. The camera operates by taking an image every 5 seconds, aligning with the system's requirement for real-time monitoring.

Clear, detailed images allow the YOLOv11 model to reliably identify smaller objects and differentiate them from background clutter, even under variable lighting conditions. The images captured are preprocessed on the Raspberry Pi,

However, there are trade-offs to consider. The higher resolution increases the computational load on the Raspberry Pi, potentially leading to processing delays, particularly when handling larger or more complex scenes. The camera's 1080p output, while beneficial for detailed object detection, requires balancing against the Pi's limited processing power.

Additionally, suboptimal lighting conditions, such as low light or harsh shadows, could impact the quality of the captured images, necessitating further adjustments in camera settings or the use of external lighting to maintain accuracy. Despite these challenges, the camera subsystem is designed to deliver consistent, high-quality visual data that is integral to the overall performance of the system.

E. Raspberry Pi processing Subsystem

The Raspberry Pi Processing Subsystem serves as the eyes and ears of the system. The Raspberry Pi 4, with its quad-core processor and 4GB of RAM, acts as the hub for image preprocessing, microphone requests/audio responses, and communication with the web server.

When its camera captures an image, the Pi first performs a basic prepressing step where it takes the pixel difference from the last frame, to determine whether the captured image has changed sufficiently for it to be worthwhile to run the object detection model and store its results in the database.

Similarly, when the microphone captures audio input, it is converted to text using a speech recognition model, and eventually sent to the web server as well.

F. Amazon S3 Cloud Storage/RPi Filesystem

The primary purpose of the filesystem is to store files which would be cumbersomely large to place in a database. We plan on encrypting all image files before storing them, on both our cloud and local backends, to ensure no malicious actor can access house photos without authentication.

The local implementation of the filesystem should suffice for the uses of a single person, but for the cloud server, which will be handling the queries of multiple users, it will be advantageous to have some of the advantages provided by Amazon's ecosystem.

VII. Test, Verification and Validation

A. Object Detection Accuracy

To verify that YOLOv11 meets the design specification of achieving at least 80% accuracy in object detection, we plan on creating our own small suite of images, which we can use to test our system. We plan on conducting 50 detection trials per object we would like to detect in a simulated environment resembling typical household conditions (a well-lit 10x10 ft room).

$$Accuracy = \frac{Number\ of\ Correct\ Detections}{Total\ Number\ of\ Objects}$$

We will repeat this test under different lighting conditions and for different object types to ensure that the model generalizes well to various environments.

B. Webserver System Latency

We can easily measure the latency of each of the components in our webserver by instrumenting our code with timers around each of our components. Measuring latency will likely not directly depend on our workload, so we can make mocked-out components which spit out random data to execute test traces.

Testing will involve a comparison between the actual capture interval and the design requirement to ensure the web server components operate within the specified bounds. This testing will also indirectly test the correctness of our implementation.

C. Query Response Time

To validate the query response time of under 20 seconds, we will test the entire system's performance from the point of speech input by the user to the system output of item location. Similarly to our webserver, we will instrument our code and measure the total time taken for the system to process the user's voice query, search the database for the object's location, and return the result to the user.

This test will be repeated multiple times under different network conditions to identify any potential delays caused by database latency, object detection processing time, or speech-to-text conversion. Achieving a consistent response time of under 20 seconds ensures that the system meets both the design and use-case requirements of providing timely feedback to users.

D. Privacy and Security

To ensure that only authorized users can access the system, we will perform security tests focusing on the authentication protocols integrated with AWS's security system. A series of mock attacks, including unauthorized device attempts to access the data, will be conducted to assess the robustness of the security measures.

We will also test the system's encryption during data storage and retrieval, ensuring that sensitive information, such as object locations, is protected during communication between the Raspberry Pi, AWS servers, and the end user. A successful test will demonstrate that privacy and security measures meet the use-case requirements for authorized access.

E. System Throughput

This test how much workload our system is able to cope with, which will be particularly important for our cloud implementation, as it will determine how many clients we are able to service with one machine. For this test, we will make mock traces with randomized data, and bombard our EC2 instance with an increasing number of requests, to see how the system copes with them. In particular, we will be measuring statistics like CPU usage, memory usage, and if we manage to reach a bottleneck, latency increases for the requests.

F. Database Latency Tests

As the database is a separate component, we would also like to independently test its performance and throughput at various levels of capacity. Testing will similarly be done by instrumenting code with timers, and running random traces.

VIII. Project Management

A. Schedule

The schedule is split up according to individual responsibilities with slack time of an average of 2 days included in the timeline. The largest time allocation is for system integration, with slack time of a week built-in. It is color coded based on the type of work (blue for documentation, red for design, green for individual progress and orange for integration). Refer to table 1.

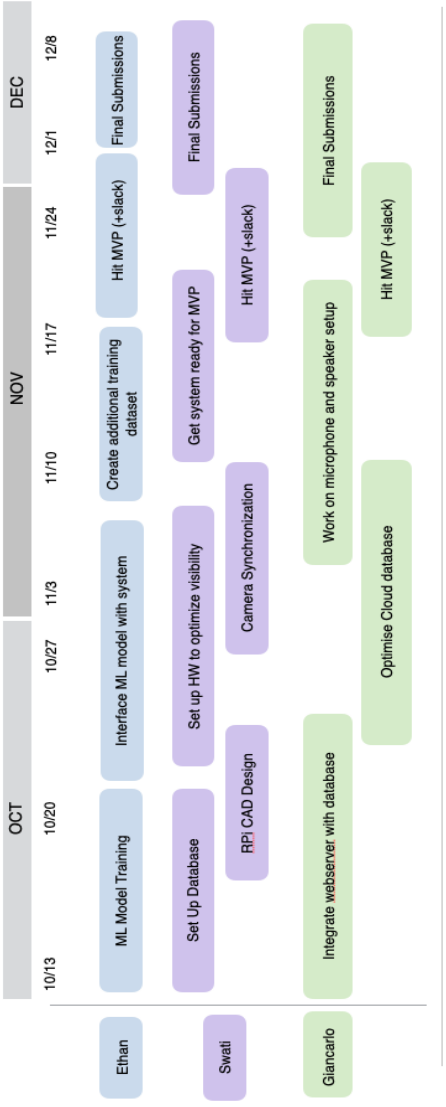

Table I: Schedule and task breakdown

B. Team Member Responsibilities

Ethan's primary responsibility in the project is the machine learning component, where his focus is on optimizing the model for accuracy, performance, and generalization. He works on improving the detection algorithms to ensure that the system can identify objects in various indoor environments with high precision. He also plays a secondary role in optimizing performance across the overall system, ensuring that both the web server and database integrate efficiently with the machine learning model.

Swati is working on setting up the database and hardware configurations, which are critical for the system's backend infrastructure. She ensures that data related to detected objects is stored correctly and accessed efficiently. In addition to database management, She will also work on preprocessing optimization, refining how raw data is prepared for object detection. Her secondary responsibility involves collaborating with Ethan and Giancarlo to troubleshoot any performance issues that arise in the database or hardware setup, ensuring smooth communication across all system subsystems.

Giancarlo is primarily responsible for the web server structure, setup, and instrumentation. His role ensures that the server can handle incoming data and process requests efficiently, providing users with quick access to object detection results. He also works closely with Ethan to integrate the machine learning model into the web server and with Swati to ensure smooth data flow from the database to the user interface. This coordination allows the system to operate seamlessly, balancing the needs of each subsystem.

C. Bill of Materials and Budget

Table 2 shows the detailed breakdown of parts that we require for our system and the estimated total cost for its development.

D. Risk Mitigation Plans

Critical risks for this project include the fact that none of us have worked with databases before so figuring out the correct configuration and the integration, post-setup will be a challenge. Secondly, the amount of data needed to train the ML model is substantially large and there are limited online datasets for the indoor images we need – hence we will need to create our own bounding boxes for hundreds of images to train the model further. Lastly, another critical risk is the final integration of the entire system, as none of us have worked on creating interfaces for multiple different systems and different datatypes.

Hence, our primary risk is training the ML model for our system to meet the design specification. These risks will be mitigated by allotting time for just for the creation of the training dataset manually and using annotation tools to speed up the process . Other than that, approaching the integration risks modularly and debugging each subsystem and subsystem interface will help us minimize complications. Lastly, figuring out how to use services that we have not used before is a matter of learning on the spot, understanding the demands of our system and implementing the necessary steps efficiently. To mitigate the database risk, we plan to use extensive documentation and tutorials provided by Amazon Web Services (AWS) for setting up and configuring the RDS database. Furthermore, dividing the team responsibilities, with each member focusing on mastering a particular service or technology, will ensure a smoother integration process.

IX. Related Work

Several existing projects and products are similar to the system we are proposing. A notable example is Tile[5], a small Bluetooth-based tracking device that helps users find misplaced objects like keys, wallets, and phones. Tile's strength lies in its simplicity and user-friendly mobile app interface, which allows users to track items using a connected smartphone. However, Tile requires physical attachment of the tracking device to each item, which our project aims to avoid by using computer vision to detect items in an environment without the need for individual trackers.

Another similar product is the Apple AirTag[6], which uses Ultra-Wideband (UWB) technology along with the "Find My" app to locate lost items. Like Tile, it also requires physical attachment to objects. Though it leverages precise location detection, it does not provide a solution for automatically tracking objects across a room in the way that our system does.

On the more technical side, systems such as Amazon Go "Just Walk Out" technology[7] stores utilize computer vision and machine learning for object detection and tracking. The Amazon Go system is much more advanced and tracks multiple users and objects in real time, relying heavily on cloud infrastructure. While impressive, the scale and complexity of Amazon Go surpasses what our project is targeting, which is smaller indoor environments.

Cortexica Vision Systems[8] also offers vision-based object recognition for retail and inventory management, similar in principle to our approach but with a focus on industrial use cases.

Our system differs from these existing solutions by focusing on a low-cost, camera-based, non-invasive solution for individual users, such as families or students, to locate commonly misplaced items without attaching tracking devices. Furthermore, the use of machine learning models for indoor object detection ensures that the solution remains scalable and adaptable to various environments.

X. Summary

Our design focuses on an intelligent object tracking system that uses computer vision and machine learning to help users locate misplaced items like keys and wallets in indoor environments. The system is built on a Raspberry Pi, running a trained YOLOv11 object detection model, and integrates with an Amazon RDS database and a web server. It is designed for low latency, ease of use, and affordability, targeting busy families and students.

Key challenges include improving object detection accuracy in various lighting conditions, optimizing the model for the

Raspberry Pi's limited hardware, and building a custom dataset for training. Additionally, ensuring smooth integration between the machine learning model, database, and web server will be essential to meet performance requirements.

Glossary of Acronyms

RPi – Raspberry Pi
AWS – Amazon Web Services
FOV – Field of View
GPU – Graphics Processing Unit
ML – Machine Learning
RDS – Relational Database Service
YOLO – You Only Look Once
VM – Virtual Machine

References

[1] "Lost and Found Statistics, Trends & Facts 2023." Lostings, 2023, www.lostings.com/lost-and-found-statistics/. newauthors.ieeeauthorcenter.ieee.org/author-tools/

[2] "The Psychology of Web Performance | the Uptrends Blog." Blog.uptrends.com, 13 June 2018, blog.uptrends.com/web-performance/the-psychology-of-web-performance/.

[3] Cascella, Marco, and Yasir Al Khalili. "Short Term Memory Impairment." PubMed, StatPearls Publishing, 2020, www.ncbi.nlm.nih.gov/books/NBK545136/.

[4] Vigderman, Aliza, and Gabe Turner. "2024 SimpliSafe Home Security Package Costs & Monitoring Plans." Security.org, 16 Sept. 2024, www.security.org/home-security-systems/simplisafe/. Accessed 12 Oct. 2024.

[5] "Tile Tracker | Bluetooth Trackers for Keys, Wallets, Phones, and More." Tile ECommerce, 2024, www.tile.com/en-us?srsltid=AfmBOoqiPlO2RfW68YqSzWMciQHgULCpPVqbTAXsK1V-UDVidQUBQluf. Accessed 12 Oct. 2024.

[6] Basappa, Prashanth. "The Technology behind Apple's AirTag." Nerd for Tech, 20 July 2021, medium.com/nerd-for-tech/the-technology-behind-apples-airtag-c7983f9322b5.

[7] Gross, Ryan. "How the Amazon Go Store's AI Works." Towards Data Science, Towards Data Science, 7 June 2019, towardsdatascience.com/how-the-amazon-go-store-works-a-deep-dive-3fde9d9939e9.

[8] "Cortexica Vision Systems." Pitchbook.com, 2024, pitchbook.com/profiles/company/60147-64#overview. Accessed 12 Oct. 2024.

Table 2: Bill of Materials

Description	Manufacturer	Quantity	Cost / item	Price Paid (in class)	Total	Total Paid (in class)
Raspberry Pi V4 8GB	Raspberry Pi	2	$75.00	$0.00	$150.00	$0.00
Raspberry Pi V5 8GB	Raspberry Pi	1	$80.00	$0.00	$80.00	$0.00
NVIDIA Jetson Nano 4GB Developer Kit	NVIDIA	1	$300.00	$0.00	$300.00	$0.00
Raspberry Pi Camera Module 3 Wide	Raspberry Pi	3	$35.00	$0.00	$105.00	$0.00
3D Printer Filament	PLA Printer Filament 1kg	1	$20.00	$0.00 (previously owned)	$20.00	$0.00
TKGOU Conference USB Microphone	TKGOU	1	$20.00	$20.00	$20.00	$20.00
USB Laptop Speaker	LIELONGREN1	1	$16.00	$16.00	$16.00	$16.00
Lomg USBC Power Cable	TONIWA	3	$14.00	$42.00	$42.00	$42.00

Grand Total: $78.00

Diagram III: Web server architecture