Project: Capstone SP23
Member Names: Rahul Singhal, Anand Todkar
NetID: rahuls11, atodkar2
Late days used: 0
Video Link: Google Drive
Gitlab Link: UIUC Gitlab
Contribution: Both Rahul and Anand contributed equally to this project. Specifically, Rahul focused on edge hardware and compute, Anand on AWS and flutter, and both on report preparation and video.

Front Porch Buddy

A modern home camera notification system for families with neighbors.

Motivation

In today's fast-paced world, there are concerns about missing out on significant events at the front door when individuals are away from home. These concerns are particularly important when it comes to the safety and well-being of children and the elderly. Installing a video doorbell or smart security camera can provide peace of mind and help individuals stay connected to their homes while they are away. The motivation of this assignment is to make our security camera smarter.

Whether it's a mail carrier dropping off an important package, a neighbor coming over to say hello, or a family member returning home after a long day, it can be frustrating to miss out on these interactions. Moreover, as parents, we often worry about our children's safety and well-being. We want to be aware of their whereabouts, and this concern is heightened when they're returning home from school or other activities.

It's not just about being aware of who is at your front door, but also about understanding the emotional state of the person coming to the door. For instance, if you have an elderly parent or a sick family member living alone, you would want to know if they are receiving regular visits from caregivers, friends, or family members. Similarly, if you have children who may be anxious or upset when they return home, you would want to be there to provide them with the emotional support they need.

Moreover, knowing who is at your front door and their emotional state can help you prioritize your responses accordingly. For example, if your child comes home crying, you may want to immediately check in with them and provide comfort. On the other hand, if it's a salesperson at your door, you may choose to ignore the notification or politely decline their offer.

In such situations, installing a video doorbell or smart security camera can provide peace of mind. These devices allow you to monitor your front door remotely and receive real-time notifications when someone approaches it. Additionally, some advanced video doorbells have built-in features such as motion detection and facial recognition, which can filter out the noise and only alert you to important events like a crying child coming home.

By leveraging modern technology, we can now stay connected to our homes, even when we're miles away. It allows us to have more control over our lives and reduces the stress of being unaware of important events that may occur at our front door.

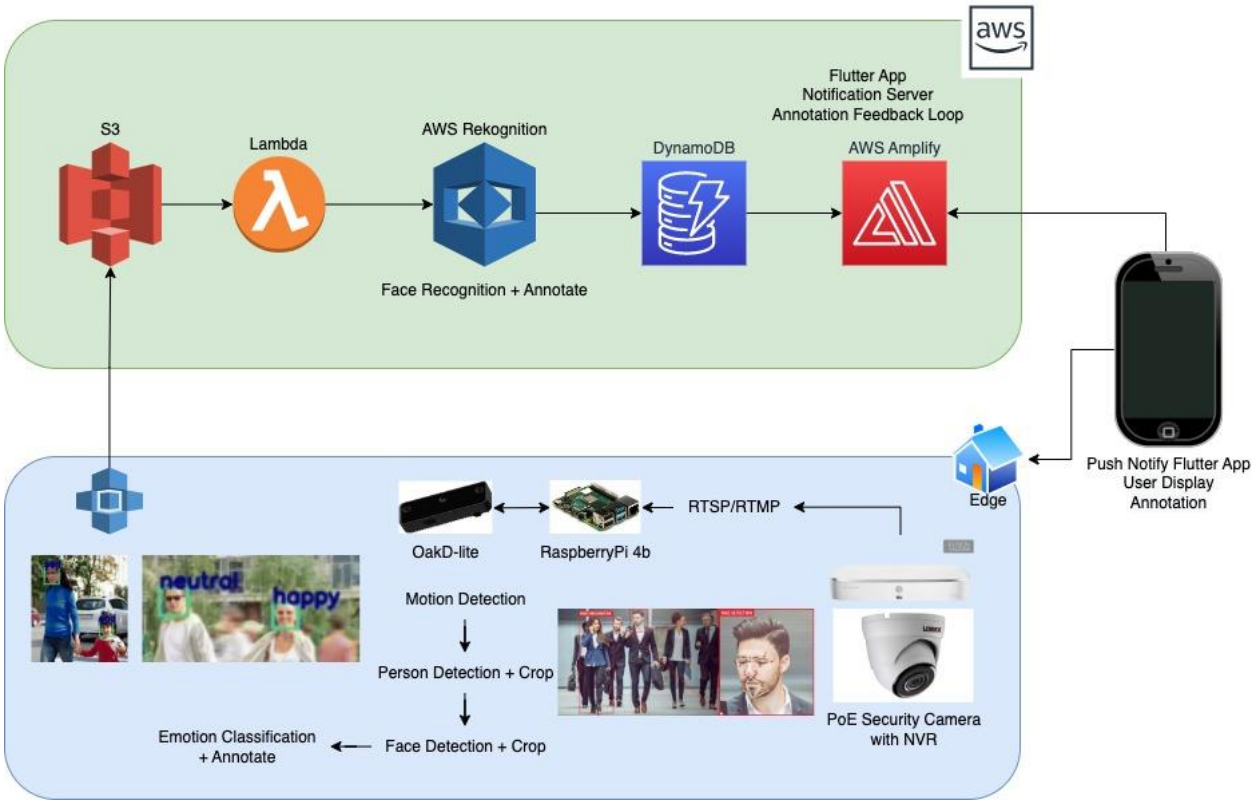
Concept

This capstone project explores the development of a system that utilizes an edge-deployed OakD-Lite camera, Raspberry Pi 4b, PoE Security Camera feed, AWS Cloud services, and a Flutter iOS app to create a smart monitoring system for the front door. The system leverages motion detection, object detection, emotion detection, and facial recognition to filter out noise and provide real-time notifications when relevant events occur. This project not only demonstrates the feasibility of such a system, but also its potential to reduce the stress of being unaware of significant events that may occur at the front door.

Proposal

Proposed Architecture

Below is proposed architecture that shows the end to end flow of a user journey. The final architecture follows this section along with an explanation of changes and reasons for each change.



The idea behind building a machine learning system using the edge deployed OakD-Lite camera, Raspberry Pi 4b, PoE Security Camera feed, Video Doorbell, AWS Cloud services, Flutter iOS app is to create a robust and comprehensive security system that can provide real-time information about the activities occurring in and around a home.

The system works by first capturing video data from the cameras, and then applying machine learning algorithms to detect any motion in the field of view. The motion detection triggers the object detection algorithm, which can identify different types of objects, such as people, animals, or vehicles. The face crop algorithm can then extract the facial features of any person detected, which can be used to recognize known individuals.

Proposed Systems

Hardware:

- PoE security camera for video capture
- OakD-Lite camera for video capture and detection models
- Raspberry Pi 4b to coordinate edge pipeline
- iOS device for user notification and data labeling

Software:

- OpenCV
- TensorFlow Lite
- Flutter

Machine Learning Models:

- Person detection
- Face segmentation
- Emotion classification
- Face instance recognition model
- (optional) Motion detection

Protocols:

- RTSP / RTMP (or reuse existing sequences from NVR)
- HTTP (required) and MQTT (optional)
- (existing installation) PoE for powering the cameras

Web Services:

- AWS Cloud for ingesting annotated data and further analysis
- AWS Rekognition or SageMaker (optional)
- Flutter app for user notification and annotation
- Greengrass and AWS IOT (optional)

Timeline

Checkpoint Date	Edge Tasks (Rahul)	Cloud Tasks (Anand)
March 19	• Setup OakD-Lite camera and Raspberry Pi 4b with PoE security camera. • Filter stream into sequences with motion detection. • Person detection model with the motion detection algorithm to detect objects only when motion is detected.	• Set up the cloud-based AWS services for ingesting and analyzing the annotated data. • Develop the user notification and labeling system using the Flutter iOS app. • Develop the machine learning models for object detection and facial recognition.

April 2	• Develop the face crop algorithm to extract the facial features of any person detected. • Prepare the system for integration with the cloud.	• Develop the user interface for the iOS app. • Develop the communication protocols between the edge and cloud, such as HTTP and MQTT.
April 16	• Test the edge computing system and make any necessary modifications. • Enhance the features of the edge computing system, if necessary.	• Test and deploy the cloud-based web services and iOS app. • Enhance the features of the cloud-based web services and iOS app, if necessary.
April 30	• Write up the report summarizing the project's goals, technical components, and results. • Develop a video summary of the project that highlights the system's features and capabilities. • Present the final deliverables to stakeholders, including a demo of the system in action.	

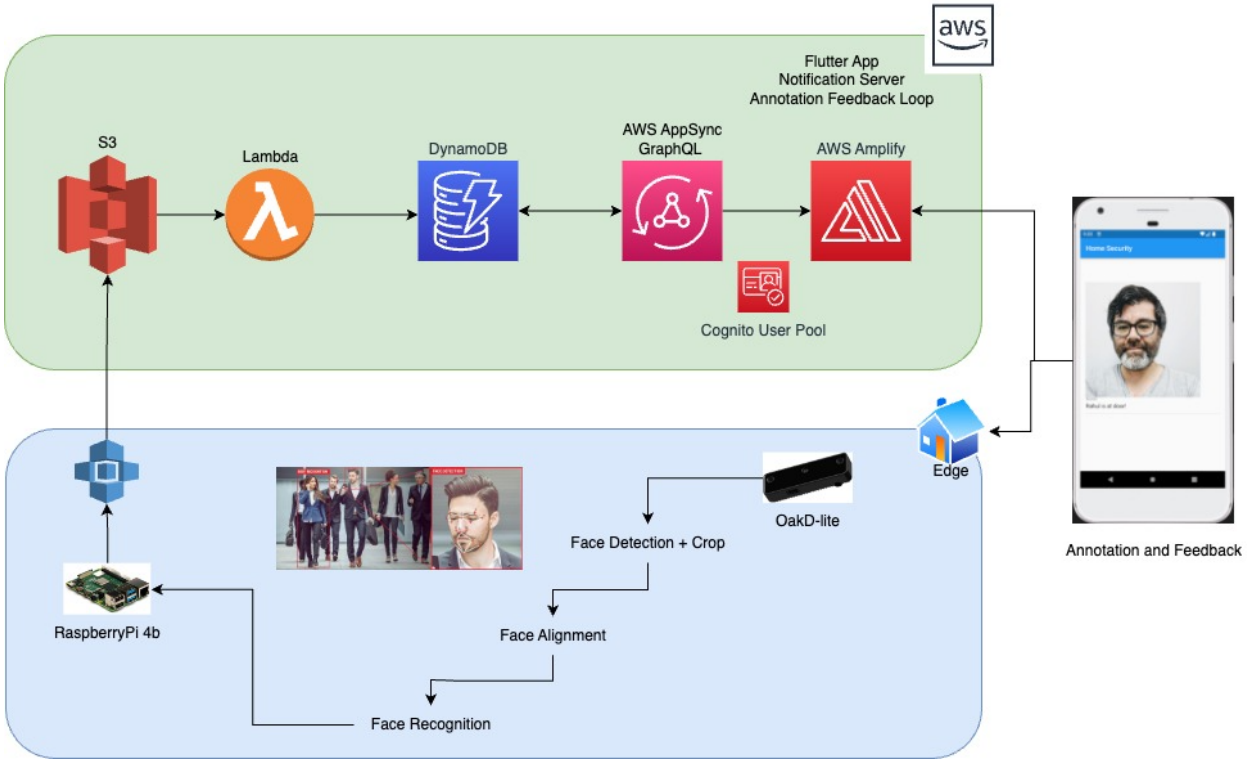
All this processing happens at the edge using the OakD-Lite camera and Raspberry Pi, which reduces latency and bandwidth requirements. Once the data is processed, it is annotated with relevant labels and sent to the cloud, where it can be further analyzed, stored, and acted upon.

In the cloud, the system can ingest the annotated data, and use machine learning models to recognize known faces and objects. When a recognized face or object is detected, the system can notify the user through the iOS app, which can display the relevant information and allow the user to take action, such as label the event, label the person or communicate with the person.

Overall, this system provides a comprehensive security solution that leverages the power of machine learning to detect, identify, and notify the user of any suspicious activities occurring in and around their home, all while reducing the computational burden on the cloud and improving response times.

Final Implementation

Final Architecture



Changes from Proposed Architecture and Why

Cloud Changes

1. In the original project idea, the face detection and recognition were likely performed on a cloud. However, in the final implementation, the processing is happening at the edge, specifically on a RaspberryPi and the OakD-lite camera. This means that the image data captured by the camera is processed on the device itself rather than being sent to a cloud and AWS Rekognition for processing. This approach has several benefits, including faster processing times, reduced bandwidth requirements, and improved privacy and security.
2. AWS AppSync is a fully managed service that makes it easy to develop GraphQL APIs by handling the heavy lifting of securely connecting to data sources like AWS DynamoDB, AWS Lambda, or any HTTP data source. GraphQL is a query language for APIs that allows clients to request only the data they need, making it an efficient and flexible way to retrieve data from a server. By using AWS AppSync with a GraphQL interface, the system can efficiently query and retrieve data from various sources.

3. AWS Cognito is a user authentication and authorization service provided by AWS. It allows users to sign up and sign in to web and mobile applications using their own username and password, as well as supporting third-party authentication providers like Facebook, Google, and Amazon. In the final implementation, AWS Cognito User pool is used to authenticate the Flutter App user, ensuring that only authorized users can access the app.
4. In the original project idea, the system was designed to send push notifications to the homeowner when an unfamiliar face was detected on the porch. However, in the final implementation, GraphQL subscription along with DynamoDB polling is used instead. GraphQL subscription allows clients to receive real-time updates when new data is available. This change is done because the push notification mechanism is more involved and requires a lot of configuration at the firebase and other centralized services. In a practical use, push notification can be implemented with additional efforts.

Edge Changes

Deviations from the proposed architecture in the edge pipeline fell into 2 categories. The source video footage using the OakD-Lite instead of the RTSP camera and the ML Models in the pipeline itself.

Source Video

About 20 hours were invested by Rahul in an attempt to stand up a reliable connection to the RTSP streams of the existing security camera systems utilizing a combination of HomeAssistant.Io and Frigate software packages. It was first discovered that the Frigate software did not work out of the box on 64bit Rasbpian using the repository and it also did not support the OakD-Lite camera for hardware acceleration, but does support a Coral TPU. The full SD image of Frigate was not tried.

An attempt was then made to utilize Home-Assitant.io to connect to the RTSP cameras, which was successful by imaging a new SD card, but adding custom functionality to this system was too involved for the purpose of this capstone project.

Next an attempt was made to access the pre-processed motion clips from the Lorex NVR system. During this process, it was discovered that Lorex is a whitelabel of Dahua systems. Neither companies provide Developer Python APIs for accessing the NVR. Further investigation of Github repositories did not yield any immediately useful results. Reverse engineering an API could be possible with additional time.

Next a direct connection with the RTSP via VLC was possible, but attempting the same via command-line also did not immediately work.

At this point, the decision was made to utilize the OakD-Lite directly as the video source. It was discovered that the OakD-Lite DepthAI API does possess the functionality to playback video footage from another source (e.g. recorded from RTSP) for detection analysis.

ML Model Pipeline

After re-architecting the Oak-D Lite as the source of video footage, it was determined that a reference implementation with a face recognition pipeline could be repurposed for this project. That pipeline includes Face Detection -> Face Alignment -> Face Recognition. The reference implementation assumed a training period of a known individual, which was an assumption that this system could not make, so effort was needed for creating an identity for

Initial investigation in interviews with neighbors revealed a privacy concern about sending images to the cloud for facial recognition. By moving recognition to the edge, it would be possible to only display images to the owner of the camera and allow that camera owner to optionally share face recognition embeddings and annotations with their neighbors via this system.

Face alignment is also a crucial component of the face recognition pipeline. Integrating emotion annotation was started and removed from the final architecture due to time constraints, but an emotion recognition reference implementation exists and is feasible as additional work to integrate into the current pipeline.

Systems

The system described above utilizes several technical hardware, software, machine learning models, protocols, and web services to provide comprehensive home monitoring and surveillance at the front door. The following is a summary of the major components of the system:

Hardware:

- OakD-Lite for video frames and ML pipeline (Intel Movidius X Visual Processing Unit)
- Raspberry Pi 4b to coordinate edge pipeline
- iOS device for user notification and data labeling

Software / API:

- DepthAI API Spatial AI platform by Luxonis (for robot and computer visual perception)
- Flutter

Machine Learning Models (OpenVINO):

- Face Detection Retail 0004
- Head Pose Estimation ADAS 0001
- Face Recognition Mobilefacenet Arcface

Protocols:

- HTTP

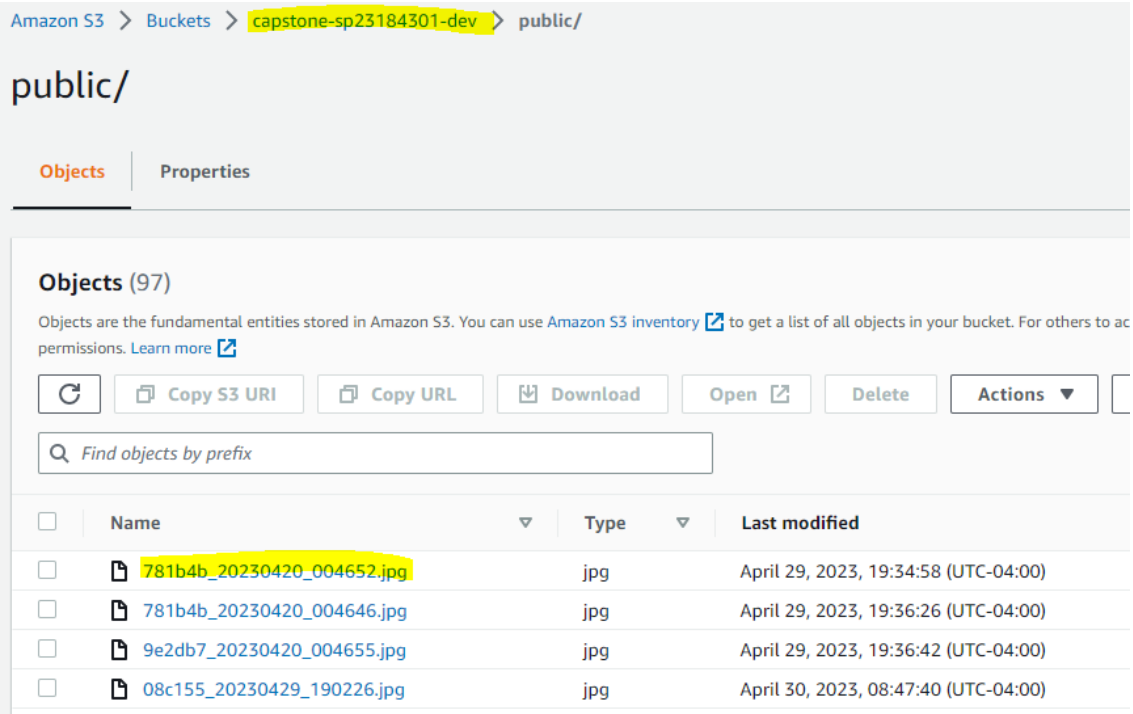
Web Services:

- AWS Cloud for ingest, notification and annotation
- Flutter app for new detection annotation and notification

AWS Cloud

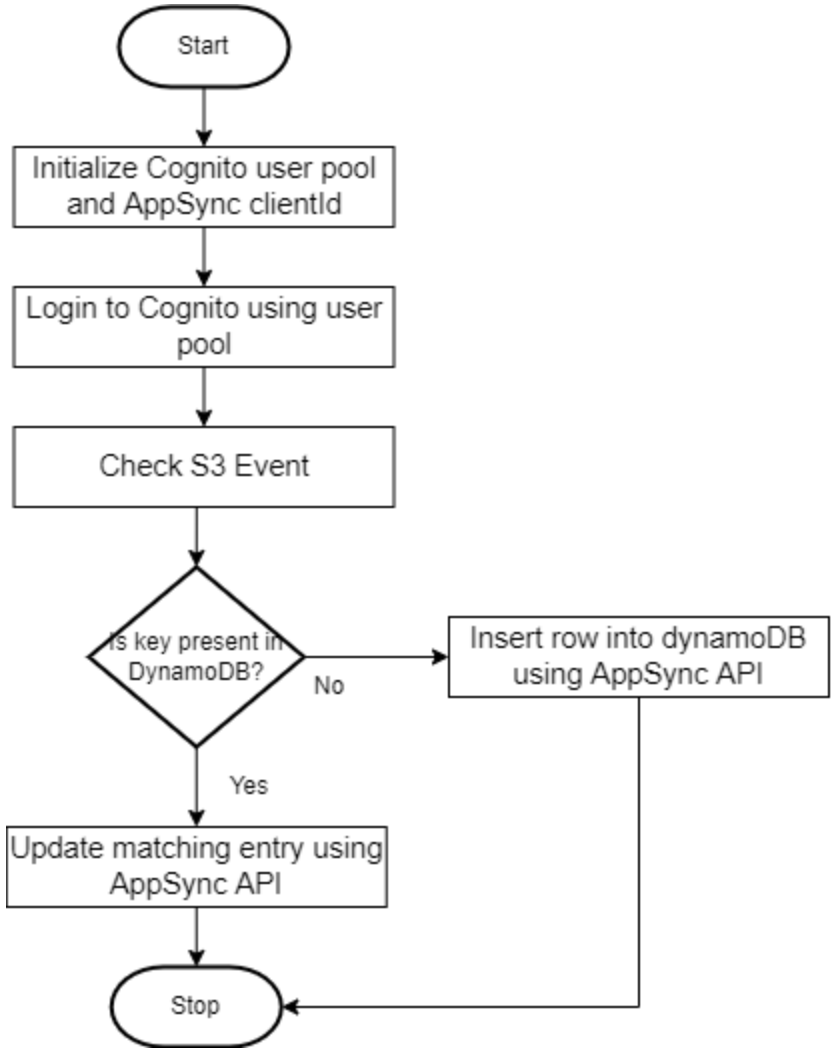
S3 Bucket - capstone-sp23184301-dev

1. In this implementation, the bucket is used to upload pictures captured by the camera on the front porch. These pictures are then processed to detect and recognize faces using the RaspberryPi and OakD-lite camera.
2. In this implementation, the access to the S3 bucket where the pictures are stored is controlled. This means that only authorized users or services can access the pictures in the bucket, ensuring that the data is secure.
3. Each picture uploaded to the bucket is stored in a specific format, namely GUID_DATE_TIME.jpg. The format includes a globally unique identifier (GUID) that identifies the person in the picture and a timestamp that helps identify when the event happened. This format allows the system to easily search and retrieve pictures based on the person's identity and when the picture was taken.
4. When a new face is detected, the system assigns a new GUID to the person and stores it in the S3 bucket along with the picture. This allows the system to easily identify and track individuals as they appear on the front porch. The timestamp helps identify when the event happened, which is useful for tracking the movements of people on the porch and detecting unusual activity, as mentioned in the original project idea.



Lambda - capstoneAppSync

The purpose of the lambda function is to read the event of S3 bucket and update DynamoDB with the event using App Sync.



Here is the pseudo algorithm explaination -

1. Initialize Cognito user pool - In this implementation, a Cognito user pool is initialized to manage user authentication for the Flutter app. This involves configuring user sign-up and sign-in options, such as email and password or social login.

2. Initialize AppSync with clientId - In this implementation, AppSync is used to provide a GraphQL interface for querying and modifying data in DynamoDB. To access AppSync, a client ID is required to authenticate with the service.

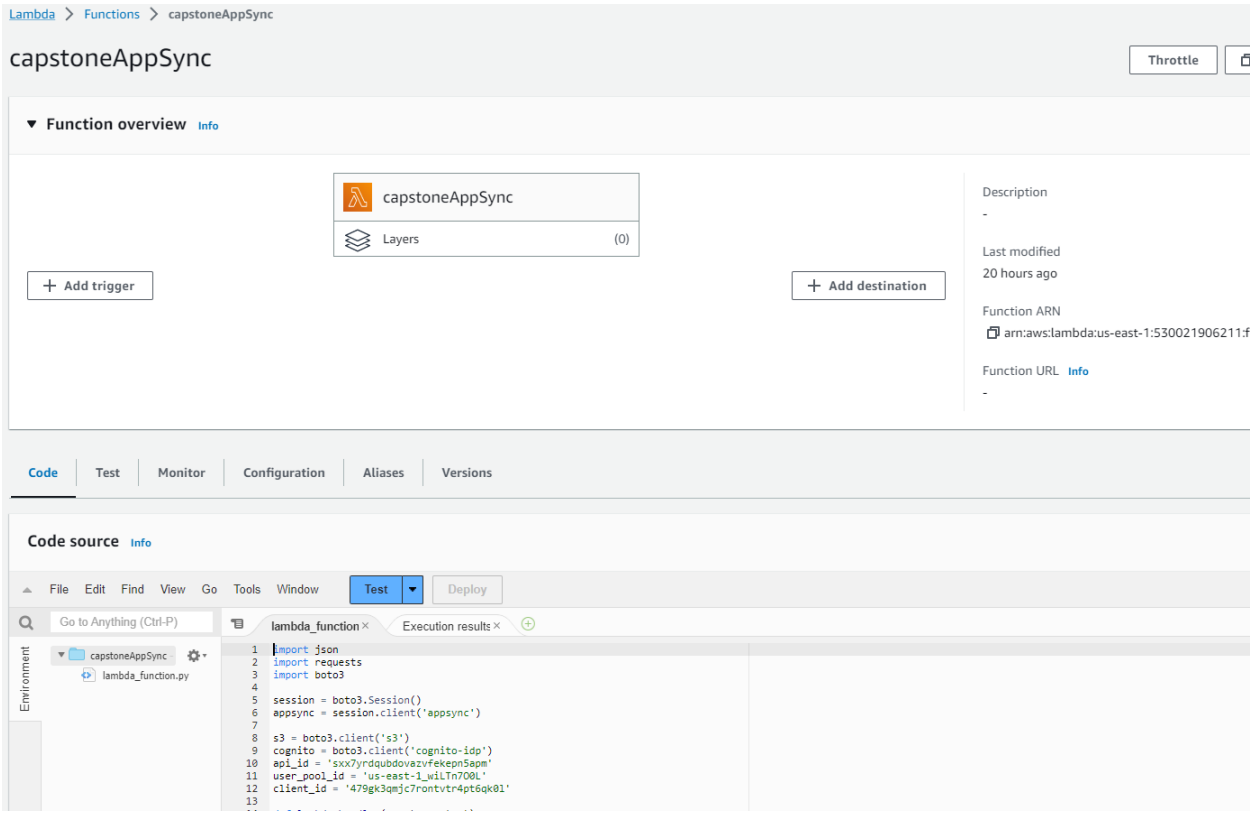
3. On S3 bucket event - An event is triggered whenever a new picture is uploaded to the S3 bucket. This event is used to initiate a series of actions that involve processing the image and updating the relevant databases.

4. Login to Cognito using user pool - When a new picture is uploaded, the request is authenticated using the Cognito user pool.

5. Get key from the bucket and parse the GUID - The key of the uploaded picture contains the GUID and timestamp information in the format specified earlier. The lambda parses the GUID from the key to identify the person in the picture.

6. Search GUID in DynamoDB using AppSync GraphQL endpoint - The lambda uses the parsed GUID to search for the person's record in DynamoDB using AppSync's GraphQL endpoint. This involves sending a GraphQL query to the endpoint that specifies the GUID as a parameter.

7. If the GUID is found in DynamoDB, the lambda updates the record with the new image using the updateCapImage API. If the GUID is not found, it means that the person is new to the application, and the lambda inserts a new record using the insertCapImage API. This allows the system to track the movements of both new and existing people on the front porch.



Amplify

AWS Amplify is a development platform offered by Amazon Web Services (AWS) that provides a set of tools and services to build scalable and secure mobile and web applications. It enables users to build AWS powered applications quickly and easily using a variety of programming languages and frameworks, including React, React Native, and Angular.

Install Amplify CLI
We need to download and install CLI on a local machine in order to use this from local development. Below steps needs to be followed specifically in order to configure it-

1. Install AWS CLI
2. Configure credentials with use of Access Key and Secret
3. Install Amplify CLI
4. Configure the user profile

Initialize Amplify
Before starting using, Amplify needs to be initialized using the below command.

```
>amplify init
```

The `amplify init` command is used to initialize an AWS Amplify project in a local directory. This command is the first step in setting up a new Amplify project and enables you to choose the services you want to use, set up your project environment, and configure your Amplify backend.

Add API
We are using AWS Amplify as a Flutter App which interacts with the DynamoDB we updated in an earlier step.

One of the key features of AWS Amplify is its integration with various AWS services, including DynamoDB.

When used together, AWS Amplify and DynamoDB provide a powerful combination for building serverless applications that can scale to handle millions of users. Here's how Amplify can be used with DynamoDB:

AWS Amplify provides a simple CLI command to create a new DynamoDB table as below

```
>amplify add api
```

Amplify automatically generates GraphQL APIs for accessing the table, along with a set of resolvers that map to DynamoDB operations like PutItem, GetItem, and Query.

Add storage

In order to access S3 or any other storage on Amplify, we need to add storage using below command

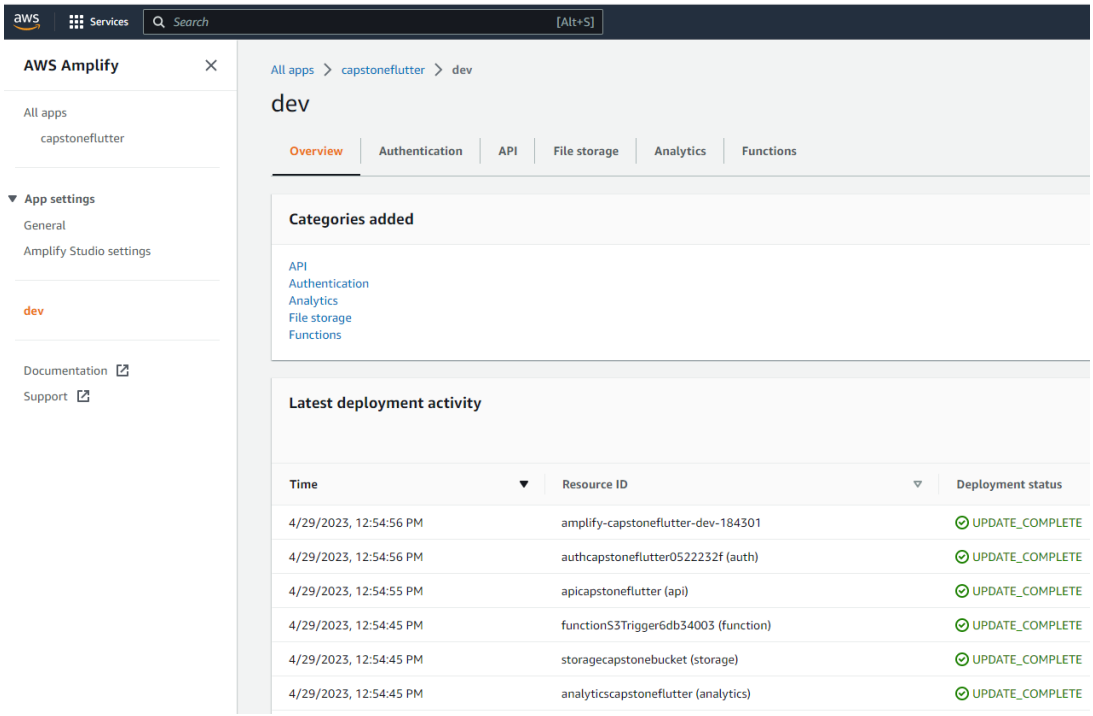
>amplify add storage

Once all applications are configured, the system needs to be pushed to AWS in order to prepare respective resources using below command.

>amplify push

When you run the amplify push command, Amplify will analyze the amplify.yml configuration file and detect any changes made to the backend configuration or resources. It will then apply these changes to the cloud environment, creating or updating the necessary resources.

Screenshots



Flutter App

The main part of this implementation is the App itself which runs using flutter framework. This framework works well with AWS Amplify in order to create a rich UI with strict bindings with AWS cloud resources.

Below packages are needed to be added in order to support the execution-

```
amplify_auth_cognito: ^1.0.0-next.0
```

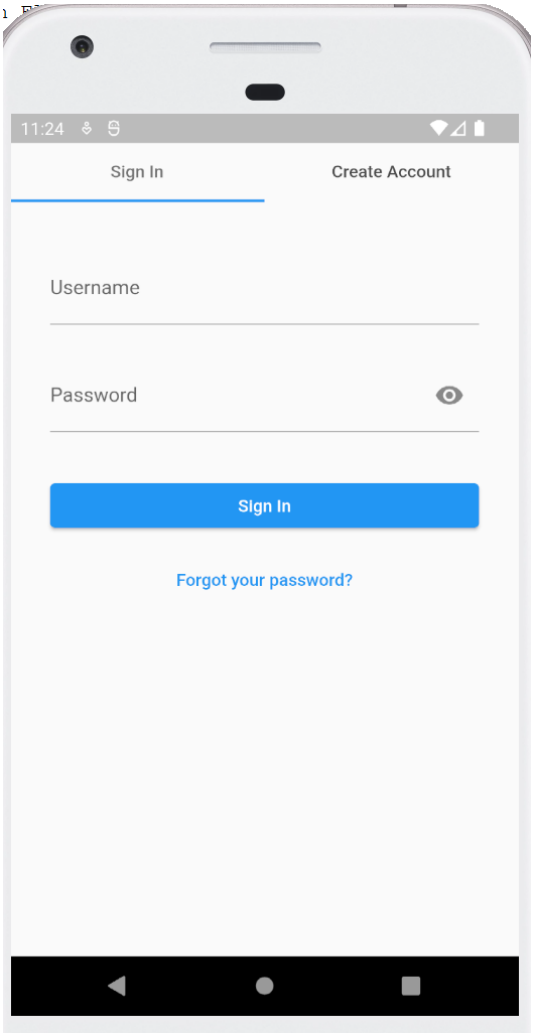
```
  amplify_flutter: ^1.0.0-next.0
  amplify_push_notifications_pinpoint: ^1.0.0-next.0
  amplify_analytics_pinpoint: ^1.0.0-next.0
  amplify_storage_s3: ^1.0.1-next.0
  flutter_aws_s3_client: ^0.6.0

  fluttertoast: ^8.2.1

  #amplify_flutter: ^1.0.0
  #amplify_auth_cognito: ^1.0.0
  amplify_api: ^1.0.0-next.0
  amplify_authenticator: ^1.0.0-next.0
  go_router: ^6.5.5
```

- amplify_auth_cognito: A package that provides authentication and user management features using Amazon Cognito for the Amplify Flutter library.
- amplify_flutter: A package that enables communication between Flutter applications and Amplify backend services.
- amplify_push_notifications_pinpoint: A package that provides push notification services for the Amplify Flutter library using Amazon Pinpoint.
- amplify_analytics_pinpoint: A package that enables tracking of user behavior and events for the Amplify Flutter library using Amazon Pinpoint.
- amplify_storage_s3: A package that provides storage services for the Amplify Flutter library using Amazon S3.
- flutter_aws_s3_client: A package that provides a simple way to connect and use Amazon S3 storage services in Flutter.
- fluttertoast: A package that provides simple toast notifications for Flutter applications.
- amplify_api: A package that provides a set of APIs for interacting with GraphQL APIs for the Amplify Flutter library.
- amplify_authenticator: A package that provides a set of UI components for authentication and user management for the Amplify Flutter library.

Authentication Screen

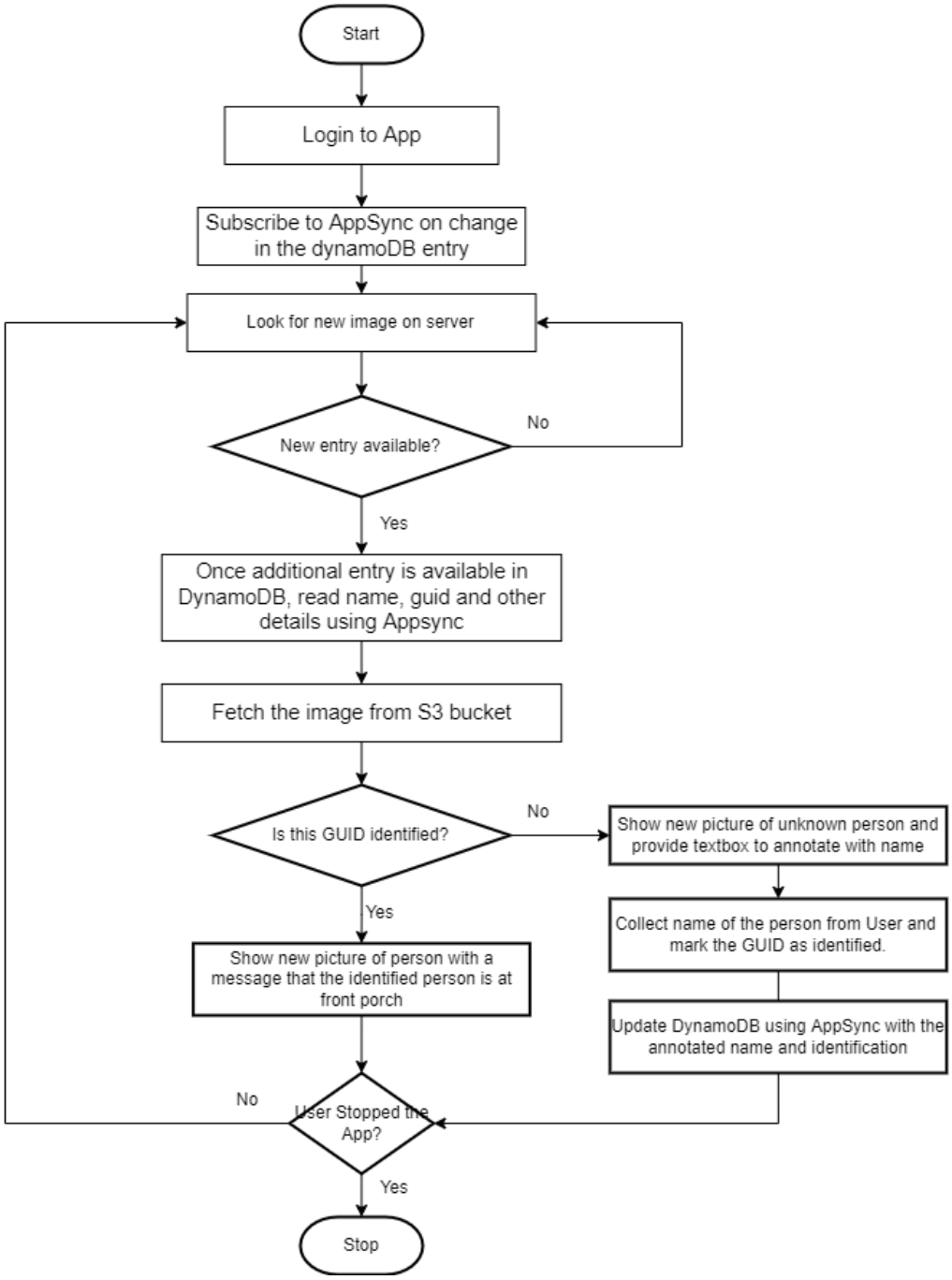


Above screen is provided by AWS Cognito authentication.

Main.dart

This is the main file at ‘lib\main.dart’ which contains the logic of implementation.

Pseudo algorithm



1. Login to App: The user logs into the Flutter application using the AWS Cognito user pool.

2. Subscribe to AppSync on change in the DynamoDB entry: The Flutter application subscribes to changes in the DynamoDB table using the AWS AppSync service.

3. Once an additional entry is available in DynamoDB, read it using Appsync: When there is a new entry in the DynamoDB table, the Flutter application reads it using the AppSync service.

4. Check GUID and identify if the name is already known: The Flutter application checks if the GUID of the new image already exists in the DynamoDB table. If it does, it means that the name of the person in the image is already known, and the application can display that name.

5. If yes, then populate that name along with the new image: If the name is already known, the Flutter application populates the name along with the new image and displays it to the user.

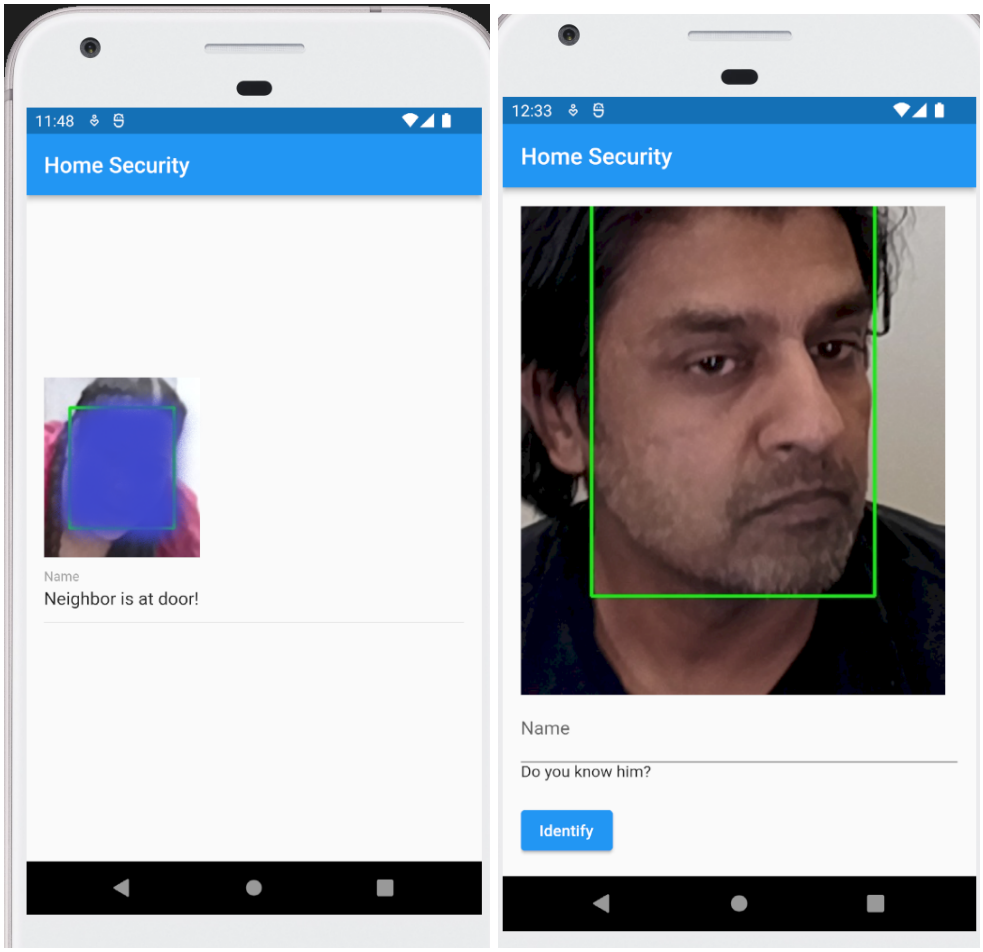
6. If not, then enable the textbox and a submit button to annotate this new image: If the name is not known, the Flutter application enables the user to enter a name for the person in the image by displaying a textbox and a submit button.

7. On user type a name of this new picture, submit the update entry to DynamoDB using AppSync: When the user enters a name for the person in the image and submits it, the Flutter application updates the DynamoDB table using the AppSync service with the new name and the corresponding GUID.

8. Show the user the success message: After updating the DynamoDB table, the Flutter application shows a success message to the user.

Results from flutter App

Below picture shows the person is at the door and identified as your neighbor. The face is faded here in the screen below for privacy purposes.



The first picture displays a scenario where a neighbor is detected by the camera at the front porch. The face detection and recognition system in place correctly identifies the neighbor, and the App displays a message indicating that the neighbor is at the door.

The second picture represents a scenario where the face detection and recognition system does not identify the person at the front porch, and the App prompts the user (presumably the homeowner) with a message asking if they recognize the person.

This is the stage where the user can add annotations to the picture by typing in the name or description of the person. Once the user submits this information, it will be sent to the AppSync endpoint which will update the information in DynamoDB. The next time a similar face is detected by the camera, the system will use the updated information to correctly identify the person and notify the user about their presence at the front porch.

Overall, this system allows for continuous learning and improvement of the face detection and recognition capabilities by leveraging the user's input to update the database with new information about people who frequent the front porch.

Steps to recreate the Amplify App and AWS infrastructure

1. Install AWS CLI and Amplify CLI
2. Install correct version of python and pythonenv
3. Install Flutter
4. Pull code from github
5. Open it in VSCode
6. Install necessary plugins on VSCode for flutter and Dart
7. In the root directory, run below commands
   amplify init
   amplify push
8. Login to App by running flutter App locally
   flutter run
9. Create a user by signing up to the prompt. Keep user information handy, we will need it while configuring lambda.
10. Create S3 bucket which will contain the images
11. Prepare Access Key and secret which has AWS S3 full access. Update Main.dart downloadImageAndDisplay method at line 233 with Access Key and secret. Required to pull images from S3.
12. Prepare Lambda with python 3.7 environment and add a trigger to receive notification when an object is added to S3 bucket.
13. Update lambda code with the code from aws/capstoneAppSync-lambda.py file.
14. Update lambda code to have correct credentials at line 28 from step 9.

With the above steps the system should be running and whenever the image in a specified format arrives in this bucket, it will be visible on Flutter App with corresponding annotation.

Edge Implementation

Setup: Hardware, OS, and Dependencies

Two Raspberry Pi 4B 8GB were utilized with Raspberry Pi OS desktop (64-bit bullseye). For development and testing a Macbook Air M1 was also used. All 3 devices have their own OakD-Lite camera. Each camera contains an Intel Movidius X VPU for accelerating ML workloads. Install instructions for RPi OS. An Argone One M.2 enclosure with SSD support was used to house the RPI4B. The added power constraints required the purchase of CanaKit power supply.

It also supports an API by Luxonis called DepthAI, *"which allows robots and computers to perceive the world like a human."* Raspberry Pi OS installation instructions are available here.

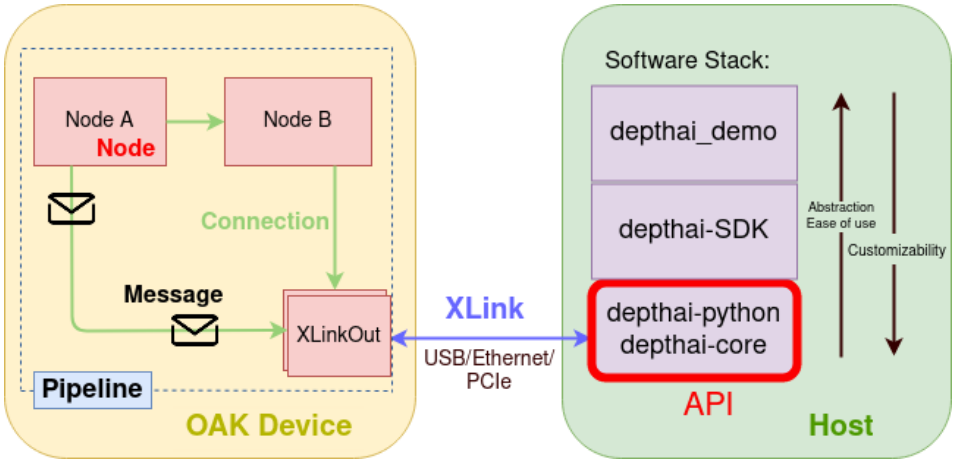
The AWS client SDK for Linux ARM also needs to be installed and can be found here. An access credential is also needed with instructions available here. In this project the same

credentials are used for all 3 devices to connect to, but as a security best practice each device should have its own credentials.

DepthAI Reference Implementation

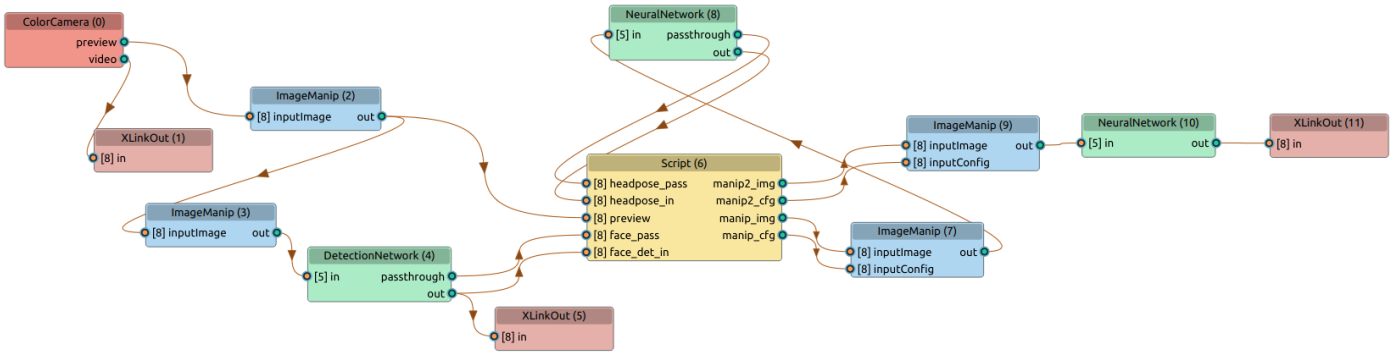
The code utilized can be found in our repository, as well as, this link. The main packages utilized are: **argparse**, **blobconverter**, **cv2**, **depthai** and **numpy**. In this reference implementation, there is a separate training step for a known user that is specified at program start using the `--name` argument. The **blobconverter** package is needed to make ML models compatible with the Movidius X VPU. The cv2 package is used for drawing text, boxes and showing each frame in succession. The **numpy** package is used as the data format for embeddings and images, it has many utility functions for storing embeddings to disk, CPU based image manipulations and embedding similarity comparisons.

The final package is **depthai**. This package does the heavy lifting for the ML pipeline. It provides functionality to define a pipeline that will run on the camera with a messaging system between nodes of the pipeline. These messages can be sent to the Host machine (RPI4) at any point in the pipeline and be combined or used individually by the Host script. This diagram shows the general architecture of a pipeline and host.



Pipeline:

The Face Recognition reference pipeline is diagrammed below.



- The green boxes represent the ML models utilized. From left to right that includes Face Detection Retail 0004, Head Pose Estimation ADAS 0001, and Face Recognition Mobilefacenet Arcface.
- The blue boxes represent Image Manipulation nodes that are required as a pre-processing step for each model. In all 4 cases, the ImageManip nodes are resizing the image for the ML models. The first ImageManip node is also caching frames from the Camera node to prevent the camera from freezing. Additional investigation into this node may yield optimization opportunities to skip blurry frames or simply reduce the number of operating frames.
- The red boxes represent Input / Output. Darkest red is the RGB Camera itself and is set to 1080P resolution, but could also be set to 4K. This device also supports 2 additional mono cameras for depth estimation. The lighter red boxes are sending frames out to the Host as messages.
- The Yellow box is a Script node and serves to coordinate between models. It's common in DepthAI pipelines to have a single Script node.

Helper Functions:

- Normalizing frames and bounding boxes
- TextHelper class for adding text to frames
- FaceRecognition class for recognizing faces using cosine similarity and saving them to a Numpy database file.
- Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. This is a fast approximation for face recognition and could be replaced with a dedicated vector database, since looping through a list would not scale for an entire neighborhood let alone a city.

Pipeline Code:

- Set up the pipeline using DepthAI nodes (e.g. ColorCamera, ImageManip, MobileNetDetectionNetwork, NeuralNetwork, XLinkOut)
- Define the connections between nodes

Running Pipeline and Output results

- Create output queues for the pipeline

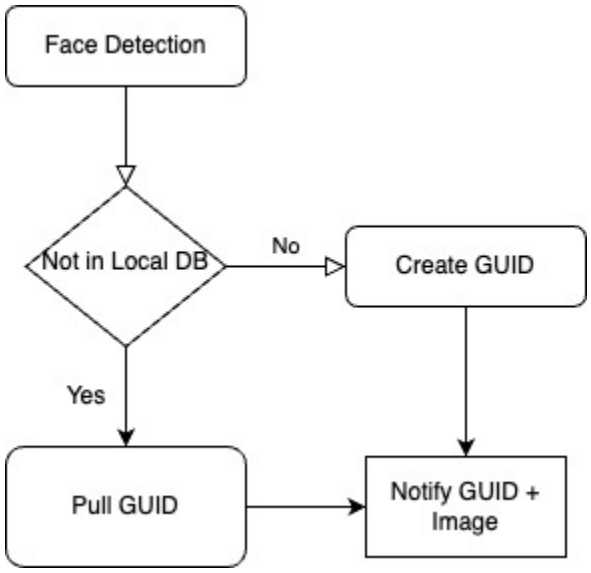
- Use a synchronization class to coordinate the output messages from different nodes in the pipeline
- Process the output messages and display the results in a video window

Custom Implementation

Additional packages include **datetime**, **boto3**, **socket**, **time** and **defaultdict**. The **boto3** and **socket** packages are required for uploading detections to the s3 bucket and determining the name of the host for the s3 bucket. The **datetime**, **time** and **defaultdict** packages are utilized to filter false positive detections, partial faces and to determine if enough time has passed to qualify as a new session for a detection.

Additional command line arguments were created for convenience including: `--skip_every_det`, `--skip_every_show`, `--display_size`, `--time_new_det`, `--skip_init_det`, and `` `--no-display` ``. Each of these were varied during experimentation and development as new features were added. In the final run detections were made on every frame with a value of `1`, but only every `10th` frame was displayed, at a resolution of `400x400`, with `5s` between detections considered a new detection, and the first `10` detections to be skipped each time to assist in filtering out partial faces.

The `new_recognition` method is modified to generate a new GUID for an UNKNOWN person and sets the name to the GUID. The new GUID is created using the uuid.uuid4() function, and only the first six characters of the generated UUID are used as the GUID.



For debugging purposes, at the start of each run, the `read_db` method prints every GUID/Name and the number of embeddings stored. This helps to understand how many new partial faces had been detected.

The `create_db` method takes a GUID parameter to specify a unique identifier for the person, otherwise it falls back to the self.name attribute. If neither of these is provided, it prints an error message and returns. It then saves the new embedding to a compressed NPZ file with the appropriate filename, and adds it to the dictionary of embeddings associated with the appropriate person. If this person is new, not already in the list of labels, it adds them.

The main loop of the reference implementation, just provides real-time detections of known pre-trained faces. The custom implementation adds significant functionality that includes expanding the bounding box, keeping a count of detections per session, retrieving the recognition GUID, and saving the new detected faces to disk and to the s3 bucket. Each of these is parameterized with skip_every_det, skip_every_show, skip_init_det, display_size, and time_new_det.

Innovation

The innovation in this project lies in combining the use of cloud technologies like AWS Amplify, AppSync, Cognito, and DynamoDB to build a scalable, secure, and efficient system that allows annotations of sensor data from Edge devices to be shared amongst the participants.

The advent of real-time ML capable edge devices provides the possibility of additional privacy and data transfer efficiency. Sensor networks can be mixed between those that are data gathering and transmitting both embeddings and images for annotation and those that are just for detecting and collecting embeddings for later comparison.

New Learnings

This project work was quite involved and needed a lot of learning new technologies. AWS Amplify and Cognito were completely new to us and we needed to understand flutter App preparation as well. All in all, the entire learning experience was quite good.

Face detection algorithms along with the protocols supported by Camera to stream images was a new area. This needed a lot of exploration on how the protocols will work and trying different combinations to make it work.

Below are the before and after knowledge levels of various technologies-

1. S3 - Before 50%, after 70%
2. Lambda - Before 50%, after 55%
3. DynamoDB - Before 20%, after 60%
4. AWS Cognito - Before 5%, after 60%
5. AWS AppSync - Before 2%, after 60%
6. AWS Amplify - Before 2%, after 60%
7. Flutter - Before 20%, after 60%
8. DepthAI - Before 0%, after 40%
9. Home-Assistant.io - Before 0%, after 25%

10. Frigate - Before 0%, after 20%
11. OakD-Lite - Before 10%, after 60%
12. RPI4 enclosures with SSD (NVME vs Sata) - Before 0%, after 70%
13. Power needs Edge devices - Before 30%, after 80%
14. RT Face Recognition Pipeline - Before 10%, after 90%
15. RTSP - Before 20%, after 60%
16. Lorex NVR - Before 10%, after 50%

Challenges

AWS implementation

1. During the implementation of the AWS architecture, there were several challenges faced by us. One of the challenges was related to Lambda function authentication and access control using Amplify. The Cognito user was assigned strict access control at the row level, making it difficult to locate objects inserted by the Flutter App on the Lambda function. To address this issue, the team had to gain a thorough understanding of the roles and policies of AWS and assign the correct permissions to the Cognito user.

2. Another challenge faced was related to API Key authentication using Amplify. Since API Key authentication is not supported by S3, accessing a bucket written by the API key authentication made the object inaccessible by the AWS Cognito user. To overcome this challenge, the team had to correct the program to use AWS Cognito authentication in all places.

3. We also faced a challenge with DynamoDB external update, which is restricted with strict access control. Rows in DynamoDB were specific to the logged-in user and cannot be accessed by AppSync with a different client ID. This was a learning experience for the team, and they had to make use of the same user to modify details. Unfortunately, this information was not indicated in the AWS documentation, and the team had to discover it through trial and error. Overall, these challenges highlight the importance of having a deep understanding of the AWS architecture and policies and being prepared to troubleshoot and adapt to unexpected issues.

Edge implementation

1. Initiating and maintaining a reliable RTSP connection over Wifi was a challenge due to the specifics of the NVR system.
2. Having enough testing data on a live camera system.
3. Compatible open source software for the proposed architecture with the hardware available.
4. GPU acceleration devices for the originally proposed architecture.
5. Learning the DepthAI Machine Learning API.

6. Standing up multiple RPI4 devices with the OakD-Lite Camera with the added power consumption needs. It took a few weeks to order and try different power adapters and USB adapters to ensure all devices received enough power, data throughput was not negatively affected and to run basic benchmark tests.

Future enhancements

There are several potential future enhancements for the above project. We talked and discussed about following potential future enhancements to the project:

1. Real-time notification: Currently, the system uses GraphQL subscription to check for updates in DynamoDB using AppSync. This could be improved by implementing real-time notification using AWS SNS or AWS EventBridge to notify the app when there is a new image or update in DynamoDB.

2. Firebase Push notification system for Android and a respective service for iOS will be used for the push notifications on the App and users can act on the notifications faster.

3. Face recognition accuracy: While the current face recognition model works well, there is always room for improvement in terms of accuracy. The system could be enhanced with a more advanced deep learning model or by using multiple models to achieve higher accuracy.

4. Cloud-based face recognition: The current implementation performs face recognition on the Edge using a Raspberry Pi and OakD-lite camera. However, cloud-based face recognition could be implemented to reduce the hardware requirements and provide more flexibility in terms of camera placement. AWS Rekognition is considered along with AWS Sagemaker which makes the ideal fit for such algorithms. Though cloud-based recognition is technically possible, it does face legal considerations depending on the use case.

5. Automated annotation: The current system requires manual annotation by the user for unknown faces. This could be improved by implementing automated annotation using techniques such as natural language processing (NLP) to extract names from user inputs or image metadata.

6. Integration with smart home devices: The system could be integrated with smart home devices such as door locks, security cameras, and lights to provide a more comprehensive home security solution.

7. Privacy and security: As with any home security system, privacy and security are paramount. The system could be enhanced with additional security features such as two-factor authentication and encryption to protect user data and prevent unauthorized access.

8. Use a scalable Vector DB system for comparing face embeddings.

9. A different face recognition model that could potentially be more accurate, using more landmarks.

10. Replace the cosine similarity metric. Maybe a cluster based approach.

11. Use depth to prevent spoofing (someone showing a face on a phone).

12. Add emotion annotation with a dedicated ML model.

13. Face detection is not enough when someone turns their head, never shows their face, or is moving too quickly. A person based detection system may work in combination with ranking based on context of day / time and whether this person is with others.

14. Cloud-based de-duplication of similar faces.

15. App enhancements for suggested annotations.

Conclusion and Practical Considerations

Overall, the Porch Buddy leverages the power of machine learning and edge computing to provide real-time monitoring and analysis of video data, and the cloud-based web services and Flutter app provide the user with timely and relevant information about activities happening at the front door.

The practical use of the project is to provide a comprehensive home monitoring and surveillance system that specifically focuses on activities happening at the front door. The system can detect, identify, and notify the user of any suspicious activities, such as unknown individuals entering the home, package deliveries, or even the behavior of pets. The system can also recognize known individuals, such as family members or caregivers, and notify the user accordingly.

The system provides peace of mind for users who may be away from home for extended periods and want to stay connected to what's happening at their front door. It also provides an added layer of security for families with young children, elderly parents, or sick family members who may require regular visits from caregivers or friends. The system's ability to monitor and analyze video data in real-time makes it an effective tool for home surveillance, and the cloud-based web services and iOS app provide timely and relevant information to the user, allowing them to take appropriate action as needed. Overall, the practical use of the project is to enhance home security, safety, and comfort, and provide a reliable and comprehensive monitoring solution for users.

References

1. Tutorial on AWS Amplify and flutter
2. AWS CLI
3. Flutter
4. Python
5. DepthAI Face Recognition Experiment Repository
6. DepthAI Emotion Recognition Experiment Repository
7. OAK-D-Lite
8. DepthAI API Documentation
9. Intel Movidius Myriad X VPU Product Brief
10. Face Detection Retail 0004
11. Head Pose Estimation ADAS 0001
12. Face Recognition Mobilefacenet Arcface
13. Home-Assistant.io
14. Frigate ML Detections
15. Argone One Enclosure
16. CanaKit Power Supply
17. Mount SSD drive to RPI4
18. Check CPU speed