Acronym

Acronym	Full Form
AV	Automated Vehicle
DFL	Data Feedback Loop
CV	Computer Vision
SAM	Segment Anything Model
GTD	Ground Truth Data
Ph	Phase
SPh	Sub-Phase
AL	Active Learning
PI	Program Increment
DSDM	Dynamic Software Development Framework

Background

Self-driving cars use a mix of sensors, cameras, and smart algorithms to move around safely.

They need two main technologies to do this: computer vision and machine learning.

1. **Computer Vision** uses cameras and sensors to take pictures and videos of everything around the AV. This includes both static and dynamic objects such as road lines, traffic lights, people, and other cars. The car then uses special techniques to understand these pictures and videos.
    a. Object Detection: The car uses advanced computer vision methods to quickly and accurately detect and classify the objects present on the surrounding pathway in real time.
    b. Object Tracking: After the car detects something, it uses object tracking technique to monitor the dynamic objects, which is crucial for path planning and collision avoidance.
2. **Machine learning**: It looks at the data from the cameras and sensors. Then it uses special algorithms to find patterns, make predictions, and learn from new information. This helps the car make smart decisions, handle new situations, and get better over time.

Scope of this analysis is only Computer Vision data, i.e; Visual Perception.

Data Feedback Loop is mandatory in every real time application to improve the performance of the system, where any misidentified or edge case can be corrected and fed back to the model to learn from the updated data.
There are some scenarios, as explained below, where the model needs real time data to predict the object accurately, which are absent in the training dataset.
Examples
1. Sometimes heavily occluded 'Stop' Signs misidentifies this.
2. Person holds 'Stop' signs without actually meant for stopping the AV.
3. School bus 'Stop' sign may predict the AV to stop it.

Besides this, following are some edge case scenarios which also needs human labelling for better model prediction, where the model may struggle to classify the object correctly and may not respond appropriately.

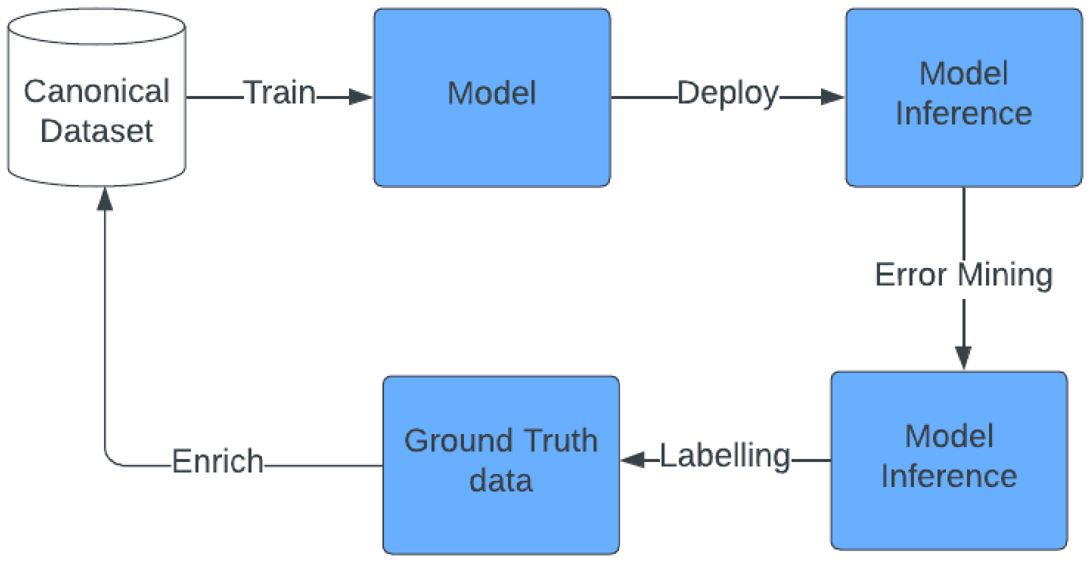
Examples

1. Cargo Bikes or Bikes with Trailers: Cargo bikes, bikes with trailers, or bikes carrying large objects can be difficult for models to recognize correctly. The model may interpret the bike as an extended vehicle, especially if the trailer or cargo changes the bike's shape significantly.
2. Motorcycles with Sidecars: Motorcycles with sidecars or small trailers present an unusual shape that can be easily confused with other types of vehicles like compact cars or three-wheeled vehicles.
3. Reflections of Vehicles and Bicycles on Shiny Surfaces: Reflections on wet roads, building windows, or shiny car surfaces can create “phantom” objects, where the model might mistakenly interpret reflections as actual vehicles or bicycles.
4. Animals Being Transported in or on Vehicles: Animals (like dogs) in the bed of a truck or in a car window can be interpreted as independent objects or mistakenly classified as pedestrians or moving obstacles.

So, here a need of **Data Feedback Loop** comes into picture.

A **data feedback loop** for an autonomous vehicle system plays a central role in ensuring continuous learning, adaptability, and improvement in vehicle performance and safety. This feedback loop serves as the engine for refining perception, decision-making, and control systems within autonomous vehicles.

In a high level, the process flow looks like below.



Initially, the CV model gets trained on commercially available or open source dataset that lacks the real world scenarios as described above. Once the model is trained and deployed in AV, it starts capturing the real time scenarios where it fails to predict, or prediction accuracy is below threshold. In this case, the model should be retrained on the newly captured data with human labelled ground truth value. The model gets retrained on enriched dataset with better prediction accuracy.

Problem Statement

To create a dynamic and intelligent data feedback loop that continuously enhances autonomous vehicle systems through real-world data, providing real-time insights, improving decision-making, and supporting iterative learning.

Scope of the Document

1. Define the vision, strategy, and roadmap for developing a scalable, secure and robust **vision perception data feedback loop** that will be responsible for the success of a fully autonomous driving system.
2. Design the functional architecture of this system, to enhance system accuracy and responsiveness under diverse driving conditions.
3. Identify and allocate necessary resources, including personnel, technology, and budget, to support development and deployment.
4. Finally, establish a comprehensive strategy for long-term management and maintenance, focusing on continuous improvement, operational efficiency, and adherence to safety and regulatory standards.

Vision

To build a secure, scalable, and intelligent data platform that empowers autonomous vehicle systems with real-time data insights, enhances decision-making, and accelerates continuous improvement, all while prioritizing safety, privacy, and compliance.

Phase	Scope	Timelines
I	1. Stream data ingestion based Data Feedback Loop with Active Learning for existing CV models 2. Train existing CV models to handle Edge cases in the same architecture, using Synthetic data generated through GAN models. 3. Monitor CV models 4. Secure the system by adopting required AI Risk mitigation controls 5. MLOps (covered with a limited scope)	Up to 1 yr
II	Additional enhancements + bug fixes with the existing architecture + POC for the new architecture	1 yr -2 yrs
III	More refined architecture to support SAM or Large Vision Models with RAG & Vision Language Models	2yrs and beyond

Assumption

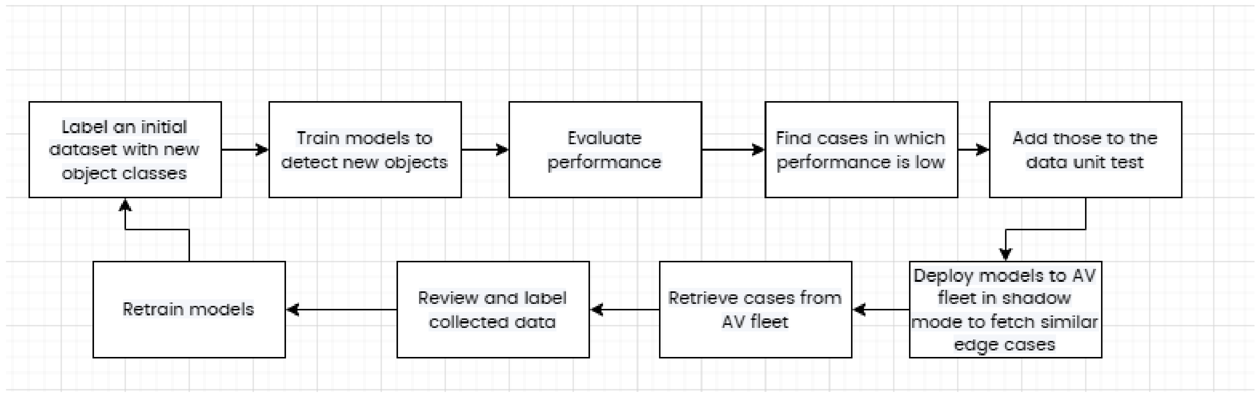
1. Initially, the CV model gets trained on commercially available or open source dataset or that lacks the real world scenarios. So, we need to build a data platform with real time data capture, once the CV model gets deployed in AV and starts collecting data, means the data lake or platform shall contain only Data Feedback Loop.
2. This data lake is for only image data and to build corresponding Observability system. It excludes other telemetry data emitted by the sensors of AV.
3. MLOps is covered in a limited version, assuming that this architecture will use Bosch’ existing MLOps platform.

Business Value [Assigned some hypothetical values]

1. **Enhanced Autonomy and Safety**: Real-world data collected from vehicles helps Bosch identify edge cases and retrain models to better recognize and respond to various driving scenarios, enhancing both driver and pedestrian safety by 15%.

2. **Reduced Development Time and Increased Model Precision**: This rapid iteration improves the precision of its computer vision models, enabling 20% faster deployment of refined updates to its fleet, giving Bosch a competitive edge in the autonomous driving space.
3. **Scalability of Autonomous Fleet Learning**: DFL will leverage data from its global fleet, allowing the company to scale its machine learning efforts based on a diverse set of driving conditions and environments, making the system scalable up to 20%.
4. **Cost Savings and Efficiency**: Bosch will reduce the need for extensive manual testing by using the data feedback loop to automatically detect and correct model errors, thereby saving cost up to 15%.

Workflow for Real time data capture with Data Feedback Loop



Label an initial dataset with new object classes
Train models to detect new objects
Evaluate performance
Find cases in which performance is low
Add those to the data unit test
Deploy models to AV fleet in shadow mode to fetch similar edge cases
Retrieve cases from AV fleet
Review and label collected data
Retrain models

1. Detect the failure scenario: Data workflow uses real-world driving examples to iteratively run machine learning algorithms, which are then used to train CV models of AV. This model would constantly be running in "shadow mode." When the driver does something different from it would have done, or the neural network signals that it doesn't know what to do in the presented scenario, it notes that event as an inaccuracy. Data Engine logs these inaccuracies into its memory, it can retroactively collect them.
2. Collect image data having similar failure scenario or contextual examples: Suppose Bosch detects enough inaccuracies under similar circumstances. In that case, Bosch can then search for similar driving conditions found in other cars in the Bosch fleet, even if it didn't detect an inaccuracy. Bosch can then harvest similar contextual examples.
3. Ground truth of the data: Next, the vehicle identifies an inaccuracy. That inaccuracy enters Bosch's Unit Tests to verify its legitimacy and that it's not the result of subpar driving by the human driver. If the inaccuracy is deemed legitimate, Bosch then asks its fleet for more examples of where the inaccuracies are found. Those examples are then correctly labelled by a human, and used to train the neural network. The network is then redeployed to the data source to collect more inaccuracies.
4. Prepare a dataset combined with predicted and GTD
5. Train the CV models with this new dataset: Using this newly formed, well-labelled data set, Bosch can re-train its neural network to better react to the scenario in which those inaccuracies were presented. Once the neural network is re-trained, it can deploy the newly revised self-driving neural network to "shadow mode" and collect new data examples for further inaccuracies.
6. Deploy these newly trained models into AVs.
7. Iterate the process.

Workflow consists of following phases
Ph-1: Stream data ingestion based Data Feedback Loop with Active Learning for existing CV models

SPh-01: Build Data Feedback Loop
SPh-02: Prepare dataset with new data through data catalogue
SPh-03: Re-train the CV model with new dataset (OOS of this document)
SPh-04: Deploy the model into edge
SPh-05: Get the model inference and model evaluation metrics
SPh-06: Continue with Phase-1.
SPh-07: AI Risk Mitigation Control
SPh-08: Non-Functional Requirements

Strategy

1. Prioritize High-Quality Data Collection
    a. Through AWS Kinesis Video streams or Apache Kafka

2. Develop a Scalable Data Ingestion and Storage System

- **Cloud-Native, Scalable Infrastructure**: Build a highly scalable, cloud-native platform with an architecture that can handle billions of data points generated daily. Utilize a hybrid storage approach that supports both edge storage for rapid processing and cloud storage for long-term analysis.
- **Data Compression and Efficient Transmission**: Implement advanced compression techniques and adaptive data transmission protocols to optimize bandwidth and reduce latency, ensuring data is transmitted without overwhelming networks.
- Apply a required **data retention and wipe out policy**, if applicable.

3. Set up a Data Cleaning and enrichment process

- **Centralized Data Lake with Distributed Processing**: Develop a data lake for centralized storage, allowing diverse data types (structured, semi-structured, unstructured) and enabling distributed processing frameworks for efficient analysis. These are for building both Data Feedback Loop and Observability system for it.

4. Set up Data Catalogue for centralized governance of data platform

- Maintain data quality, lineage, data discovery and data access framework.
- Initiate data labelling process.

5. Foster a Collaborative Ecosystem and Open Standards

- **Open APIs and SDKs for Third-Party Integrations**: Develop APIs that enable integration with third-party applications and analytics platforms to support V2X (Vehicle-to-Everything) communications and real-time information exchange.
- **Encourage Industry Collaboration**: Participate in open-source and industry-wide initiatives to set standardized data formats and protocols, fostering innovation and creating an ecosystem where autonomous data can be leveraged collectively.

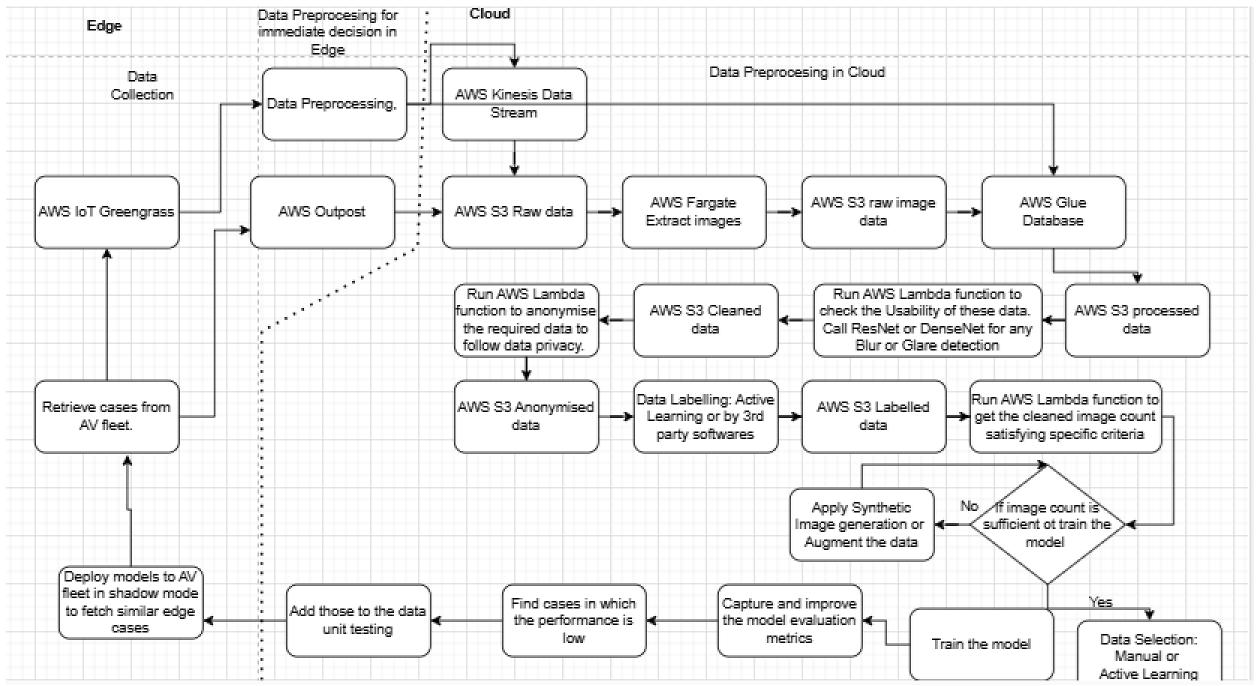
6. Build a Transparent, Robust, and Secure Platform

- **Compliance-First Design**: Design the platform to align with data privacy regulations (GDPR, CCPA) and industry standards (ISO 26262, UL 4600). Implement privacy-preserving techniques like anonymization and differential privacy to protect user data.
- **Transparent and Auditable System**: Introduce transparency in data handling practices, enabling users to access data logs related to their journeys, creating trust with end-users and aligning with regulatory demands.

- **End-to-End Security**: Secure data at every layer, with encryption protocols for data at rest and in transit, and stringent access controls to prevent unauthorized data access.

Phase-1

Technical Architecture (SPh-01, SPh-02, SPh-03)


Edge
Data Preprocesing for immediate decision in Edge
Cloud
Data Preprocesing in Cloud
Data Collection
Data Preprocessing,
AWS Kinesis Data Stream
AWS IoT Greengrass
AWS Outpost
AWS S3 Raw data
AWS Fargate Extract images
AWS S3 raw image data
AWS Glue Database
AWS S3 processed data
Run AWS Lambda function to check the Usability of these data. Call ResNet or DenseNet for any Blur or Glare detection
AWS S3 Cleaned data
Run AWS Lambda function to anonymise the required data to follow data privacy.
AWS S3 Anonymised data
Data Labelling: Active Learning or by 3rd party softwares
AWS S3 Labelled data
Run AWS Lambda function to get the cleaned image count satisfying specific criteria
If image count is sufficient ot train the model
No
Apply Synthetic Image generation or Augment the data
Yes
Data Selection: Manual or Active Learning
Train the model
Capture and improve the model evaluation metrics
Find cases in which the performance is low
Add those to the data unit testing
Deploy models to AV fleet in shadow mode to fetch similar edge cases
Retrieve cases from AV fleet.

1. Image data gets collected from the AV through AWS Greengrass.
2. Then these data gets ingested through AWS Outpost for local data processing.
3. In a streaming data pipeline for autonomous vehicles, **object detection** is typically performed on the **cloud** using AWS services,
4. **Cloud Processing** (for complex tasks):
    a. **AWS SageMaker** or **AWS Lambda**: For more resource-intensive models, raw image data is sent to the cloud. AWS Lambda functions or SageMaker inference endpoints perform object detection, tagging detected objects in metadata before storing or streaming it further.
        i. Then these data comes to AWS Cloud Infrastructure through Kinesis Video streams or Apache Kafka. Kinesis Video Streams extracts frames from the live video to an S3 bucket. Alternatively, a Lambda function extracts frames of the uploaded video clips and store those into PNG format in another AWS S3- raw data bucket with timestamp.
        ii. AWS EMR shall be used for
            1. Image preprocessing tasks such as resizing, normalizing, and filtering images.
            2. Data cleansing operations to remove noisy data, correct image labels, or eliminate corrupted files.
            3. Extracting metadata (e.g., timestamps, GPS coordinates, lighting conditions) associated with each image. This information is valuable for labelling and training
            4. Normalizing thousands of images to standardize pixel values, resolutions, and colour spaces for consistency in model training.

        5. Dimension reduction of high-resolution images to optimize storage and computational requirements using algorithms like PCA on Spark MLlib.
        6. Using EC2 Spot Instances to reduce costs for compute-heavy tasks, making it suitable for image data processing pipelines that may need significant computational resources.
        7. Allocating resources dynamically based on data volume and processing demand, providing flexibility for scaling up during peak processing times.
5. Perform Usability test on these image data, using either DenseNet or ResNet for any glare and blur detection.
6. Extract only usable data and store those into AWS S3- Usable data.
7. To query these data, we need to perform ETL and store those in AWS Glue. Detail database design is described in the following section.
8. Store the cleaned data in another S3-silver bucket. Complete process should be orchestrated by Airflow.
9. Next step is to label these images. This can be done in three ways
    a. AWS Rekognition is an open source service offered by AWS.
    b. Any third party annotation softwares such as Scale.ai or V7.ai.
    c. Through Active Learning (This will be discussed in next section)
10. Once images are labelled, corresponding metadata will be stored in any NoSQL DB and images will be stored in an aws S3 bucket.
11. Now, we need to check whether we have sufficient images available for model training or not.
12. We can use AWS Lambda function to check this condition.
13. In case, there are no sufficient images, then we need to augment the available images or need to create synthetic images. Detail synthetic image creation process is described in next section. (**Note**: If, Synthetic Image generation cannot be ready for first release, then we can use augmented images for this release, and can integrate Synthetic Images in next release.). The integration can be done by calling synthetic image model API endpoint exposed through FastAI, inside the AWS Lambda function.
14. Then using AWS Lambda and Amazon Rekognition, apply, data anonymisation principles to adhere to data privacy principles.
15. Next step is to prepare the dataset for model training. This can happen in two ways
    a. Manually: By building a data catalogue for data discovery, a data scientist can create dataset by searching with the criteria. For this, data should be stored in a Graph database with Knowledge Graph.
    b. There are different sampling techniques for image data selection
        i. A python code can be written to select the data without any class imbalance.
    c. Active Learning: Active Learning is a machine learning approach where the model intelligently selects the most informative data points from a large pool of unlabelled data for human annotation. Instead of passively learning from all available data, the model actively queries the data that it finds most challenging or uncertain, thereby improving its performance more efficiently.
16. Train models to detect new objects and track the following model evaluation metrics
    a. Evaluate performance
    b. Precision,
    c. Recall,
    d. F1 Score,
    e. Mean Average Precision (mAP) and
    f. IoU.
17. Find cases in which performance is low
18. Add those to the data unit test. Objective of the unit case is to check the performance of those failed cases are enough improved to get accepted.
19. Deploy models to car fleet in shadow mode to fetch similar edge cases
20. Retrieve cases from AV fleet.
21. Review and label collected data
22. Retrain models
23. Repeat the above steps.

Database Design

To store and query image data from autonomous vehicles in **AWS Glue** with images stored in **AWS S3**, here’s a schema design and sample queries.

1. Schema Design in AWS Glue

Define a Glue table with the following columns for structured storage of image metadata in Amazon S3:

- image_id (String): Unique identifier for each image.
- vehicle_id (String): Identifier for the vehicle that captured the image.
- timestamp (Timestamp): Date and time the image was captured.
- location (String): GPS coordinates or area name.
- day_night (String): ‘Day’ or ‘Night’ based on timestamp or lighting.
- objects_detected (Array of Strings): Objects identified in the image, e.g., ["car", "bicycle", "pedestrian"].
- file_path (String): S3 path to the actual image file.

2. Sample Glue Queries

To query images with cars and bicycles taken at night:

```
-- Retrieve all images containing cars and bicycles at night
SELECT image_id, file_path
FROM image_data
WHERE 'car' IN objects_detected
  AND 'bicycle' IN objects_detected
  AND day_night = 'Night';

-- Count the number of nighttime images with cars and bicycles
SELECT COUNT(*) AS car_bicycle_night_images
FROM image_data
WHERE 'car' IN objects_detected
  AND 'bicycle' IN objects_detected
  AND day_night = 'Night';
```

AWS Lambda function to check the image count

```
import boto3

# Initialize S3 client
s3 = boto3.client('s3')

# Constants
S3_BUCKET = 'your-image-bucket'
S3_PREFIX = 'images/'  # Folder path in S3

# Function to check image count
def check_image_count():
    response = s3.list_objects_v2(Bucket=S3_BUCKET, Prefix=S3_PREFIX)
    return response.get('KeyCount', 0)

def lambda_handler(event, context):
    image_count = check_image_count()
    print(f"Image count in S3 bucket: {image_count}")
```

Integration of Synthetic image generation model API endpoint

1. Create FastAI api endpoint

```
pip install fastapi uvicorn torch torchvision
# gan_api.py

from fastapi import FastAPI
import torch
from torchvision.utils import save_image
from io import BytesIO
from fastapi.responses import StreamingResponse

app = FastAPI()

# Load pre-trained GAN model here (replace with your model path)
model = torch.load("path_to_your_trained_gan_model.pth")
model.eval()  # Set to evaluation mode

@app.get("/generate-image")
async def generate_image():
    # Generate synthetic image
    noise = torch.randn(1, 100)  # Example: noise vector for GAN input
    with torch.no_grad():
        fake_image = model(noise)

    # Convert to an image response
    buffer = BytesIO()
    save_image(fake_image, buffer, format="JPEG")
    buffer.seek(0)
    return StreamingResponse(buffer, media_type="image/jpeg")

uvicorn gan_api:app --reload
```

2. API integration

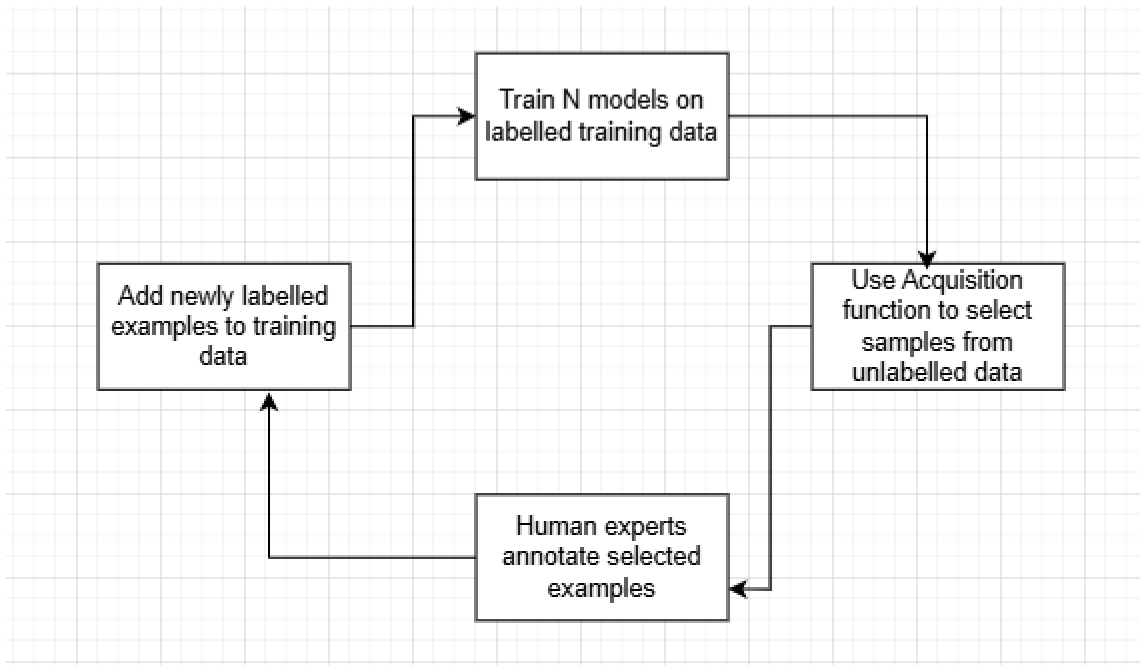
```
import boto3

# Initialize AWS clients
s3 = boto3.client('s3')
sagemaker = boto3.client('sagemaker')

# Constants
```

```
MIN_IMAGE_COUNT = 1000  # Define the minimum required images
s3_bucket = 'your-image-bucket'
s3_prefix = 'images/'

# Check image count in S3
def check_image_count():
    response = s3.list_objects_v2(Bucket=s3_bucket, Prefix=s3_prefix)
    return response.get('KeyCount', 0)

# Trigger synthetic image generation
def trigger_synthetic_generation():
    response = sagemaker.start_notebook_instance(NotebookInstanceName='synthetic-image-generator')
    return response

def lambda_handler(event, context):
    image_count = check_image_count()
    if image_count < MIN_IMAGE_COUNT:
        print(f"Only {image_count} images found. Generating synthetic images...")
        trigger_synthetic_generation()
    else:
        print("Sufficient images available.")
```

Active learning (needs to be explored more in depth)

This process will be used for both data selection and data labelling.

It is an ML approach that involves an iterative process of selecting and annotating the most informative data to train a model. Given a small set of labelled data and a large set of unlabelled data, active learning improves model performance, reduces labelling effort, and integrates human expertise for robust results, which has improved mean average precision around 3 to times than without active learning in a cost-effective manner.


Train N models on labelled training data
Use Acquisition function to select samples from unlabelled data
Human experts annotate selected examples
Add newly labelled examples to training data

- Training Pipeline: At first, an image labelling model is set up, trained on a small set of manually labelled data, and will be used in the labelling pipeline.
- Labelling Pipeline: Then, the labelling pipeline takes a small subset of unlabelled data from AWS S3 bucket and outputs annotated images with the cooperation of above image labelling model and human expertise.
- Then, the labelling pipeline and training pipeline can be iterated gradually with more labelled data to enhance the model's performance.
- In the labelling pipeline, an Amazon S3 Event Notification is invoked when a new batch of images comes into the Unlabelled Datastore S3 bucket, activating the labelling pipeline. The model produces the inference results on the new images. A customized judgement function selects parts of the data based on the inference confidence score or other user-defined functions. This data, with its inference results, is sent for a human labelling job on Amazon SageMaker Ground Truth created by the pipeline. The human labelling process helps annotate the data, and the modified results are combined with the remaining auto annotated data, which can be used later by the training pipeline.
- Model retraining happens in the training pipeline, where we use the dataset containing the human-labelled data to retrain the model. A manifest file is produced to describe where the files are stored, and the same initial model is retrained on the new data. After retraining, the new model replaces the initial model, and the next iteration of the active learning pipeline starts.

Synthetic Image Generation (needs to be explored more in depth)

Gather high-quality AV images (roads, vehicles, bicycles, etc.). Use a dataset like Waymo, BDD100K, or similar.

Preprocess the images (resize, align) and format them to fit StyleGAN2's input specifications.

Train the model and, generate the images.

Check the QC passes the generated image quality

Pass a sample of these images for manual Human Check.
If it satisfies QC check, then we can apply Active Learning for data selection to train the CV model.

Integrate with the current pipeline

Once the model is ready, we can integrate it with the existing pipeline to generate images, as explained above.

Deploy the model (SPh-04)

We can refer to the existing process flow for ML model deployment into edge.

1. Train an existing ML model and make it available in ONNX format
2. Create AWS IoT Greengrass components using the generated files in ONNX format
3. Deploy IoT Greengrass components to target edge devices.

Monitor model inference and model evaluation metrics (SPh-05)

Once the model is deployed into production, model prediction starts by classifying all the objects found in the pathway, with corresponding Object Detection mode. In this phase, following model evaluation metrics get continuously monitored in order to detection any degradation.

Performance Analysis

- Precision,
- Recall,
- F1 Score,
- Mean Average Precision (mAP)
- Intersection-over-union (IoU),
- Panoptic quality

The performance of these models get deteriorated with the progress of time due to various reasons. The process is called Drift in Machine Learning language. Drift refers to the phenomenon where computer vision models gradually become less effective over time as the environmental factors and target objects undergo changes.

Drifts
There are various kinds of drifts such as

- Data drift: change in statistical data distributional.
  - Image drift: Image data drift occurs when image properties change over time. For instance, certain images may have poor lighting, different background environments, different camera angles, etc.
  - Occlusion: Occlusion happens when another object blocks or hides an image's primary object of interest. It causes object detection models to classify objects wrongly and reduces model performance.
  - Lack of annotated samples: CV models often require labeled images for training. However, finding sufficient domain-specific images with correct labels is challenging.
  - Sensitive use cases: CV models usually operate in safety-critical applications like medical diagnosis and self-driving cars. Minor errors can lead to disastrous consequences.
- Model drift: drop in model performance due to drift.
- Conceptual Drift: the relationship between the target variable and input features changes.

Explainability

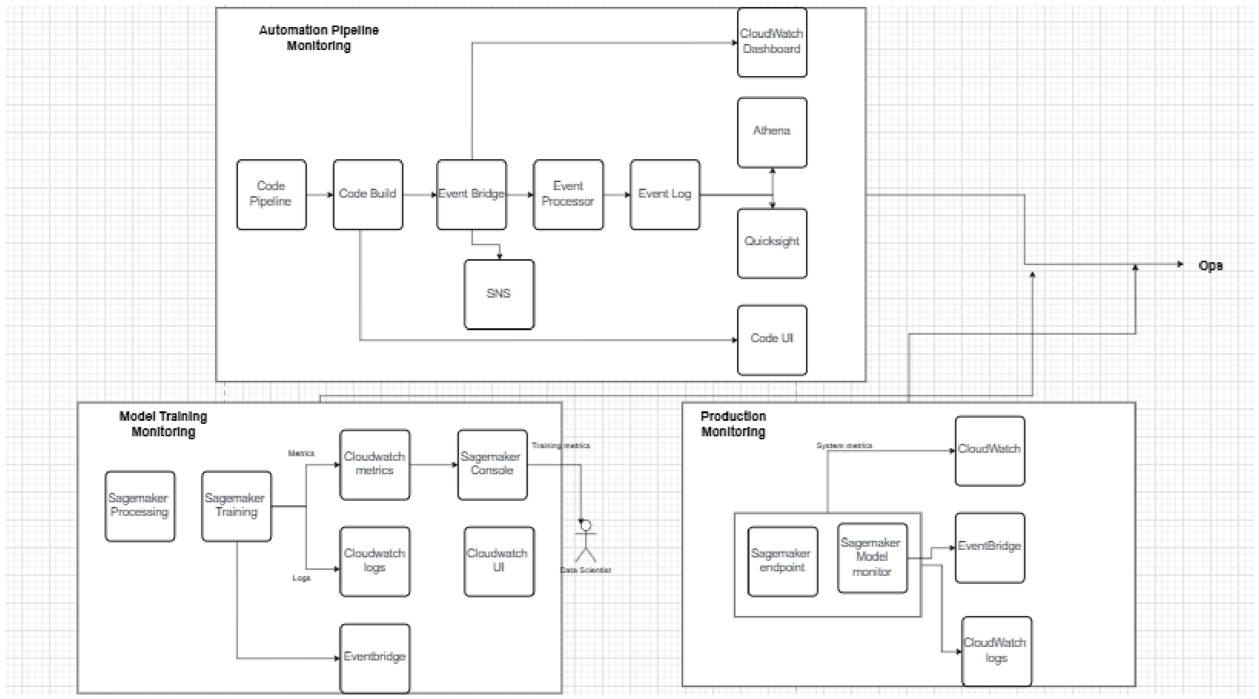
1. Integrated gradients.
2. XRAI

3. Grad-Cam

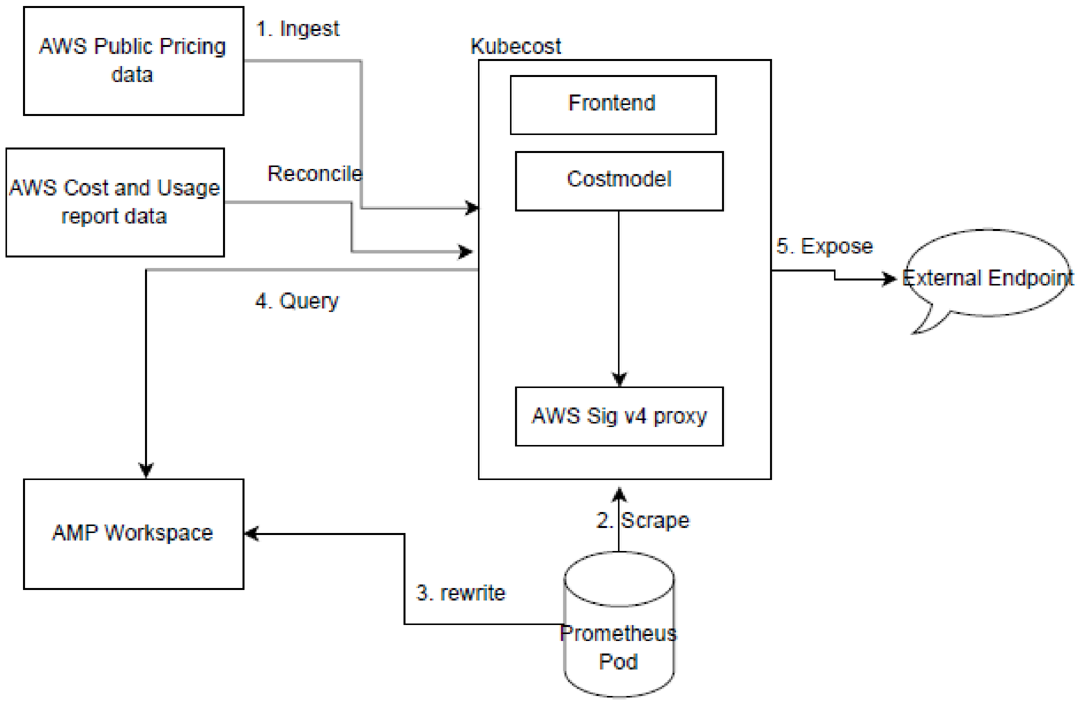
By the course of time, if the production data distribution gets changed from the trained data distribution, then the model should be retrained on the new production data in order to avoid the drift. Various statistical tests such as Kolmogorov-Smirnov (K-S) test, Population Stability Index and Page-Hinkley method can be applied to test these drifts. For instance, in autonomous vehicles the PSI can monitor the distributional changes in object categories between training and real-world driving datasets, ensuring the model's performance remains stable.

Model drift refers to the phenomenon where a machine learning model's performance deteriorates over time due to changes in the underlying data distribution. Correcting model drift involves updating the model to adapt to these changes and maintain accuracy.

Automation Pipeline, Model Training and Production Monitoring



Cost Management:



The core components of Kubecost are

1. Frontend

2. Prometheus

3. Costmodel

4. AWS Sig V4 proxy

When Kubecost is deployed in AWS EKS

1. Costmodel retrieves public pricing data from aws billing api.

2. Prometheus scrapes K8 cluster and cost- analyzer metrics

3. Prometheus re-writes the metrics into AMP workspace

4. Cost model queries the metrics from AMP through a sidecar container which using aws Sig v4 proxy to authenticate with AMP. It performs cost allocation calculations and exposes the metrics

5. Frontend routes requests to cost model to query cost allocation data then expose AWS EKS cluster cost and efficiency on K8 dashboard

.

AI Risk Mitigation Control (SPh-06)

The following is a snapshot of different stages of AI system development, system components in each phase, potential security risks for each component and corresponding mitigation control

1. Data Operations
   a. Raw data
      i. Insufficient access controls
      ii. Missing data classification
      iii. Poor data quality
      iv. Ineffective storage and encryption
      v. Lack of data versioning
      vi. Insufficient data lineage
      vii. Lack of data trustworthiness
      viii. Data legal
      ix. Stale data
      x. Lack of data access logs
   b. Data preparation
      i. Preprocessing integrity
         1. SSO with IdP and MFA to limit who can access your data and AI platform
         2. Sync users and groups to inherit your organizational roles to access data
         3. Restrict access using IP access lists to limit IP addresses that can authenticate to your data and AI platform
         4. Restrict access using private link as a strong control that limits the source for inbound requests
         5. Control access to data and other objects for permissions model across all data assets to protect data and sources
         6. Enforce data quality checks on batch and streaming datasets for data sanity checks and automatically detect anomalies before they make it to the datasets
         7. Capture and view data lineage to capture the lineage all the way to the original raw data sources
         8. Explore datasets and identify problems
         9. Source Code Control
         10. Secure model features to reduce the risk of malicious actors manipulating the features that feed into Model training
         11. Data-centric MLOps and LLMOps promote models as code
         12. Monitor audit logs
      ii. Feature manipulation
         1. SSO with IdP and MFA to limit who can access your data and AI platform
         2. Sync users and groups to inherit your organizational roles to access data
         3. Restrict access using IP access lists to limit IP addresses that can authenticate to your data and AI platform
         4. Restrict access using private link as a strong control that limits the source for inbound requests
         5. Secure model features to prevent and track unauthorized updates to features and for lineage or traceability
         6. Data-centric MLOps.
      iii. Raw data criteria
         1. SSO with IdP and MFA to limit who can access your data and AI platform
         2. Sync users and groups to inherit your organizational roles to access data
         3. Restrict access using IP access lists to restrict the IP addresses that can authenticate to Databricks
         4. Restrict access using private link as strong controls that limit the source for inbound requests

5. Use access control lists to control access to data, data streams and notebooks
6. Data-centric MLOps for unit and integration testing.

iv. Adversarial partitions

1. SSO with IdP and MFA to limit who can access your data and AI platform
2. Sync users and groups to inherit your organizational roles to access data
3. Restrict access using IP access lists to restrict the IP addresses that can authenticate to Databricks
4. Restrict access using private link as strong controls that limit the source for inbound requests
5. Track and reproduce the training data used for ML model training to track and reproduce the training data partitions and the human owner accountable for ML model training, as well as identify ML models and runs derived from a particular dataset
6. Data-centric MLOps for unit and integration testing

c. Datasets

i. Data poisoning

1. SSO with IdP and MFA to limit who can access your data and AI platform
2. Sync users and groups to inherit your organizational roles to access data
3. Restrict access using IP access lists to restrict the IP addresses that can authenticate to your data and AI platform
4. Restrict access using private link as strong controls that limit the source for inbound requests
5. Control access to data and other objects for permissions model across all data assets to protect data and sources
6. Enforce data quality checks on batch and streaming datasets for data sanity checks, and automatically detect anomalies before they make it to the datasets
7. Capture and view data lineage to capture the lineage all the way to the original raw data sources
8. Secure model features
9. Track and reproduce the training data used for ML model training and identify ML models and runs derived from a particular dataset
10. Share data and AI assets securely
11. Audit actions performed on datasets
12. Monitor audit logs

ii. Ineffective storage and encryption

1. Encrypt data at rest
2. Encrypt data in transit
3. Control access to data and other

iii. Label flipping

1. Encrypt data at rest
2. Encrypt data in transit
3. Control access to data and other objects for metadata encryption across all data assets

d. Catalog and governance

1. Lack of traceability and transparency of model assets
2. Control access to data and other objects for permissions model across all data assets to protect data and sources
3. Enforce data quality checks on batch and streaming datasets for data sanity checks, and automatically detection anomalies before they make it to the datasets
4. Capture and view data lineage to capture the lineage all the way to the original raw data sources
5. Secure model features

            6. Track and reproduce the training data used for ML model training and identify ML models and runs derived from a particular dataset
            7. Govern model assets for traceability
            8. Monitor audit logs
        ii. Lack of end-to-end ML lifecycle
            1. Manage end-to-end machine learning lifecycle for measuring, versioning, tracking model artifacts, metrics and results
            2. Data-centric MLOps unit and integration testing
            3. Monitor data and AI system from a single pane of glass

2. Model Operations
    a. ML algorithm
        i. Lack of tracking and reproducibility of experiments
            1. Track ML training runs for documenting, measuring, versioning, tracking model artifacts including algorithms, training environment, hyperparameters, metrics and results
            2. Data-centric MLOps promote models as code and automate ML tasks for cross-environment reproducibility
            3. Monitor audit logs
        ii. Model drift
            1. Track training data with MLflow and Delta Lake to track upstream data changes
            2. Secure model features to track changes to features
            3. Monitor data and AI system from a single pane of glass for changes and take action when changes occur. Have a feedback loop from a monitoring system and refresh models over time to help avoid model staleness.
        iii. Hyperparameters stealing
            1. Track ML training runs in the model development process, including parameter settings securely
            2. Use access control lists via workspace access controls
            3. Data-centric MLOps employing separate model lifecycle stages by UC schema
        iv. Malicious libraries
            1. Third-party library control to limit the potential for malicious third-party libraries and code to be used on mission-critical workloads.
    b. Evaluation
        i. Evaluation data poisoning
            1. SSO with IdP and MFA to limit who can access your data and AI platform
            2. Sync users and groups to inherit your organizational roles to access data
            3. Restrict access using IP access lists to restrict the IP addresses that can authenticate to your data and AI platform
            4. Restrict access using private link as strong controls that limit the source for inbound requests
            5. Control access to data and other objects for permissions model across all data assets to protect data and sources
            6. Enforce data quality checks on batch and streaming datasets for data sanity checks, and automatically detect anomalies before they make it to the datasets
            7. Capture and view data lineage to capture the lineage all the way to the original raw data sources
            8. Evaluate models to capture performance insights for language models
            9. Trigger actions in response to a specific event via automated jobs to notify human-in-the-loop (HITL)
            10. Data-centric MLOps unit and integration testing
        ii. Insufficient evaluation data

        1. Build models with all representative, accurate and relevant data sources to evaluate on clean and sufficient data

c. Model build
    i. Backdoor machine learning/Trojaned model
        1. SSO with IdP and MFA to limit who can access your data and AI platform
        2. Use access control lists to limit who can bring models and limit the use of public models
        3. Data-centric MLOps promote models as code using CI/CD. Scan third-party models continuously to identify hidden cybersecurity risks and threats such as malware, vulnerabilities and integrity issues to detect possible signs of malicious activity, including malware, tampering and backdoors. See resources section for third-party tools.
        4. Register, version, approve, promote and deploy models and scan models for malicious code when using thirdparty models or libraries
        5. Manage end-to-end machine learning lifecycle
        6. Control access to data and other objects
        7. Run models in multiple layers of isolation. Models are considered untrusted code: deploy models and custom
        8. Restrict outbound connections from models to prevent attacks to exfiltrate data, inference requests and responses
        9. Monitor audit logs
    ii. Model assets leak
        1. Control access to models and model assets
        2. SSO with IdP and MFA to limit who can access your data and AI platform
        3. Sync users and groups to inherit your organizational roles to access data
        4. Restrict access using IP access lists that can authenticate to your data and AI platform
        5. Restrict access using private link as strong controls that limit the source for inbound requests
        6. Control access to data and other objects for permissions model across all data assets to protect data and sources
        7. Data-centric MLOps to maintain separate model lifecycle stages
        8. Manage credentials securely to prevent credentials of data sources used for model training from leaking through models
        9. Monitor audit logs
    iii. ML supply chain vulnerabilities
        1. Build models with all representative, accurate and relevant data sources to minimize third-party dependencies for models and data where possible
        2. Pretrain a large language model (LLM) on your own IP
        3. Use hardened runtime for machine learning
        4. Third-party library control
        5. Data-centric MLOps promote models as code using CI/CD. Scan third-party models continuously to identify hidden cybersecurity risks and threats such as malware, vulnerabilities and integrity issues to detect possible signs of malicious activity, including malware, tampering and backdoors. See resources section for third-party tools.
        6. Evaluate models and validate (aka, stress testing) to verify reported function and disclosed weaknesses in the models
        7. Restrict outbound connections from models to prevent attacks to exfiltrate data, inference requests and responses
    iv. Source code control attack
        1. Source code control to control and audit your knowledge object integrity
        2. Third-party library control for third-party library integrity
        3. Restrict outbound connections from models to prevent attacks to exfiltrate data, inference requests and responses

d. Model management

        i. Model attribution
            1. Control access to data and other objects for permissions model across all data assets to protect data and sources
            2. Create model aliases, tags and annotations for documenting and discovering models
            3. Build MLOps workflows with human-in-the-loop (HITL) , model stage management and approvals
            4. Share data and AI assets securely
        ii. Model theft
            1. SSO with IdP and MFA to limit who can access your data and AI platform
            2. Sync users and groups to inherit your organizational roles to access data
            3. Restrict access using IP access lists that can authenticate to your data and AI platform
            4. Restrict access using private link as strong controls that limit the source for inbound requests
            5. Control access to data and other objects for permissions model across all data assets to protect data and sources
            6. Control access to models and model assets
            7. Encrypt models
            8. Secure model serving endpoints to prevent access and compute theft
            9. Share data and AI assets securely
            10. Streamline the usage and management of rate-limit APIs
            11. Manage credentials securely to prevent credentials of data sources used for model training from leaking through models
            12. Use clean rooms to collaborate in a secure environment
            13. Monitor audit logs
        iii. Model lifecycle without HITL
            1. Control access to data and other objects for permissions model across all data assets to protect data and sources
            2. Control access to models and model assets
            3. Create model aliases, tags and annotations
            4. Build MLOps workflows with human-in-the-loop (HILP) with permissions, versions and approvals to promote models to production
            5. Data-centric MLOps promote models as code using CI/CD
        iv. Model inversion
            1. SSO with IdP and MFA to limit who can access your data and AI platform
            2. Sync users and groups to inherit your organizationals role to access data
            3. Restrict access using IP access lists that can authenticate to your data and AI platform
            4. Restrict access using private link as strong controls that limit the source for inbound requests
            5. Control access to data and other objects for permissions model across all data assets to protect data and sources
            6. Control access to models and model assets
            7. Encrypt models
            8. Secure model serving endpoints
            9. Monitor audit logs
3. Model Deployment & Serving
    a. Model Serving inference requests
        i. Model inversion
            1. SSO with IdP and MFA to limit who can access your data and AI platform
            2. Sync users and groups to inherit your organizational roles to access data
            3. Restrict access using IP access lists that can authenticate to your data and AI platform
            4. Restrict access using private link as strong controls that limit the source for inbound requests

            5. Control access to data and other objects for permissions model across all data assets to protect data and sources
            6. Control access to models and model assets
            7. Encrypt models
            8. Secure model serving endpoints
        ii. Denial of service (DOS)
            1. SSO with IdP and MFA to limit who can access your data and AI platform
            2. Sync users and groups to inherit your organizational roles to access data
            3. Restrict access using IP access lists that can authenticate to your data and AI platform
            4. Restrict access using private link as strong controls that limit the source for inbound requests
            5. Control access to data and other objects for permissions model across all data assets to protect data and sources
            6. Control access to models and model assets
            7. Store and retrieve embeddings securely to integrate data objects for security-sensitive data that goes into
    b. Model Serving inference responses
        i. Lack of audit and monitoring inference quality
            1. Track model performance to evaluate quality
            2. Set up monitoring alerts
            3. Set up inference tables for monitoring and debugging models to capture incoming requests and outgoing responses to your model serving endpoint and log them in a table. Afterward, you can use the data in this table to monitor, debug and improve ML models and decide if these inferences are of quality to use as input to model training.
            4. Monitor audit logs
        ii. Output manipulation
            1. Encrypt models for model endpoints with encryption in transit
            2. Secure model serving endpoints
        iii. Black-box attacks
            1. Encrypt models for model endpoints with encryption in transit
            2. Secure model serving endpoints
4. Operations and Platforms
    a. ML operations
        i. Lack of MLOps — repeatable enforced standards
            1. Evaluate models to capture performance insights for language models
            2. Trigger actions in response to a specific event to trigger automated jobs to keep human-in-the-loop (HITL)
    b. ML platform
        i. Lack of vulnerability management
            1. Platform security — vulnerability management to build, deploy and monitor AI/ML models on a platform that takes responsibility seriously and shares remediation timeline commitments
        ii. Lack of penetration testing and bug bounty
            1. Platform security — penetration testing and bug bounty to build, deploy and monitor AI/ML models on a platform that takes responsibility seriously and shares remediation timeline commitments. A bug bounty program removes a barrier researchers face in working with Databricks.
        iii. Lack of incident response
        iv. Unauthorized privileged access
        v. Poor SDLC
        vi. Lack of compliance

Non-Functional Requirements(SPh-08)

1. Data Engineering
   a. Data Ingestion Latency: < 100 ms per image.
   b. Storage Reliability: 99.999% availability for cloud data storage.
   c. Data Throughput: > 500 images per second.
2. Model Training
   a. Training Time: < 2 hours per model iteration.
   b. Resource Utilization: 85% GPU efficiency.
   c. Model Accuracy (Baseline): ≥ 90% on test datasets.
3. Model Building and Optimization
   a. Inference Latency: < 50 ms per image.
   b. Model Size: ≤ 100 MB for deployment efficiency.
   c. Energy Consumption: ≤ 15 Watts on edge devices.
4. Deployment
   a. Deployment Time: < 10 minutes per model version.
   b. Scalability: Capable of 10,000 simultaneous edge deployments.
   c. Rollback Time: < 5 minutes to revert to previous version.
5. Monitoring
   a. Uptime: 99.9% availability of the monitoring system.
   b. Error Detection Latency: < 5 seconds to detect critical errors.
   c. Data Accuracy Drift Detection: Identify 5% drift within 1 hour.

Roadmap

https://docs.google.com/spreadsheets/d/10f6aHshluHjAPn0Q9lad2QRwPUFCfFMjaWJXivurrHM/edit?gid=0#gid=0

Technology Assessment

https://docs.google.com/spreadsheets/d/10f6aHshluHjAPn0Q9lad2QRwPUFCfFMjaWJXivurrHM/edit?gid=33273134#gid=33273134

Project Management

https://docs.google.com/spreadsheets/d/10f6aHshluHjAPn0Q9lad2QRwPUFCfFMjaWJXivurrHM/edit?gid=1026204855#gid=1026204855

Framework

We can introduce Dynamic System Development Methodology in our project, if Bosch does not follow Scaled Agile Framework.

But a major challenge always comes in requirement finalisation phase. By following frequent stakeholder collaboration, required POC and requirement prioritisation techniques, this challenge can be sufficed, and the requirements shall be frozen at least 3 weeks prior to the starting of PI, so that engineering team will get enough time to brainstorm the scope of next PI.

A small formula which may help in addressing the challenge of technology selection and release deadlines,

1. POC for technology assessment should be done prior to one PI of actual planned implementation PI and should be approved by the approver with required InfoSec certificates. For instance, if implementation of data pipeline is planned for PI-2, then the technology stack should be approved within the end of PI-1 along with the design doc.
2. In next PI, means in PI-2, the data pipeline implementation should start.
3. Each feature must be clearly defined with Acceptance Criteria, OKR, Timelines.

4. Give enough privilege to Engg team to come up with effort estimation and any technical dependencies\risk which are not identified.
5. Provide required trainings and supporting documents.
6. Most Important: Collaborate and communicate continuously to identify any challenges the team is facing.
7. Continuously follow up and monitor.

This framework has proved to be succeeded.

Phase- II (Vision after 1 year)

Minor enhancements + Bug Fixes + Integration with Active Learning and Synthetic Image generation endpoints, if not integrated in Phase- I. Integration approaches are provided in respective sections.

Phase- III (Vision after 2 years)

This architecture is proposed based on LAECIPS, a Large Vision Model Assisted Adaptive Edge-Cloud Collaboration framework, developed to improve real-time perception in IoT applications, such as autonomous driving, by combining the strengths of edge devices and cloud resources. By blending **edge and cloud models** and continuously refining the edge model with cloud feedback, LAECIPS enables **scalable, real-time, and accurate IoT-based perception**.

Ref: https://arxiv.org/pdf/2404.10498

Limitation of existing edge-cloud deployment model

1. The tight coupling between the large and small models limits the system flexibility of the current methods for fully leveraging large vision models.
2. The collaboration strategy needs to be further optimized for both high accuracy and low latency, while demonstrating its capability to adapt to the dynamic IoT environment.
3. Inference outputs from a large vision model (e.g., SAM) may lack semantic labels and thus need to be combined with the edge model inference results.

Features

1. It enables flexible utilization of both large and small models in an online manner to solve this problem.
2. We can design a hard input mining-based edge-cloud co-inference strategy that achieves higher accuracy and lower task processing latency.
3. The technique enhances model robustness, as the model progressively learns from edge cases, becoming better at handling diverse or ambiguous data.
4. A continual training of the small model to fit in with the dynamic environmental changes in the IoT environment.
5. We can analyze the theoretical generalization capability of LAECIPS to prove the feasibility of incorporating large vision models, edge small models, and edge-cloud coinference strategies into this framework in a plug-and-play manner.

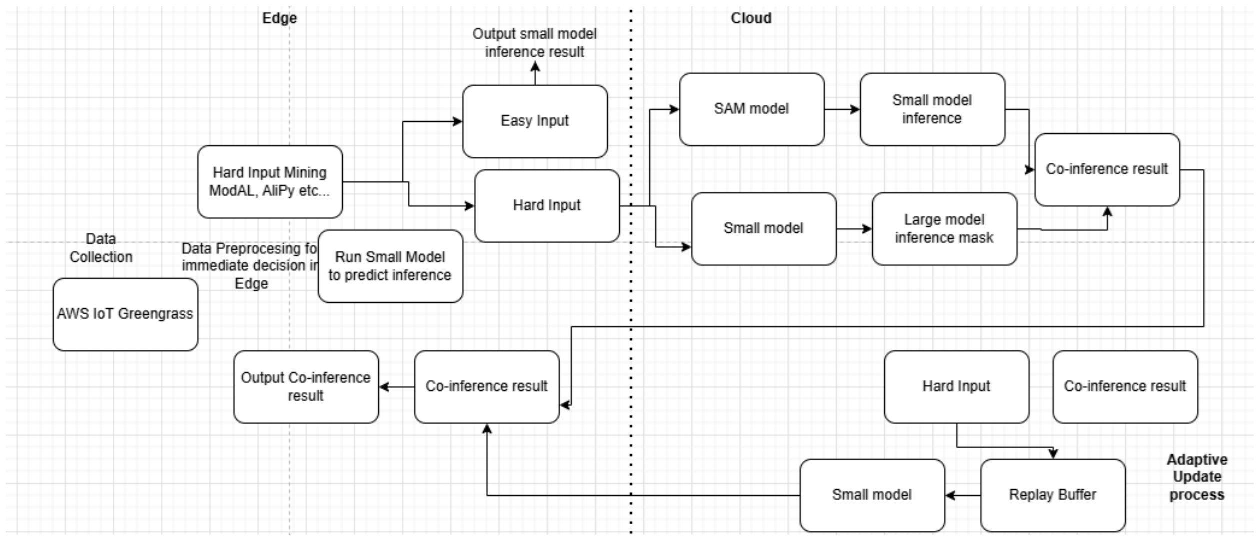
Aspect	LVMs (Large Vision Models)	CNNs (Convolutional Neural Networks)
Performance	Higher accuracy and generalization but slower	Competitive performance, faster inference, task-specific

Computational Cost	High GPU/TPU requirements, costly at scale	More efficient, can run on mobile/edge devices
Scalability	Highly scalable with vast data and new tasks	Scales well on fixed-size tasks, less generalizable
Deployment Complexity	Higher infrastructure cost, complex to deploy	Mature tooling, easier optimization, and deployment
Latency	Slower inference needs optimization	Typically faster, suited to real-time applications
Interpretability	Harder to interpret, post-hoc analysis	More interpretable via feature/activation maps
Cost	Higher deployment and compute costs	Generally cheaper in both training and inference
Use Case Fit	Best for multimodal, few-shot, large-scale tasks	Suitable for specific, well-defined image tasks

We will consider the example of SAM, how it can improve our AV system. is a versatile vision model primarily designed for **image segmentation** tasks, where the goal is to identify and delineate specific objects or regions within an image.

Features of SAM: Object Segmentation, Instance Segmentation,Interactive Segmentation, Semantic Segmentation, Zero-Shot Segmentation, Part Segmentation,Video Frame Segmentation (with modification), Foreground-Background Segmentation, Complex Scene Segmentation, Fine-Grained and Detailed Segmentation.

Workflow of LAECIPS



1. In steps 1 and 2, a lightweight model performs inference on incoming data to produce preliminary results.

2. In steps 3 and 4, the hard input mining module classifies this data into "easy" and "hard" inputs based on accuracy.
3. Step 5 outputs results for easy inputs directly, reducing latency, while hard inputs are sent to the cloud for enhanced accuracy.
4. In steps 6 and 7, the small and large models jointly process hard inputs, and their results are fused for co-inference.
5. In steps 8 and 9, co-inference results are returned to the edge and saved in the cloud's replay buffer. Once the buffer reaches a set threshold, the cloud retrains the edge model using this data, and
6. In step 10, the updated model is deployed back to the edge.

In this phase, ML skilled resources are more required as compared to Data Engineer, as data pipeline would be stable by this time.

References
https://gitlab.com/ashishkaul/shared-pipeline-library
https://www.surfing.ai/blog/self-driving-car-data.html#:~:text=Types%20of%20data%20used%20in%20self-driving%20cars%3A%20Sensors%3A,Weather%20and%20traffic%20conditions%20inform%20adaptive%20driving%20decisions.
https://blog.roboflow.com/what-is-object-tracking-computer-vision/#:~:text=Object%20detection%20and%20tracking%20are%20both%20computer%20vision,and%2C%20follow%20objects%20over%20frames%20in%20a%20video.
https://www.quora.com/Should-a-machine-learning-model-be-retrained-each-time-new-observations-are-available
https://www.kai-waehner.de/blog/2020/08/07/apache-kafka-handling-large-messages-and-files-for-image-video-audio-processing/
https://ieeexplore.ieee.org/document/10518077