

Project: Binsense AI

Automated Bin Item Verification System using Computer Vision and Machine Learning

Project Overview

This project aims to develop a highly accurate and fast computer vision model by using the Amazon Bin Image Dataset (ABID) containing images and metadata to verify if the items with their respective quantities are present in the image of the bin.

Motivation :

The e-commerce industry is growing unprecedentedly, leading to an increased demand for efficient inventory management and order fulfillment processes. One critical aspect of this process is accurately identifying items within bins or baskets, determining their quantities, estimating packaging sizes, and updating inventory availability in real time. Manual identification of items, their size estimation and feeding this information into the inventory management system are time-consuming; leading to suboptimal packaging and inefficient inventory management. The rapid growth of e-commerce demands innovative solutions for optimizing logistics, reducing packaging waste, and enhancing operational efficiency and customer satisfaction. Automating the item and its quantity validation, packaging size estimation and simultaneously updating inventory availability can significantly improve operational efficiency. This project aims to do this automation by leveraging computer vision technology. The objective is to develop a solution with intrinsic features like simple interactive UI with a very well-integrated computer vision model.

Project Scope

☐ **Objective:**

- To create a computer vision model that can accurately verify the presence and quantity of items in a bin image.
- To create a simple and intuitive UI that allows users to interact with the model. The UI should allow users to select items and quantities, display the most appropriate image from the dataset and show the model's verification results.
- The goal is to thoroughly evaluate the performance of the computer vision model.
- The project should include a comprehensive report detailing the model's performance and results.

☐ Deliverables:

- Trained machine learning model
- Web-based interface to allow users to select items and quantities, display the most appropriate image from the dataset and show the model's verification results.
- Evaluation Metrics
- Documentation and report on the project's progress and results

EDA - Data Collection and Preprocessing

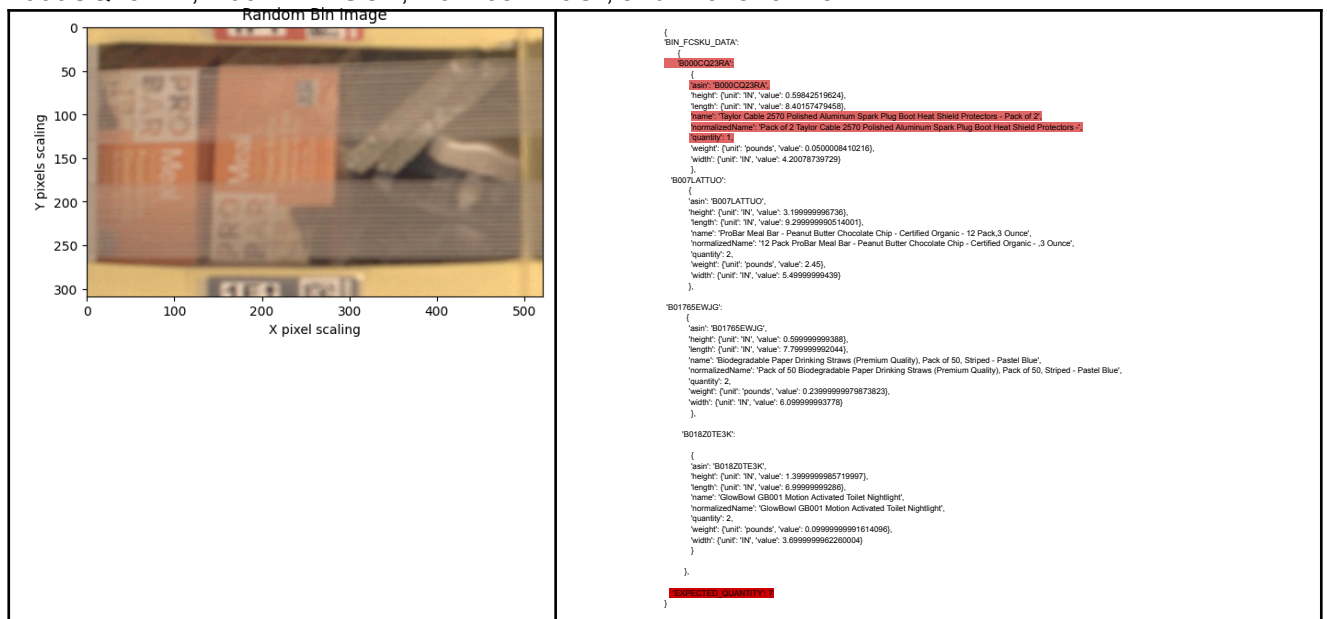
Dataset : The Amazon Bin Image Dataset contains over 535K images and metadata from bins of a pod in an operating Amazon Fulfillment Center. The bin images in this dataset are captured as robot units carrying pods as part of normal Amazon Fulfillment Center operations. The details of the dataset is available in <https://registry.opendata.aws/amazon-bin-imagery/>

Images : The dataset features various images of bins containing multiple object categories and a varying number of instances. Each bin image has corresponding metadata, including the object category identification (Amazon Standard Identification Number, ASIN), quantity, size, weight, and other details. The bin sizes differ depending on the object sizes they contain. Note that the tapes in front of the bins are used to prevent items from falling out, but may occasionally obscure the objects. Additionally, objects may be heavily occluded by other objects or have limited visibility due to the image's viewpoint. Below are some examples of bin images that were downloaded from the dataset -



Metadata : Here is an example of a metadata file . Each file contains information about the items present in the bin. The corresponding metadata for each bin image includes the item identification (Amazon Standard Identification Number, ASIN), quantity, height, length, width, weights. This is an example of image(jpg) and metadata(json) pair for 103372.jpg. This image contains 4 different object categories. For each category, there is one instance. So, "EXPECTED_QUANTITY" is 7, and for three items the "quantity" field was 2 and one was 1.

Unique identifier("asin") is assigned to each object category, e.g. here "B000CQ23RA", "B007LATTUO", "B01765EWJG", and "B018Z0TE3K".



Below dataset was used for this project :

https://docs.google.com/spreadsheets/d/1rZfFrHEbfX_b-3ofEIDxLQUFNDWtmDarXBrU4nn4Luw/edit#gid=601918728

The dataset is a list of images and metadata file names without .jpg and .json extension from the AWS S3 bucket which you should download and use for this project.

Dataset Statistics :

Statistics!

Total Images: 3875

Average expected quantity in a bin: $17827/3875$: 4.6005161290322585

The number of instances: 5280

Description	Total	Train	Validation
The number of images	3875	3487	388
Average quantity in a bin	4.60	4.60	4.52
The number of object categories	5280	5003	938

Figure 1 : Figure 1 shows the distribution of object repetition. Approximately 2500 object categories (out of 5280) showed up only once across the entire dataset, and roughly

1800 object categories showed up twice.

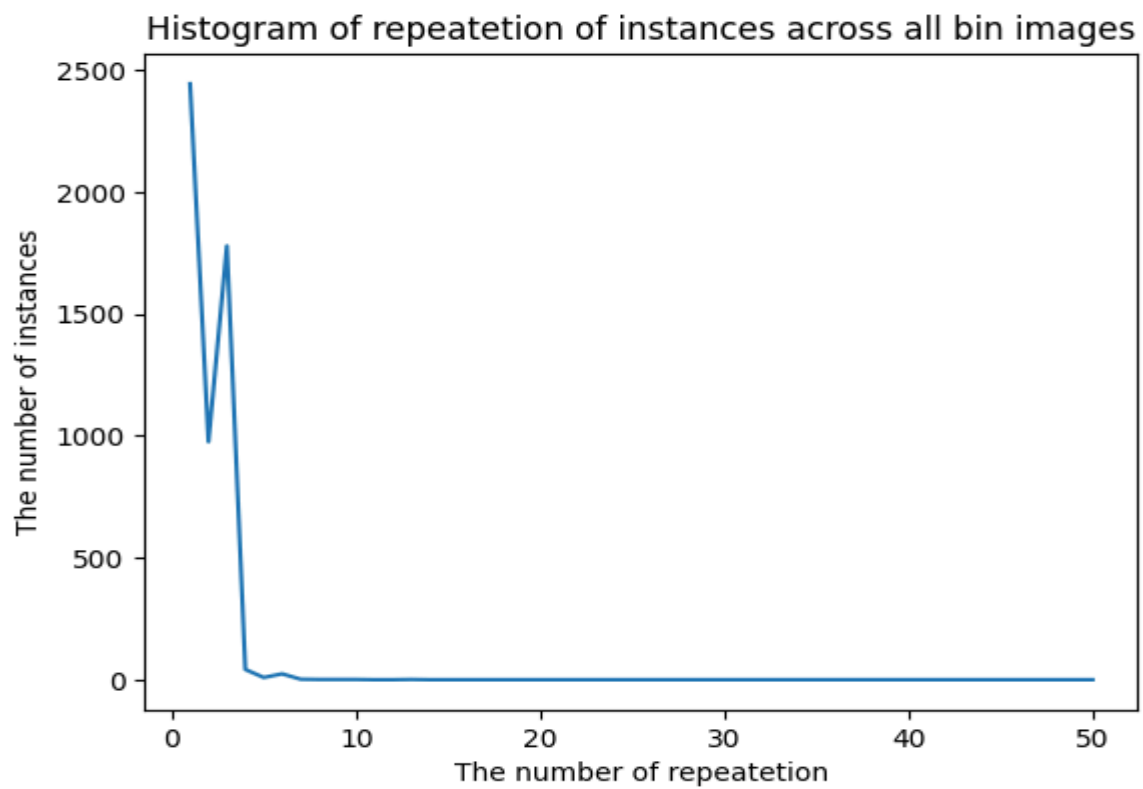
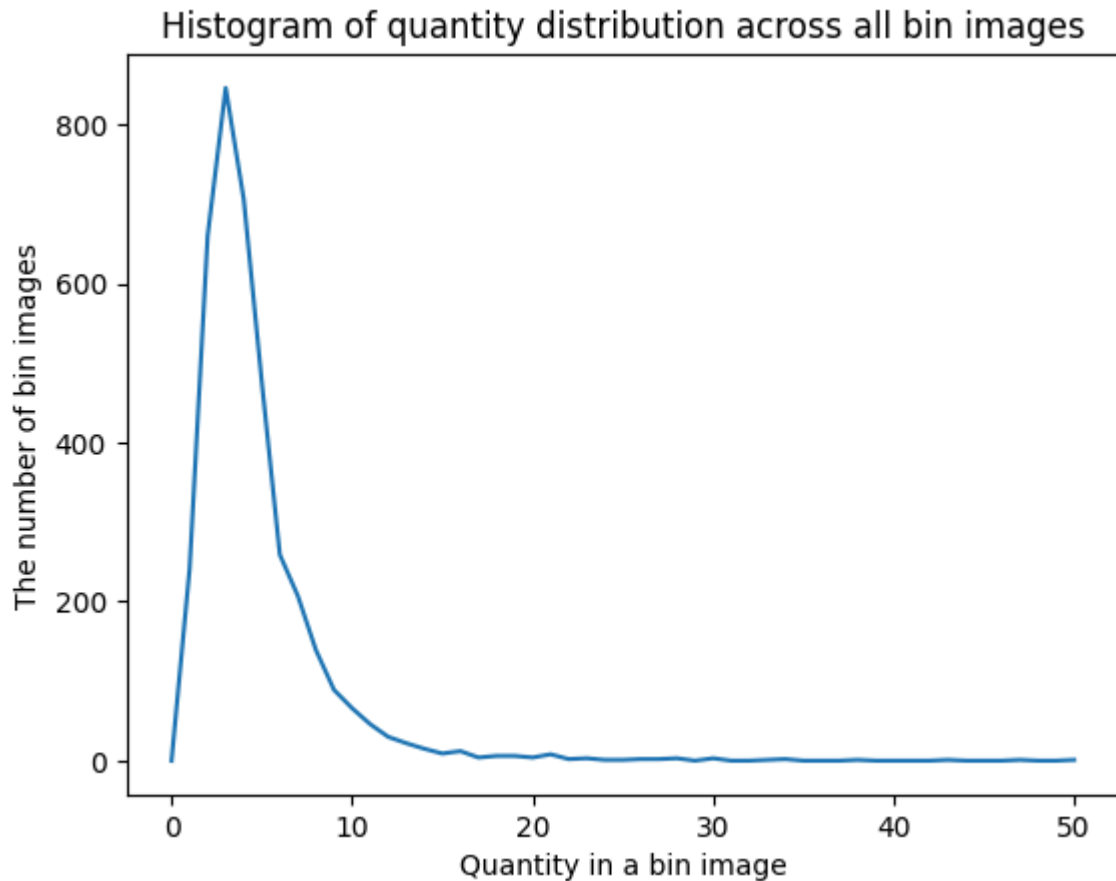


Figure 2: Figure 2 shows the distribution of quantity in a bin(90% of bin images contain less than 10 object instances in a bin). The average expected quantity in a Bin is 5 .



Data preparation

- Downloading data and preprocessing

Below two files are used to download and preprocess the image and metadata files -

```
${project_root_dir}/src/binsenseai/pipeline> python3 stage_01_data_ingestion.py
```

```
${project_root_dir}/src/binsenseai/pipeline> python3 stage_01_data_ingestion.py
```

It performs below activities :

1. Read the downloaded excel file data using the function **read_excel_to_list()**
2. Download the image and metadata files from Amazon Bin Image Dataset and placed it in below location by the function **s3_read_write()** -

`{project_root_dir}/binsenseai/artifacts/data_ingestion/`

- `amazon_bin_images`
- `amazon_bin_metadata`

3. Generate below two files using function **make_metadata()** :

file location - {project_root_dir}/artifacts/data_transformation

- metadata.json - This file contains all metadata in a single file
- instances.json - This file contains a list of all categories and image indices that contain the object.

They will be useful for investigating metadata of the datasets and used to generate task specific metadata files.

4. The training and validation sets are prepared by using function **split_train_val_data()** which will randomly split the files and generate 'random_train.txt' and 'random_val.txt' files.

file location - {project_root_dir}/artifacts/data_transformation

5. There are two computer vision models trained on the datasets. One is to verify if the object is present in the image and another is to verify object's presence and its quantity

For Object verification task , we have used function **make_obj_verification_data()** to generate below metadata files to be used during training -

location - {project_root_dir}/artifacts/data_transformation

- obj_verification_train.json
- obj_verification_val.json

For Object verify and count quantity tas,, we have used function **make_obj_num_verification_data()** to generate below metadata files to be used during training -

location - {project_root_dir}/artifacts/data_transformation

- obj_num_verification_train.json
- obj_num_verification_val.json

Code Snippet :

```

src > binsenseai > pipeline > stage_01_data_transformation.py > ...
1  from src.binsenseai.config.configuration import ConfigurationManager
2  from src.binsenseai.utils.load_data_S3 import DataReadAndTransformation
3  from src.binsenseai import logger
4
5
6
7  STAGE_NAME = "Data Ingestion stage"
8
9  class DataReadingAndTransformationPipeline:
10     def __init__(self):
11         pass
12
13     def main(self):
14         config = ConfigurationManager()
15         data_ingestion_config = config.get_data_transformation_config()
16         data_ingestion = DataReadAndTransformation(config=data_ingestion_config)
17         data_list = data_ingestion.read_excel_to_list("artifacts/data_ingestion/binsimages.xlsx", "Sheet1") #binsimages
18         data_ingestion.convert_to_string_with_zeros(5) # 5 number width
19         data_ingestion.s3_read_write(data_list) # 5 number width
20         data_ingestion.make_metadata('artifacts/data_ingestion/amazon_bin_images', 'artifacts/data_ingestion/amazon_bin_metadata') # 5 number width
21         data_ingestion.split_train_val_data('artifacts/data_ingestion/amazon_bin_images', 'artifacts/data_ingestion/amazon_bin_metadata')
22         data_ingestion.make_obj_num_verification_data('artifacts/data_transformation/random_train.txt',
23                                                     'artifacts/data_transformation/random_val.txt',
24                                                     'artifacts/data_transformation/metadata.json',
25                                                     'artifacts/data_transformation/instances.json')
26         data_ingestion.make_obj_verification_data('artifacts/data_transformation/random_train.txt',
27                                                  'artifacts/data_transformation/random_val.txt',
28                                                  'artifacts/data_transformation/metadata.json',
29                                                  'artifacts/data_transformation/instances.json')
30
31
32
33
34
35
36 if __name__ == '__main__':
37     try:
38         logger.info(f">>>>> stage {STAGE_NAME} started <<<<<")
39         obj = DataReadingAndTransformationPipeline()
40         obj.main()
41         logger.info(f">>>>> stage {STAGE_NAME} completed <<<<<\n\nx=====x")
42     except Exception as e:
43         logger.exception(e)
44         raise e

```

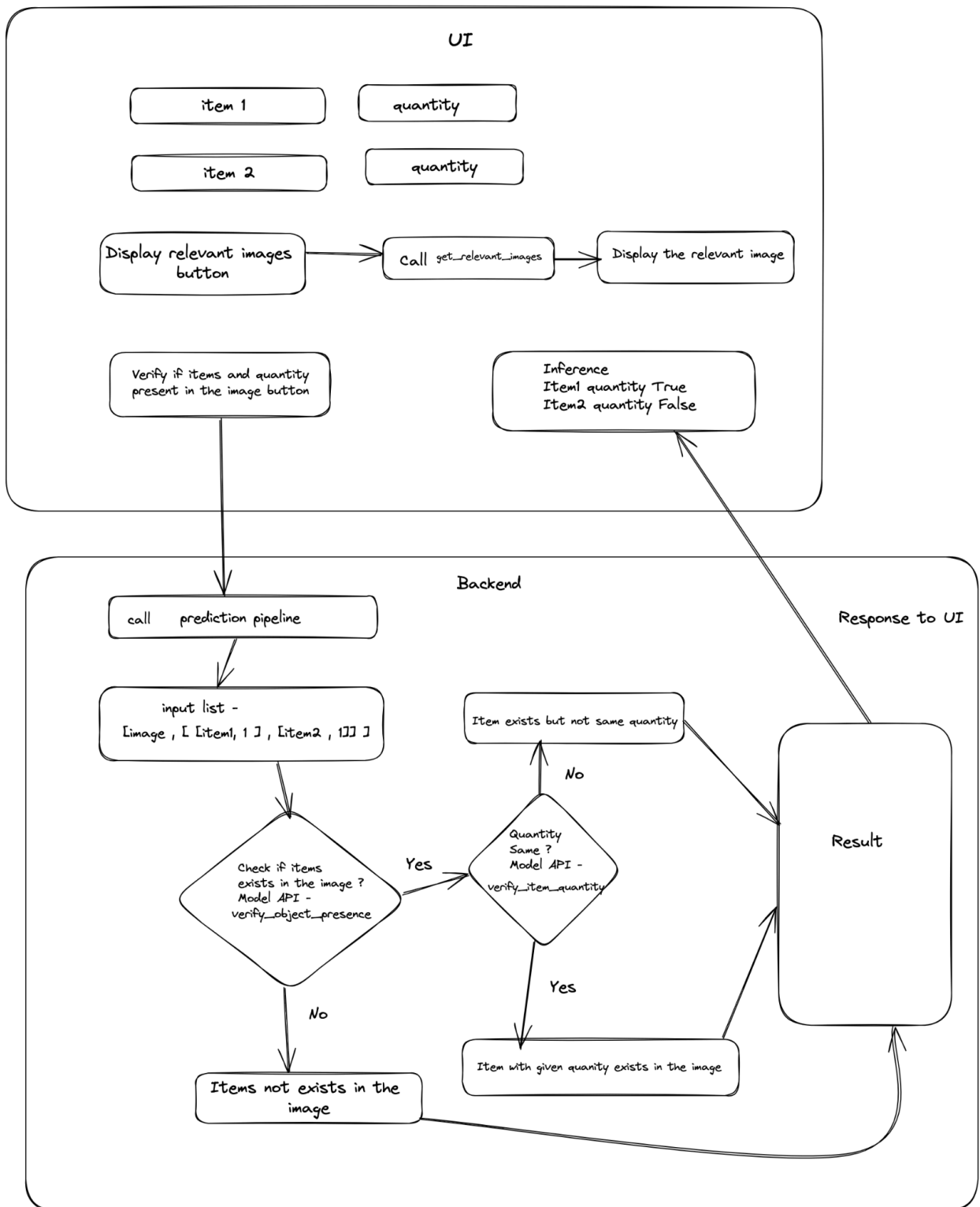
Resizing images

We resize all images into 224x224 for convenient training purposes. The resized images are placed in below location

{project_root_dir}/binsenseai/artifacts/data_ingestion/bin-images-resize

Architecture :

- App Workflow:



- **CV Model:**

Siamese Network with ResNet34 Backbone for Object Verification: This architecture is chosen mainly because the number of object categories is huge. So we used siamese network ResNet34 Backbone to learn how to compare the images instead of modelling all individual object categories.

Training Object Verification files -

File Name	Path	Purpose
train_obj_verify.py	{project_root}/src/binsenseai/components/object_verification/	Train and Validate the model
siamese.py	same location as above	Siamese Network Architecture define
data_utils_obj_verify.py	same location as above	Image and Metadata loader for training and valid

Input

- Anchor Image (e.g., image1)
- Positive Image (e.g., image2)
- Negative Image (e.g., image3)

ResNet34 Backbone (Branch 1)

- Conv1
- Conv2
- Conv3
- Conv4
- Conv5
- Avg Pool
- Flatten

ResNet34 Backbone (Branch 2)

- Conv1
- Conv2
- Conv3
- Conv4
- Conv5

- Avg Pool
- Flatten

Shared Weights

- The two branches share the same weights

Distance Metric

- L2 Distance or Cosine Similarity

Output

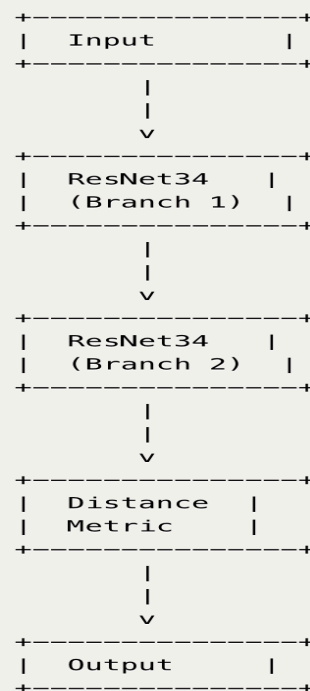
- Distance between Anchor and Positive Image
- Distance between Anchor and Negative Image

Training

- Contrastive Loss (e.g., Triplet Loss)

Note:

- The ResNet34 backbone is pre-trained on ImageNet and frozen.
- The output of each branch is a feature vector (embedding).
- The distance metric calculates the similarity between the embeddings.
- The contrastive loss trains the network to minimize the distance between similar images and maximize the distance between dissimilar images.



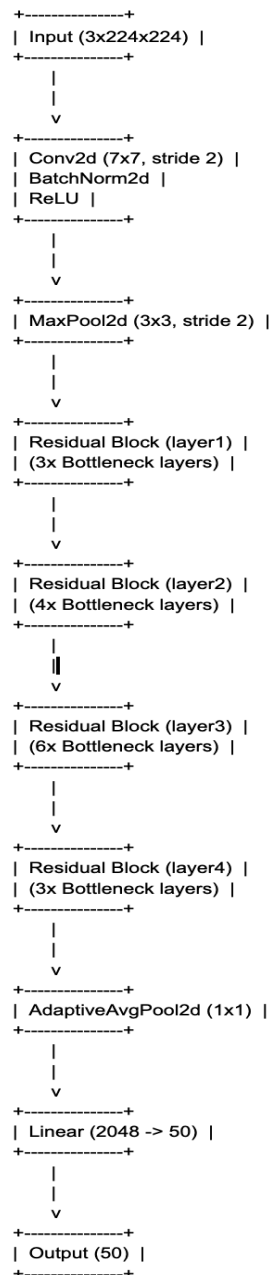
ResNet-50 architecture for Object detection and quantity count: We choose ResNet-50 architecture for object detection and object quantity count in the bin image considering its below advantages -

- Solves the problem of vanishing gradients
- Deeper neural networks.
- Improved accuracy
- Faster training times

Training Object Detection and Quantity Count files -

File Name	Path	Purpose
train_obj_quantity_count.py	{project_root}/src/binsenseai/components/object_quantity_verification/	Train and Validate the model
data_utils_obj_quantity.py	same location as above	Image and Metadata loader for training and valid

The ResNet-50 architecture diagram is given below :



User Interface

At a high level, we design the user experience so that people who search for a product (or products) with a particular quantity (or quantities) will get image(s) with high certainty of containing with exact match product(s) and quantity(ies)

The UI first calls into the backend and database, retrieves images then runs checks against our ML model to verify that those images actually contain the items and quantities. This dual-verification approach allows us to minimize any mistake that may happen in image metadata (stored in database) and detection error by the ML model. The only time that our

web app makes a mistake is when both the ML model and the metadata make a mistake, which is a very unlikely event.

Technologies

Gradio and tabulate packages are used to design a simple UI that allow users to add multiple items with quantity , display the most relevant image that contains the items and generate and display the verification result predicted by the CV models in the app.

Steps to use the app from UI :

Link: <http://100.25.33.208:7860/>

1. After opening the link in a browser , it will take to app landing page as shown below -

App Landing Page -

Not Secure — 100.25.33.208

Add Item to Cart

Item Name

Quantity

0

Add to Bin

Display the most Relevant Item Image

Display Image

Display Relevant Image

Validate if items and its respective quantity in order exists in the bin image

Validate Item Order and Bin Image

Use via API · Built with Gradio

2. User can select item and quantity and click the "Add to Bin" button to add the {item,quantity} pair to the bin . Similar way, user can add multiple items {item,quantity}.

Not Secure — 100.25.33.208

Add Item to Cart

Item Name

I & SOAP, 5pcs Mini Sampler Set (02) - Guest Soap - Travel Soap - 100% N

Quantity

1

Add to Bin

Added Items to Bin


Product Name	ASIN	Quantity
DIY YOUR WORLD USB Powered Water Dancing Speaker Music Box Speaker for PC Laptop MP3 MP4 Cell Phone	B00CRABRSU	1
Elegant Comfort 1500 Thread Count Wrinkle & Fade Resistant Egyptian Quality Ultra Soft Luxurious 4-Piece Bed Sheet Set, Queen, Gray	B00DV8AJLS	1
I & SOAP, 5pcs Mini Sampler Set (02) - Guest Soap - Travel Soap - 100% Natural & Organic Materials - Handcrafted Herbal Soap - Gentle and Effective Facial, Hand and Body Cleansing Soap Bars - Deeply Moisturizing Soft Soap - **Sodium Lauryl Sulfate(SLS), Paraben and Phthalate FREE - 100% Satisfaction GUARANTEED - SPMS02	B00H95M72S	1

3. After adding the items , click on the "Display Image" button to display the most relevant images that contain all the items.

Display the most Relevant Item Image

Display Image

Display Relevant Image



4. After retrieving the relevant image, click on "Validate Item Order and Bin Image" button to display the results of the CV models to validate if items and its respective quantity in order exists in the bin image.

Validate if items and its respective quantity in order exists in the bin image

Validate Item Order and Bin Image

Inference Result

Product Name	ASIN	Quantity	In Stock
DIY YOUR WORLD USB Powered Water Dancing Speaker Music Box Speaker for PC Laptop MP3 MP4 Cell Phone	B00CRABRSU	1	True
Elegant Comfort 1500 Thread Count Wrinkle & Fade Resistant Egyptian Quality Ultra Soft Luxurious 4-Piece Bed Sheet Set, Queen, Gray	B00DV8AJLS	1	True
I & SOAP, 5pcs Mini Sampler Set (02) - Guest Soap - Travel Soap - 100% Natural & Organic Materials - Handcrafted Herbal Soap - Gentle and Effective Facial, Hand and Body Cleansing Soap Bars - Deeply Moisturizing Soft Soap - **Sodium Lauryl Sulfate(SLS), Paraben and Phthalate FREE - 100% Satisfaction GUARANTEED - 5PMS02	B00H95MT2S	1	True

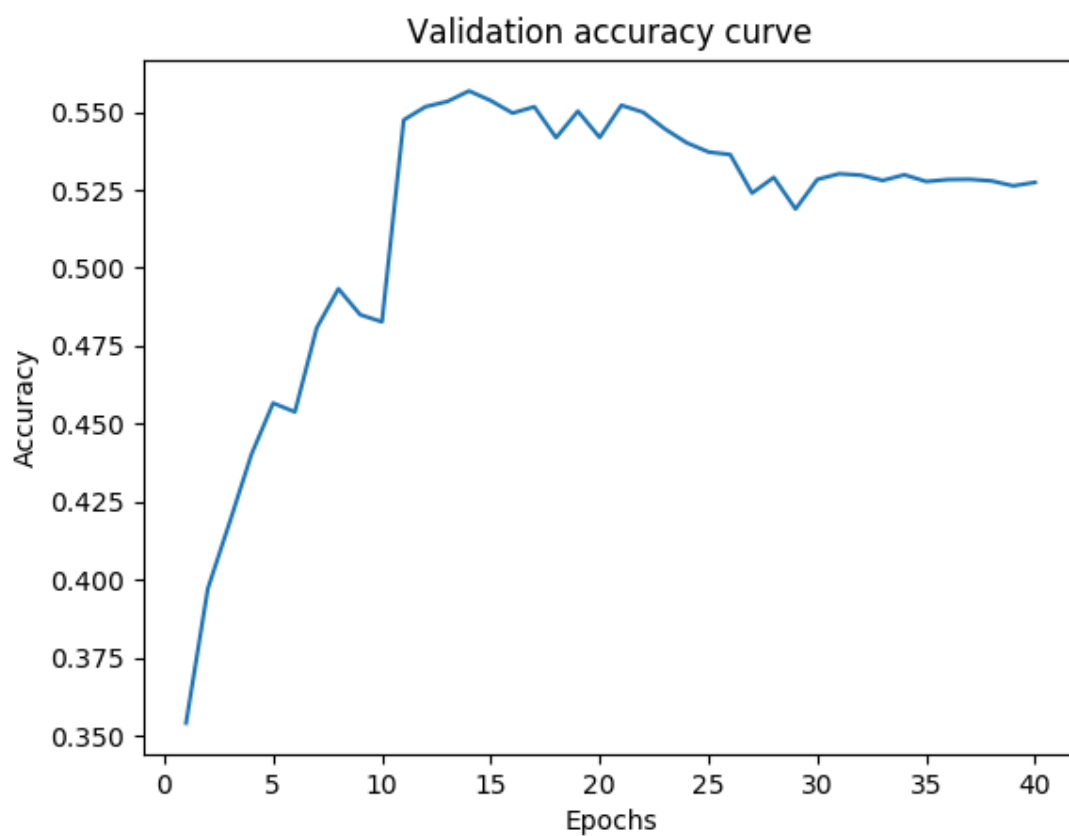
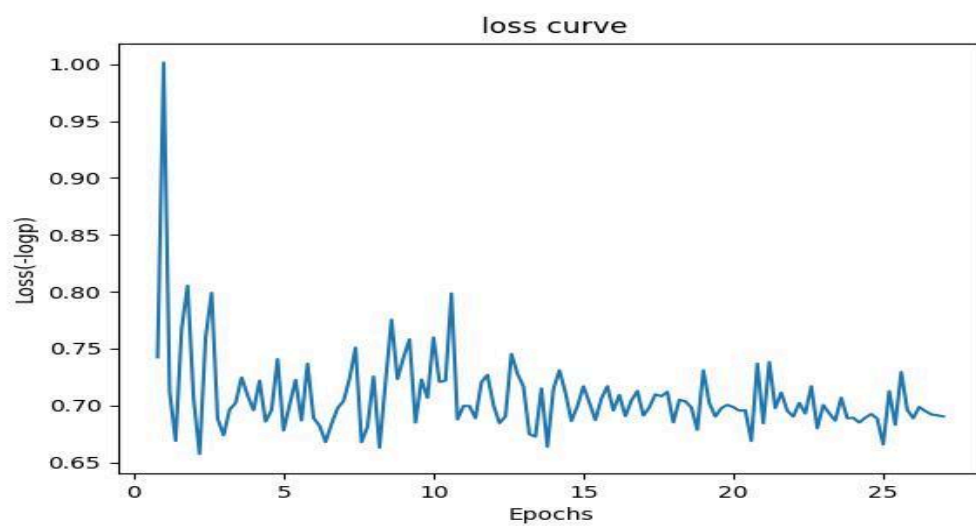
Decisions and Milestones

- Initial research: Understanding dataset structure and content
- EDA: Defining scope and refining understanding
- Image pre-processing: Standardizing image sizes
- Data preparation: Preparing data for CV modeling
- CV model selection: Selecting the most accurate model
- Hyperparameter tuning: Optimizing model parameters
- UI development: Designing and developing user-friendly UI
- Integration: Integrating UI and CV model
- Testing: Thorough testing with positive and negative test cases
- MLOps integration: Automating deployment, monitoring, and retraining

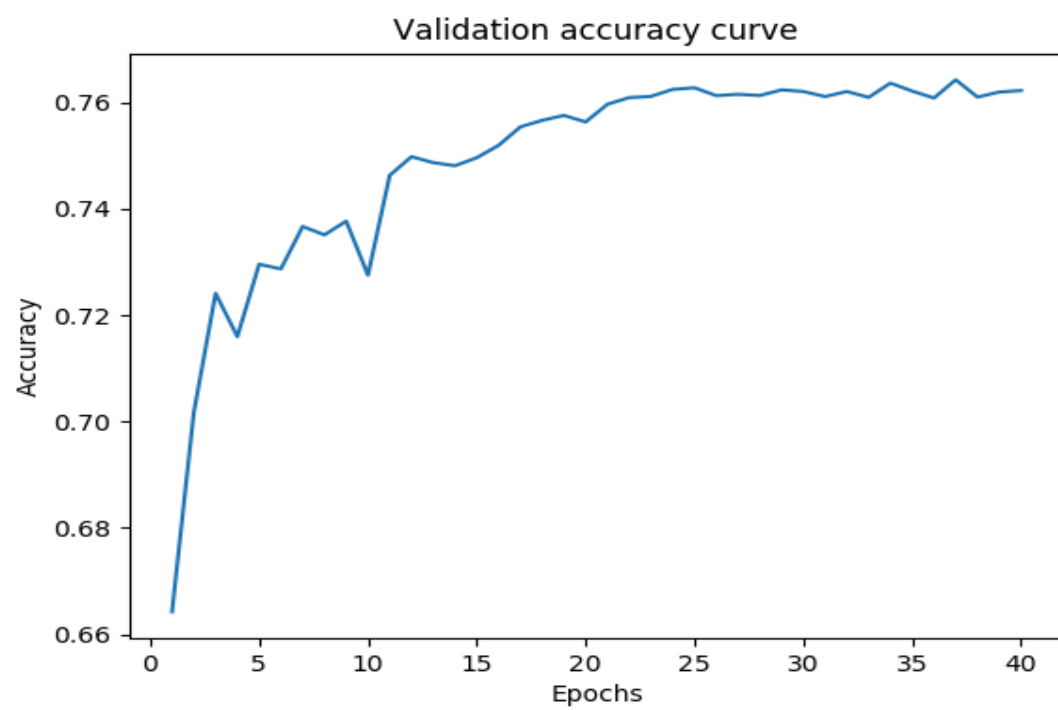
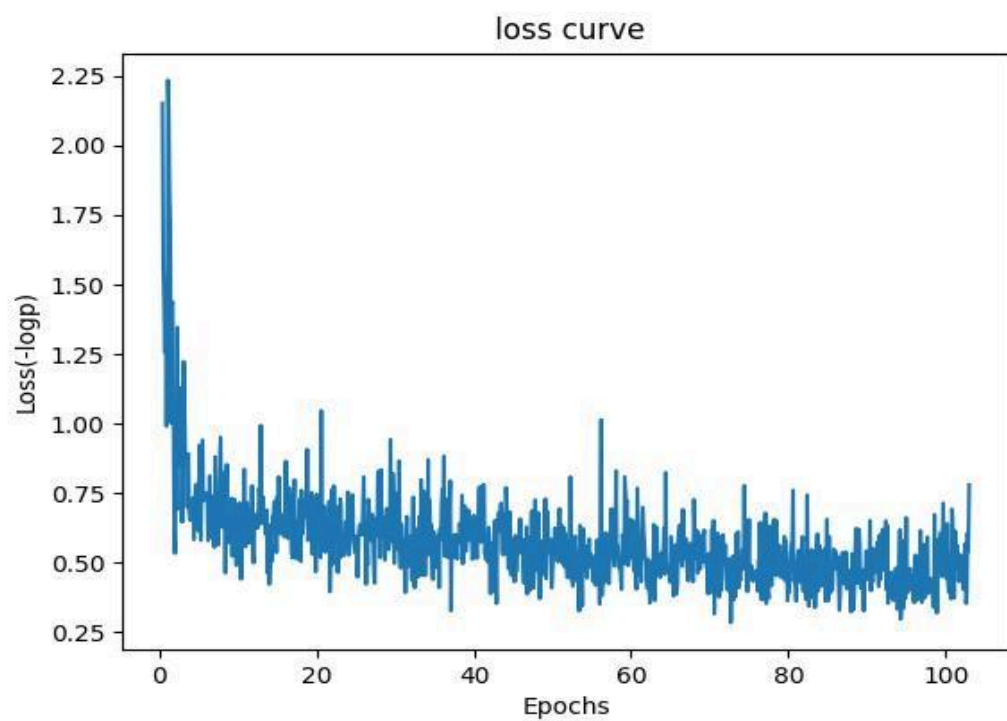
Model Selection and Hyperparameter Tuning

- CV models considered:
 - Siamese Network with ResNet34 Backbone
 - resnet50
- Hyperparameter tuning experiments: [model predict good at 21 epoch with these parameters for both object verification and object quantity verification :
 - Normalize : mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225]
 - Batch size: 64
 - Criterion :
 - For resnet50 : [Linear](#)
 - For resnet34 : [BCELoss](#)
 - Optimizer : SGD
 - Learning-rate: 0.001
 - Momentum: 0.9
 - Precision : 0.554

Object verification



Object Quantity Verification



Evaluation and Results

- CV model performance metrics:
 - For Object Verification Model Training used to train Siamese Network with ResNet34 Backbone -

```

❏ evaluate_obj_verify.py
abid > object_verification > ❏ evaluate_obj_verify.py > ...
1  import json
2  import numpy as np
3
4  with open('./obj_verification_val.json') as f:
5      val_list = json.loads(f.read())
6
7  n = 0
8  correct = 0
9
10 with open('./obj_verification_result.txt') as f:
11     for line in f:
12         pred = int(line)
13         gt = int(val_list[n][2])
14         correct = correct + int(pred==gt)
15         n = n+1
16
17 print('accuracy')
18 print('%d/%d (%F)' %(correct, n, float(correct)/n))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1

```

❏ (assignment-env) ubuntu@ip-172-31-84-132:~/abid/object_verification$ python3 evaluate_obj_verify.py
accuracy
200/400 (0.500000)
❏ (assignment-env) ubuntu@ip-172-31-84-132:~/abid/object_verification$

```

- For Object Verification and Quantity Count to train resnet50 model -

```

1  # make_obj_verification_data.py
2  # verify_item_quantity.py
3  # object_count_evaluate.py
4
5  abid > object-quantity-verification > # object_count_evaluate.py > ...
6
7  1
8  2 import json
9  3 import numpy as np
10 4
11 5 with open('/home/ubuntu/abid/dataset/obj_num_verification_val.json') as f:
12 6     val_list = json.loads(f.read())
13 7
14 8
15 9 n = 0
16 10 perclass_correct = np.zeros(50)
17 11 perclass_dist = np.zeros(50)
18 12 perclass_N = np.zeros(50)
19 13
20 14 with open('obj_quantity_count_result.txt') as f:
21 15     for line in f:
22 16         pred = int(line)
23 17         gt = int(val_list[n][3])
24 18         perclass_correct[gt] = perclass_correct[gt] + int(pred==gt)
25 19         perclass_dist[gt] = perclass_dist[gt] + np.power(pred-gt,2)
26 20         perclass_N[gt] = perclass_N[gt] + 1
27 21         n = n+1
28 22
29 23 print('accuracy')
30 24 print('%d/%d (%f) %d(perclass_correct.sum(), perclass_N.sum(), perclass_correct.sum()/perclass_N.sum()))'
31 25 print('RMSE(Root mean squared error)')
32 26 print(np.sqrt(perclass_dist.sum()/perclass_N.sum()))
33 27 print('Per class accuracy')
34 28 print(perclass_correct,perclass_N)
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

- UI functionality: <http://100.25.33.208:7860>

MLOps Considerations:

This part is a crucial part when dealing with ML model deployment and making it ready for use through API.

- **Model deployment:** integrating your model into an existing production environment where it can take in an input and return an output. Technic used:
 - **Github** (create, store, manage and share code as well as versioning control) and,
 - **Docker** (deliver software in packages called containers).
 - **ECR** (AWS managed container image registry service) .
 - **GitHub Actions** will be employed for seamless CI/CD integration.

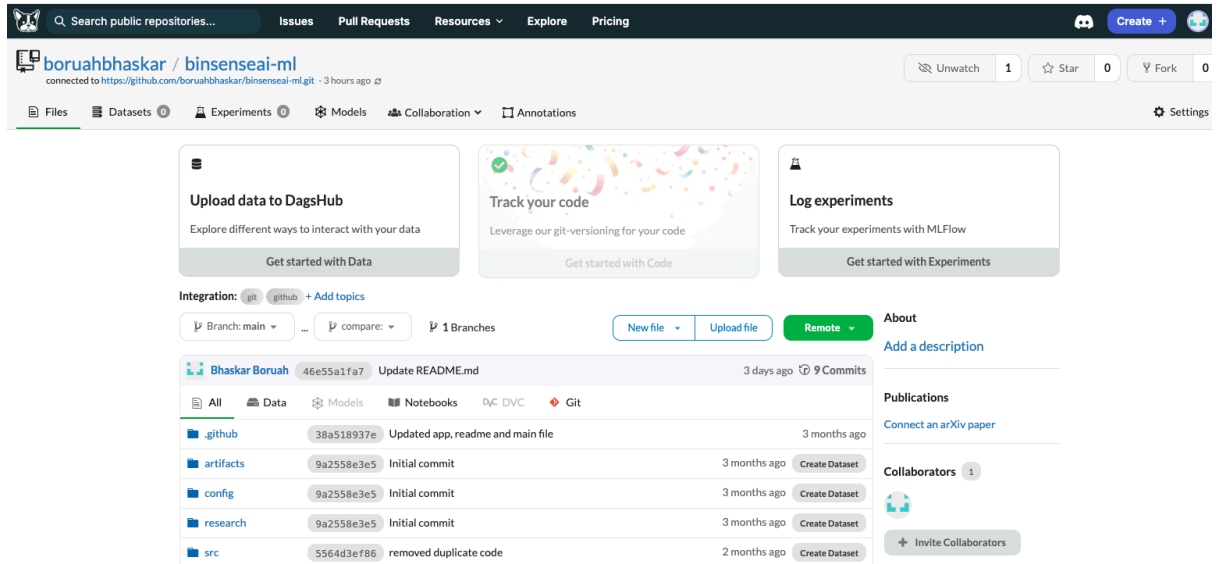
The screenshot shows the GitHub Actions interface for a workflow named 'removed duplicate code #5'. The workflow is in a 'Success' state, triggered by a push to the 'main' branch. The summary card displays a total duration of 5m 48s and a billable time of 12m. Below the summary, a 'main.yaml' file is shown with a job named 'on: push' containing three steps: 'Continuous Integration' (10s), 'Continuous Delivery' (10m 39s), and 'Continuous-Deployment' (5m 35s). The 'Annotations' section shows 10 warnings, including a deprecation notice for 'Continuous Integration' actions.

The screenshot shows the detailed view of the 'Continuous Integration' job, which succeeded on June 28 in 10s. The job log is displayed in a dark theme, showing the following steps and commands:

- Set up job** (1s)
- Checkout Code** (9s)
- Lint code** (0s)
 - 1 ▶ Run echo "Linting repository"
 - 4 Linting repository
- Run unit tests** (0s)
 - 1 ▶ Run echo "Running unit tests"
 - 4 Running unit tests
- Post Checkout Code** (0s)
 - 1 Post job cleanup.
 - 2 /usr/bin/git version
 - 3 git version 2.45.2
 - 4 Temporarily overriding HOME='/home/runner/work/_temp/0e990f05-3d9b-48c7-a8cd-1930f2de89f7' before making global git config changes
 - 5 Adding repository directory to the temporary git global config as a safe directory
 - 6 /usr/bin/git config --global --add safe.directory /home/runner/work/binsenseai-ml/binsenseai-ml
 - 7 /usr/bin/git config --local --name-only --get-regexp core.sshCommand
 - 8 /usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'core.sshCommand' && git config --local --unset-all

- **Model monitoring:**

MLFlow - Setup completed but not able to perform experiment on EC2 instance due low processing power



- **Model retraining -**

The general strategy and schedule for CV (Computer Vision) model retraining is given below. For this project, we have not implemented a model retraining module due to time constraint .

1. **Continuous Monitoring:**

- Monitor model performance and data distribution regularly (e.g., daily/weekly).
- Use metrics like accuracy, precision, recall, and F1-score to track performance.

2. **Retraining Schedule:**

- Daily/Weekly: Retrain the model with new data to adapt to minor data drift.
- Monthly: Perform a full retraining cycle with a new dataset to adapt to significant data drift.
- Quarterly: Update the model architecture or hyperparameters to incorporate new techniques or improvements.

3. **Retraining Triggers:**

- Performance threshold: Retrain when the model's accuracy drops below a set threshold (e.g., 5%).

- Data drift detection: Retrain when significant data drift is detected (e.g., changes in data distribution).
- New data availability: Retrain when a substantial amount of new data becomes available.

4. Retraining Process:

- Incremental retraining: Update the existing model with new data.
- Full retraining: Train a new model from scratch with the updated dataset.
- Transfer learning: Use pre-trained models as a starting point and fine-tune on the new dataset.

5. Model Versioning:

- Versioning: Keep track of model versions and their corresponding performance metrics.
- Rollback: Roll back to a previous version if the new version performs poorly.

6. Automation:

- Automate monitoring: Use tools like Prometheus, Grafana, or TensorBoard to monitor model performance.
- Automate retraining: Use tools like Apache Airflow, TensorFlow, or PyTorch to automate the retraining process.

Conclusion

- Developed computer vision models that can accurately verify the presence and quantity of items in a bin image.
- Designed and developed a user-friendly UI for user input and image processing
- Integrated MLOps frameworks for automated deployment and monitoring
- Demonstrated creativity and innovative mindset throughout the project