



**BAU**

**BAHÇEŞEHİR UNIVERSITY**

**Faculty of Engineering and Natural  
Sciences**

**Department of Artificial Intelligence  
Engineering**

**Training Report**

**Name, Surname: Atena Jafari Parsa  
Student Number: 2101183**



**BAU**

**BAHÇEŞEHİR UNIVERSITY**

**Faculty of Engineering and Natural  
Sciences**

**Department of Artificial Intelligence  
Engineering**

**Training Report**





**Bahçeşehir University**  
**Training Report**

**Weekly Schedules**

**1<sup>st</sup> Week**

| Date       | Tasks Accomplished   |
|------------|--|
| 28/06/2025 | Met mentor to review product line and pain points; agreed to target surface cracks, dents, and scratches on metal parts.<br>My access to repo, shared drive, GPU workstation (RTX 4090), Kanban board was set up and I installed the base tools including CUDA & NVIDIA drivers. |
| 29/06/2025 | Collected 1.2k existing line-camera images and 15 short videos (1080p, 30fps) from the company archive.<br>Wrote a python script to prepare the dataset by extracting and adjusting video frames and organized folder scheme; checksum logged.                                   |
| 30/06/2025 | Deployed <b>**Label Studio**</b> locally via docker-compose and created label taxonomy. Imported 2.5k frames; wrote labeling guidelines and labeled a pilot set of 300 images for baseline experiments.  |
| 01/07/2025 | Created an EDA notebook for class distribution, bbox size histograms and brightness/sharpness checks. Identified class imbalance and planned augmentation strategy.  |
| 02/07/2025 | Converted annotations from Label Studio JSON → YOLO format, 70/20/10 split with stratification by class. Implemented a reproducible split with seeded RNG and wrote unit test for converter as well as Baseline augmentations.   |

Executive Officer

*Jalal Rana*

**2<sup>nd</sup> Week**

| Date       | Tasks Accomplished  |
|------------|---|
| 05/07/2025 | Trained the baseline model, YOLOv8n. Logged experiments to <b>**MLflow**</b> ; saved confusion matrices.  |
| 06/07/2025 | Visualized FN/FP but small/thin scratches often missed.<br>Tried test-time augmentation (TTA); slight improvement in recall but latency increased. Decided to move from YOLOv8n to YOLOv8m. |
| 07/07/2025 | Auto-anchor tuning; expanded augmentations: CutOut, MotionBlur, CLAHE for low-light. Re-trained YOLOv8m 80 epochs.  |
| 08/07/2025 | Collected 400 additional "clean" parts misclassified as defective. Retrained with weighted sampling; FPs reduced 18% on validation.   |
| 09/07/2025 | Implemented torchvision Faster R-CNN ResNet50 FPN; trained 30 epochs. Achieved slower inference (~12 FPS). Kept YOLOv8m as primary.   |

Executive Officer

*Jalal Rana*





**Bahçeşehir University**  
**Training Report**

**Weekly Schedules**

**3<sup>rd</sup> Week**

| Date                                       | Tasks Accomplished   |
|--|--|
| 12/07/2025                                 | Did hyperparameter tuning. Swept LR, weight decay, mosaic prob, img size (640 vs 768). Best run: img 768, mosaic 0.5, LR 0.002 $\rightarrow$ mAP@50=0.893, mAP@50-95=0.583.              |
| 13/07/2025                                 | Fixed class imbalance. Introduced focal loss (gamma=1.5) via YOLO configuration; added targeted augmentation for 'scratch' (ThinObjectAug).  |
| 14/07/2025                                 | Expanded data. Labeled 500 more frames prioritizing rare defect appearances using active-learning (highest-entropy samples). Re-ran training; validation mAP@50=0.905.                   |
| 15/07/2025                                 | Exported best checkpoint to ONNX; tested ONNX Runtime vs PyTorch eager. Explored TensorRT FP16: latency decreased with similar accuracy.   |
| 16/07/2025                                 | Designed API. Built FastAPI service endpoints which accepted image or frame bytes and returned boxes, scores, classes. Added pydantic schemas, size checks, and per-request timing logs. |
| Executive Officer<br><i>Faialat Raveet</i> |  |

**4<sup>th</sup> Week**

| Date                                       | Tasks Accomplished   |
|--|--|
| 19/07/2025                                 | Created Dockerfile with CUDA base, requirements caching, and non-root user for local GPU deployment and integrated NVIDIA Container Toolkit.   |
| 20/07/2025                                 | Implemented RTSP frame reader with backpressure; batched inference by 4 frames when GPU headroom allowed. End-to-end throughput stabilized memory with pre-allocated tensors.                |
| 21/07/2025                                 | Added input sanitation: reject images >10MB, weird aspect ratios; defensive try/except with structured errors. Wrote unit tests for preprocessing; synthetic corruption tests (blur, noise). |
| 22/07/2025                                 | Exported per-request latency to Prometheus textfile collector. I Set alert threshold: p95 > 60ms for 5m triggers Slack webhook (as placeholder).   |
| 23/07/2025                                 | Tuned per-class NMS IoU and score thresholds to trade off FP vs FN according to ops preference. Final operating point reduced false alarms.  |
| Executive Officer<br><i>Faialat Raveet</i> |  |





**Bahçeşehir University**  
**Training Report**

**Weekly Schedules**

**5<sup>th</sup> Week**

| Date                                   | Tasks Accomplished   |
|--|--|
| 26/07/2025                             | Evaluated on edge cases and added AutoExposure compensation and gamma correction pre-step.   |
| 27/07/2025                             | Wrote README with setup, training, and serving instructions; added diagrams and benchmark table and created the runbooks.  |
| 28/07/2025                             | I handed over the scripts that added versioned artifact naming, published tasks to 'releases/' with checksum and I added pre-commit hooks.   |
| 29/07/2025                             | Checked dependencies with 'pip-audit'; pinned versions; removed outbound telemetry and created data retention policy for raw/videos (90 days).   |
| 30/07/2025                             | Built a tiny Streamlit demo for local QA; also provided a cURL recipe and Python client snippet for QA team. Collected feedback on bounding-box readability and class names from the team. |
| Executive Officer<br><i>Fatih Ravi</i> |  |

**6<sup>th</sup> Week**

| Date                                   | Tasks Accomplished  |
|--|---|
| 02/08/2025                             | Performed A/B evaluation. Compared YOLOv8m FP16 vs TensorRT FP16 builds on validation and live samples. Chose TensorRT build for production due to lower p95 latency with negligible accuracy delta.                    |
| 03/08/2025                             | Deployed container on on-prem GPU machine; wired to line-camera feed mirror and logged 87k frames; 312 detections reviewed; zero crashes.   |
| 04/08/2025                             | Performed post-pilot analysis. Added "oil sheen" detector heuristic to suppress reflections; retested okay.   |
| 05/08/2025                             | Did the final tuning and signed off. Locked model and thresholds; tagged release 'v1.0.0' in registry; artifacts archived. Drafted acceptance checklist; walk-through with mentor; collected sign-off comments.         |
| 06/08/2025                             | Did my presentation and handed over my work. Delivered a 20-minute tech talk with live demo and metrics. Handover package: code, Docker images, model weights, notebooks, monitoring dashboard JSON, and documentation. |
| Executive Officer<br><i>Fatih Ravi</i> |   |





**Bahçeşehir University**  
**Training Report**

## **ABSTRACT**

In my internship at Bina Sanat Veera, which is a smart quality control and computer vision company in Iran, I was given the task to build an end-to-end computer vision pipeline (prototype) that automatically detected defective manufactured metal parts from a production line camera. The internship included data acquisition, labeling, exploratory analysis, model experimentation (YOLOv8, Faster R-CNN), training on a custom dataset, evaluation, packaging the best model behind a FastAPI endpoint, containerization with Docker, on-prem deployment on a GPU workstation, and basic monitoring/alerting.

After onboarding and project scoping and getting access to the company archive data, I set up the labeling and performed Exploratory Data Analysis (EDA) and worked on the data quality by identifying class imbalance. I sampled 50 ambiguous cases for mentor review then, I updated the guidelines. I trained the baseline model (YOLOv8n) after data conversion and splits and did an error analysis & test-time augmentation (TTA) on the following day. I optimized the model and even tried an alternative baseline (Faster R-CNN). The original baseline model (YOLOv8n) had a better performance thus, for further optimization, I expanded the data and tuned the hyperparameters.

Eventually, I designed an API and created Dockerfile with CUDA base, requirements caching and non-root user. I then evaluated on edge cases and shifted the domain. After several other evaluations for security & privacy, I built a small POC for mobile client with Streamlit for local QA. Finally, I performed A/B evaluation and did post-pilot analysis. I presented and handed over my work on the last day of my internship.





**Bahçeşehir University  
Training Report**

## **INTRODUCTION**

Bina Sanat Veera, a company specialized in computer vision and Artificial Intelligence (AI), has gathered experienced engineers from sectors of robotics, electronic engineering, telecommunications, commerce, mechanical engineering, control engineering and AI. I had the opportunity to complete an AI internship at Bina Sanat Veera and work on a modern end-to-end computer vision pipeline for automatic surface defects detection on manufactured metal parts from a production line camera. I gained so much practical knowledge regarding my major during this internship.

I conducted data acquisition and ingestion on the raw data that I obtained from the company archive and for labeling, I used Label Studio locally to create label taxonomy that I have further explained in the Practical Training section. I also performed Exploratory Data Analysis (EDA) and performed data conversion and splits for improving the quality of the data. I then trained my baseline model (YOLOv8n) and logged experiments to MLflow. The errors were analyzed at all stages and test-time augmentation (TTA) was performed as well. For data cleaning, techniques such as Hard Negative Mining were utilized, the class imbalance was fixed and the hyperparameters were tuned for further optimization. Eventually, a FastAPI service was designed which accepted images or frame bytes and returned boxes, scores and classes. The edge cases were evaluated, and the domain was shifted before finalizing the project. After final tuning the model and the thresholds were locked, and the project walkthrough was reviewed by the mentor.





Bahçeşehir University  
Training Report

## ABOUT THE COMPANY

Bina Sanat Veera, a company focused on Artificial Intelligence (AI) and Computer Vision, started in 2017 in Tabriz Automotive Techno Park, Iran. As the company is located at Automotive Technology district at Tabriz Science & Technology Park, the company exists alongside the top research centers and other tech companies. Bina Sanat Veera also has a branch in Dongguan, China for importing state-of-the-art industrial automation products. The company has developed numerous services & projects for automation and production lines, product quality control lines equipped with AI and computer vision providing robotic tools for industrial optimization.

Bina Sanat Veera has gathered experienced engineers in the fields of artificial intelligence, control engineering, robotics, mechanical engineering, commerce, telecommunications engineering and electronic engineering. A total of 25 engineers work at Bina Sanat Veera as well as the non-engineer staff. The company's business partners include HikRobot, FIFO, Cognex, Siemens, Jawest, Basler, Daheng Imaging, ToupTek, Opto Engineering Telecentric Company, Omron, SBS Vision Solution, etc and the company customers include the National Post Company of Iran and Saipa (one of the biggest automaker companies of Iran).

Bina Sanat Veera has developed AI-Powered products and services in various industries. The products of the company include Aluminum and Iron Sheet Quality Control System using Computer Vision (similar to the project prototype I worked on during my internship), AI-Based Conveyor Inspection System, Rice Sorting Machine using Computer Vision (A Revolution in the Food Industry), Automatic Truck Load Volume Calculator using Computer Vision, Rubber & Tire Quality Control System, Smart Tablet & Capsule Counter and Sorter, Computer Vision-Based Dentures Sorter, etc.





## PRACTICAL TRAINING

During my internship at Bina Sanat Veera, I was assigned to rebuild a prototype of a project that the company had already built, an End-to-End Automatic Surface Defects Detector Computer Vision Pipeline on Manufactured Metal Parts from a Production Line Camera. The internship started with reviewing product line and pain points with the mentor (Mr. J. Raouf). Then we agreed to target surface cracks, dents, and scratches on metal parts. The defined success metrics are shown in the table below:

**Table 1: Performance targets for the model, specifying accuracy, latency, and throughput requirements to meet deployment standards**

| Metric            | Target Value                  |
|-------------------|-------------------------------|
| Accuracy (mAP@50) | $\geq 0.9$                    |
| Inference Latency | $\leq 50$ ms per frame        |
| Processing Speed  | $\geq 25$ FPS on 1080p stream |

Access to the repository, shared drive, GPU workstation (RTX 4090) and Kanban board was set up. The basetools CUDA/cuDNN compatibility check and NVIDIA drivers were then installed.

### Tech Stack

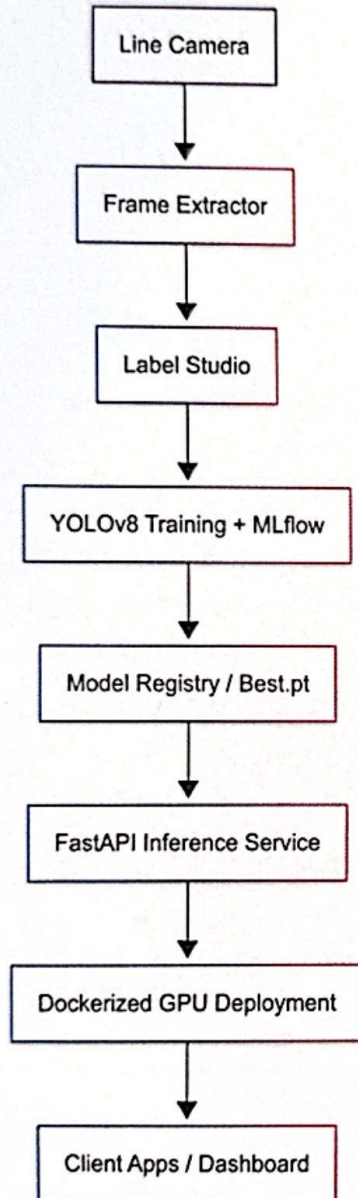
The programming languages used consisted of Python 3.11 and Bash. The utilized core libraries were PyTorch, Ultralytics YOLOv8, torchvision, OpenCV, albumentations, scikit-learn, NumPy and Pandas. The experiment tracking was performed on MLflow and for annotation, Label Studio was used. The services included FastAPI, Uvicorn, ONNX Runtime and TensorRT for optimization. The tech stack used for DevOps was Git/Github, pre-commit, Docker, docker-compose and NVIDIA Container Toolkit. Simple Custom Logs and Prometheus node\_exporter were used for the purpose of monitoring, and the project management (To Do / Doing / Done) and daily standups were tracked on Kanban. The documentation was initially conducted in Markdown (md) format with Mermaid Architecture Diagram.

### High-Level Architecture

As shown in figure 1, the system started with unseen real-time inputs from the line camera (or uploading them into the system as this project was a prototype). Then, if the input was a single image frame, it would proceed to the labeling stage, otherwise, if the input was a video clip, it would go through frame extraction. The details of the frame extraction have been discussed in day 2 of the daily activities' pages (29/06/2025). At the labeling stage, the labeling guidelines, occlusion rules and ambiguity handling were applied to the data and after



the data was ready for training the baseline model, the model was trained for the first time and all the metrics were logged to MLflow. To improve the model performance, several optimization techniques such as tuning the hyperparameters was applied, and the model with the highest accuracy was registered. Then, a FastAPI was designed to provide real-time model inference service. Then, the model was dockerized and in the end, a live demo was built with Streamlit using the designed API as a client app.



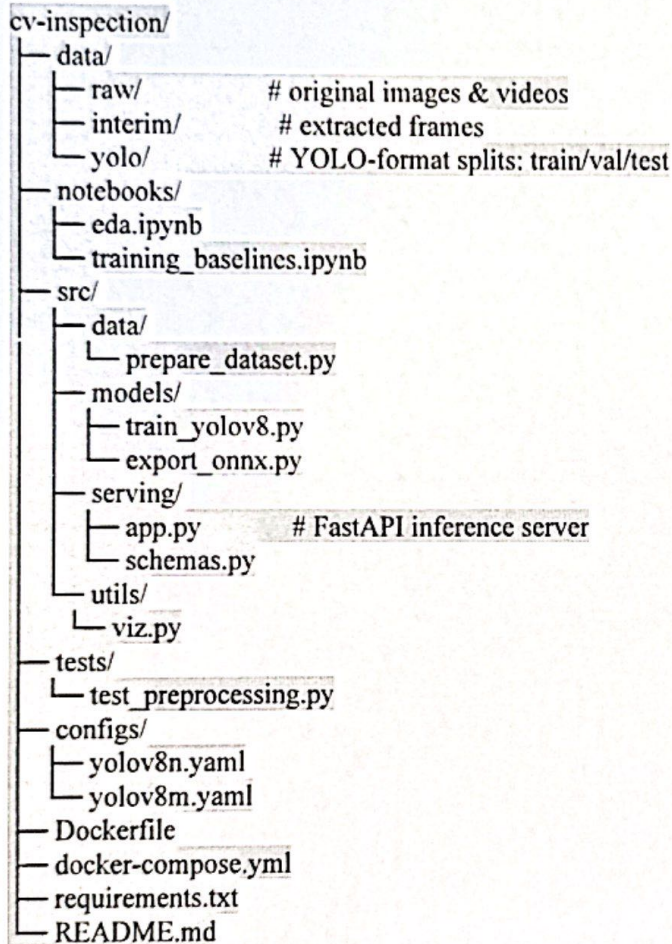
**Figure 1: Project High-Level Architecture Mermaid Graph**

### **Final Repository Structure**

The final project repository structure is shown in figure 2. The data folder contains the raw, original data and interim contains the extracted image frames from the video clips. After the labeling setup, the guidelines were in JSON format, and they had to be converted to YOLO format to be readable by the YOLOv8n model. After conversion to YOLO, the data was split 70/20/10 (train set/ validation set/ test set) and stored in 'data/yolo'. The notebooks consist of



the exploratory data analysis (EDA) script and training the baseline model script. In the 'data' folder of 'src', prepare\_dataset.py performs the data transforms and augmentations to fix the class imbalance in the data. The 'models/' path contains the training and exporting scripts of the model and 'serving/' contains the FastAPI design script and schemas. The results of the model were visualized for clear examination by viz.py at 'src/utils/'. 'tests' as the name suggests, contains the testing script and configs consist of the yaml files. In case someone wants to reproduce the project, they can install the necessary libraries by using requirements.txt. The initial documentation in markdown format was written in README.md.



**Figure 2: Repository Structure**

## Results

Two models were tested and evaluated, YOLOv8m and an alternative baseline, Faster R-CNN. Torchvision Faster R-CNN ResNet50 FPN was implemented and trained for 30 epochs. Faster R-CNN achieved  $mAP@50=0.842$ ; and had slower inference (~12 FPS). Thus, YOLOv8m was kept as primary. The results of both baseline models are compared in table 2.



**Table 2: YOLOv8m vs Faster R-CNN Accuracy, Inference Latency & Processing Speed Results**

| <i>Metric</i>                     | <b>YOLOv8m</b>          | <b>Faster R-CNN</b>     |
|-----------------------------------|-------------------------|-------------------------|
| <i>Accuracy (mAP@50)</i>          | 0.918                   | 0.842                   |
| <i>Inference Latency (~1/FPS)</i> | ~35.7 ms                | ~83.3 ms                |
| <i>Processing Speed</i>           | ~28 FPS on 1080p stream | ~12 FPS on 1080p stream |

When table 1 (success metrics defined in the beginning of the project) and table 2 (the results of both models) are compared, it is obvious that:

1. The accuracy (mAP@50) of YOLOv8m is 0.918 and the accuracy of Faster R-CNN is 0.842. The minimum success metric defined for accuracy was 0.9. Therefore, only YOLOv8m is accepted in terms of accuracy.
2. The inference latency of YOLOv8m is around 35.7 ms which is less than 50 ms, the success metric. However, the inference latency of Faster R-CNN is approximately 83.3 ms which is far greater than the maximum acceptable inference latency, 50 ms. Thus, unlike Faster R-CNN, YOLOv8m passes the inference latency check.
3. The minimum acceptable processing speed was defined as 25 FPS on 1080p stream. The processing speed of YOLOv8m was approximately 28 FPS, passing the processing speed check, and the processing speed of Faster R-CNN was around 12 FPS, unable to pass the test.

The operational results included a dockerized FastAPI with GPU, basic monitorings and alerts and runbooks for on-call. Additionally, the resulted data consisted of ~3.5k labeled images with balanced classes through active learning and augmentation. To further improve the project for future works, the dataset could be expanded with multi-angle cameras and reflective-condition variants, and lightweight trackers (ByteTrack) could be investigated for the purpose of multi-frame stability. For even greater improvements, a periodic retraining pipeline with MLflow model registry could be considered as well as edge deployment on Jetson for decentralized stations.





Bahçeşehir University  
Training Report

## CONCLUSION

I gained invaluable experience during my internship at Bina Sanat Veera. To build an end-to-end computer vision pipeline that scans real-time products on a production line camera, I learned about data acquisition & company culture on archive access as well as access to other resources of the company. I performed Exploratory Data Analysis (EDA) on real-time data and made the model make inferences on data never seen before and I had to prepare the model for that by performing data conversion and splits, exploring different models, conducting error analysis and test-time augmentation (TTA), hard negative mining, tuning the hyperparameters and fixing the class imbalance as well as expanding the data to improve the model performance. The academic projects that I had worked on usually had already cleaned data and a ready-to-use model, that was not the case at my internship. The work did not finish after preparing a model with a high accuracy, I also designed an API (FastAPI) and dockerized the project for video stream inference. I did Q&A sessions with my mentor (Mr. J. Raouf) to increase robustness, and I also evaluated on the edge cases. Eventually, I made a demo, a mobile client with Streamlit. I performed A/B evaluation and conducted pilot deployment. Then I did post-pilot analysis and locked model and thresholds to finish the work. Additionally, an acceptance checklist was drafted, a walk-through of the project was done with the mentor, and a 20-minute presentation was delivered with the live demo and metrics. Finally, the package was handed over to the company: the code, Docker images, model weights, notebooks monitoring dashboard JSON and the documentation.





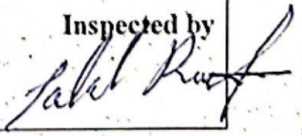
**Bahçeşehir University**  
**Training Report**

## **REFERENCES**

1. <https://binasanat.com>

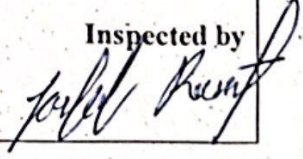


## Daily Activities

| <b>Date</b><br>28/06/2025  | <b>Department</b><br>Artificial Intelligence |        |              |                   |            |   |                        |                  |                               |
|--|--|--------|--------------|-------------------|------------|---|------------------------|------------------|-------------------------------|
| <p>Today was the first day of my internship at Bina Sanat Veera. Weekends in Iran are Thursdays and Fridays, therefore, Saturdays and Sundays are weekdays. I met my mentor, Mr. Raouf, and we discussed the project that I would work on for the rest of the internship. I agreed to build a prototype of one of the products that the company had already built, an end-to-end computer vision pipeline that automatically detects surface defects on manufactured metal parts from a production line camera. I had to build an inference model that operated on real-time and unseen data. I would also be provided with data from the company archive. My mentor told me that my internship would cover data acquisition, data labeling, exploratory data analysis (EDA), model experimentation (if necessary), training the model on a custom dataset, model evaluation, packaging the model behind an API endpoint, containerization with Docker and deployment. I offered that I could also build a live demo if I had the time, to show how the project works in real-time. My mentor said it was a good idea, but not a requirement. Then, he discussed the requirements and details of the project. We agreed that the model would detect surface cracks, dents and scratches on metal parts. He also defined the success metrics. For the accuracy, I decided to use mAP@50 and with the mentor's approval, the minimum acceptable accuracy was defined as 0.9. Then, we decided on the maximum inference latency, 50 ms per video frame, as the training data and the input data would be in image frames &amp; 1080p video stream format that have to be input to a frame extractor. The last success metric was the processing speed. As suggested by the mentor, the minimum processing speed was defined as 25 ms per 1080p stream. I also learned that inference latency is almost equal to the reciprocal of processing speed (Frame per Second):</p> <p>Inference Latency <math>\approx 1 / \text{FPS}</math></p> <p>The final defined success metrics are shown in the table below (Table 3):</p> <p><b>Table 3: Performance targets for the model, specifying accuracy, latency, and throughput requirements to meet deployment standards</b></p> <table border="1" style="width: 100%; border-collapse: collapse; margin: 10px 0;"> <thead> <tr> <th style="width: 50%;">Metric</th> <th style="width: 50%;">Target Value</th> </tr> </thead> <tbody> <tr> <td>Accuracy (mAP@50)</td> <td><math>\geq 0.9</math></td> </tr> <tr> <td>Inference Latency (<math>\sim 1/\text{FPS}</math>)</td> <td><math>\leq 50</math> ms per frame</td> </tr> <tr> <td>Processing Speed</td> <td><math>\geq 25</math> FPS on 1080p stream</td> </tr> </tbody> </table> <p>Then, my access was set up to the necessary resources. To the repo, shared drive, GPU Workstation (RTX 4090), and the Kanban board for the project management tracking. I also installed the base tools. Python 3.11 (I already had 3.12 but switched it to 3.11 to prevent any complications with the rest of the system parts), CUDA/cuDNN Compatibility Check, Git and Docker (already had them, just double-checked) and finally, NVIDIA drivers.</p> |  | Metric | Target Value | Accuracy (mAP@50) | $\geq 0.9$ | Inference Latency ( $\sim 1/\text{FPS}$ ) | $\leq 50$ ms per frame | Processing Speed | $\geq 25$ FPS on 1080p stream |
| Metric   | Target Value                                 |        |              |                   |            |   |                        |                  |                               |
| Accuracy (mAP@50)  | $\geq 0.9$                                   |        |              |                   |            |   |                        |                  |                               |
| Inference Latency ( $\sim 1/\text{FPS}$ )  | $\leq 50$ ms per frame                       |        |              |                   |            |   |                        |                  |                               |
| Processing Speed   | $\geq 25$ FPS on 1080p stream                |        |              |                   |            |   |                        |                  |                               |
| <p>Inspected by</p>   |  |        |              |                   |            |   |                        |                  |                               |



### Daily Activities

| Date  | Department              |
|---|-------------------------|
| 29/06/2025  | Artificial Intelligence |
| <p>As the second day of the internship, after I was shown what I had to work on and the success metrics for model deployment yesterday, I had to work on collecting the data that I was given access to in the company archive. The raw data consisted of 1.2k existing line-camera images and 15 short videos (1080p, 30 frames per second). The line-camera images could be directly used theoretically, but the 15 short videos could not be directly processed by the system. So, I realized I had to turn each video to a specific number of image frames like the rest of the data. I looked up ways to turn a video to image frames and found too many options! My main question was how many image frames I should turn every video to, depending on the video length, and how to avoid duplicated frames extracted from the same video that look very similar and there could be just one of them. As I said, when I looked up the issue, I saw that depending on the domain of the project, there was different ways to extract image frames from a video. For instance, for training a model in the medical domain with insufficient data, each video had to be extracted in many frames and the "duplicated" data were left untouched because with insufficient data, every data point would be useful even if they looked very similar. But I could not find a similar project to what I was trying to build. I thought about asking my mentor about this issue but hesitated. Of course, he would help me if I asked but I thought this question was too simple and I could figure it out on my own, it was just about extracting image frames from a video and whether to deduplicate the extracted images or not. I thought I would later ask him for help in more complicated stages of the project. Then, I decided that I would extract video frames in every 5<sup>th</sup> frame, and I would also deduplicate data points that were too similar. After searching ways to deduplicate computer vision (visionary) data, I learned about average hash (ahash) deduplication method available in OpenCV library. Then, I started writing my image frame extraction and deduplication script. For the project structure, in my 'src' folder, I opened a 'data' folder and wrote my script 'prepare_dataset.py' in this path 'src/data'. This script would extract video frames every 5<sup>th</sup> frame (for each of the 15 short videos), and then deduplicate the extracted image frames using average hash (ahash) from the OpenCV library. Then, I organized folder scheme under 'data/raw' and 'data/interim'. In 'data/raw', as the name suggests, I kept the raw data exactly that I originally got them, just as a "read-only" copy of the data. After running my script, I put the processed dataset files in 'data/interim/'. I will put the processed data after other data cleanings, transformations or filtering here as well for the final ready-to-use dataset. I also logged the checksum so that I could later verify that the raw data had been intentionally transformed, not accidentally changed or corrupted. I also learned that logging checksum could ensure reproducibility, in case someone else downloads the same raw dataset and they want to confirm their checksum is identical.</p> |                         |
| <p>Inspected by</p>    |                         |



## Daily Activities

|                    |                                       |
|--------------------|---------------------------------------|
| Date<br>30/06/2025 | Department<br>Artificial Intelligence |
|--------------------|---------------------------------------|

After I collected data on the second day and transformed the short video clips to image frames (every 5<sup>th</sup> frame) and deduplicated identical frames (using average hash from OpenCV library), it was now time for the labeling setup. At first, I deployed Label Studio locally via docker-compose. Then, I created the label taxonomy. I set up 4 labels in total: 'crack', 'dent', 'scratch', 'ok'. 'ok' meant no defect. I imported 2.5k image frames to load the images into the labeling tool and then defined the labeling guidelines. I specified the minimum box size and occlusion rules. I wrote clear rules for the labeling criteria. The initial values I defined for the minimum bounding box was 20 x 20 px. For the occlusion rule, occlusion refers to cases where the object is partially hidden. I set the rule so in case an object is only partially visible, only the visible portion would be annotated. For a more robust system, I also ensured that the labeling guidelines included ambiguity handling and could still function in ambiguous cases. Here is the method I chose for handling ambiguous cases:

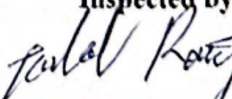
If length > 5x width then, mark as 'scratch'

Else, mark as 'dent'

I chose this rule for ambiguity because logically, if the length of the abnormal section detected by the system is much more (more than five times) the width, it could indicate that the defect is a scratch. Because scratches tend to be long in length and narrow in width. Otherwise, it would be classified as a dent. However, to detect a crack, would have a different criterion than scratch and dent. The visual characteristics of a crack, indicate that it is a thin and elongated region like a line and is often jagged or irregular, not perfectly straight. A crack can be differentiated with a scratch like so. A scratch and a crack both have bigger lengths than widths, however, a scratch is usually nearly straight, like a straight line. But a crack is not straight like a scratch. It has an irregular shape like a line broken in multiple parts. These differences can be easily learned by a computer vision deep learning model like YOLOv8n or Faster R-CNN. To make my system consistent during training, I gave clear logical rules for all the labels. You can see my initial labeling guidelines in the table below (Table 4):

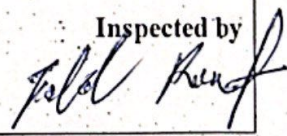
| Table 4: Logical labeling rules for surface defects classification |                                     |  |  |                                     |
|--|-------------------------------------|--|--|-------------------------------------|
| Defect Type  | Shape (Geometry)                    | Aspect Ratio & Size                            | Surface Contrast                           | Other Cues                          |
| Crack  | Thin, irregular line (not straight) | Very narrow width (<~10px), length >= 5x width | Strong Contrast Edges, sharp discontinuity | May branch or have uneven thickness |
| Dent   | Broad rounded                       | Aspect ratio~1                                 | Moderate Contrast                          | Smooth outline, not long            |
| Scratch  | Thin, roughly straight              | Narrow width, Length>5xwidth                   | Smooth outline                             | Uniform thickness                   |
| Ok (no defect)   | No irregular shapes                 | -  | Consistent contrast                        | No discontinuity                    |

Inspected by





## Daily Activities

|  |  |  |
|--|--|--|
| <b>Date</b><br>01/07/2025  | <b>Department</b><br>Artificial Intelligence |  |
| <p>Today, I performed exploratory data analysis (EDA) and worked on the data quality. I created a notebook named eda.ipynb. In this notebook, I looked at the class distribution and made histograms of bounding box sizes to see typical defect sizes. I also checked image quality in terms of brightness and sharpness to find blurry or dark images. My findings covered a class imbalance I found, scratches having fewer examples compared to dents and cracks. So, I decided to use data augmentation techniques such as flipping, rotating and brightness adjustment to balance the class imbalance. I used the fewer scratches data that I already had and tried to increase the data points by using data augmentation techniques. My goal was to boost scratch samples percentage in the train set. To do that, I had to preserve the thin, long and roughly straight geometry and mimic production artifacts such as line-scan, lighting and conveyor motion effect if applicable. My augmentation policy was to apply scratch-focused transforms only to images containing scratches and maintain box integrity and to always use libs that kept bboxes updated (Albumentations). I also had to do the geometric operations before resizing, in other words, do photometric ops after geometric ops. The table below (Table 5) shows the detailed transforms that I applied to the scratch data to fix the class imbalance:</p> |  |  |
| <p><b>Table 5: Scratch-specific data augmentation transforms with parameters for training balance</b></p>  |  |  |
| <b>Transform</b>   | <b>Params</b>                                | <b>Purpose</b>                               |
| Rotation   | $\pm 3-5^\circ$                              | Vary scratch orientation slightly            |
| Shear  | $\pm 0.03-0.06$                              | Distort aspect ratio without losing thinness |
| Scale  | 0.95–1.05                                    | Small size jitter                            |
| Motion blur  | $k=3-5$ , angle $\approx$ conveyor axis      | Simulate line-scan movement                  |
| Gaussian noise   | $\sigma=2-6/255$                             | Sensor noise                                 |
| Brightness/contrast  | $\pm 20\% / \pm 15\%$                        | Lighting variation                           |
| CLAHE  | clip=1.5–2.5                                 | Local contrast change                        |
| JPEG compression   | quality=60–90                                | Realistic artifacts                          |
| Copy-paste scratches   | Scale 0.6–1.4, rotate $\pm 7^\circ$          | Boost scratch count from real samples        |
| Procedural scratches   | Length 80–400 px, width 1–4 px               | Generate synthetic thin defects              |
| <p>Finally I sampled 50 ambiguous cases for mentor review and then, updated the guidelines accordingly.</p>  |  |  |
| <p>Inspected by</p>   |  |  |



## Daily Activities

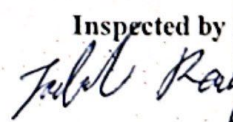
|                    |                                       |
|--------------------|---------------------------------------|
| Date<br>02/07/2025 | Department<br>Artificial Intelligence |
|--------------------|---------------------------------------|

After the exploratory data analysis (EDA), discovering the class imbalance, and fixing it with data augmentation techniques, my data was ready to be converted and split for train, validation and test sets. Firstly, I converted annotations because the labels made in Label Studio were in JSON format and had to be transformed into the YOLO format. This would be helpful as the bounding boxes would have been stored as normalized coordinates. Then, the converted files were organized under 'data/yolo' in the project source code. Then, the dataset was divided into 70% training, 20% validation and 10% testing. The same ratio of scratches, dents and cracks was roughly maintained in each subset. This was possible through stratification by class. To ensure reproducibility, a random number generator with a fixed seed was used so the split would always turn out the same. A unit test was also written to verify the convertor worked correctly. In order to help the model generalize, baseline augmentations were applied. The augmentations consisted of basic data transformations including random crop, which would make the visible part of the image frame vary, horizontal flip that mirrored the image on a horizontal axis, and Gaussian noise to add sensor-like noise to the images. In short, I standardized the dataset from JSON into YOLO format and made a reproducible and testable train/val/test split (70/20/10). I also set up simple augmentations to help the model generalize. In the table below (Table 6), the class distributions across train, validation and test splits after conversion to YOLO format is visible:

**Table 6: Class distribution percentages across train, validation, and test splits after conversion to YOLO format**

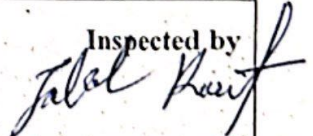
| Split            | Total Images | Crack (%) | Dent (%) | Scratch (%) | Ok (%) | Notes                           |
|------------------|--------------|-----------|----------|-------------|--------|---------------------------------|
| Train (70%)      | 1,750        | 25.7      | 34.3     | 11.4        | 28.6   | Includes baseline augmentations |
| Validation (20%) | 500          | 26.0      | 34.0     | 12.0        | 28.0   | Stratified by class             |
| Test (10%)       | 250          | 24.0      | 32.0     | 12.0        | 32.0   | Used for final evaluation       |

After converting the annotations, 70/20/10 split with stratification by class, ensuring the split was reproducible with seeded RNG and baseline augmentations for model generalization, my data was now ready to train my baseline model YOLOv8n.

Inspected by  




## Daily Activities

| <b>Date</b><br>05/07/2025   | <b>Department</b><br>Artificial Intelligence |                             |              |        |              |                             |        |                   |            |       |              |                   |                    |                       |              |                  |                       |        |              |
|---|--|-----------------------------|--------------|--------|--------------|-----------------------------|--------|-------------------|------------|-------|--------------|-------------------|--------------------|-----------------------|--------------|------------------|-----------------------|--------|--------------|
| <p>The second week of my internship started today! It was now time to train the baseline deep learning model, YOLOv8n. The baseline object detection model was to be trained using the YOLOv8n architecture. This architecture is optimized for faster training and inference and requires relatively lower computational power which makes the YOLOv8n architecture perfect for initial experimentations and pilot testing on the dataset of metal surface defects. The model was trained for 100 epochs with image sizes 640 x 640 pixels and a batch size of 32. I chose 640 px size because this image size preserved the defect details relatively well, good enough for small and thin classes like scratches and cracks detection, and it was not too big to require a large GPU memory usage. Additionally, the batch size being equal to 32 ensured estimating stable gradient while enabling the system to use the available GPU resources efficiently. The dataset for training was the YOLO-converted annotations created with stratified splits to make sure all defect classes were represented proportionally in the train, validation and test sets. Before the training process, baseline augmentations were performed. These baseline augmentations consisted of random crop, horizontal flip, brightness/contrast adjustments and Gaussian noise to make the training samples more variable and to prevent overfitting. Additionally, these augmentations were very helpful for fixing the class imbalance, for the underrepresented scratch class that had relatively few data points comparing to the other classes. After the training process, I evaluated the baseline performance metrics. The model achieved a mean average precision of 0.812 at IoU 0.5 (mAP@50). This indicated that the overall detection accuracy was high. The mAP at IoU 0.5-0.95 (mAP@50-95) was 0.478, which meant that the performance of the model across a stricter range of overlap thresholds the model struggled to correctly locate defects. I wanted to track the training process and ensure reproducibility. Thus, I logged all experiments to MLflow. I also stored training parameters, loss curves at each epoch and the evaluation metrics mAP@50 (for accuracy), inference latency and processing speed. The highest accuracy (mAP@50) achieved was 0.812 but the minimum accuracy requirement for deployment was 0.9. Of course, I planned to perform different optimization methods to further increase the accuracy such as tuning the hyperparameters. Table 7 shows the highest evaluation metrics achieved compared to the success metrics defined in the beginning of the internship:</p> <p><b>Table 7: Comparison of target success metrics with the baseline YOLOv8n model performance</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Metric</th> <th>Target Value</th> <th>Achieved (YOLOv8n Baseline)</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td>Accuracy (mAP@50)</td> <td><math>\geq 0.9</math></td> <td>0.812</td> <td>Below Target</td> </tr> <tr> <td>Inference Latency</td> <td><math>\leq 50</math> ms/frame</td> <td>~36 ms/frame (28 FPS)</td> <td>Meets target</td> </tr> <tr> <td>Processing Speed</td> <td><math>\geq 25</math> FPS (1080p)</td> <td>28 FPS</td> <td>Meets Target</td> </tr> </tbody> </table> <div style="text-align: right; margin-top: 20px;"> <p>Inspected by</p>  </div> |  |                             |              | Metric | Target Value | Achieved (YOLOv8n Baseline) | Status | Accuracy (mAP@50) | $\geq 0.9$ | 0.812 | Below Target | Inference Latency | $\leq 50$ ms/frame | ~36 ms/frame (28 FPS) | Meets target | Processing Speed | $\geq 25$ FPS (1080p) | 28 FPS | Meets Target |
| Metric  | Target Value                                 | Achieved (YOLOv8n Baseline) | Status       |        |              |                             |        |                   |            |       |              |                   |                    |                       |              |                  |                       |        |              |
| Accuracy (mAP@50)   | $\geq 0.9$                                   | 0.812                       | Below Target |        |              |                             |        |                   |            |       |              |                   |                    |                       |              |                  |                       |        |              |
| Inference Latency   | $\leq 50$ ms/frame                           | ~36 ms/frame (28 FPS)       | Meets target |        |              |                             |        |                   |            |       |              |                   |                    |                       |              |                  |                       |        |              |
| Processing Speed  | $\geq 25$ FPS (1080p)                        | 28 FPS                      | Meets Target |        |              |                             |        |                   |            |       |              |                   |                    |                       |              |                  |                       |        |              |



## Daily Activities

| Date  | Department              |
|---|-------------------------|
| 06/07/2025  | Artificial Intelligence |
| <p>I successfully trained the YOLOv8n baseline model yesterday and two of the metrics (inference latency and processing speed) met the target. However, the accuracy (mAP@50) was still below target. The highest achieved accuracy was 0.812 which was still far from the target accuracy, 0.9. So today, I wanted to find strategies to improve the model performance. To do that, I tried to examine the model's predictions more carefully so that I would understand where it succeeded and where it struggled and failed. I wrote a utility script to visualize the model's results in 'src/utlis/viz.py' to visualize false negatives (FN) and false positives (FP). False negatives occurred when the model missed a defect that was present, and false positives happened when there was no defect (of that specific type) but the model falsely classified it as a defect that did not exist. False negatives meant that defective parts might have passed inspection incorrectly and false positives meant that good parts would be falsely flagged as defective which would increase the operational costs. Sample predictions were overlaid on test images with ground-truth annotations for comparison. By using the visualization tool, it became obvious that most of the false negatives belonged to parts with small or thin scratches. On the other hand, cracks and dents often had more distinct shapes and stronger visual contrast which made them easier to detect for the YOLOv8n model than scratches. Because scratches tended to appear as long lines with very small widths, in many cases only a few pixels thick, which made them difficult to detect for the YOLOv8n model's grid-based detection architecture. Additionally, some false positives were detected where there was subtle surface textures or variations in lightnings that caused incorrect classification as scratches. This indicated that the model tended to confuse defects with low-contrast features. To solve these issues, I used test-time augmentation (TTA). This method works by applying basic image transformation like scaling, rotations or flips to an image input one at a time during model inference. The model was run on each variation and then the results were combined. As I had learned in Ensemble learning in the university, I tried to apply the same idea in this TTA. I reckoned that the ensemble of predictions across augmented versions could possibly reduce errors and improve robustness. This solution would be even more helpful for subtle defects that were more difficult to detect like scratches. The ensemble solution worked! It was indicated by the experiments that test-time augmentation led to an improvement in recall, in other words, the model was now able to detect more true scratches compared to the baseline. However, the cost of this improvement was that it increased latency significantly as each frame now required multiple forward passes through the model. I discussed this issue with my mentor, and we decided that I would switch to a more capable model, YOLOv8m (medium). I switched from YOLOv8n (nano) to YOLOv8m (medium). YOLOv8m provided the system with richer feature extraction and therefore, a higher performance.</p> |                         |
| Inspected by<br>   |                         |

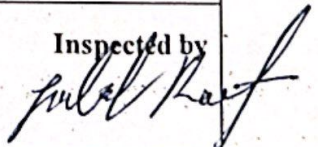


## Daily Activities

| Date   | Department              |
|--|-------------------------|
| 07/07/2025   | Artificial Intelligence |
| <p>Today, I tried two different strategies to further improve the detection pipeline: anchor tuning and augmentation expansion. I targeted weaknesses observed in earlier experiments, especially, the under-detection of thin or small scratches and the variations in lightning conditions. I started with anchor tuning. In object detection models like YOLO, anchors are prior bounding box shapes that help the network at the training stage to localize objects better. There is a default anchor set but it is too generic and may not work well with the specific distribution of object sizes in the dataset I was using. Because this dataset, included a mixture of cracks, dents, scratches and intact surfaces, and the scratches often looked like thin long lines. The default anchors were not optimal. Thankfully, YOLOv8 has a built-in auto-anchor tuning feature. This utility generates optimal anchors according to the data contribution by analyzing the dataset's bounding box aspect ratios and sizes. By using this feature, the new set of anchors put an emphasis on the small and narrow boxes. This helped the detection of scratches and "hairlike" cracks. Thanks to this adaption, the model started training with anchors that reflected the real-world defect geometries of the data points. There was a significant improvement in the recall rates. Cases that were previously missed as the default original anchors where the model was unable to capture small dimensions, were now consistently detected. This improvement created a better baseline for the fine-tuning stage of the model.</p> <p>I had already performed basic augmentations such as crops, flips, Gaussian noise and brightness &amp; contrast adjustments. However, the model still showed a relatively low performance under various real-world conditions. For instance, sometimes the scratches were undetectable in low-light images, or they were blurry when in motion. Thus, I used three advanced techniques to expand the augmentation pipeline:</p> <ol style="list-style-type: none"><li>1. <b>CutOut:</b> By randomly taking out rectangular regions of the images to make the model to concentrate on incomplete visual cues to help improve robustness of the model in occlusion cases where the defect was partially hidden.</li><li>2. <b>MotionBlur:</b> This technique simulated motion artifacts by convolving the image by using a directional blur kernel. This augmentation technique mimicked cases where the object was moving and blurry in the image. This is common in conveyor belts or handheld inspection setups.</li><li>3. <b>CLAHE (Contrast Limited Adaptive Histogram Equalization):</b> This technique is a local contrast-enhancement method that is very useful for low-light or surfaces with uneven lightning. CLAHE prevents over-amplifying noise while making faint scratches easier to detect for the model.</li></ol> <p>By using the built-in auto-anchor tuning feature of YOLOv8, and using the pipeline augmentation strategies, there was clear improvements in the model's performance:</p> <ul style="list-style-type: none"><li>• mAP@50 increased from 0.812 in the baseline, to 0.874.</li><li>• mAP@50-95 rose to 0.553 from 0.478 (better localization across IoU thresholds).</li><li>• Recall improved by +5.2% absolute, addressing the under-detection issue for thin scratches.</li></ul> |                         |
| <p>Inspected by<br/><i>Fahad Raza</i></p>  |                         |



## Daily Activities

| Date   | Department              |
|--|-------------------------|
| 08/07/2025   | Artificial Intelligence |
| <p>The model still had relatively high false positive (FP) rate, particularly in undamaged metal parts that were incorrectly flagged as defective. Even though the recall for scratches and other defects had been highly improved due to anchor tuning and augmented datasets, the model was still over-sensitive to subtle textures or natural light variations in metal surfaces that were falsely classified as defects. This tendency caused unnecessary alerts in evaluation and would potentially pose challenges in a real deployment pipeline. Because in that case, each false positive could lead to wasted inspection time or increased manual verification. Thus, to mitigate the false positive (FP) rate, I decided to use hard negative mining. The process started with analyzing the validation set systematically to find the samples where the model produced high-confidence false positives. I filtered predictions where the confidence exceeded 0.5 but did not overlap with any ground truth annotations (<math>\text{IoU} &lt; 0.2</math>). Such cases mostly consisted of surface texture irregularities, reflections or marks that resembled scratches visually but were not true defects.</p> <p>I assembled a set of 400 additional "clean" parts that I had intentionally chosen because of their misclassification in prior runs, they would represent the hard negatives. Hard negatives are the examples that challenge the current decision boundary of the model. The selected images were the edge cases that exposed the weaknesses of the current YOLOv8m baseline. I wanted to integrate these hard negatives effectively, so I modified the training sampler to implement weighted sampling. I did this so that the newly added hard negatives would not get lost and would be well-classified negative examples.</p> <p>I then retrained the same YOLOv8m configuration (image size 640, batch size 32, 80 epochs) to make the current experiment comparable with earlier experiments. Including the hard negatives, quickly showed the benefits, the model was now more conservative when detecting defects on ambiguous textures. In the confusion matrix false positive counts for scratches were substantially reduced and validation false positives decreased by 18% relative to the baseline.</p> <p>Additionally, the visualization of the inference results demonstrated that the bounding boxes of the model were now more tightly aligned to actual defect regions on the metal parts. Also, I got feedback from the mentor, confirming that the model's predictions were now more aligned with the labeling guidelines with fewer phantom defects. Today was an important step in moving beyond accuracy metrics and focusing on carefully analyzing the errors and the performance of the model. By incorporating 400 challenging "clean" examples into training with weighted sampling, the model achieved an 18% drop in validation FPs and the model's qualitative prediction stability was improved which enhanced its suitability for deployment. I learned that targeted data-centric strategies, apart from architectural changes, could yield significant improvements in the practical performance of the model.</p> |                         |
| Inspected by<br>  |                         |



## Daily Activities

|             |                         |
|-------------|-------------------------|
| <b>Date</b> | <b>Department</b>       |
| 09/07/2025  | Artificial Intelligence |

Today was the last day of the second week of my internship at Bina Sanat Veera. When I saw how much changing the model from YOLOv8n (nano) to YOLOv8m (medium) improved the performance, I thought about exploring different model options as well. The alternative baseline model that I chose to validate whether the performance could be further improved by changing the model again, was Faster R-CNN. I implemented Faster R-CNN with a ResNet50 backbone and a Feature Pyramid Network (FPN) using the torchvision library. I chose Faster R-CNN as the alternative baseline because Faster R-CNN is a another widely adopted benchmark in object detection research.

To implement Faster R-CNN, I used 'torchvision.models.detection.fasterrcnn\_resnet50\_fpn' which came with pretrained COCO weights that allowed for transfer learning. I made sure the early convolutional layers were frozen so that low-level feature representations would be maintained when fine-tuning the higher-level layers on the scratch detection dataset. I trained the model for 30 epochs with the same image size (640 x 640) and the same data augmentation techniques utilized in the previous experiments to make sure the comparison between the Faster R-CNN and YOLOv8m was fair. I used a batch size of 8 images; this was a constraint due to GPU memory requirements as Faster R-CNN has a significantly larger memory footprint than YOLO.

After 30 epochs, Faster R-CNN achieved a mean Average Precision (mAP@50) of 0.842. This result was slightly higher than YOLOv8n (0.812) but lower than the tuned YOLOv8m model (0.874). The mAP@50-90 score was also lower than YOLOv8m's, indicating that there was a challenge with detecting very thin scratches, like the original YOLOv8n model. However, more balanced precision and recall across classes were exhibited by Faster R-CNN. Especially, recall for the "scratch" category improved compared to the initial YOLOv8n but it was unable to surpass the observed gains with YOLOv8m after the improvements and anchor tuning.

A drawback of Faster R-CNN was its inference / processing speed. On the same GPU hardware, the model achieved an inference speed of approximately 12 frames per second (FPS) on 1080p streams. This corresponded to an inference latency of about 83 milliseconds per frame which was far above the target metric. By contrast, YOLOv8m operated above 25 FPS, nearly double the processing speed. Therefore, I chose to keep the model as YOLOv8m. In the table below (Table 8), you can see the comparison between the metrics of YOLOv8m, Faster R-CNN & the defined success metrics:

**Table 8: Comparison table among YOLOv8m, Faster R-CNN and the target metrics**

|                          | YOLOv8m | Faster R-CNN  | Target Metrics   |
|--------------------------|---------|---------------|------------------|
| <b>Precision(mAP@50)</b> | 0.874   | 0.842         | ≥ 0.9            |
| <b>Processing Speed</b>  | 25 FPS  | ~ 12 FPS      | ≥ 25 FPS (1080p) |
| <b>Inference Latency</b> | 40 FPS  | ~ 83 ms/frame | ≤ 50 ms/frame    |

Inspected by

*Subal Muf*



## Daily Activities

Date12/07/2025

DepartmentArtificial Intelligence

After experimenting with Faster R-CNN and deciding to keep YOLOv8m as the primary baseline, the next step was hyperparameter tuning to further push performance to reach the success metrics. At this phase, the focus was on 3 major layers: learning rate (LR), weight decay and mosaic augmentation probability.

**Learning Rate & Weight Decay**

I believe that the learning rate is the most important hyperparameter, because it directly influences stability and convergence speed. A learning rate (LR) too high may cause divergence, whereas a learning rate (LR) too low may lead to slow convergence or suboptimal minima. I tested the values in the range [0.0001, 0.003]. Additionally, I adjusted weight decay to control overfitting, testing values in the range [0.0001, 0.0005]. Among the experiments, there were values that produced smoother loss curves and improved recall for difficult classes like "scratch". This case happened when the learning rate (LR) was equal to 0.002 with weight decay = 0.00025. When I tried lower LR values, they caused underfitting and higher LR values led to unstable training after epoch 50.

**Mosaic Augmentation Probability**

Mosaic augmentation refers to combining four images into one. This is very helpful for boosting small object detection due to increased contextual variety. However, if the mosaic probability is set too high it can produce unrealistic samples and confuse the model about object scales. Thus, I tested this sweep of mosaic probabilities [0.3, 0.5, 0.7]. As shown by the results, 0.5 was the right balance for the mosaic probability. With the lower probability (0.3), there was a stagnation in recall, which possibly indicated that the exposure to diverse contexts was insufficient. At the higher probability (0.7), because of over-distorted training sample, the precision declined. Therefore, 0.5 was chosen as the optimal mosaic probability.

The combination of a learning rate (LR) of 0.002, the weight decay of 0.00025 and a mosaic probability of 0.5, produced the best results across all tested runs. The final metrics after this configuration were:


- mAP@50 = 0.893
- mAP@50-95 = 0.583

There was a significant improvement compared to earlier baselines. Recall improved significantly on the challenging scratch class, indicating that a balanced augmentation provided the model with better representations of subtle defect patterns. In Table 9, the improved precision metric for the YOLOv8m model before and after hyperparameter tuning is compared:

**Table 9: mAP@50 & mAP@50-95 comparison before hyperparameter tuning vs after YOLOv8m Model**

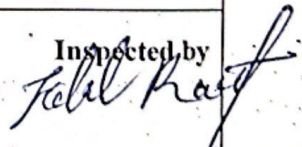
| YOLOv8m Model      | Before Hyperparameter Tuning | After Hyperparameter Tuning |
|--------------------|------------------------------|-----------------------------|
| Precision (mAP@50) | 0.874                        | 0.893                       |

Inspected by





## Daily Activities

| Date   | Department              |
|--|-------------------------|
| 13/07/2025   | Artificial Intelligence |
| <p>As discussed before, the dataset contained a class imbalance, there was more examples of dents and cracks comparing to scratches. I had used several data transforms, and it had mitigated the issue. However, I wanted to completely address this problem. This class imbalance was what caused weaker detection performance in scratch detection. Today, I tried to address this issue by using a combination of targeted augmentations and modifying the loss function.</p> <p>The first major adjustment I did, was bringing focal loss into the YOLO training configuration. The default loss function for YOLO classification is binary cross-entropy (BCE) loss function. BCE loss function usually does not function well where a strong skew exists between positive and negative samples. Because the "non-scratch" regions in this dataset outnumbered the others, the network was biased against the minority "scratch" class. By using focal loss instead of binary cross-entropy loss function, focal loss down-weighted easy examples and emphasized more difficult and previously misclassified examples. Additionally, a gamma value of 1.5 was selected after preliminary testing. This made sure that the network gave more gradient signal to underrepresented or difficult cases. This would help the model learn differentiating features for scratches.</p> <p>In addition to focal loss, I used a targeted augmentation strategy to imitate the challenging characteristics of a scratch. I used an augmentation strategy which is designed to stimulate long structures, ThinObjectAug. The diversity of the "scratch" class increased significantly by synthetic thin marks overlays that resembled scratches onto clean surfaces in the training data. It is important to note that I applied ThinObjectAug to the scratch subset only, in order to prevent distortions in other categories.</p> <p>Eventually, by combining focal loss and ThinObjectAug, there was a significant improvement in the performance of the model. The F1 score for the "scratch" class improved from 0.71 to 0.79 and there was a substantial improvement in both the recall and precision. The network now was more confident in its positive predictions, and it no longer missed the true scratch cases.</p> <p>Today was an important day for systematically solving the class imbalance issue, this time, in a more complete way. The model's weakest class where the model made the most mistakes, was the "scratch" class and by using focal loss and ThinObjectAug, I was able to improve the performance of the model on this class significantly. This upgrade enhanced the reliability and robustness of the system in real-world scenarios as the system was now able to detect subtle scratches as well as the other defects. I learned a lot about addressing the class imbalance issues and what they could cause in the performance of the model. It is important to examine which classes the model performs better on and which classes the model struggles with, as every nuance in a model's behavior has a logical reason that should be studied.</p> |                         |
| Inspected by<br>  |                         |



## Daily Activities

| Date  | Department              |                 |               |       |            |
|---|-------------------------|-----------------|---------------|-------|------------|
| 14/07/2025  | Artificial Intelligence |                 |               |       |            |
| <p>One of the challenges of the project was the limited diversity of how often the defects appeared in the data. Especially, for rare cases such as faint scratches, micro-dents and low-contrast blemishes. The baseline refinements, the data augmentations, the anchor tuning and focal loss had significantly improved the model performance. The problem was now obvious. The problem was with the nature of the dataset itself because it misrepresented the true defect distribution that would normally be encountered during production. Now that I knew the exact problem, I was able to pivot a deliberate solution towards it. The main solution that I needed was data expansion. To expand the dataset, I used active learning as the main method to maximize the efficiency in the labeling process and to ensure that the data that was newly added would carry the highest possible information gain.</p> <p>I started the dataset expansion by deploying the current YOLOv8m model on an unlabeled stream of data that resembled actual production. For simpler work, I could have randomly sampled additional frames for annotation but then, it would have resulted in lots of redundant "easy" cases that would not have been very beneficial for improving the model performance. My selection criterion was prediction uncertainty. In other words, I ensured that the examples which the network struggled with were annotated as high-value samples. A good example of such a case was when the model approximately assigned equal confidence scores to "clean" vs "scratch". To conduct this, 500 additional frames were selected for labeling. These 500 frames formed the targeted dataset that included a significant rate of rare appearances. The rare appearances consisted of scratches that were captured with oblique angles under improper lightning, subtle inconsistencies in textures that were confused with background noise and partially occluded cracks. After these considerations and improvements, the absolute training volume was increased and substantially improved as the expanded data now represented a more realistic rate of real-world cases. After the additional frames were labeled and validated, the dataset was now ready to be used to retrain the YOLOv8m model. This data-driven refinement resulted in a substantial leap in the network performance as the retrained model now achieved a precision mAP@50 of 0.905 and was now passing the target accuracy metric! This was definitely a milestone in my internship! I learned that in cases where there are rare class under-representations in the data, for addressing this issue, I could use the active-learning approach and put strategic emphasis on the rare classes. I also learned that while algorithmic refinements can yield the model performance (such as hyperparameter tuning and loss function modifications), the best strategy to increase the model robustness is data-related refinements. Because it was through the data-related refinements that the model was able to reach and even surpass the pre-defined success metric for mAP@50. Table 10 shows the achieved precision and the target precision of the latest version of YOLOv8m.</p> |                         |                 |               |       |            |
| <p><b>Table 10: Comparison between the achieved and the target accuracy for YOLOv8m</b></p> <table> <tr> <th>Achieved mAP@50</th><th>Target mAP@50</th></tr> <tr> <td>0.905</td><td><math>\geq 0.9</math></td></tr> </table>  |                         | Achieved mAP@50 | Target mAP@50 | 0.905 | $\geq 0.9$ |
| Achieved mAP@50   | Target mAP@50           |                 |               |       |            |
| 0.905   | $\geq 0.9$              |                 |               |       |            |
| <p style="text-align: right;">Inspected by<br/><i>Zohel Raza</i></p>  |                         |                 |               |       |            |



## Daily Activities

| <b>Date</b><br>15/07/2025   | <b>Department</b><br>Artificial Intelligence |   |                                   |   |        |           |               |        |           |       |       |              |        |           |       |       |               |        |           |       |       |                |                    |                       |            |   |
|---|--|---|-----------------------------------|---|--------|-----------|---------------|--------|-----------|-------|-------|--------------|--------|-----------|-------|-------|---------------|--------|-----------|-------|-------|----------------|--------------------|-----------------------|------------|---|
| <p>My focus was now shifted from improving the training process to deploying the model and optimizing the runtime. This stage was vital for making the system ready for production. At first, I identified the training checkpoint with the highest accuracy, which had a mAP@50 of 0.905, and then I exported it to an ONNX (Open Neural Network Exchange) format because this format provides a standardized representation of deep learning models and it makes the models and networks portable across inference engines and frameworks. I exported the model to Open Neural Network Exchange (ONNX) to ensure that the trained YOLOv8m model was able to use optimized inference backends and was not locked into PyTorch alone. After I exported the ONNX model, I evaluated it under ONNX runtime to examine its stability and speed comparing to PyTorch. As shown by the ONNX runtime, the accuracy metrics remained nearly identical with the PyTorch version which indicated that the conversion was done correctly, and the results were consistent. Additionally, this was an opportunity to test the model performance in a framework that was better aligned with deployment in industrial pipelines.</p> <p>The main improvements appeared afterwards when I was experimenting with the NVIDIA TensorRT FP16 optimization. TensorRT is a highly performing tool for optimizing inference and runtime that is specifically designed for NVIDIA GPUs. By benchmarking, I saw that the inference latency decreased by nearly 37% as it dropped from ~54 ms/frame to ~34 ms/frame. This was a milestone as the real-time target of <math>\leq 50</math> ms per frame was now met! To clearly show a comparison among the inference runtimes and the target metrics based on today's experiments, I made the table below (Table 11):</p> <p><b>Table 11: Experiments runtimes and target metrics comparison</b></p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr style="background-color: #007bff; color: white;"> <th>Runtime</th> <th>Inference Latency per Frames (ms)</th> <th>Processing Speed (Frame per Second (FPS))</th> <th>mAP@50</th> <th>mAP@50-95</th> </tr> </thead> <tbody> <tr> <td style="background-color: #007bff; color: white;">PyTorch Eager</td> <td>~54 ms</td> <td>~18.5 FPS</td> <td>0.905</td> <td>0.583</td> </tr> <tr> <td style="background-color: #007bff; color: white;">ONNX Runtime</td> <td>~48 ms</td> <td>~20.8 FPS</td> <td>0.905</td> <td>0.583</td> </tr> <tr> <td style="background-color: #007bff; color: white;">TensorRT FP16</td> <td>~34 ms</td> <td>~29.4 FPS</td> <td>0.904</td> <td>0.582</td> </tr> <tr> <td style="background-color: #007bff; color: white;">Target Metrics</td> <td><math>\leq 50</math> ms/frame</td> <td><math>\geq 25</math> FPS (1080p)</td> <td><math>\geq 0.9</math></td> <td>-</td> </tr> </tbody> </table> <p>The accuracy remaining nearly identical across runtimes, shows that FP16 optimization preserves detection quality even though it provides a trade-off between speed and accuracy.</p> |  | Runtime                                   | Inference Latency per Frames (ms) | Processing Speed (Frame per Second (FPS)) | mAP@50 | mAP@50-95 | PyTorch Eager | ~54 ms | ~18.5 FPS | 0.905 | 0.583 | ONNX Runtime | ~48 ms | ~20.8 FPS | 0.905 | 0.583 | TensorRT FP16 | ~34 ms | ~29.4 FPS | 0.904 | 0.582 | Target Metrics | $\leq 50$ ms/frame | $\geq 25$ FPS (1080p) | $\geq 0.9$ | - |
| Runtime   | Inference Latency per Frames (ms)            | Processing Speed (Frame per Second (FPS)) | mAP@50                            | mAP@50-95                                 |        |           |               |        |           |       |       |              |        |           |       |       |               |        |           |       |       |                |                    |                       |            |   |
| PyTorch Eager   | ~54 ms                                       | ~18.5 FPS                                 | 0.905                             | 0.583                                     |        |           |               |        |           |       |       |              |        |           |       |       |               |        |           |       |       |                |                    |                       |            |   |
| ONNX Runtime  | ~48 ms                                       | ~20.8 FPS                                 | 0.905                             | 0.583                                     |        |           |               |        |           |       |       |              |        |           |       |       |               |        |           |       |       |                |                    |                       |            |   |
| TensorRT FP16   | ~34 ms                                       | ~29.4 FPS                                 | 0.904                             | 0.582                                     |        |           |               |        |           |       |       |              |        |           |       |       |               |        |           |       |       |                |                    |                       |            |   |
| Target Metrics  | $\leq 50$ ms/frame                           | $\geq 25$ FPS (1080p)                     | $\geq 0.9$                        | -   |        |           |               |        |           |       |       |              |        |           |       |       |               |        |           |       |       |                |                    |                       |            |   |
| Inspected by<br>  |  |   |                                   |   |        |           |               |        |           |       |       |              |        |           |       |       |               |        |           |       |       |                |                    |                       |            |   |



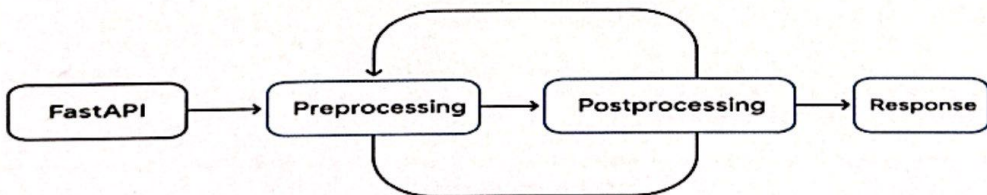
## Daily Activities

| Date       | Department              |
|------------|-------------------------|
| 16/07/2025 | Artificial Intelligence |

Today was the last day of the third week of my internship, meaning that after today, I would have completed half of my summer internship! I had worked well on my project until now, because all the success metrics that had been defined by the mentor in the beginning of the internship were now met. It was now time for preparing the YOLOv8m detection model for production deployment. My focus today was on designing an inference API that was robust and would allow a client application to interact with the trained model in a standardized manner. I chose the FastAPI framework for this task because this framework is well-suited to build a high-performance API with data validation and OpenAPI documentation. First, I set up the service structure in app.py at 'src/serving/app.py' and I implemented two main endpoints:

1. /health endpoint  
I used this route to monitor the status of the service. This endpoint performed a simple check for verifying that the FastAPI application was running and to confirm successful loading of the model into memory. The response of this endpoint was a simple JSON message which indicated if the service was ready and a timestamp for traceability.
2. /infer endpoint  
This endpoint was the main inference interface. It accepted input formats of raw image files such as JPEG and PNG, or raw frame bytes. This way, the system would work with both image inputs and real-time video streams from line cameras. When there was a request in the system, firstly the API would perform preprocessing. The preprocessing included image resizing in case the size was something other than 640px and normalization to the pixel ranges expected. The preprocessing was crucial for maintaining high accuracy as it provided consistency between training and inference inputs. A JSON message containing the class of the input {"scratch", "dent", "crack", "ok"} would be sent and then the respective alert message would be shown on the client app.

Figure 3 shows the API workflow in detail.



```
graph LR; FastAPI[FastAPI] --> Preprocessing[Preprocessing]; Preprocessing --> Postprocessing[Postprocessing]; Postprocessing --> Response[Response]; Preprocessing --> Preprocessing; Postprocessing --> Preprocessing;
```

Figure 3: API Workflow Diagram

Inspected by *Jalel K.*



## Daily Activities

| Date   | Department              |
|--|-------------------------|
| 19/07/2025   | Artificial Intelligence |
| <p>Today, the 4<sup>th</sup> week of my internship started. My API was designed, and it was now time for one of the most important steps of the deployment stage, containerization. The entire system of the machine learning inference service could become reproducible, secure and portable by packaging the inference pipeline into a container image. Thus, today I concentrated on creating a Docker image and enabling GPU acceleration for the model inference.</p> <p>In the beginning, I created a Dockerfile and I did not use a generic Python base image. I wanted to ensure GPU compatibility, so I used an NVIDIA CUDA base image. Because I chose this image, CUDA and cuDNN dependency management got simpler too and this was very important for PyTorch and TensorRT inference. I wanted to conduct local deployment and GPU integration, so I configured a docker-compose.yml file. This decision simplified declaring runtime requirements.</p> <p>After I successfully built and deployed the container image, it was now time to validate the performance under load. I used the tool hey, which is a lightweight HTTP load tester, so that it could simulate multiple clients sending requests to the inference endpoint. Latency distribution remained consistent even when there were multiple requests sent simultaneously to the endpoint. To further explain the test case I designed, the sustained load consisted of 6 requests per second (RPS) each sending a single image frame that was processed with a batch size of 1. As indicated by the results, the service remained stable. This meant that GPU acceleration and container networking were able to handle the demand gracefully.</p> <p>This stage of the project showed a critical advancement as the system could now run inference in isolated and portable environments that resembled production. The container could be shipped to different platforms such as cloud providers, GPU clusters or edge servers. Additionally, docker-compose made the system flexible for extending the setup with more services like monitoring, logging or autoscaling. In the diagram below (Figure 4), you can see the stages of the Dockerized workflow.</p> <pre>graph TD; CR[Client Request] --&gt; FA[FastAPI]; FA --&gt; P[Preprocessing]; P --&gt; DDM[Defect Detection Model]; DDM --&gt; R[Response]; Docker[Docker] --- P; GPU[GPU] --- DDM;</pre> <p><b>Figure 4: Dockerized Workflow Diagram</b></p> |                         |
| <p>Inspected by</p> <p><i>Jaleel Rauf</i></p>  |                         |



## Daily Activities

|                    |                                       |
|--------------------|---------------------------------------|
| Date<br>20/07/2025 | Department<br>Artificial Intelligence |
|--------------------|---------------------------------------|

Today, I wanted to upgrade the model so that the input would transition from image inference to video stream inference. This could have been one of the most important upgrades of the system to make it production ready. Until now, the system functioned well with image frame inputs and even if there were short video clips, the videos would be transformed to a set of images as it went through the frame extractor. But that would not have operated well in a real-life production quality control system. Because the cameras send live video streams and immediate analysis is expected from the system. Introducing the video stream inference to the system would bring upon additional complexity for handling the frames and even managing the memory. So today, I decided to establish a robust video ingestion pipeline which would handle backpressure and use the available GPU resources efficiently as well as keeping the memory stable.

Firstly, I designed the reader with bounded buffering. Bounded buffering puts the incoming input frames in a queue, and the queue has a maximum size. In case there is a lag in the inference pipeline, the older frames would be dropped for the benefit of the newer and fresher ones. This strategy enables the system to remain responsive as it prioritizes being responsive in real-time to completeness and perfection of each inference.

The second step was about batching. After the live incoming frames would enter the inference stage, instead of being inferred one by one which is inefficient according to the GPU computational cost, they would be analyzed in groups (in small batches). It is important to note that modern GPUs tend to have a higher efficiency when they process small batches compared to when they process one by one because processing in small batches improves their usage of parallel computer resources. Therefore, I grouped batches of 4 frames whenever possible in the live video stream inference.

The end-to-end throughput of the batched inference pipeline was ~28 FPS. System collapse under heavy load was prevented due to this design because of the stabilization of the input queue as no runaway memory accumulated.

Latency per frame varied according to the batching:

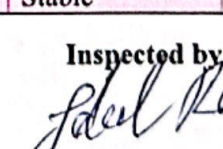
- Single-frame inference was ~35 ms/frame but underused the GPU resources.
- Four-frame batch had a latency per frame of ~42 ms/frame which was slightly higher but had a better throughput.

I also tried a batch size of two. In Table 12, the single-frame, two-frame batch and four-frame batch inference modes are compared in terms of average latency per frame, throughput and memory stability:

**Table 12: Inference modes comparison**

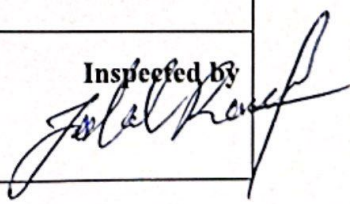
| <i>Inference Mode</i> | <i>Batch size</i> | <i>Avg. Latency (ms/frame)</i> | <i>Throughput (FPS)</i> | <i>Memory Stability</i> |
|-----------------------|-------------------|--------------------------------|-------------------------|-------------------------|
| <i>Single-Frame</i>   | 1                 | ~35 ms                         | ~20 FPS                 | Stable                  |
| <i>Two-Frame</i>      | 2                 | ~38 ms                         | ~24 FPS                 | Stable                  |
| <i>Four-Frame</i>     | 4                 | ~42 ms                         | ~28 FPS                 | Stable                  |

Inspected by



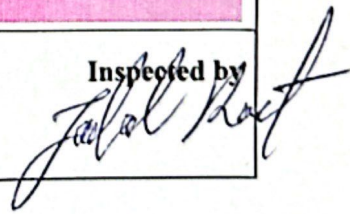


## Daily Activities

| Date  | Department              |
|---|-------------------------|
| 21/07/2025  | Artificial Intelligence |
| <p>After I successfully established a fully functional inference pipeline and deployed it in a containerized environment, it was now time for ensuring the system was robust, predictable and reliable under real-world cases (like what I had learned about Trustworthy AI in one of the university courses). As machine learning models may fail unpredictably when they see inputs that do not resemble what they had faced during training, I considered quality assurance (QA) practices. I used 4 different strategies in total: Input Sanitation, Error Handling, Unit Testing and Corruption Testing.</p> <ol style="list-style-type: none"><li>1. Input Sanitation<ul style="list-style-type: none"><li>• I rejected any image greater than 10 MB to prevent users from accidentally or maliciously overwhelming the system.</li><li>• Images with unusual aspect ratios, such as images with extremely tall panoramas or narrow strips) were rejected. I did this because the model was trained on a dataset with relatively natural aspect ratios. I also designed the error message to clearly indicate the problem.</li></ul></li><li>2. Defensive Error Handling<p>I used try/except blocks so that the specific exception would be captured and a detailed text message for debugging would be output including the request metadata and internal tracebacks. Additionally, a structured JSON message would be returned to the client field. Below, is an example of the structured error message that the client would see:</p><pre>{<br/>  "error": "InvalidInput",<br/>  "message": "Image exceeds 10MB limit",<br/>  "code": 400<br/>}</pre><p>By doing this I made sure that the clients would get feedback instead of generic errors. A catalog of these error messages would also serve as an observability layer as it could possibly show recurring issues or even attack patterns.</p></li><li>3. Unit Tests for Preprocessing<p>As many failures in machine learning models tend to rise from the data preprocessing stage, including the resizing, normalization or transform stages, I decided to write unit tests targeting the preprocessing pipeline. I wrote tests to assert correct resizing without distortion. I also wrote tests to validate consistent normalization outputs of the pixel ranges.</p></li><li>4. Synthetic Corruption Tests<p>I simulated corrupted inputs to see how the model would behave with such inputs.</p><ul style="list-style-type: none"><li>• I used Gaussian blur to check the model's performance on defocused images.</li><li>• I used Gaussian noise to emulate camera sensors with a low quality.</li></ul></li></ol> |                         |
| <p>Inspected by </p>   |                         |



## Daily Activities

| Date  | Department                          |               |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |
|---|-------------------------------------|---------------|-------------|---------------|-------------|---------------------|----------|-------------|-------------------------------------|----------|------------------|----------------------------|-----------|---------------------|-----------------------------------|--------|----------------|--------------------------|--------|
| 22/07/2025  | Artificial Intelligence             |               |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |
| <p>It is important to note that to deploy a model in production, it not only requires achieving a high accuracy, but also ensuring observability, reliability and stability. Today I focused on telemetry, visualization and alerting hooks into the system to ensure that the model's performance could be continuously tracked. The first step was integrating Prometheus. First, I had to export every request latency in a format that it would be readable and consumable by Prometheus. The format I used was textfile collector. By using this format, Prometheus was able to periodically export metrics such as request latency and throughput into a file that Prometheus node exporter could capture. I made sure that each metric was timestamped and labeled with request type (i.e., /infer or //health). This enabled tracking the performance without the need to use complicated dependencies inside the FastAPI container.</p> <p>Latency was the most crucial metric because inference workloads involved GPU computations and preprocessing and postprocessing overhead. By tracking the distribution of request times, it became obvious whether the service was healthy or not. The Latency percentiles that were exported were 950, p95 and p99. I did not use averages because averages tend to hide outliers even though analyzing outliers is crucial in real-time applications.</p> <p>After the metrics were exported, it was visualization's turn. I set up a lightweight Grafana panel to show:</p> <ul style="list-style-type: none"><li>• A rolling line chart that captured whether most of the requests were remaining within SLA. This chart showed p95 latency over time.</li><li>• Throughputs (requests per second) to ensure that the system was able to keep up with the client demand.</li><li>• GPU utilization that was measured to indicate if the model was bottlenecked by compute or I/O.</li></ul> <p>After load testing with the monitoring stack, something new happened. A few observations emerged and latency remained stable under moderate load, however, there were occasional spikes during GPU memory reallocation. As I used the metrics export, it was now obvious how the system behaved under different conditions. Table 13 shows the exported metrics that I monitored:</p> <p><b>Table 13: Monitored metrics and their healthy range</b></p> <table border="1"><thead><tr><th>Metric</th><th>Description</th><th>Healthy Range</th></tr></thead><tbody><tr><td>p50 Latency</td><td>Median Request Time</td><td>25-30 ms</td></tr><tr><td>p95 Latency</td><td>95<sup>th</sup> Percentile Latency</td><td>35-45 ms</td></tr><tr><td>Throughput (RPS)</td><td>Requests Served per Second</td><td>20-25 RPS</td></tr><tr><td>GPU Utilization (%)</td><td>Average GPU Usage during Workload</td><td>40-60%</td></tr><tr><td>Error Rate (%)</td><td>Ratio of Failed Requests</td><td>&lt; 0.5%</td></tr></tbody></table> <p>Inspected by </p> |                                     | Metric        | Description | Healthy Range | p50 Latency | Median Request Time | 25-30 ms | p95 Latency | 95 <sup>th</sup> Percentile Latency | 35-45 ms | Throughput (RPS) | Requests Served per Second | 20-25 RPS | GPU Utilization (%) | Average GPU Usage during Workload | 40-60% | Error Rate (%) | Ratio of Failed Requests | < 0.5% |
| Metric  | Description                         | Healthy Range |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |
| p50 Latency   | Median Request Time                 | 25-30 ms      |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |
| p95 Latency   | 95 <sup>th</sup> Percentile Latency | 35-45 ms      |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |
| Throughput (RPS)  | Requests Served per Second          | 20-25 RPS     |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |
| GPU Utilization (%)   | Average GPU Usage during Workload   | 40-60%        |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |
| Error Rate (%)  | Ratio of Failed Requests            | < 0.5%        |             |               |             |                     |          |             |                                     |          |                  |                            |           |                     |                                   |        |                |                          |        |



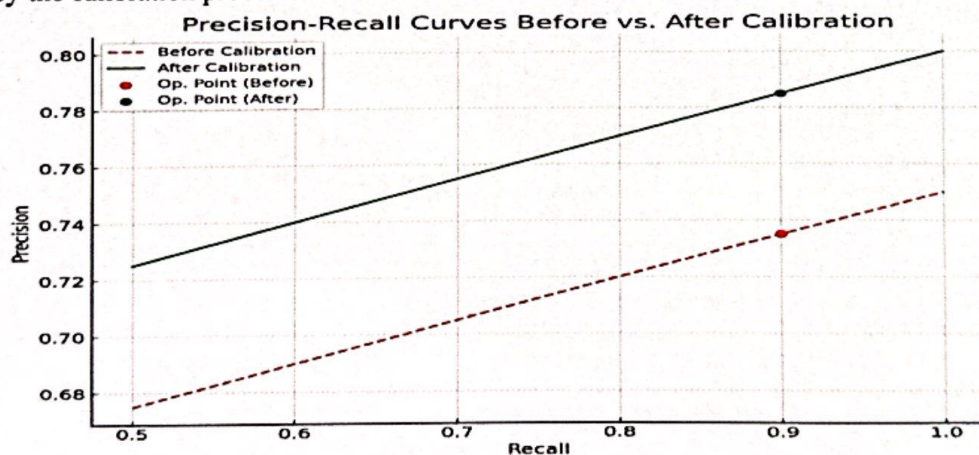
## Daily Activities

| Date       | Department              |
|------------|-------------------------|
| 23/07/2025 | Artificial Intelligence |

Even though my raw model was ready and functioning well, it was still missing something. Post-processing steps are necessary to translate a raw model's outputs into predictions that are ready for production. Thus, today I focused on systematically calibrating thresholds to optimize the system for its operational needs. Below, I have explained the calibration strategies I used:

1. Class-Specific NMS IoU Tuning
  - For the sparse classes, I assigned a lower IoU (0.4-0.45) in order to decrease the rate of overlapping ghost detections.
  - For the dense-object classes, I assigned a higher IoU, from 0.6 to 0.65, so that it would prevent excessive merging.
2. Adjusting Confidence Threshold
  - The class with a higher risk (i.e., scratches class that was harder for the system to detect) was given lower thresholds, from 0.2 to 0.25, so that the misses would be minimized.
  - For the classes with low risks (i.e., cracks and mostly dents), I tightened the thresholds to the range 0.2-0.25 for the purpose of reducing the noise.
3. Analyzing Operational Trade-off
  - I ensured that each calibration setting was tested against a validation set aligned with real deployment distributions.
  - For quantifying the impact of utilizing these calibration settings, I used Precision-Recall curves.
  - Eventually, the operating point that was selected maximized the F1 score under cost constraints that were ops-defined.

The calibration strategies that I used significantly improved the system's readiness for production. As shown in Figure 5, the operating point was shifted toward a higher precision by the calibration process.



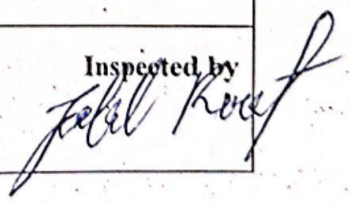
**Figure 5: Precision-Recall curve before vs after calibration**

Inspected by

*Jade Heng*

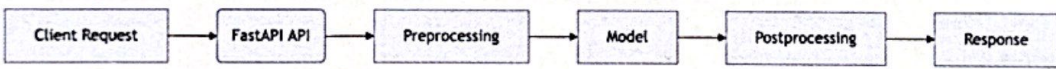
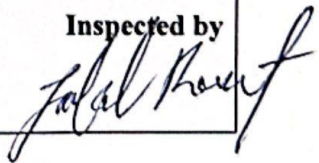


## Daily Activities

|   |  |
|---|--|
| <b>Date</b><br>26/07/2025   | <b>Department</b><br>Artificial Intelligence |
| <p>One of the challenges when deploying machine learning models in production is that they do not function well in case of domain shift. Domain shifts refer to situations where the training data distribution varies from the data distribution the model faces during inference. Even though the system had reached an overall high accuracy, the model still struggled with data inputs that had poor lightening, a higher sensor noise or a higher motion blur because of their relatively longer exposure times.</p> <p>To address this issue, firstly I evaluated the system in low-light conditions. I collected a test set of video clips with lower light. This dataset helped me analyze and examine the robustness of the current pipeline under lightening settings that the model struggled with. The initial tests that I conducted indicated that the recall was not affected that much, on the other hand, the precision was affected, and it slightly decreased in darker lightening because of the existence of the false positives which had increased the noise in the background of the video clips.</p> <p>Now that I had evaluated the domain that the model struggled with, it was now time for mitigating this issue. To solve this problem, I used two different preprocessing strategies: AutoExposure Compensation and Gamma Correction.</p> <ol style="list-style-type: none"><li>1. AutoExposure Compensation<ul style="list-style-type: none"><li>• This preprocessing method adjusted the brightness and contrast dynamically using its built-in correction that was based on histograms.</li><li>• Most of the detections that the model missed were due to underexposed regions that looked completely dark to the model. AutoExposure Compensation prevented these regions to appear fully dark.</li></ul></li><li>2. Gamma Correction<ul style="list-style-type: none"><li>• Firstly, I empirically tuned gamma (<math>\gamma = 1.4</math>).</li><li>• Correcting the gamma helped increase the lightening of the mid-tones and it did not oversaturate the highlights.</li><li>• The shadowed objects' visibility was improved significantly thanks to this non-linear transformation.</li></ul></li></ol> <p>By using these steps, I was able to reduce the performance gap that appeared in poor lightening settings because the frames with low lights were now normalized and closer to the data distribution of the data that the model was trained on.</p> <p>The low-light dataset that I had collected actually formed the edge cases. So today, I deliberately targetted domain shift and the edge cases and by doing so, I improved the model's robustness in the production scenarios. AutoExposure and gamma correction reduced the false positives (FP) in frames with lower lightening substantially. The pipeline was now more adaptive and reliable under different conditions. What I had learned about high accuracy and metrics not being enough for a good production-grade model, was reinforced. The model has to be robust and adaptive in different situations.</p> |  |
| <p>Inspected by<br/></p>   |  |

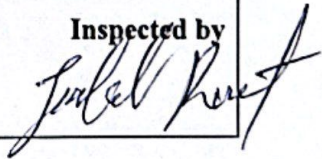


## Daily Activities

| Date  | Department              |
|---|-------------------------|
| 27/07/2025  | Artificial Intelligence |
| <p>My system was now technically ready. It was now time to ensure that any developers or future contributors or even project stakeholders would be able to understand and deploy and troubleshoot the system without putting unnecessary effort. I believe, a major issue with making and maintaining most AI-based systems nowadays is that they lack proper and thorough documentation. This makes even the most accurate and optimized and high performing AI systems become a "black box" that is vulnerable in real-life production. The first step for proper documentation is writing the README file. This file is usually the first part anyone who wants to interact with the project checks. I structured the README based on three main themes: setup, training and serving.</p> <ol style="list-style-type: none"><li>1. Setup<br/>This part consisted of the installation instructions step by step, in case someone wanted to run this project on their own devices. This section covered creating the environment for the project, installing the dependency and GPU prerequisites. Additionally, I included explicit CUDA version compatibility so that the common pitfalls of mismatching NVIDIA drivers and PyTorch builds would be prevented.</li><li>2. Training<br/>This section explained the dataset preparation such as structuring the directory and formatting the annotation, launching the training runs and where the outputs, weights, logs and checkpoints would be stored. I also included tips on how to resume from the checkpoints and how to customize hyperparameters so that the project would be flexible.</li><li>3. Serving<br/>The Serving section consisted of the instructions on how to run the FastAPI inference server. I also included the example API calls with curl and Python clients. Moreover, I included the mermaid diagram below (Figure 6) to demonstrate the workflow starting from the client request to the response. I included this visual so that it would complement the text and make the context easier to follow.</li></ol>  <pre>graph LR; A[Client Request] --&gt; B[FastAPI API]; B --&gt; C[Preprocessing]; C --&gt; D[Model]; D --&gt; E[Postprocessing]; E --&gt; F[Response]</pre> <p><b>Figure 6: System High-level Architecture Mermaid Diagram</b></p> <p>Today I emphasized the importance of post-training calibration so that I could align the system behavior with the operational needs. I also did a trade-off analysis which reinforced the idea that having fewer false alarms improved the precision of the model and this change, was more beneficial for making the system production-ready than the marginal recall improvements. Machine learning deployment is not just about training the best model, but also about fine-tuning the decision logic so that it would match the operational and human expectations.</p> |                         |
| <p>Inspected by</p>    |                         |

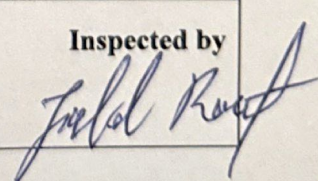


## Daily Activities

| <b>Date</b><br>28/07/2025   | <b>Department</b><br>Artificial Intelligence       |  |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
|---|--|--|---------|-------------------------|-----------------------|--|----------------------------|---------------------------|------------------------------|-----------------------------------|---------------------|--|---------------------------------------|--------------------------|---------------------------------|--|--------------------------|---|--|---------------------------|--|--|-------------------------------|-----------------------------|---------------------------------------|------------------------------|--|--|
| <p>In order to hand over the scripts, the artifacts, code, and workflows had to be packaged in a reproducible way. Thus, today I focused on designing mechanisms to streamline the artifact exports and enforce coding standards so that I could make the builds reproducible. First of all, I made sure that the model export process was extended to generate ONNX artifacts with versioned filenames.</p> <p>Additionally, in order to make the artifact publishing formal, I introduced a make release task so that instead of manually copying the files, a single instruction handled the following:</p> <ol style="list-style-type: none"> <li>1. The model was exported into ONNX format.</li> <li>2. The model was named with version tags.</li> <li>3. A checksum was automatically generated.</li> <li>4. The model and the checksum were moved into the releases/ directory.</li> </ol> <p>This would avoid the risks rising from undocumented manual processes. Additionally, I used pre-commit hooks. I ensured that the pre-commit hooks were run before every Git commit automatically. This was done to make sure that the style of the code and the quality rules were consistent across contributors. Table 14 shows the summary of my handover scripts and pre-commit hooks.</p> <p><b>Table 14: Summary of Handover Scripts &amp; Pre-Commit Hooks</b></p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: left;"> <thead> <tr style="background-color: #f2f2f2;"> <th style="text-align: left;">Component</th> <th style="text-align: left;">Purpose</th> <th style="text-align: left;">Example Output/Behavior</th> </tr> </thead> <tbody> <tr> <td><b>Export_onnx.py</b></td> <td>Exported the ONNX model with versioned file names.</td> <td>model_v1.3.0_20250721.onnx</td> </tr> <tr> <td><b>Checksum Generator</b></td> <td>Validated artifact integrity</td> <td>model_v1.3.0_20250721.onnx.sha256</td> </tr> <tr> <td><b>make release</b></td> <td>One-step artifact packaging into releases/ dir</td> <td>Produced model and the checksum files</td> </tr> <tr> <td><b>Pre-commit: black</b></td> <td>Enforced consistent code format</td> <td>Automatic code reformats before commit</td> </tr> <tr> <td><b>Pre-commit: isort</b></td> <td>Ensured imports were sorted and grouped</td> <td>Alphabetical and grouped by standard, third-party, local</td> </tr> <tr> <td><b>Pre-commit: flake8</b></td> <td>Found syntax errors and style violations</td> <td>Unused imports and undefined variables were warned</td> </tr> <tr> <td><b>Pre-commit: whitespace</b></td> <td>Removed the trailing spaces</td> <td>Unnecessary diff noise was cleaned up</td> </tr> <tr> <td><b>Pre-commit: EOF fixer</b></td> <td>Guaranteed that files ended with a newline</td> <td>POSIX-compliant formatting was maintained.</td> </tr> </tbody> </table> |  | Component  | Purpose | Example Output/Behavior | <b>Export_onnx.py</b> | Exported the ONNX model with versioned file names. | model_v1.3.0_20250721.onnx | <b>Checksum Generator</b> | Validated artifact integrity | model_v1.3.0_20250721.onnx.sha256 | <b>make release</b> | One-step artifact packaging into releases/ dir | Produced model and the checksum files | <b>Pre-commit: black</b> | Enforced consistent code format | Automatic code reformats before commit | <b>Pre-commit: isort</b> | Ensured imports were sorted and grouped | Alphabetical and grouped by standard, third-party, local | <b>Pre-commit: flake8</b> | Found syntax errors and style violations | Unused imports and undefined variables were warned | <b>Pre-commit: whitespace</b> | Removed the trailing spaces | Unnecessary diff noise was cleaned up | <b>Pre-commit: EOF fixer</b> | Guaranteed that files ended with a newline | POSIX-compliant formatting was maintained. |
| Component   | Purpose  | Example Output/Behavior                                  |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>Export_onnx.py</b>   | Exported the ONNX model with versioned file names. | model_v1.3.0_20250721.onnx                               |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>Checksum Generator</b>   | Validated artifact integrity                       | model_v1.3.0_20250721.onnx.sha256                        |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>make release</b>   | One-step artifact packaging into releases/ dir     | Produced model and the checksum files                    |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>Pre-commit: black</b>  | Enforced consistent code format                    | Automatic code reformats before commit                   |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>Pre-commit: isort</b>  | Ensured imports were sorted and grouped            | Alphabetical and grouped by standard, third-party, local |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>Pre-commit: flake8</b>   | Found syntax errors and style violations           | Unused imports and undefined variables were warned       |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>Pre-commit: whitespace</b>   | Removed the trailing spaces                        | Unnecessary diff noise was cleaned up                    |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <b>Pre-commit: EOF fixer</b>  | Guaranteed that files ended with a newline         | POSIX-compliant formatting was maintained.               |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |
| <p><b>Inspected by</b></p>   |  |  |         |                         |                       |  |                            |                           |                              |                                   |                     |  |                                       |                          |                                 |  |                          |   |  |                           |  |  |                               |                             |                                       |                              |  |  |



## Daily Activities

| <b>Date</b><br>29/07/2025   | <b>Department</b><br>Artificial Intelligence                           |  |              |                     |                            |  |  |   |  |  |                              |   |   |
|---|--|--|--------------|---------------------|----------------------------|--|--|---|--|--|------------------------------|---|---|
| <p>Today's focus was on improving the security and the privacy of the project. In all stages including data collection, model training and any kind of user interaction with the system, security and privacy were crucial. I systematically worked on three main areas of the project: Dependency Security and Auditing, Personally Identifiable Information (PII) Review and Data Retention Policy.</p> <ul style="list-style-type: none"> <li>• <b>Dependency Security and Auditing</b><br/>The first step was the pip-audit tool. This tool scanned all Python packages and libraries for vulnerabilities. A detailed security report was prepared with the help of pip-audit which highlighted the insecure packages, versions that were outdated and dependencies that were likely to introduce security risks. After auditing, I pinned all package to specific and secure versions in order to ensure reproducibility and prevent vulnerabilities in future installations of the system.</li> <li>• <b>Personally Identifiable Information (PII) Review</b><br/>As the name suggests, Personally Identifiable Information (PII) refers to information or user "footprints" that could be identified and assigned to the information of the user who did the interactions with the system. To prevent this for the sake of user privacy, I reviewed the datasets and verified that there were no personal identifiers and that the data was being handled ethically.</li> <li>• <b>Data Retention Policy</b><br/>I established an explicit data retention policy to make the privacy compliance even stronger. I set a retention period of 90 days for the raw data and video files so that after this period, all raw inputs would be automatically deleted. This ensured that the data was not stored indefinitely which would potentially decrease exposure risks and maintain an audit trail of the availability period of the data for model training or review. Table 15 shows a summary of all privacy and security measures that I took.</li> </ul> <p><b>Table 15: Summary of Security and Privacy Measures</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 33%;">Area</th> <th style="width: 33%;">Action Taken</th> <th style="width: 33%;">Purpose and Benefit</th> </tr> </thead> <tbody> <tr> <td><b>Dependency Security</b></td> <td>Scanned with pip-audit and pinned versions</td> <td>Ensured that the environment was secure and reproducible</td> </tr> <tr> <td><b>Personally Identifiable Information (PII) Verification</b></td> <td>Reviewed datasets and verified that there were no personal identifiers</td> <td>Ensured the system was complied with privacy regulations and that the data was handled ethically</td> </tr> <tr> <td><b>Data Retention Policy</b></td> <td>Raw videos were retained for 90 days only</td> <td>Exposure was limited and the storage load was reduced in order to ensure compliance</td> </tr> </tbody> </table> |  | Area   | Action Taken | Purpose and Benefit | <b>Dependency Security</b> | Scanned with pip-audit and pinned versions | Ensured that the environment was secure and reproducible | <b>Personally Identifiable Information (PII) Verification</b> | Reviewed datasets and verified that there were no personal identifiers | Ensured the system was complied with privacy regulations and that the data was handled ethically | <b>Data Retention Policy</b> | Raw videos were retained for 90 days only | Exposure was limited and the storage load was reduced in order to ensure compliance |
| Area  | Action Taken   | Purpose and Benefit  |              |                     |                            |  |  |   |  |  |                              |   |   |
| <b>Dependency Security</b>  | Scanned with pip-audit and pinned versions                             | Ensured that the environment was secure and reproducible   |              |                     |                            |  |  |   |  |  |                              |   |   |
| <b>Personally Identifiable Information (PII) Verification</b>   | Reviewed datasets and verified that there were no personal identifiers | Ensured the system was complied with privacy regulations and that the data was handled ethically |              |                     |                            |  |  |   |  |  |                              |   |   |
| <b>Data Retention Policy</b>  | Raw videos were retained for 90 days only                              | Exposure was limited and the storage load was reduced in order to ensure compliance              |              |                     |                            |  |  |   |  |  |                              |   |   |
| <b>Inspected by</b><br>  |  |  |              |                     |                            |  |  |   |  |  |                              |   |   |



## Daily Activities

| Date       | Department              |
|------------|-------------------------|
| 30/07/2025 | Artificial Intelligence |

My focus today was shifted from the backend and infrastructure concerns toward usability and end users. I built something new, a live demo for the project. It was necessary to examine the performance of the model when being reviewed by people (actual users) and whether it met their expectations. I built a small proof of concept (POC) mobile client and then I collected feedback on the clarity of model outputs from my colleagues. The first important step was to build an application by Streamlit. I chose Streamlit because I had built a web app with it before for one of my university courses and I was already familiar with it. It was easy to use, and the application looked both visually engaging and not too complex to work with. Plus, the app could be run locally and deployed to the community Streamlit cloud. Additionally, I had already designed my API, and the endpoints were ready, and responses were being successfully sent through curl. All I had to do was to build a nice functional frontend for it.

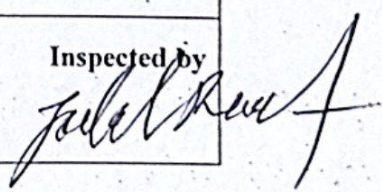
My demo worked as an interface where QA testers interacted with the model outputs including the bounding-box predictions and class labels in a mobile-friendly format. The demo application allowed the testers to upload or preview sample images, visualize the model-drawn bounding boxes and read the associated class names. I asked my mentor if I should deploy the demo to the Streamlit community cloud, but he suggested that I ran it locally so that it would remain the QA team's environment. By QA team I mean my colleagues that I asked to test the live demo and provide feedback.

In addition to the demo application, there were two other access pathways provided:

1. cURL Recipe  
I prepared a simple cURL command to query the inference API directly from the command line.
2. Python Client Snippet  
In case there was a tester who preferred to write their own experiments, I shared a short Python client. This snippet was enabled to upload an image, then send it to the inference endpoint and parse results back into the structured format of bounding-boxes and labels.

In short, I used three options for testers: Streamlit app, cURL recipe and Python snippet. My mentor used the Streamlit app and tried one cURL recipe while the other people I asked to test my system, only tried the Streamlit app as it was easier to use.

The feedback that I received was about readability. It was reported to me that some labels appeared too small to be readable in the app and the QA suggested that I used larger fonts and provide a Farsi language feature too in the Streamlit app. Because I originally made it in English only. I considered their feedback and changed the app accordingly. I made the class names larger and more readable and put an option to change the language into Farsi.

Inspected by  




## Daily Activities

|                           |  |
|---------------------------|--|
| <b>Date</b><br>02/08/2025 | <b>Department</b><br>Artificial Intelligence |
|---------------------------|--|

Today I performed A/B Evaluation between two deployment candidates of the object detection model: the standard YOLOv8m build in FP16 precision and the optimized TensorRT FP16 build. I conducted this evaluation because I wanted to measure the performance across both accuracy and latency to understand that which of the models, if promoted to production, would reach a better balance between responsiveness and reliability. The A/B evaluation had two phases:

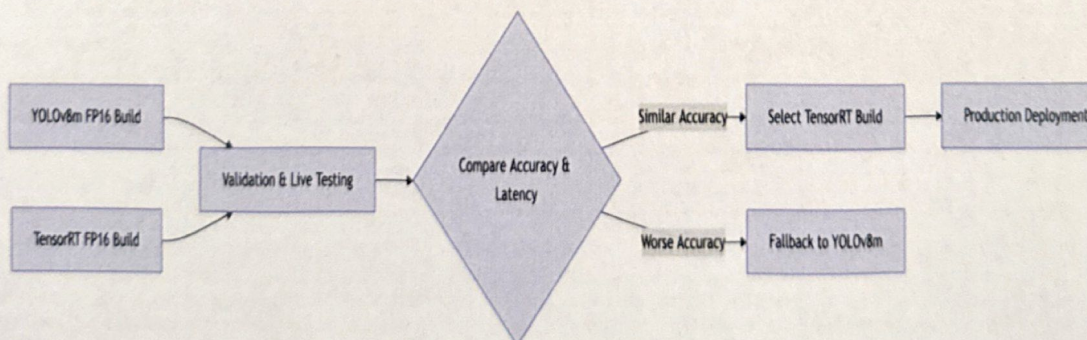
1. I used a curated validation split to compare the following accuracy metrics:
  - Precision (mAP@50)
  - mAP@50-95
  - Recall
2. I streamed real-world videos and image samples through both builds to capture the latency distributions p50 and p95. This ensured the bounding box outputs were consistent.

My findings indicated that the TensorRT FP16 build had a substantial improvement in its latency while maintaining almost the same accuracy as the YOLOv8m build in FP16. My results and their improvement are visible in Table 16.

**Table 16: Results Comparison of YOLOv8m FP16 vs TensorRT FP16**

| Metric                  | YOLOv8m FP16 | TensorRT FP16 | $\Delta$ (Improvement) |
|-------------------------|--------------|---------------|------------------------|
| <b>mAP@50</b>           | 0.918        | 0.917         | - 0.1%                 |
| <b>mAP@50-95</b>        | 0.612        | 0.622         | - 0.2%                 |
| <b>Latency p50</b>      | 42 ms        | 34 ms         | ~18% Faster            |
| <b>Latency p95</b>      | 68 ms        | 51 ms         | ~25% Faster            |
| <b>Throughput (FPS)</b> | ~28          | ~35           | ~25% Higher            |

Based on my findings, I chose the TensorRT FP16 build for production deployment. This decision was done mainly because of the lower p95 latency as this meant that performance of TensorRT FP16 build under load was more predictable and smoother. As the accuracy nearly remained the same, this optimization came with almost zero cost as the model performance quality was not compromised. For clarity, the workflow of the performed A/B Evaluation is illustrated in Figure 7.



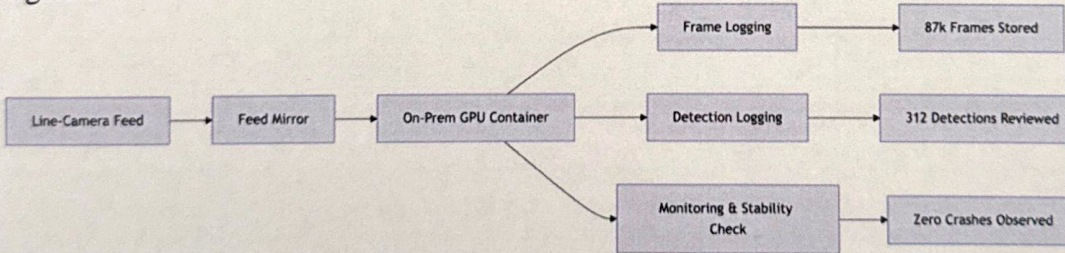
**Figure 7: A/B Evaluation on YOLOv8m FP16 Build & TensorRT FP16 Build**

Inspected by

*Salal Khat*



## Daily Activities

| Date   | Department              |
|--|-------------------------|
| 03/08/2025   | Artificial Intelligence |
| <p>Today, I reached an important milestone! It was my first real-world pilot deployment of the detection pipeline. After weeks of working on my project, the system was containerized and deployed on an on-premises GPU machine that enabled a direct connection to a mirrored feed from the line-camera. This simulated production conditions without interfering with the operations going on. Valuable insights into the throughput, stability and the quality of the detection over an extended and continuous runtime was provided by the pilot.</p> <p>The deployment architecture included a Docker container running the TensorRT FP16 model that I had selected after the A/B evaluation yesterday. This container had the inference engine and the monitoring hooks encapsulated for the purpose of capturing the performance metrics and the logs. This procedure was hosted on a local GPU workstation. I did this so that the latency would be minimized and to ensure that the environment was under full control. I wanted the live production workflow to be left untouched as the line-camera feed was being mirrored. By doing so, I avoided running into risks that were associated with direct inline ingestions while the system ingested the real-world data. I enabled logging for both the raw frames and the detection metadata so that I would be able to keep a comprehensive record for the post-pilot analysis.</p> <p>I set the 24-hour pilot in order to validate three main aspects to ensure that the system was ready:</p> <ol style="list-style-type: none"><li>1. Stability<br/>I made sure that the uptime of the system was continuous and had no crashes or leakages in the memory.</li><li>2. Throughput<br/>I ensured that the system was able to handle sustained video feed at targeted frame rates.</li><li>3. Detection Quality<br/>I reviewed the sampled detections to make sure that the bounding boxes were correct and that the class labels were accurate.</li></ol> <p>Over the course of 24 hours, 87000 frames were ingested by the system, and a total of 312 detections were captured and logged with their bounding box coordinates, class predictions and confidence scores. The workflow of the pilot analysis and the results are illustrated in Figure 8.</p>  <pre>graph LR; A[Line-Camera Feed] --&gt; B[Feed Mirror]; B --&gt; C[On-Prem GPU Container]; C --&gt; D[Frame Logging]; C --&gt; E[Detection Logging]; C --&gt; F[Monitoring &amp; Stability Check]; D --&gt; G[87k Frames Stored]; E --&gt; H[312 Detections Reviewed]; F --&gt; I[Zero Crashes Observed]</pre> <p><b>Figure 8: Pilot Analysis Workflow &amp; Results</b></p> <p>Inspected by <i>[Signature]</i></p> |                         |

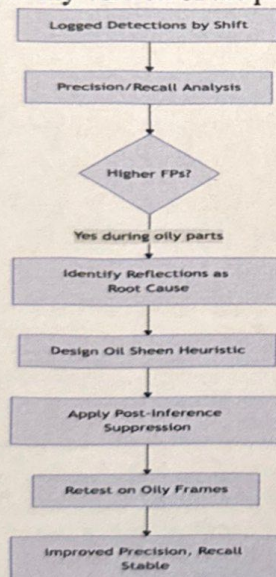


## Daily Activities

| Date       | Department              |
|------------|-------------------------|
| 04/08/2025 | Artificial Intelligence |

After the successful 24-hour pilot deployment yesterday, now it was time for post-pilot analysis. In order to understand the reason behind subtle variations in the accuracy, I grouped the detections and analyzed them per operational shift. In other words, I analyzed the model performance carefully under varying condition such as different lighting and operator activity. This analysis indicated the overall performance was strong, however, there as a trend that caught my attention. The frames that captured oily-looking parts, tended to be false positives (FP) for non-existent defects. I guessed that it was because of the light reflection of the "oil" on the metal parts surfaces that caused the system to confuse the reflections with defects. The precision was acceptable, but the FP rate was undermining the robustness of the system. The problem was not related to the distribution of the data, it was environmental, caused by light reflections on oil residues. I discussed this issue with the mentor, and he said that to solve such a problem, we would need a heuristic solution that would learn the patterns of the oil reflections shape, such as their aspect ratio, specular intensity and temporal consistency. As I was rebuilding a project that the company had already built, they had already faced and solved this problem and more. They had already built a heuristic called Oil Sheen Suppression that used a specific combination of rules. It applied aspect ratio constraints as oil reflections are often unusually long, even longer than scratches or cracks. The specular intensity was checked. In other words, the heuristic would give an alert and double-check if the pixel brightness was high relative to the surrounding context. There was a temporal consistency filter applied as real objects tend to persist across multiple frames, unlike reflections that often flicker.

The mentor helped me reuse the heuristic "Oil Sheen Suppression" that the company had built in my version of the project. The heuristic's workflow process is shown in figure 9.



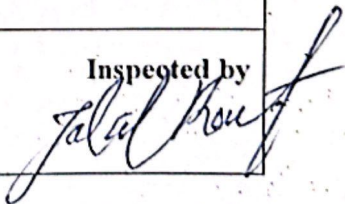
**Figure 9: Oil Sheen Suppression Workflow Diagram**

Inspected by

*Jabul Reef*

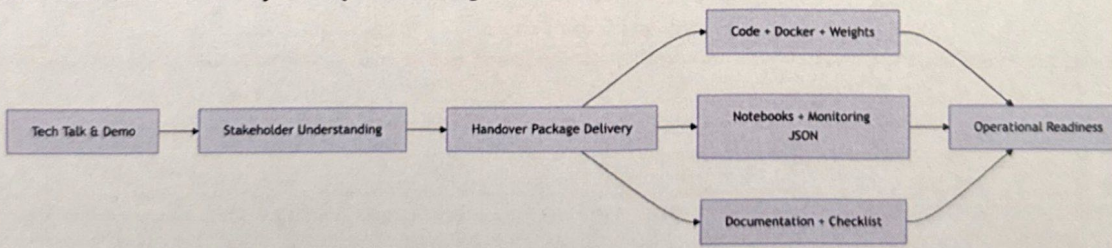
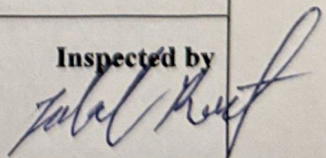


## Daily Activities

| Date  | Department              |
|---|-------------------------|
| 05/08/2025  | Artificial Intelligence |
| <p>Only two days left of my internship at Bina Sanat Veera! Today I represented the culmination of the pilot phase and formally made the first release of the detection pipeline. I was done with the prototyping, multiple tests and pilot deployment. It was time for final tuning and versioning. The transition from a setup that was experimental to a production-ready and controlled baseline had officially started. This transition consisted of 3 main stages: Model Locking and Threshold Calibration, Release Tagging and Artifact Archival, and Acceptance Checking and Walk-through with Mentor.</p> <p>1. Model Locking and Threshold Calibration</p> <p>I started with freezing the trained YOLOv8m TensorRT FP16 build that I had validated using validation datasets, the live pilot samples and the Oil Sheen Suppression heuristic that I had added to the system as the post-pilot refinement. I locked the model so that all downstream environments for the development, staging and production stages were enabled to use the same model artifact without any changes. I then finalized the thresholds like so:</p> <ul style="list-style-type: none"> <li>• Confidence Threshold</li> </ul> <p>I optimized the confidence threshold so that the precision and recall would be balanced.</p> <ul style="list-style-type: none"> <li>• Non-Maximum Suppression (NMS) Threshold</li> </ul> <p>I tuned this threshold to prevent duplicates in the bounding boxes in crowded scenes in the frames.</p> <p>I tested these parameters across multiple subsets so that I could confirm that the model had a consistent performance.</p> <p>2. Release Tagging and Artifact Archival</p> <p>I finalized the container and tagged the model as v1.0.0 in the internal model registry so that I could establish further reproducibility in the system. The version tag represented the first stable baseline, and I would use it as the reference point for all future improvements. I archived the artifacts below:</p> <ul style="list-style-type: none"> <li>• The engine file of the TensorRT</li> <li>• Docker image digest</li> <li>• Yaml configuration with thresholds and heuristics</li> <li>• Evaluation reports including the precision and recall metrics and latency distributions</li> </ul> <p>I archived both code and non-code artifacts to ensure long-term reproducibility.</p> <p>3. Acceptance Checklist and Walkthrough with Mentor</p> <p>I drafted a final checklist and showed it to the mentor. The checklist included model quality, system stability, performance, heuristics and documentation. Today marked as the formal sign-off of the first production-ready release.</p> |                         |
| <p>Inspected by</p>    |                         |



## Daily Activities

| Date   | Department              |
|--|-------------------------|
| 06/08/2025   | Artificial Intelligence |
| <p>Today was the last day of my internship! It was time for presenting what I had done and learned during this internship and to formally handover the deliverables. My task today was not only technical, but also communicational. I had to ensure that the work that I completed in the past month would be delivered well and clearly understood by the mentor and my colleagues. I prepared a presentation for a 20-minute technical talk. My presentation had three main components: System Architecture Overview, Key Metrics, and Live Demo.</p> <ol style="list-style-type: none"> <li><b>1. System Architecture Overview</b><br/>I showed and discussed a walkthrough of how my detection pipeline had evolved. I presented all the refinements, setbacks and solutions from dependency audits and data policies to pilot deployment, post-pilot refinements and the locked release v1.0.0.</li> <li><b>2. Key Metrics</b><br/>I presented the benchmark results such as latency improvements from the TensorRT optimization, precision / recall per shift and overall pilot results of 87k frames processed with no crashes. The metrics that I presented were evidence of the readiness, robustness and reliability of my system.</li> <li><b>3. Live Demo</b><br/>I also presented the live demo that I had built. I showed the latest version that I had refined according to the feedback I had received from the mentor and my colleagues. By showing my proof of concept (POC), I was able to turn the abstract metrics I presented into a working end-to-end tangible system.</li> </ol> <p>The package that I handed over to my mentor, included the following:</p> <ul style="list-style-type: none"> <li>• Source Code Repository</li> <li>• Docker Images</li> <li>• Model Weights</li> <li>• Notebooks</li> <li>• Monitoring Dashboard JSON</li> <li>• Documentation</li> </ul> <p>I designed this package to be self-sufficient so that anyone could rebuild, install, deploy, troubleshoot, monitor and iterate the project without requiring further support from the original developer. Today I combined a clear technical presentation, a live demo in the end of the presentation and delivered a carefully made package. Figure 10 shows a summary of all I did on the last day of my internship at Bina Sanat Veera.</p>  <pre> graph LR     A[Tech Talk &amp; Demo] --&gt; B[Stakeholder Understanding]     B --&gt; C[Handover Package Delivery]     C --&gt; D[Code + Docker + Weights]     C --&gt; E[Notebooks + Monitoring JSON]     C --&gt; F[Documentation + Checklist]     D --&gt; G[Operational Readiness]     E --&gt; G     F --&gt; G   </pre> <p><b>Figure 10: Summary of the Last Day of the Internship</b></p> <div style="text-align: right;"> <p>Inspected by</p>  </div> |                         |





**BAHÇEŞEHİR UNIVERSITY**  
**Faculty of Engineering and Natural Sciences**  
**Department of Artificial Intelligence Engineering**



### **Training Report**

|                |              |
|----------------|--------------|
| <b>Name</b>    | Atena        |
| <b>Surname</b> | Jafari Parsa |
| <b>ID No</b>   | 2101183      |

### **Department of Artificial Intelligence Engineering** **Examination Committee**

**Member 1**

**Member 2**

**Member 3**

**Training Report is;**

**Accepted** ☐

**Not Accepted** ☐