AutoML CI/CD/CT Capstone Project Proposal Report

Sepehr Heydarian, Archer Liu, Elshaday Yoseph, Tien Nguyen

Table of contents

1 Executive Summary

Bayes Studio, a Vancouver-based startup focused on wildfire detection, faces challenges updating their computer vision models as new image data becomes available. Their current manual labeling workflow delays model improvement and deployment. Our Capstone product is an automated, reproducible pipeline that begins with pre-labeling using `YOLOv8`, verified with Meta's Segment Anything Model (`SAM`). Ambiguous cases are routed to a human-in-the-loop system via Label Studio. The verified data is then used to retrain and update the model. The model undergoes distillation and quantization for edge-device utilization. The final deliverable is a fully automated CI/CD pipeline that continuously updates an edge-deployable YOLO model.

2 Introduction

Wildfires are devastating environmental disasters that threaten ecosystems and communities. Early detection is critical to limiting wildfire spread and minimizing damage (Defence Research and Development Canada 2022).

Our Capstone partner, Bayes Studio, is a Vancouver-based startup that develops advanced AI tools for environmental monitoring and early wildfire detection. A key challenge they face is keeping their models up to date as new image data becomes available. Currently, images are manually labeled using a `YOLO` object detection model, then used to retrain and redeploy the system. As new data arrives frequently, this manual process leads to delays, and slower model updates. Ultimately, this reduces the system's effectiveness for real-time wildfire detection.

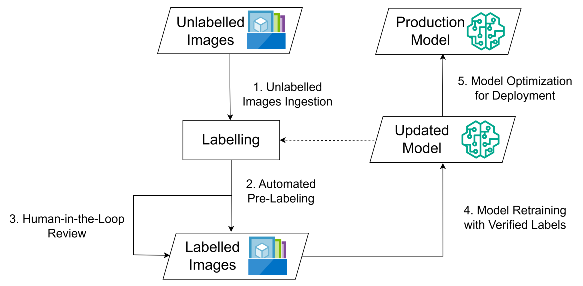
To address this, our product is an automated pipeline. When new images arrive, they will be pre-labelled using the existing `YOLO` model. A second model, Meta's Segment Anything Model (`SAM`), will verify the labels. If both models agree, the image is accepted, otherwise it will be flagged for human review through an interface like Label Studio. Accepted images will then be used to retrain the model, followed by distillation and quantization for edge deployment.

Our final deliverable is a reproducible, automated pipeline deployed via GitHub Actions that outputs an updated `YOLO` model file(`.pt`).

3 Proposed Pipeline

3.1 High-Level Overview of the Proposed Data Product

To address the current bottleneck, we propose an iterative data pipeline that integrates automation, human feedback, and model optimization for continuous improvement.



Unlabelled Images
Production Model
1. Unlabelled Images Ingestion
5. Model Optimization for Deployment
Labelling
Updated Model
2. Automated Pre-Labeling
3. Human-in-the-Loop Review
4. Model Retraining with Verified Labels
Labelled Images

Figure 1: High-level overview of the data pipeline

1. **Unlabelled Image Ingestion**
   The pipeline begins by collecting raw, unlabelled images from the partner's data sources (`fetch_data.py`).
2. **Automated Pre-Labelling**
   AI models generate initial labels for key objects (e.g., `Fire`, `Smoke`, etc), reducing the need for manual annotation (`labelling.py`).
3. **Human-in-the-Loop Review**
   Experts review and correct the ambiguous pre-labels to ensure high-quality annotations with minimal manual effort (`human_intervention.py`).
4. **Model Retraining with Verified Labels**
   Verified labelled images are used to retrain the wildfire detection model, improving its accuracy and adaptability over time (`augmentation.py` and `train.py`).
5. **Model Optimization for Deployment**
   Techniques like **distillation** and **quantization** compress the model, making it suitable for deployment on edge devices (`distill_quantize.py`).

Note that the names and implementations of the Python scripts will evolve throughout the course of development.

3.2 Unlabelled Images Ingestion

Efficiently handling a growing stream of unlabelled images requires a standardized ingestion process that supports local prototyping and future cloud integration. We will implement this step in `fetch_data.py`.

3.2.1 Input

This step will ingest unlabelled image data, which serves as the input for the entire system. Around 500 new images are added each month, containing five object classes: `Fire`, `Smoke`, `Lightning`, `Vehicle`, and `Person`.

3.2.2 Process

`fetch_data.py` loads data locally (for dev/test purposes) but can later fetch from cloud storage.

3.2.3 Output

The following image directories should be produced:

- `raw/images` – used as input for the pre-labelling stage
- `raw/distilled_images` – used during the model distillation step

These folders are assumed to be locally accessible for the prototype.

3.3 Automated Pre-Labeling

Manual labeling is expensive and time-consuming. To reduce annotation effort, we implement an automated pre-labeling system in `labeling.py` that leverages both object detection and image segmentation.

3.3.1 Input

The input consists of unlabeled wildfire-related images from the project dataset.

3.3.2 Process

Object Detection and Segmentation (`labeling.py`)

A YOLOv8 model (Bochkovskiy, Wang, and Liao 2020) is applied to each image to generate bounding boxes, class labels (e.g., fire, smoke, vehicle), and confidence scores. These predictions are then passed to the Segment Anything Model (SAM) (Kirillov et al. 2023), which generates pixel-level segmentation masks using the YOLO-generated bounding boxes and labels as prompts.

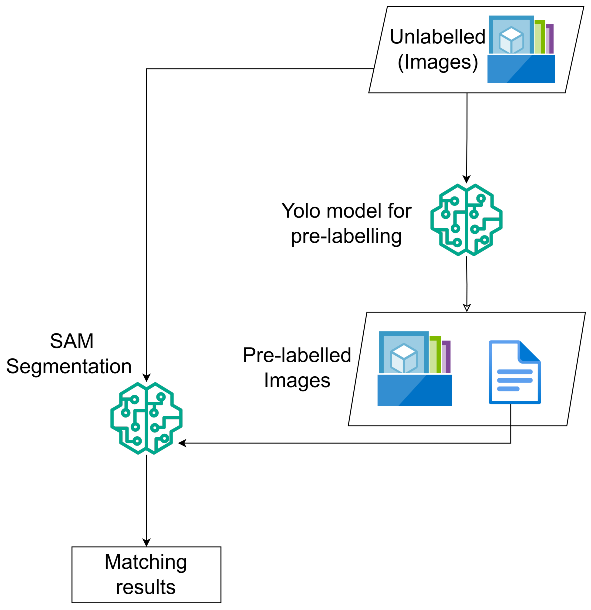


Figure 2: Overview of the proposed labeling pipeline combining YOLO for object detection, SAM for segmentation, and a matching process.

Matching and Filtering (`matching.py`)

To reconcile YOLO's bounding boxes with SAM's segmentation masks, the `matching.py` script computes the Intersection over Union (IoU) between each box and corresponding mask. If IoU falls below a predefined threshold, the annotation is flagged as uncertain and passed to the manual review stage.

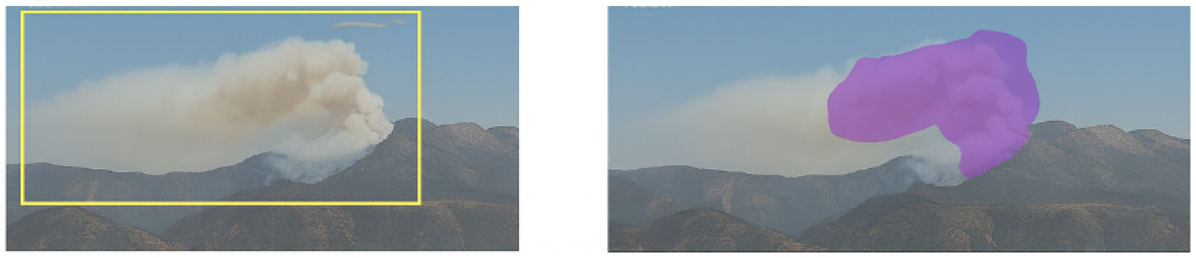


Figure 3: Visual comparison of YOLO's bounding box and SAM's segmentation mask, highlighting the need for a matching criterion.

3.3.3 Output

Images with high-confidence matches are automatically labeled. Low-confidence matches are routed to a manual review step, described in the following section.

3.4 Human-in-the-Loop Review

Some predictions from the pre-labeling stage may be ambiguous or inaccurate. These cases are handled in `human_intervention.py`, which integrates with **Label Studio**, an open-source annotation platform (Heartex 2021).

3.4.1 Input

The input includes uncertain or low-confidence image annotations from the matching stage.

3.4.2 Process

Human annotators review and correct the predicted bounding boxes and segmentation masks using a customized Label Studio interface that displays both YOLO and SAM outputs. This manual verification ensures that the final annotations are of high quality and suitable for training.

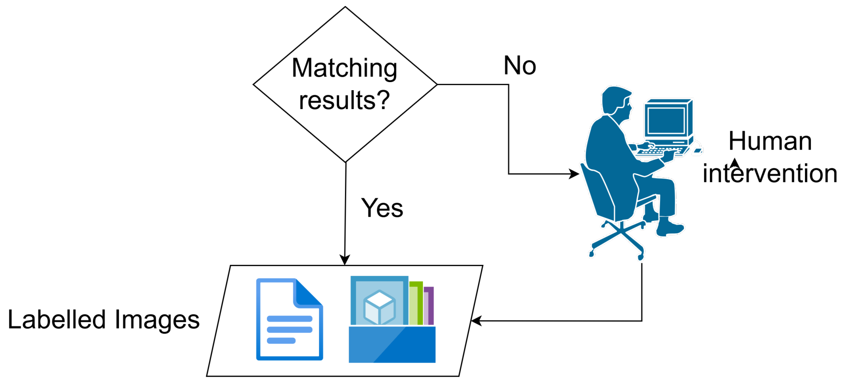
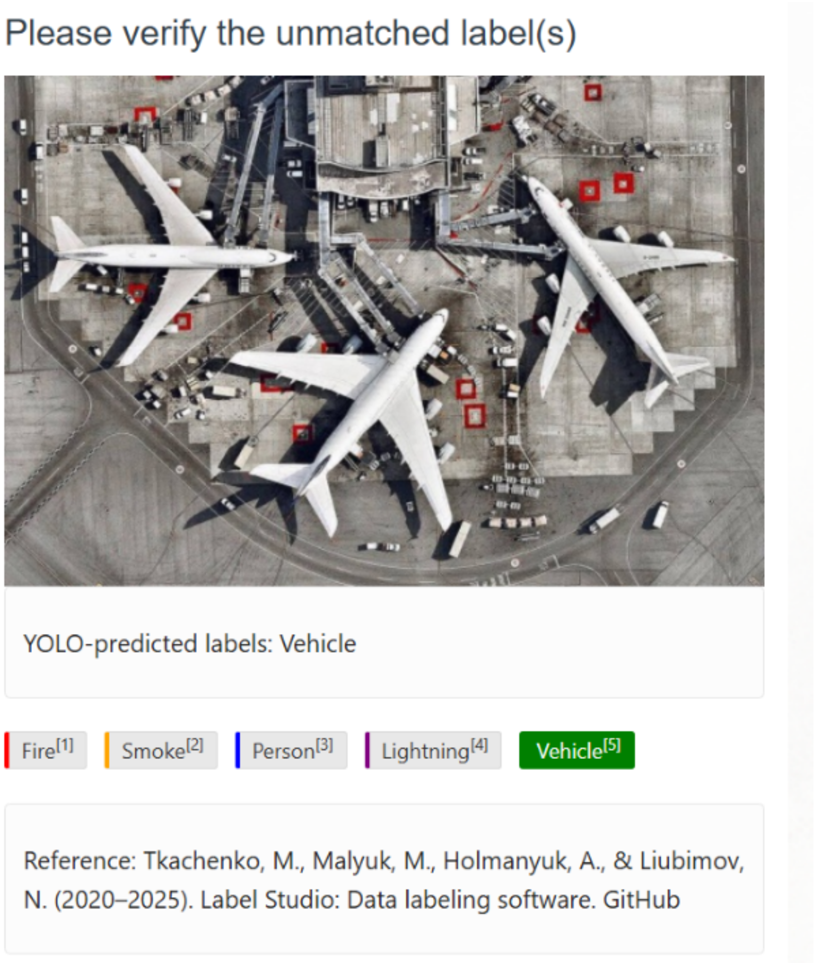


Figure 4: Human-in-the-loop flow using Label Studio to validate flagged predictions.



Please verify the unmatched label(s)

YOLO-predicted labels: Vehicle

Reference: Tkachenko, M., Malyuk, M., Holmanyuk, A., & Liubimov, N. (2020–2025). Label Studio: Data labeling software. GitHub

Figure 5: Label Studio interface displaying pre-labeled objects for reviewer validation.

3.4.3 Output

The output is a validated dataset of labeled wildfire images ready for training a downstream YOLOv8 model.

3.5 Model Retraining with Verified Labels

As new labeled images are added, the model may become outdated if not retrained. The pipeline should track data volume and trigger retraining once it exceeds a defined threshold.

3.5.1 Input

The input includes:

- Labeled images (stored in the `processed/labeled_images/` directory)
- Augmentation configuration file
- Training configuration file

3.5.2 Process

1. Data Augmentation (`augmentation.py`)

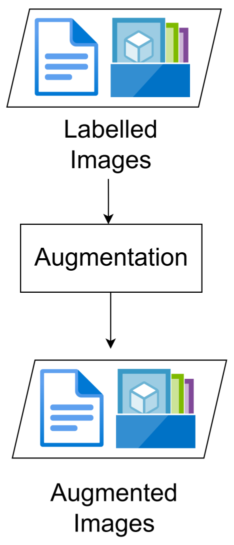


Figure 6: Overview of the augmentation pipeline

This script increases the size and diversity of our dataset through common augmentation techniques (e.g., flipping, brightness/contrast adjustment, noise injection), as specified in the configuration file. These augmentations help reduce overfitting and improve generalization.

2. Model Training and Retraining (`train.py`)

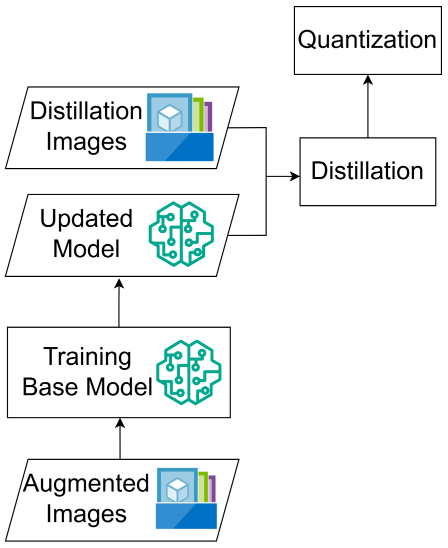


Figure 7: Overview of the model training, distillation, and quantization pipeline

This script fine-tunes the YOLOv8 base model using the augmented dataset and a configuration file, and produces a full trained model file (`.pt`), which is reused in future labeling iterations as the updated pre-labeling model, and passed to the next stage for optimization. Retraining is triggered when new data exceeds a predefined threshold.

3.5.3 Output

- Full trained model (`full_trained_model.pt`)
- Updated training configuration file

3.6 Model Optimization for Deployment

Updating optimized models in production can interrupt downstream processes if not managed carefully. To address this, we automate distillation, quantization, and version-controlled registration to ensure seamless integration.

3.6.1 Input

The input to this stage includes:

- Full trained model (`full_trained_model.pt`)

- Distillation images (stored in the `raw/distilled_images` directory)
- Distillation and quantization configuration files

3.6.2 Process

1. Model Distillation and Quantization (`distill_quantize.py`)

This script optimizes the trained model in two stages. First, it performs **distillation** using the distillation images to produce a distilled model that is smaller and faster. Then it applies **quantization** to further reduce model size, enabling deployment on lightweight platforms.

2. Deployment (`save_model.py`)

This script finalizes the pipeline by registering all model versions in the model registry, with the quantized model marked as the production-ready version for deployment.

3.6.3 Output

- Distilled model (`distilled_model.pt`)
- Quantized model (`quantized_model.pt`)
- Updated distillation configuration file

4 Timeline

With our proposed data pipeline, we've laid out our timeline across 7 tasks:

Task	Description	Date
1	Set up project structure; define pipeline architecture.	May 5 - May 9
2	Implement pre-labeling and SAM verification; build human review UI .	May 12 - May 15
3	Apply data augmentation and integrate model training.	May 19 - May 23
4	Add model distillation and quantization for deployment.	May 26 - May 30
5	Conduct end-to-end testing of the full pipeline.	June 2 - June 6
6	Submit runnable data product and supporting documentation	June 9 - June 11
7	Refine finalize product and written report based on partner feedback.	June 14 - June 25

References

Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. "YOLOv4: Optimal Speed and Accuracy of Object Detection." https://arxiv.org/abs/2004.10934.

Defence Research and Development Canada. 2022. "DRDC Tests Early Detection Wildfire Sensors." https://science.gc.ca/site/science/en/blogs/defence-and-security-science/drdc-tests-early-detection-wildfire-sensors.

Heartex. 2021. "Label Studio: Data Labeling Platform." https://labelstud.io/.

Kirillov, Alexander, Eric Mintun, Nikhila Ravi, et al. 2023. "Segment Anything." https://segment-anything.com/.