**MLOps Final Project**

Abdullah I19-1772

Majid Ahmed I19-1796

**Project Title: Polyglot Interpreter**

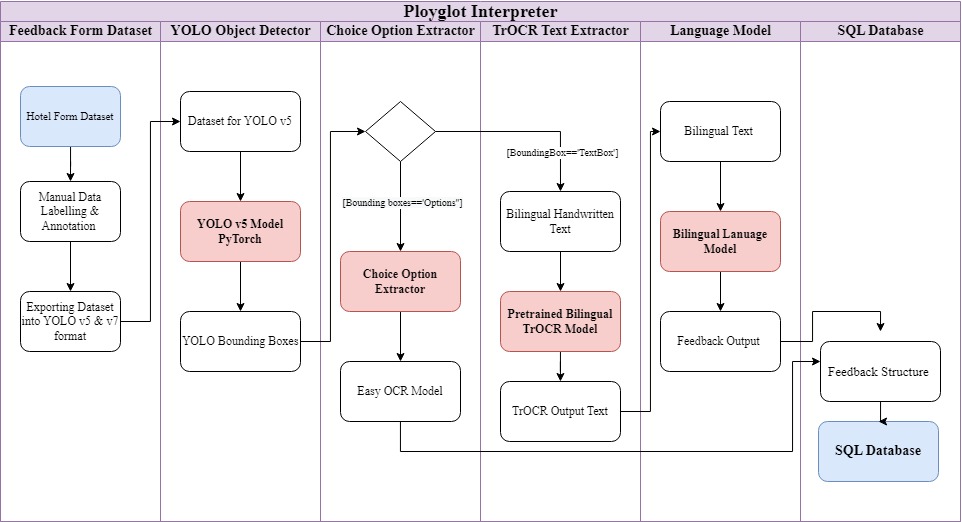
**Project Description:**

Hand-filled hotel feedback forms accumulate daily, overwhelming staff and managers. Consequently, these valuable customer insights often go unexplored, as they are either shelved or discarded due to time constraints. The task at hand is the automatic detection and extraction of hand marked heterogeneous checkboxes (e.g. checkbox, radio button, Yes/No, numeric rating) and multilingual handwritten text (e.g. English, Urdu). Digitization of handwritten forms will allow hotels to gain meaning insights from customer feedback and improve their services.

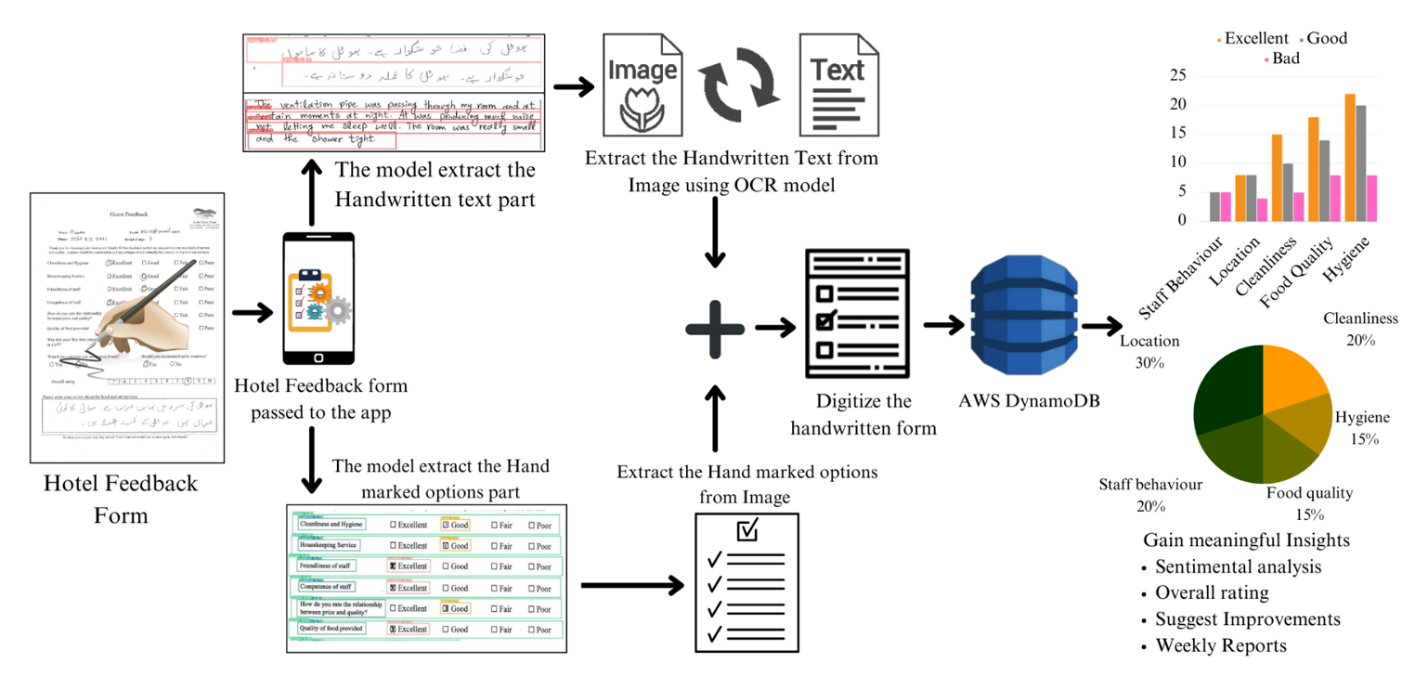
To digitize hotel feedback forms, we need to extract user credentials, user ratings e.g. checkboxes, radio buttons, numeric rating and user comments that can be written in Urdu and English language both. To extract user ratings, we have proposed efficient object detection methodologies. For extraction of handwritten text, we have proposed lightweight custom OCR models for English and Urdu language.

Once these methodologies are combined we are able to digitize the complete form and store it in a database. We can run multiple queries and apply different analytic techniques on the stored data. The analysis will be viewed on a dashboard for hotel employees to gain insights and improve customer services.

**Flow of Project:**



**High Level Architecture of the project:**

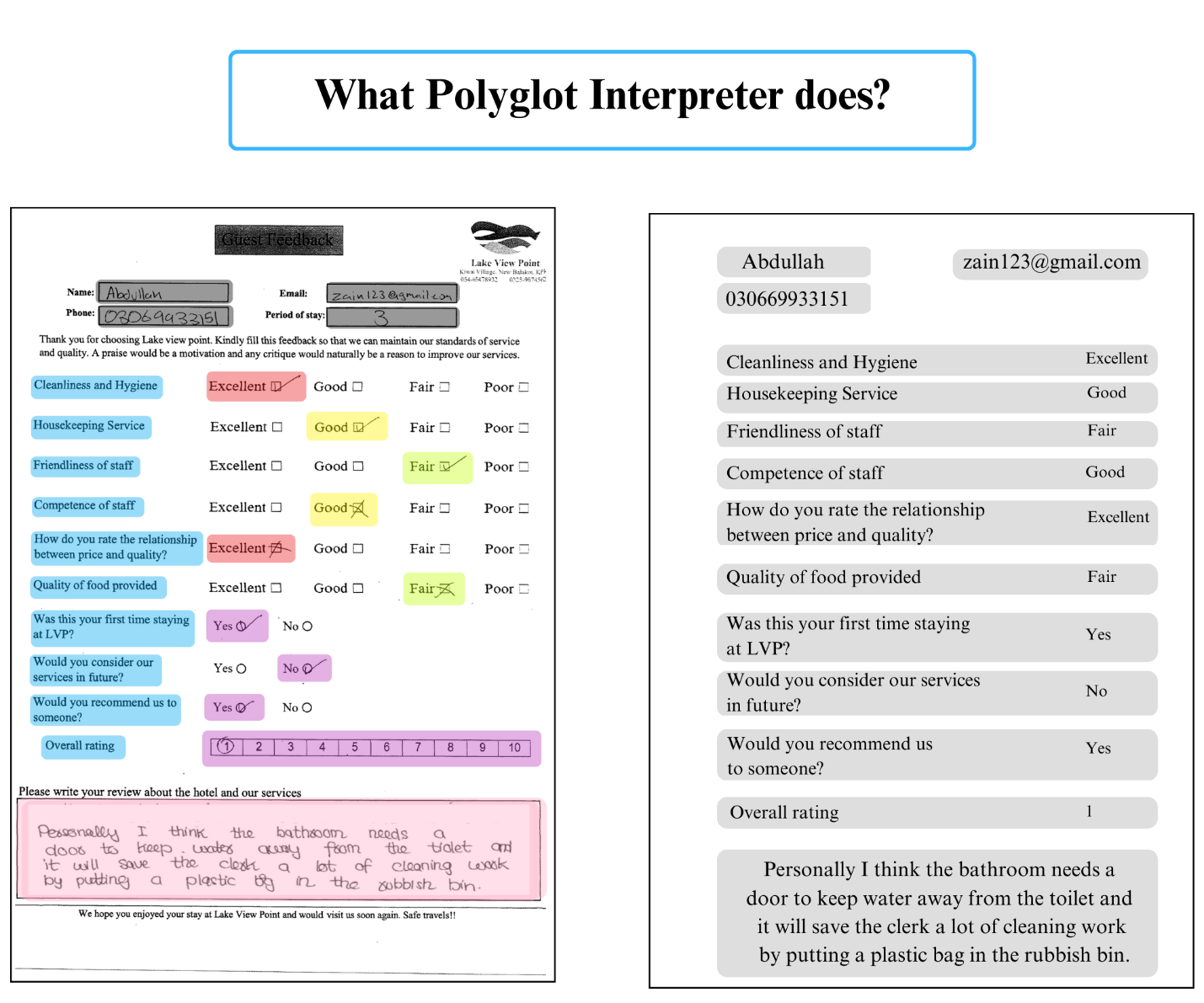
****

The Polyglot Interpreter is a comprehensive solution for digitizing hotel feedback forms, handling both structured and unstructured data, such as multiple-choice questions and handwritten text. A robust dataset was created to train the system, generating four subsets to focus on different aspects of the digitization process: detection and localization of data, line segmentation of text, Optical Character Recognition (OCR) for handwriting, and extraction of numeric data.

Different models were trained for each stage of the process, with the YOLO v5 model used for data localization and line segmentation, and custom OCR models for handwritten text conversion. User credentials were digitized with TrOCR, while multiple-choice questions were segmented and digitized using EasyOCR. Handwritten comments were processed with a combination of TrOCR and UrduLiteOCR.

EasyOCR was chosen over TrOCR for digitizing the questions due to its speed and ease of handling multi-line text. All digitized data was stored in a dedicated database for future analysis and insights. The Polyglot Interpreter therefore represents an innovative solution for transforming analogue customer feedback into actionable, digital insights in the hospitality industry.

**Project Expected Outcome:**

****

**Goals of MLOps Project:**

The goals of this project are to create a Mlops pipeline and containerization for Polyglot interpreter. The major reasons being, so that us students can get hands on practice of how to implement complex MLops pipelines. The second reason being so that this application be delivered and shared with other people without any hassle. We are going to containerize this application and make it deliverable to other people. Another reason is that in future if the model starts to degrade, or we come across new data samples we can easily retrain and deploy the model without going through the hassle of training each model, creating datasets and loading them or orchestrating the whole pipeline hassle free.

Let’s start the implementation of our project using MLops tools and best practices.

**Project stages:**

Let’s break down the project into major stages.

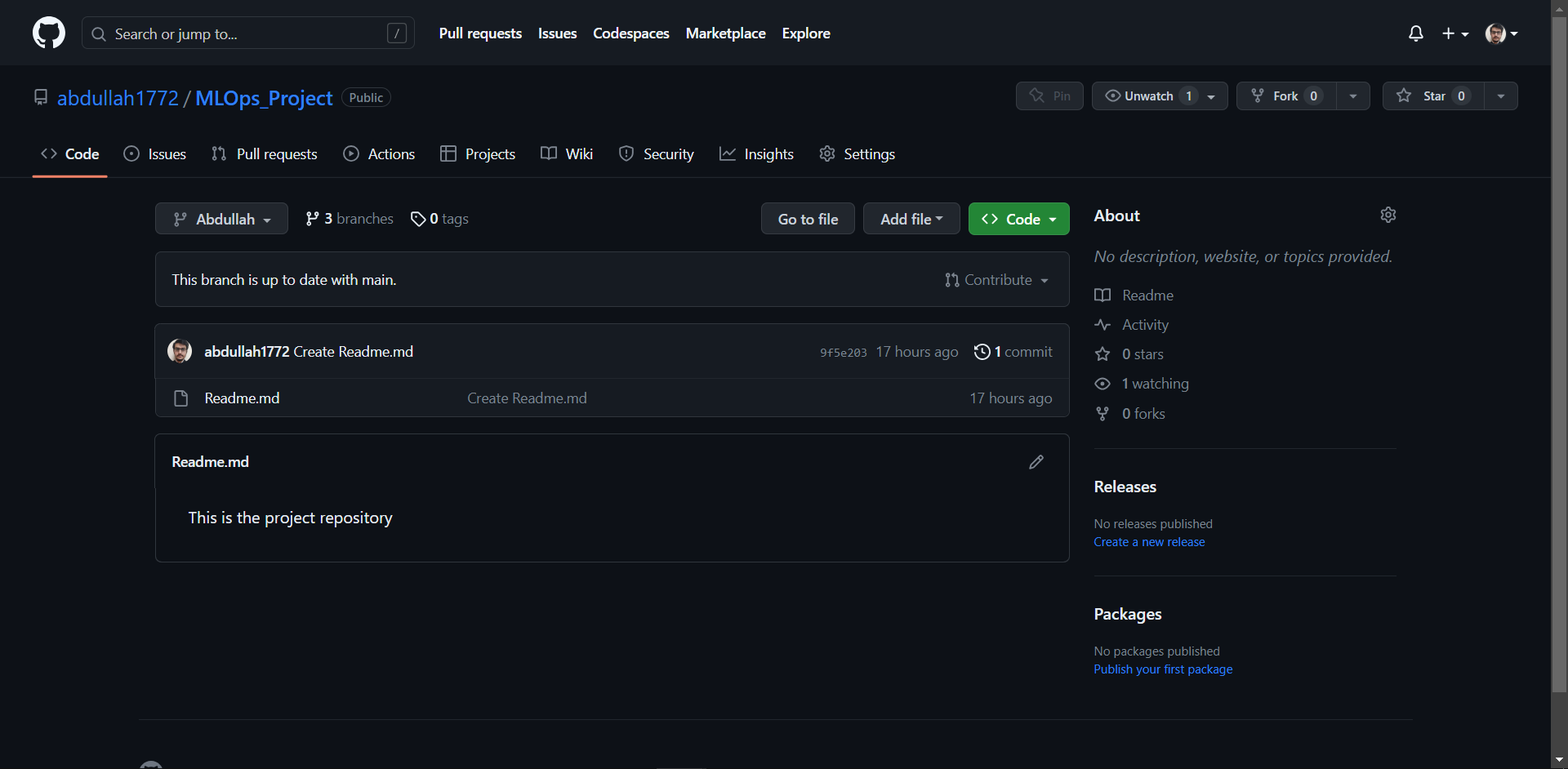
1. Data versioning
2. Training YOLO models
3. Training OCR models
4. Creating an inference\prediction app
5. Containerizing and making the app delivery ready

**1.** **Data versioning**

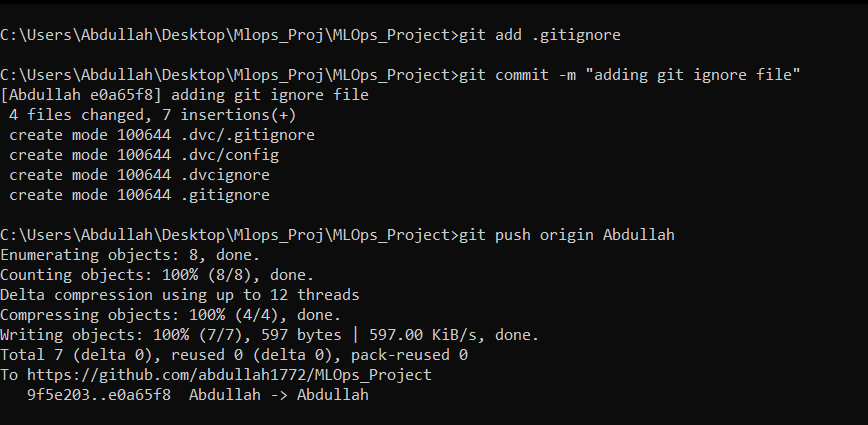
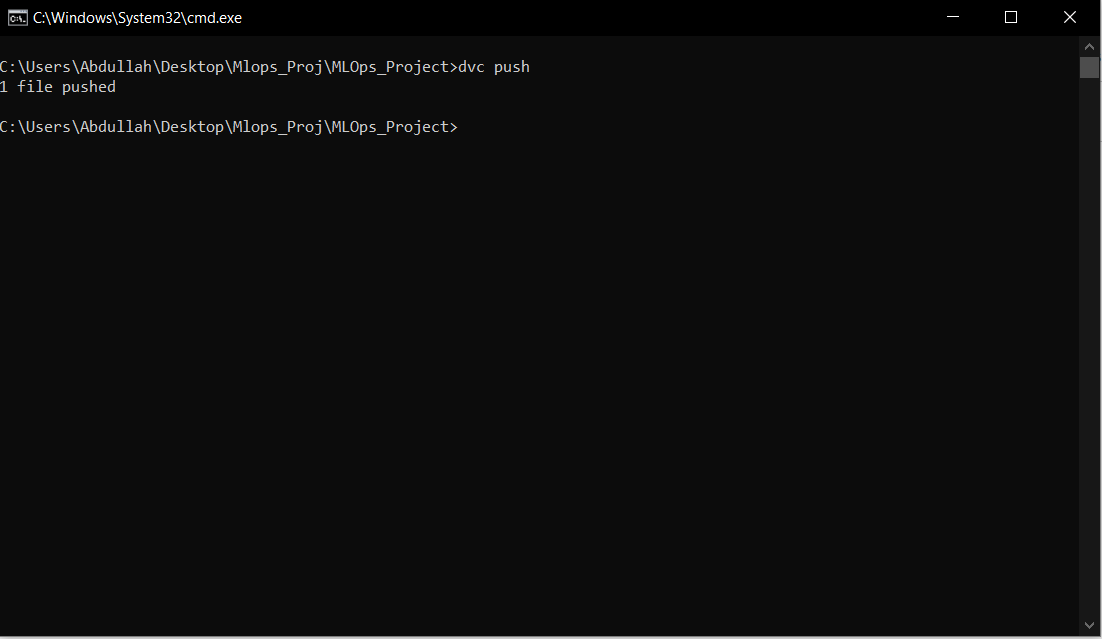
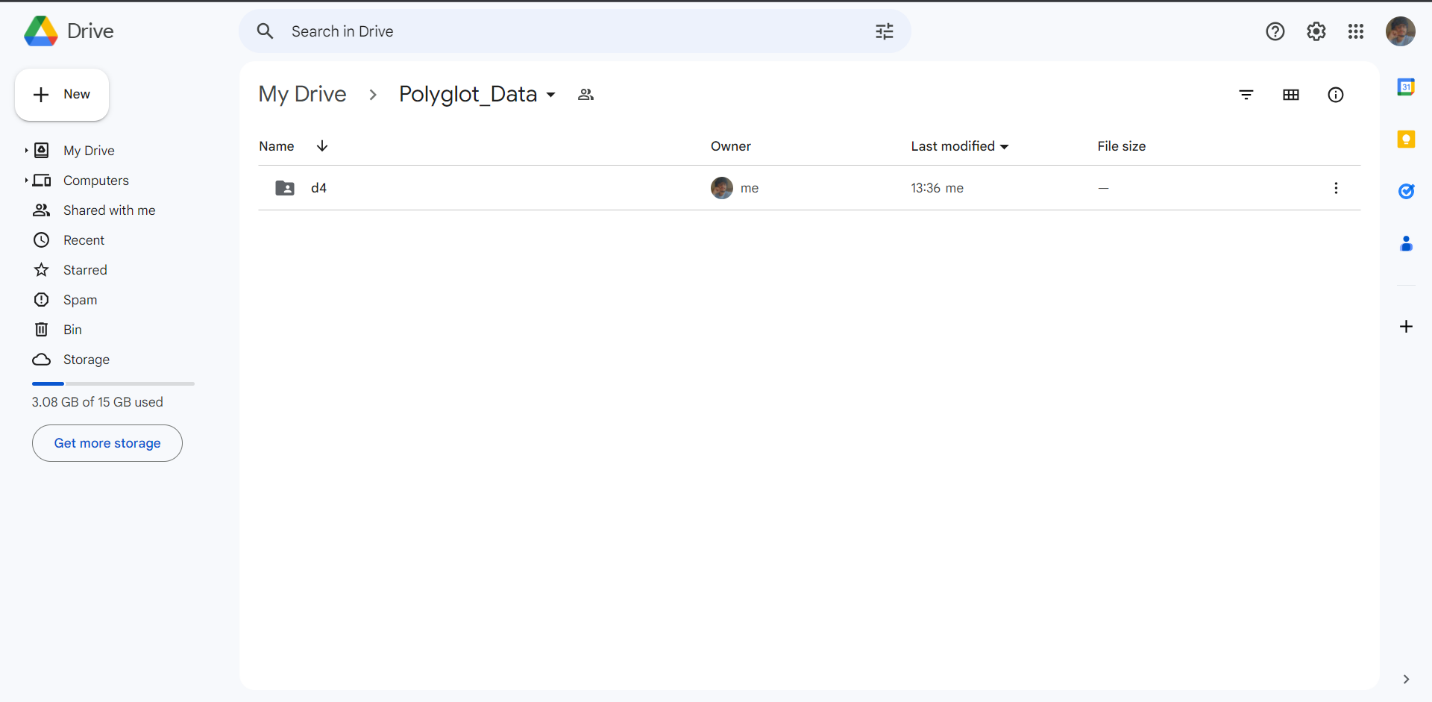
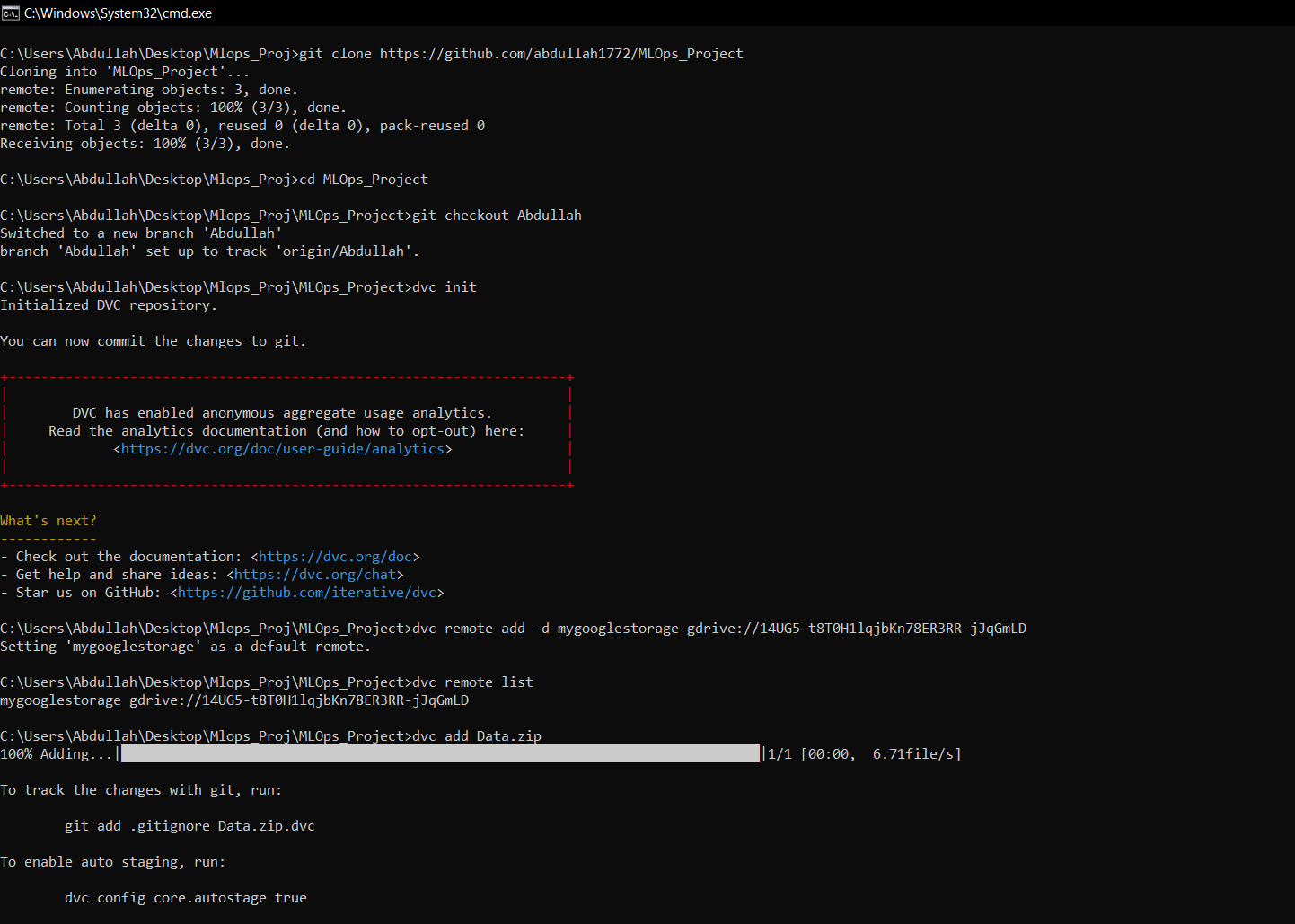
As mentioned above we have 4 different versions of dataset. 3 for training yolo models. 1 for training OCR model.

We are using DVC for versioning dataset.

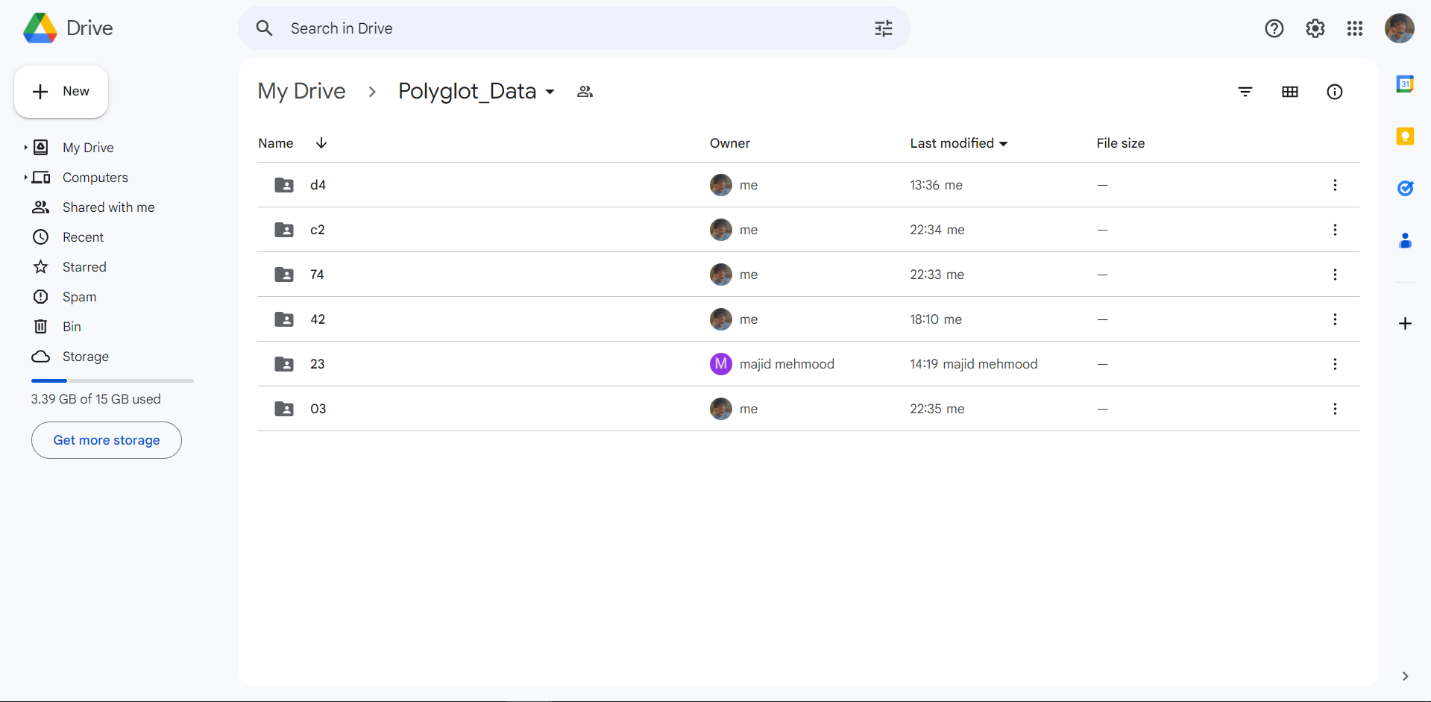
We are going to keep the .dvc files of the datsets on github repo. While the actual dataset is going to be stored on the Google drive. In the following screenshots you can see that we have pushed the dataset to google drive and github using dvc.



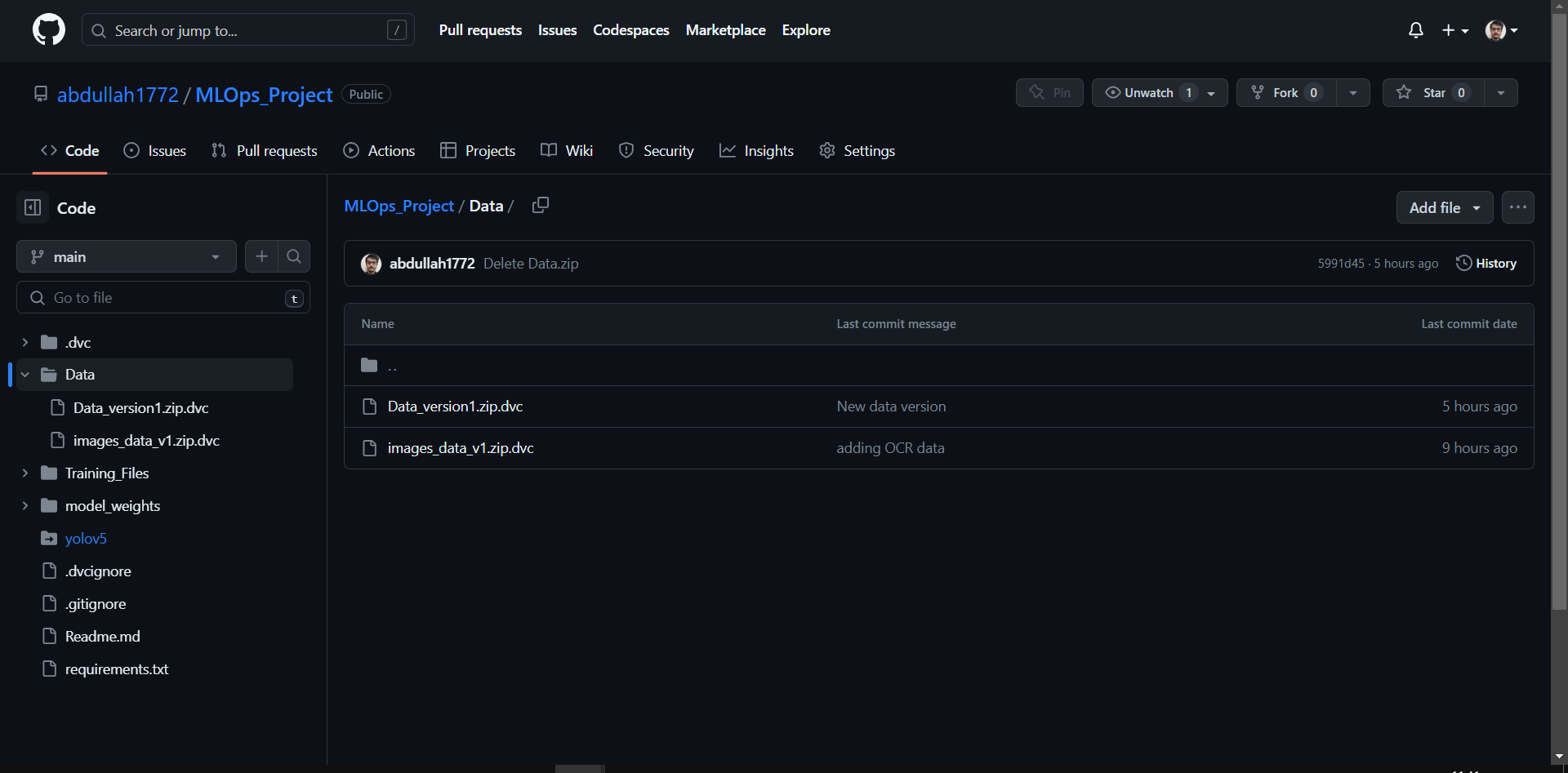
First we initialized our github repo and created 3 branches. Main, Abdullah and Majid. Main for final deployment and one branch for each team members.

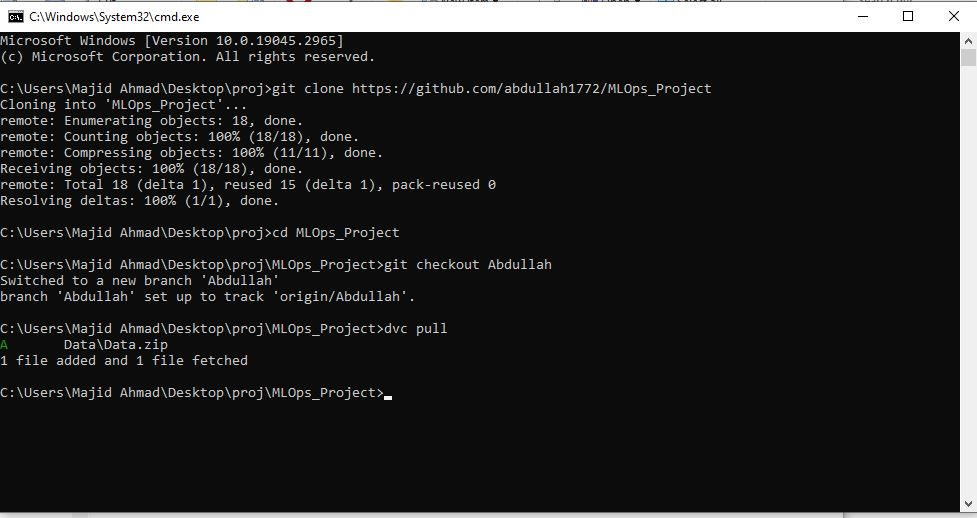


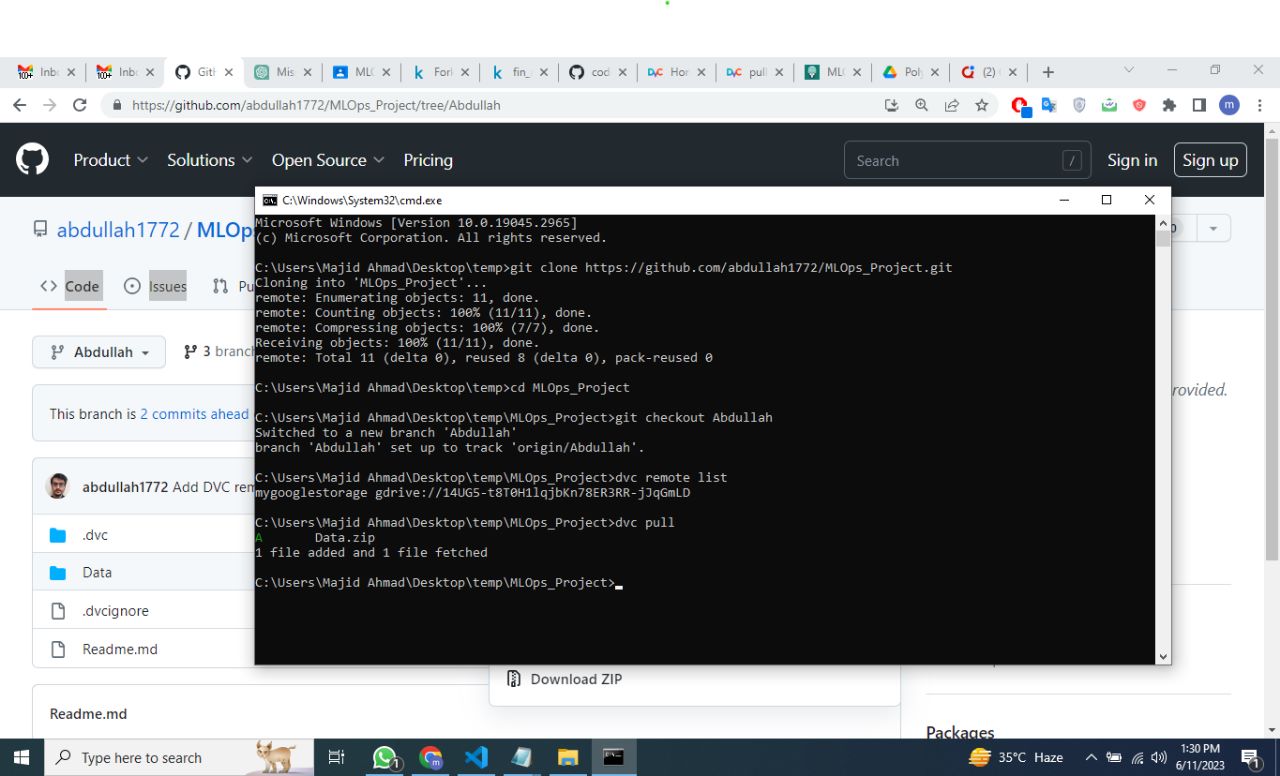
Here is the final version of our datafiles:



As you can see we have pushed multiple data files on gdrive. Some of those files are model weights we’ll get to them later.







Abdullah was responsible for pushing data for YOLO training.

Majid was responsible for pushing data for OCR training.

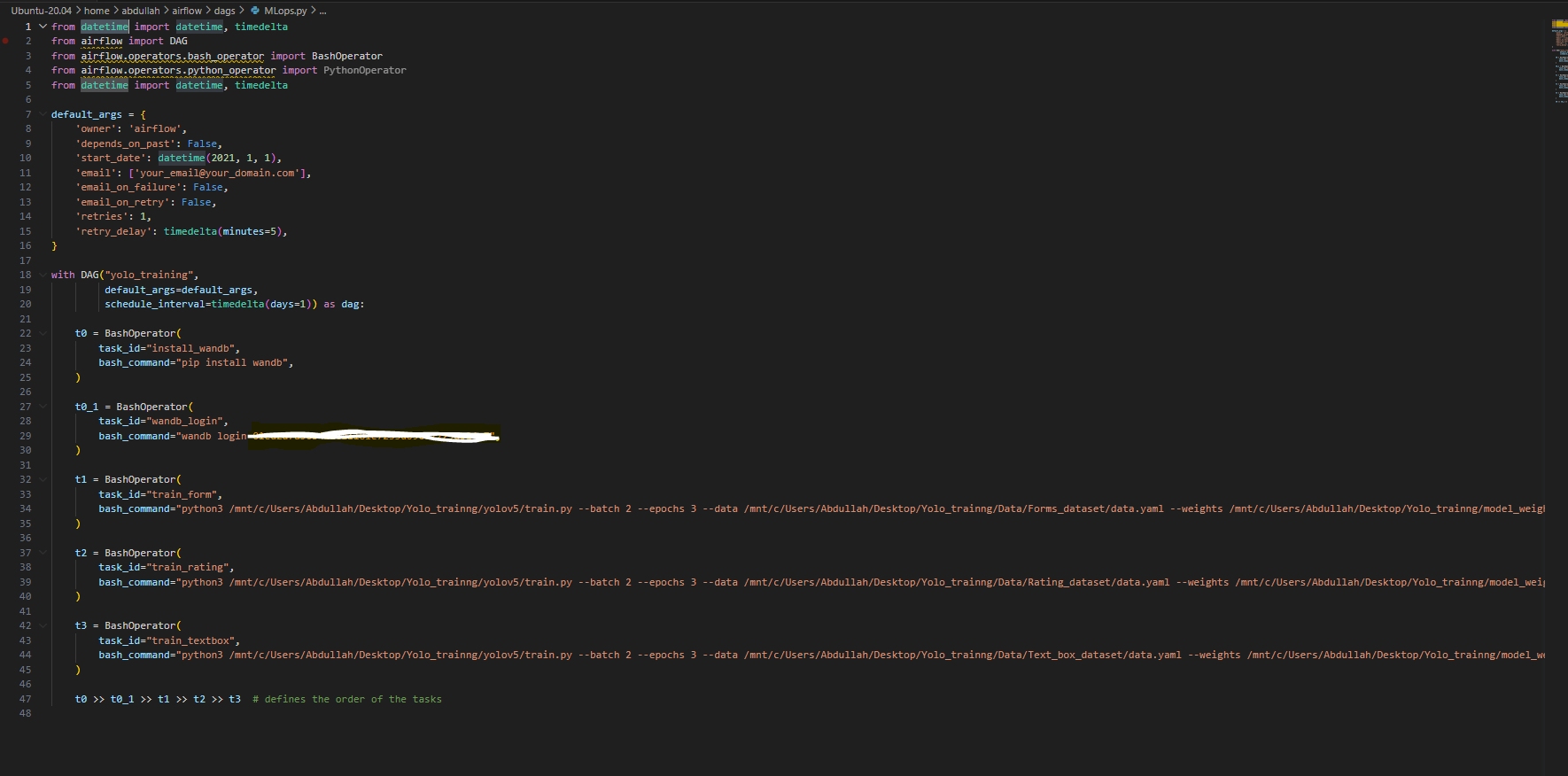
**2. Training YOLO models**

Once we had our datasets setup, it was time for training YOLO models. We are going to train 3 different YOLO models. One for form predictions, One for Textbox prediction and One for Rating scale predictions.

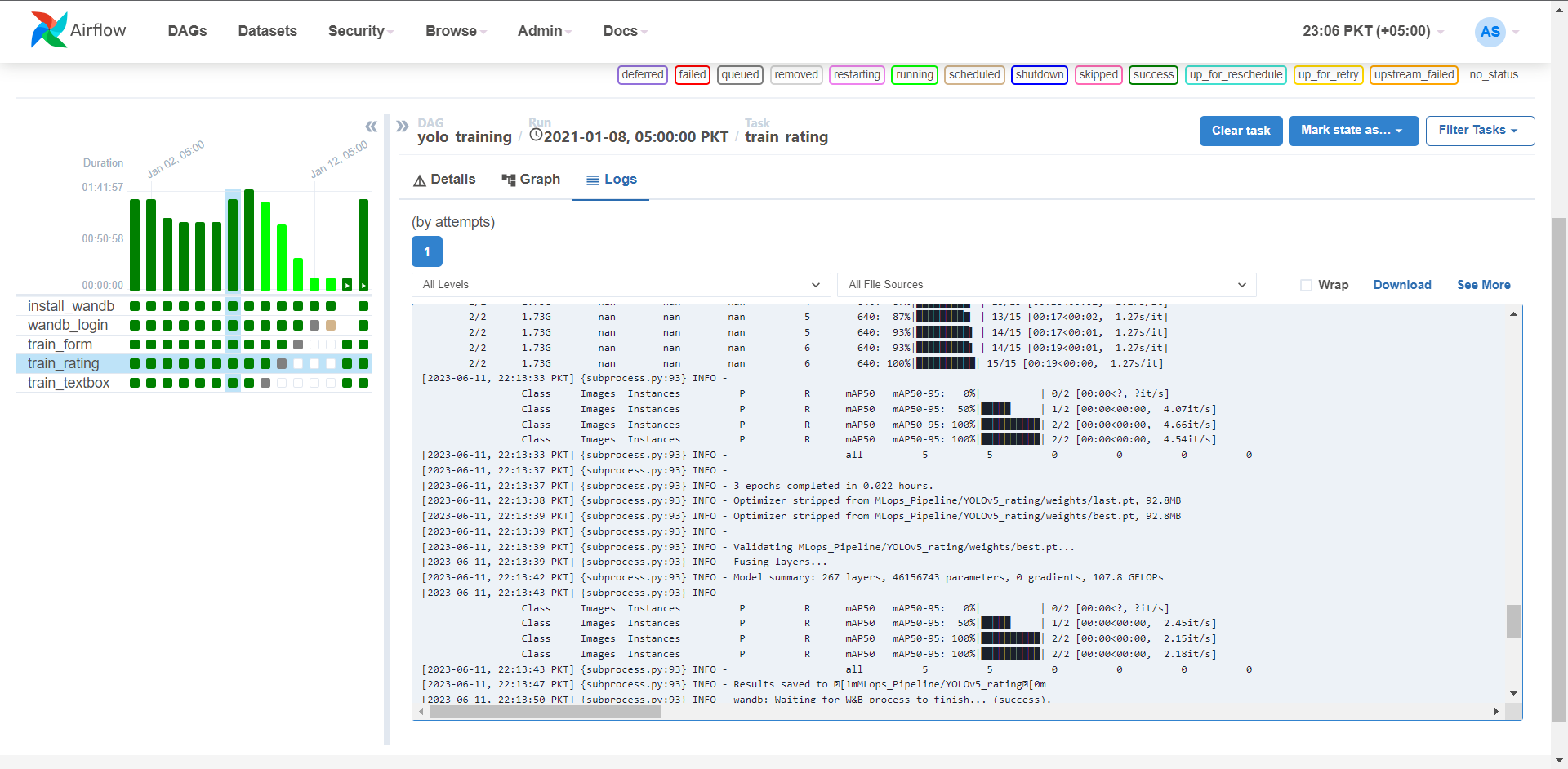
For achieving this task we are going to use Apache Airflow. Airflow will allow us to schedule and train these models. We are going to integrate Wandb with YOLO models for metric, parameters and the model weights. We are also going to save the weights on github using dvc and gdrive.

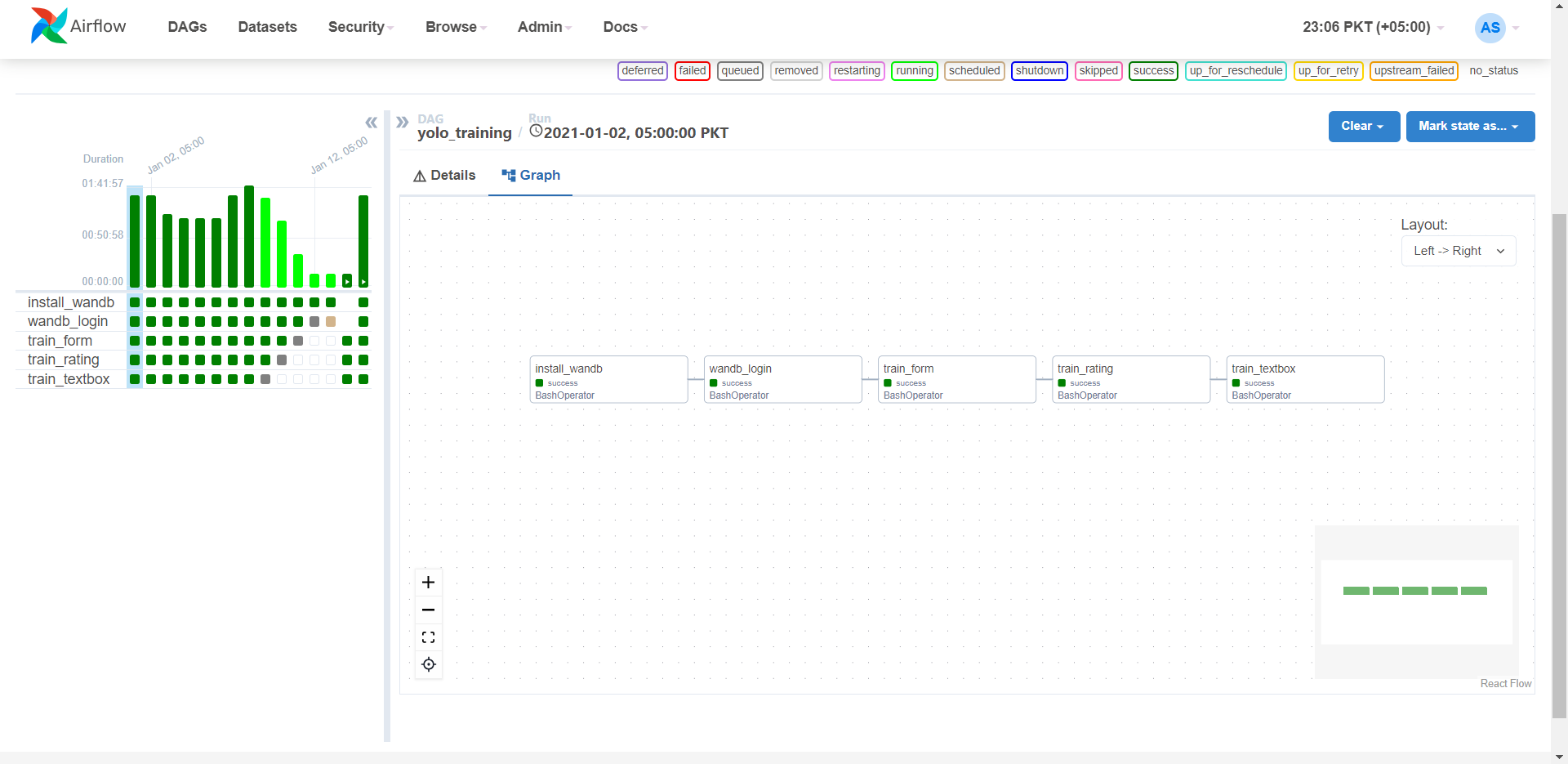
To accomplish the task of training and scheduling we created a .py file. We created a total of 5 tasks, 2 for initializing Wandb and 3 for the 3 YOLO models.

You can see the tasks below.



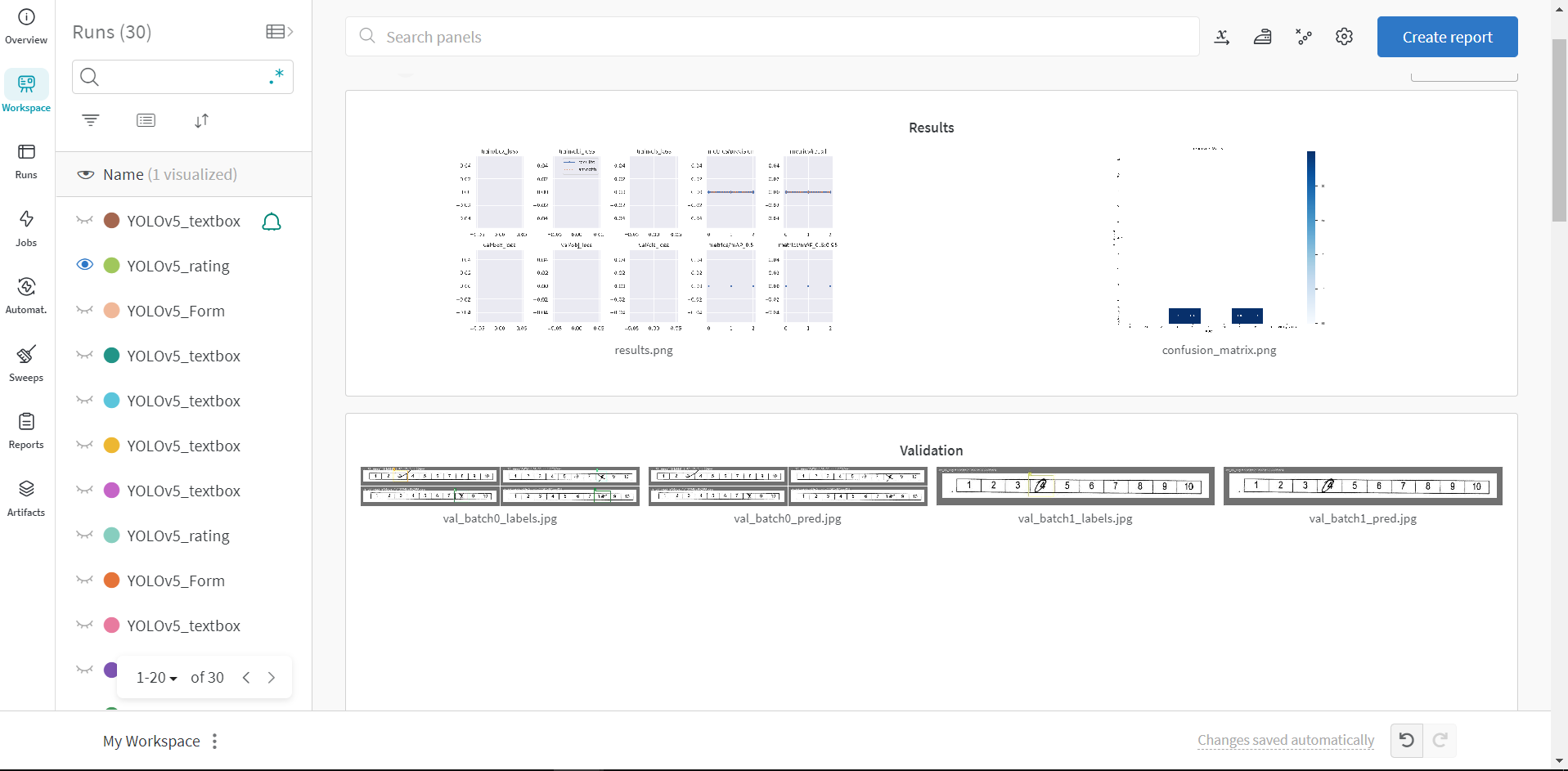
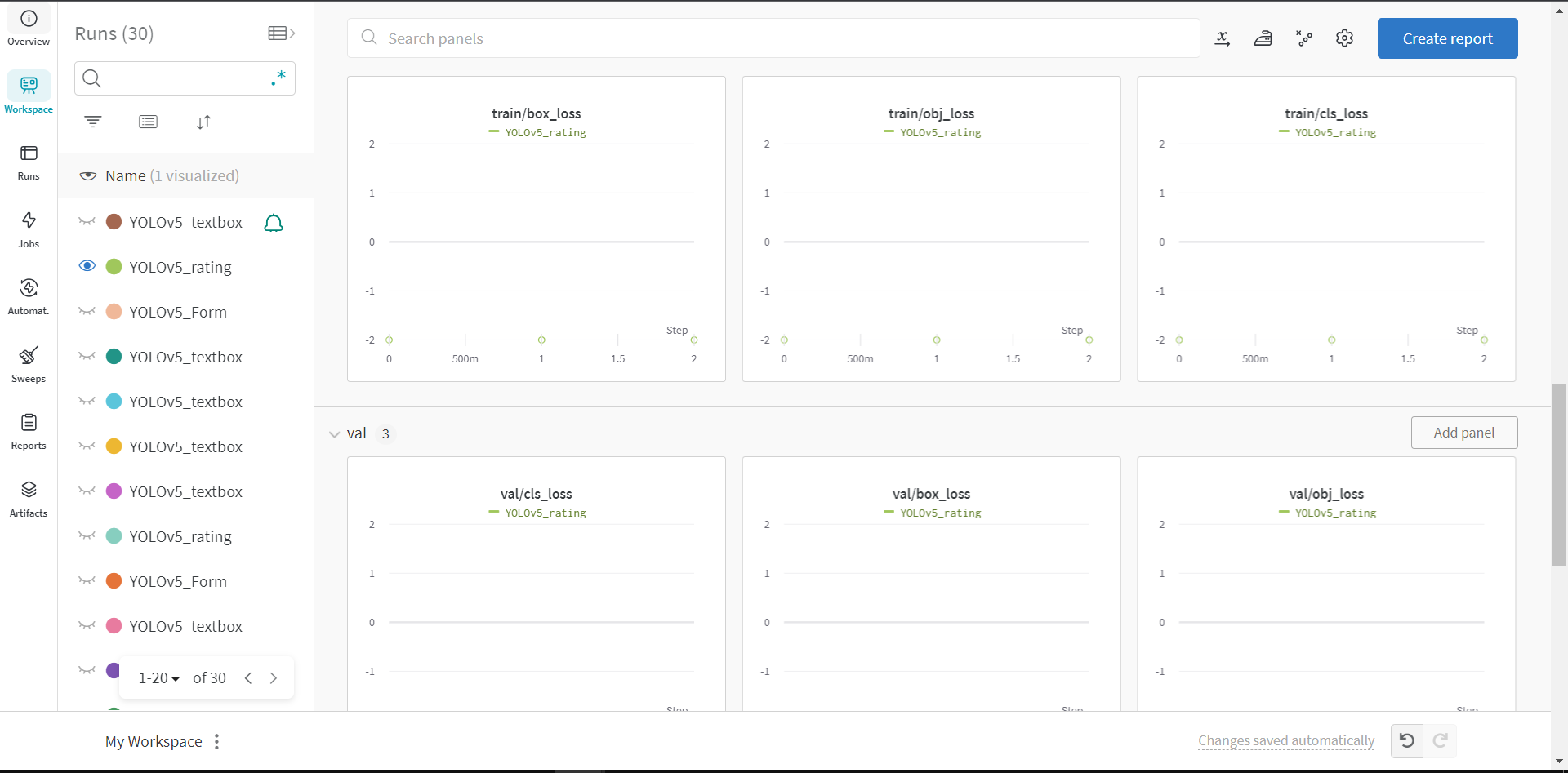
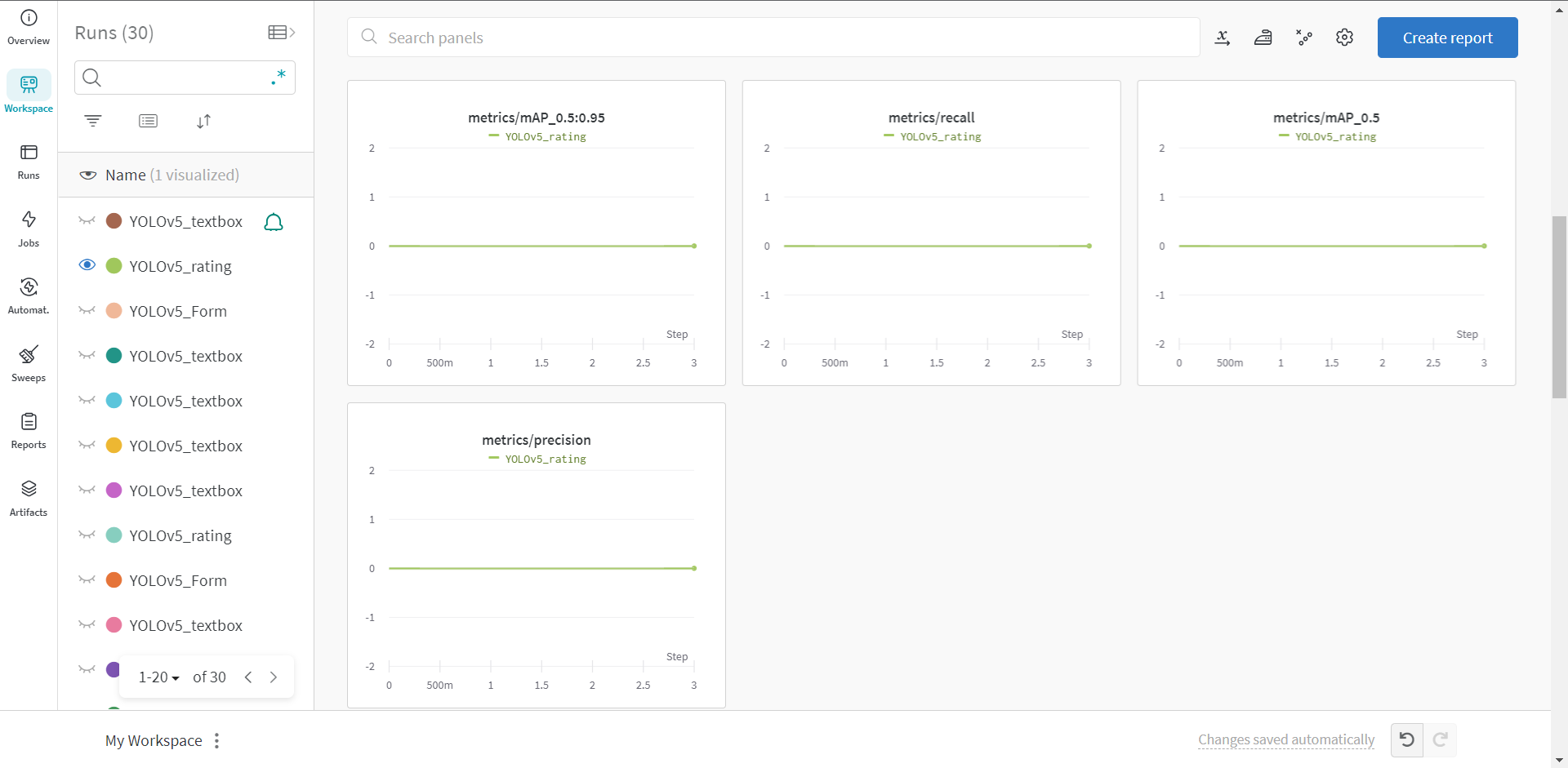
Next we launched the airflow webserver and the scheduler and ran our DAG.



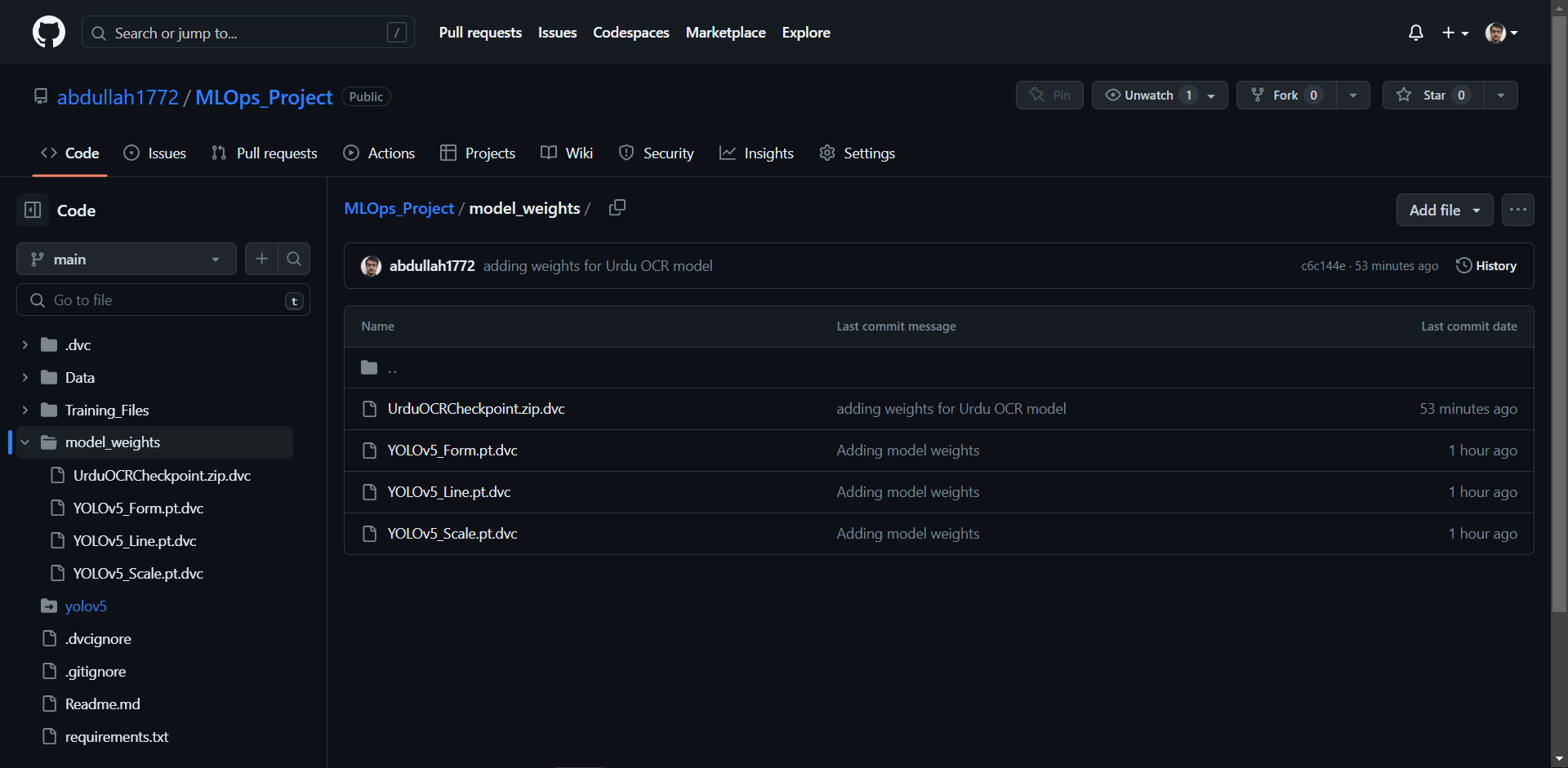
****

As you can see we trained the models multiple times using different parameters and dataversions.

Our logs were saved on Wandb.



Once we have completed the training we, choose the most recent weights, pushed them to dvc and remote storage. We pushed the DVC files to Github.



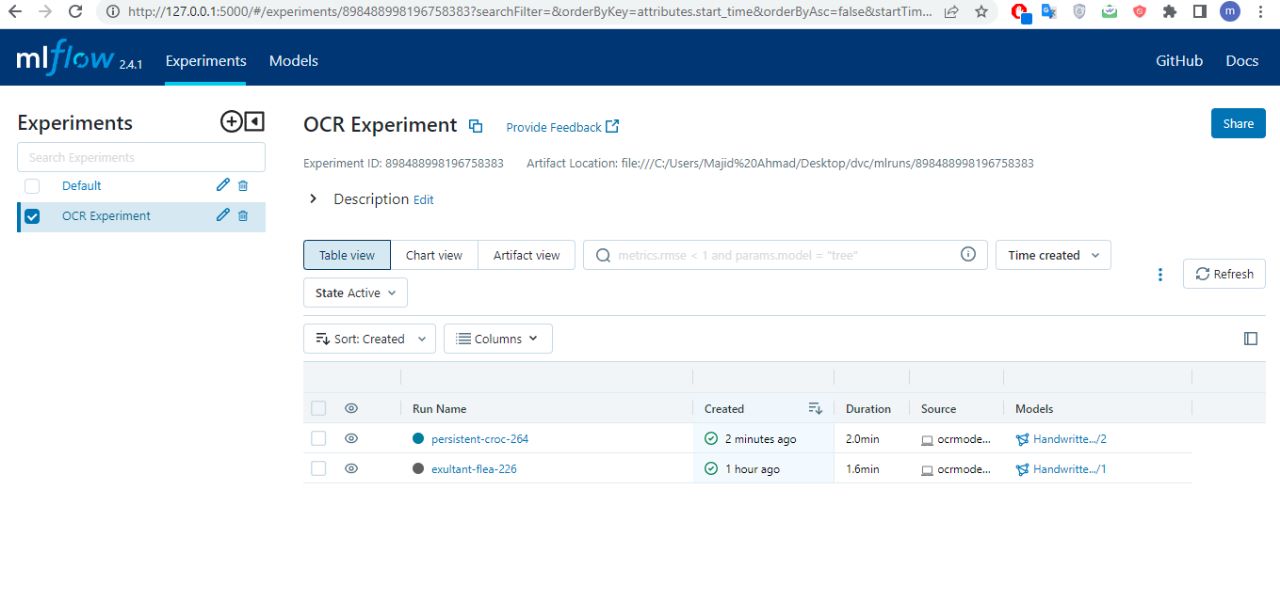
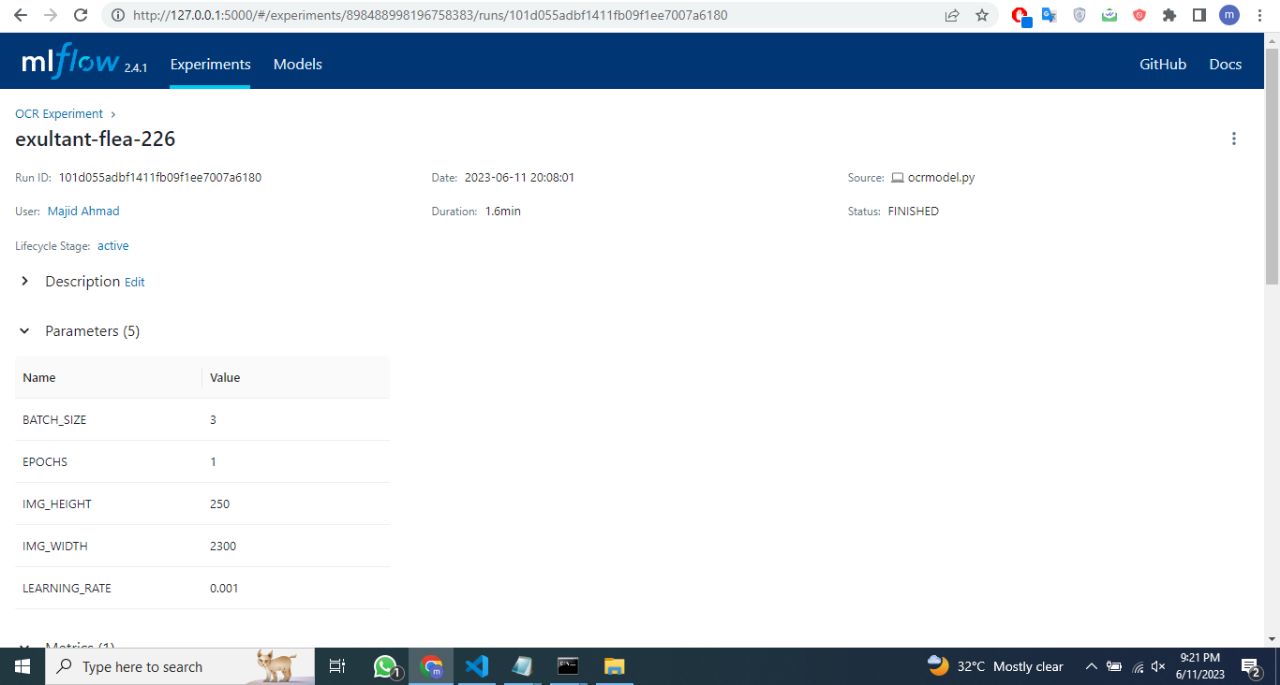
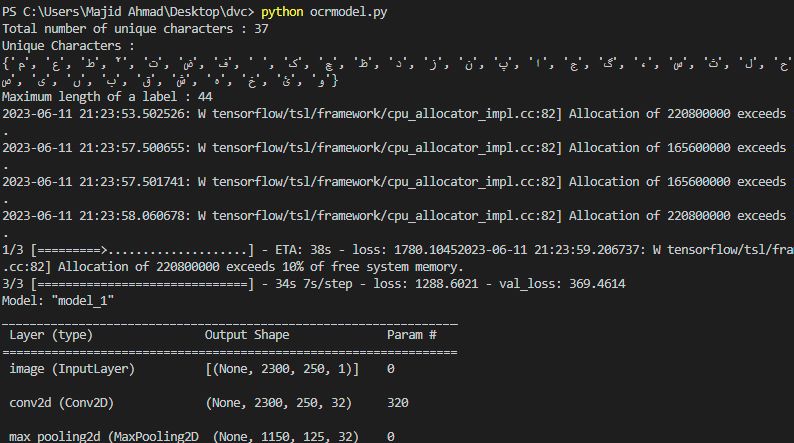
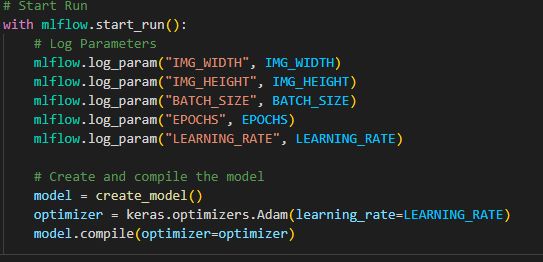
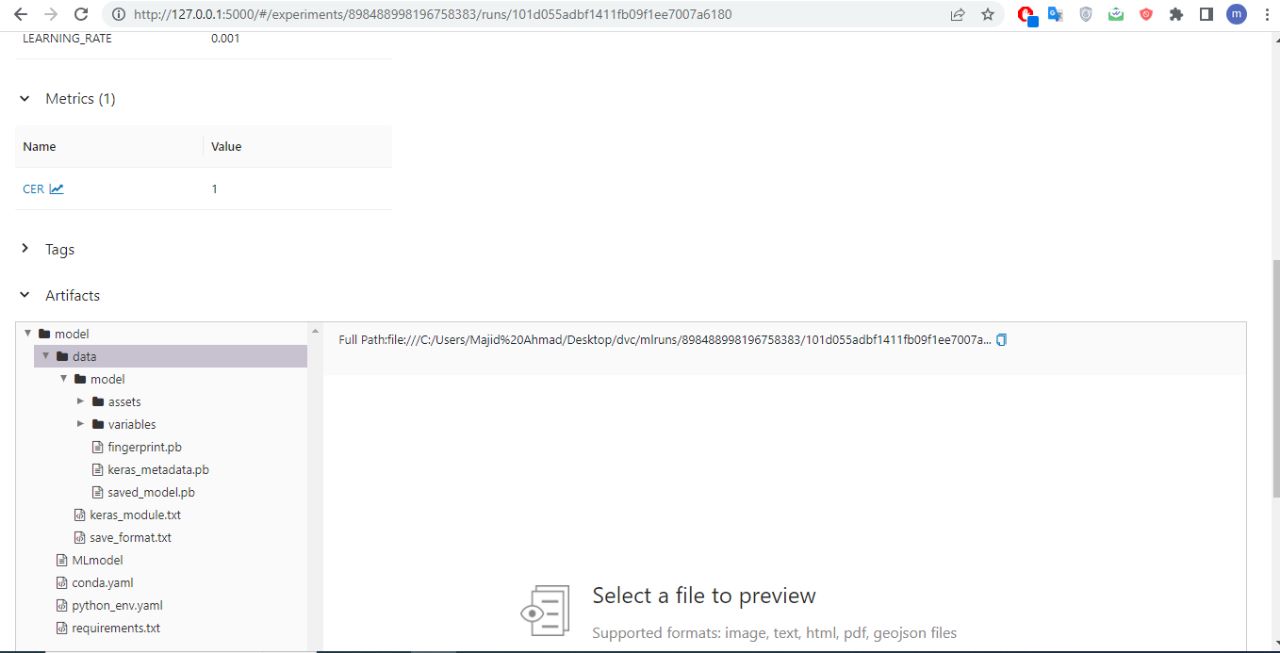
As you can see we pushed the dvc files of our trained weights on github.

We showed in the previous images as well, where multiple files were stored on gdrive, those were the weights. Each weight is above 50mbs and github only allows less than 50 mbs files. Using this technique we showed that we can share weights using Wandb and DVC.

**3.** **Training OCR mode**

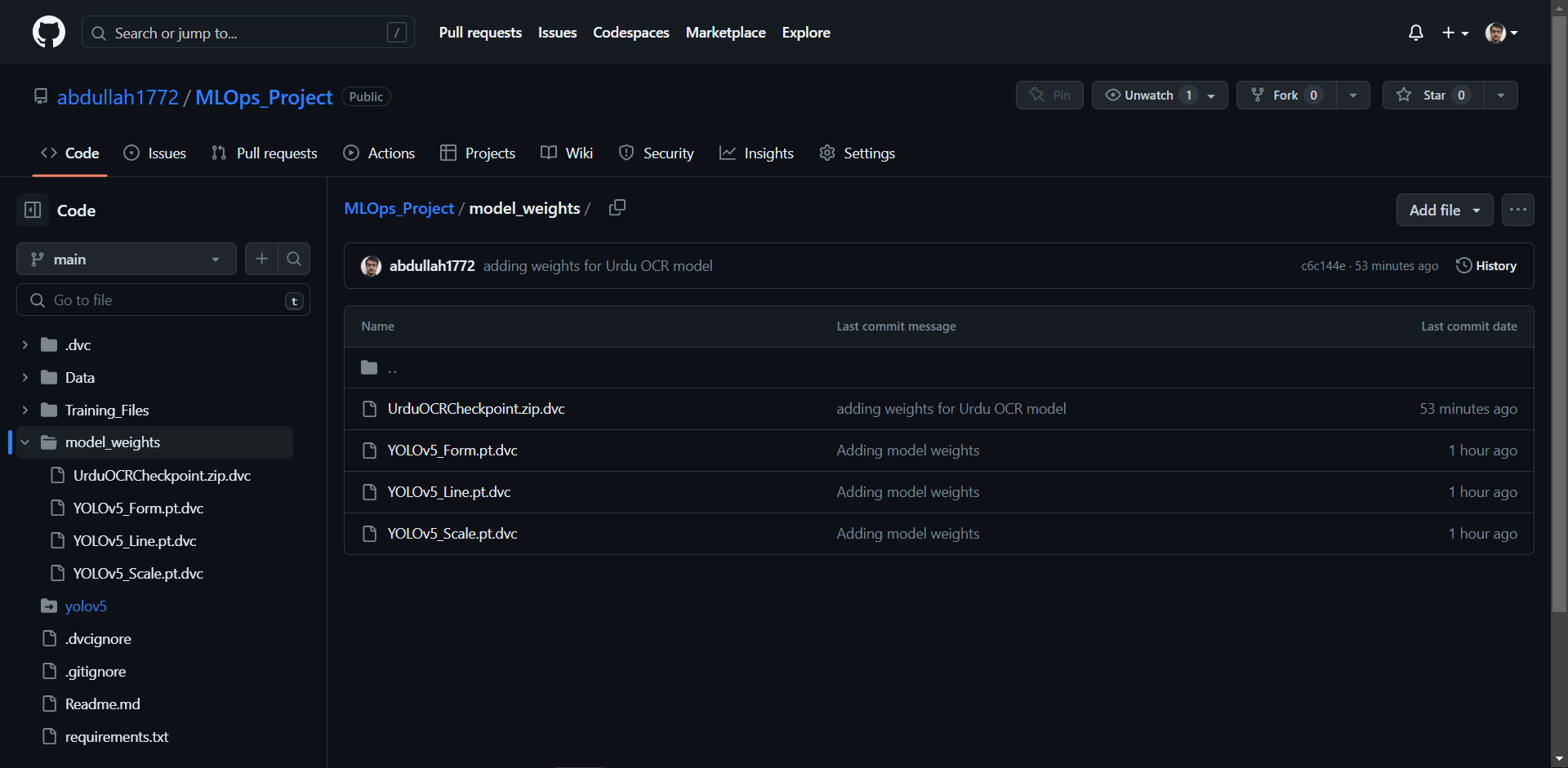
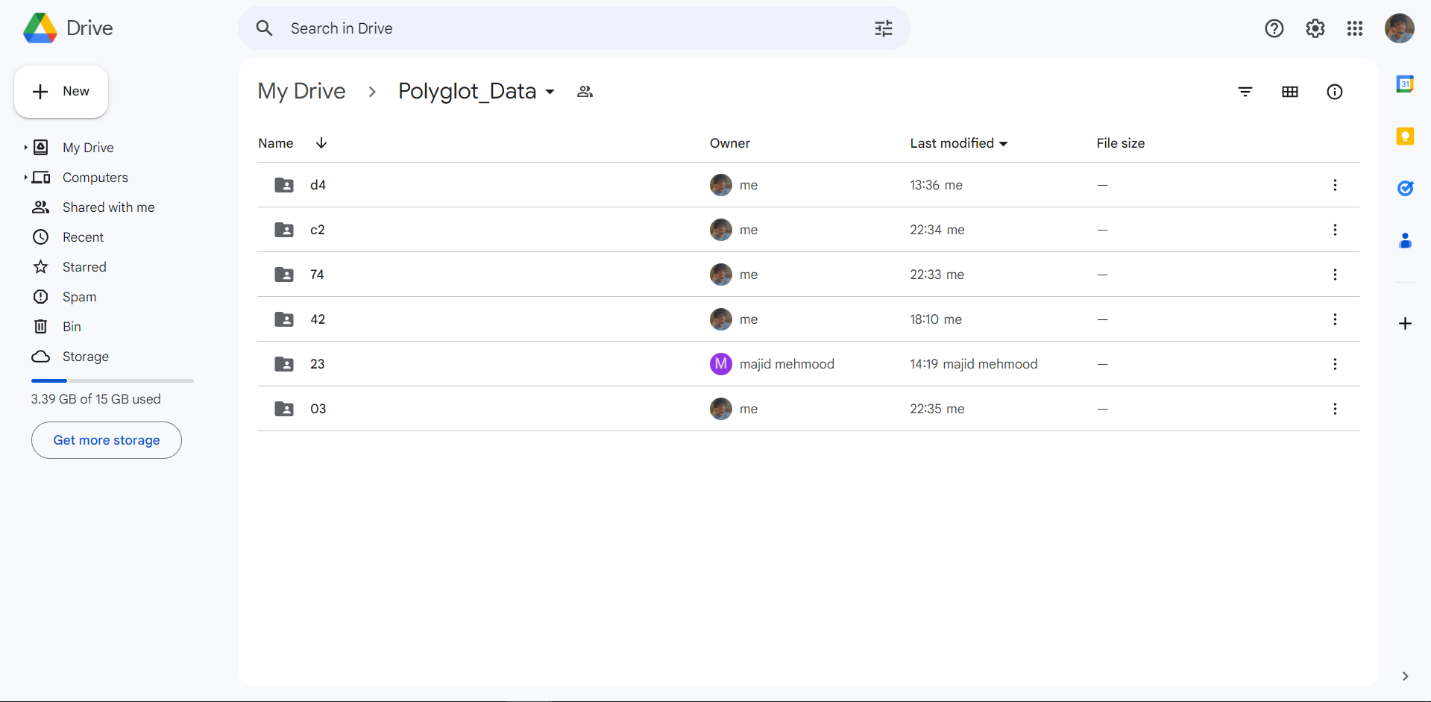
The next crucial step in our pipeline is the training of our OCR. We are going to use Mlflow to log weights and save models. There are 2 main reasons for that. Our Ocr code, is not airflow compatible and we also wanted to add MLflow to our project as well to demonstrate our command over it.

The OCR is a built from scratch and is still in the process of compilation changes and under production, so producing a airflow compatible code is not possible as of right now. We are going to use MLflow to train our OCR.



Below you can see that we have created multiple experiments for MLflow with different parameters and different versions of dataset.

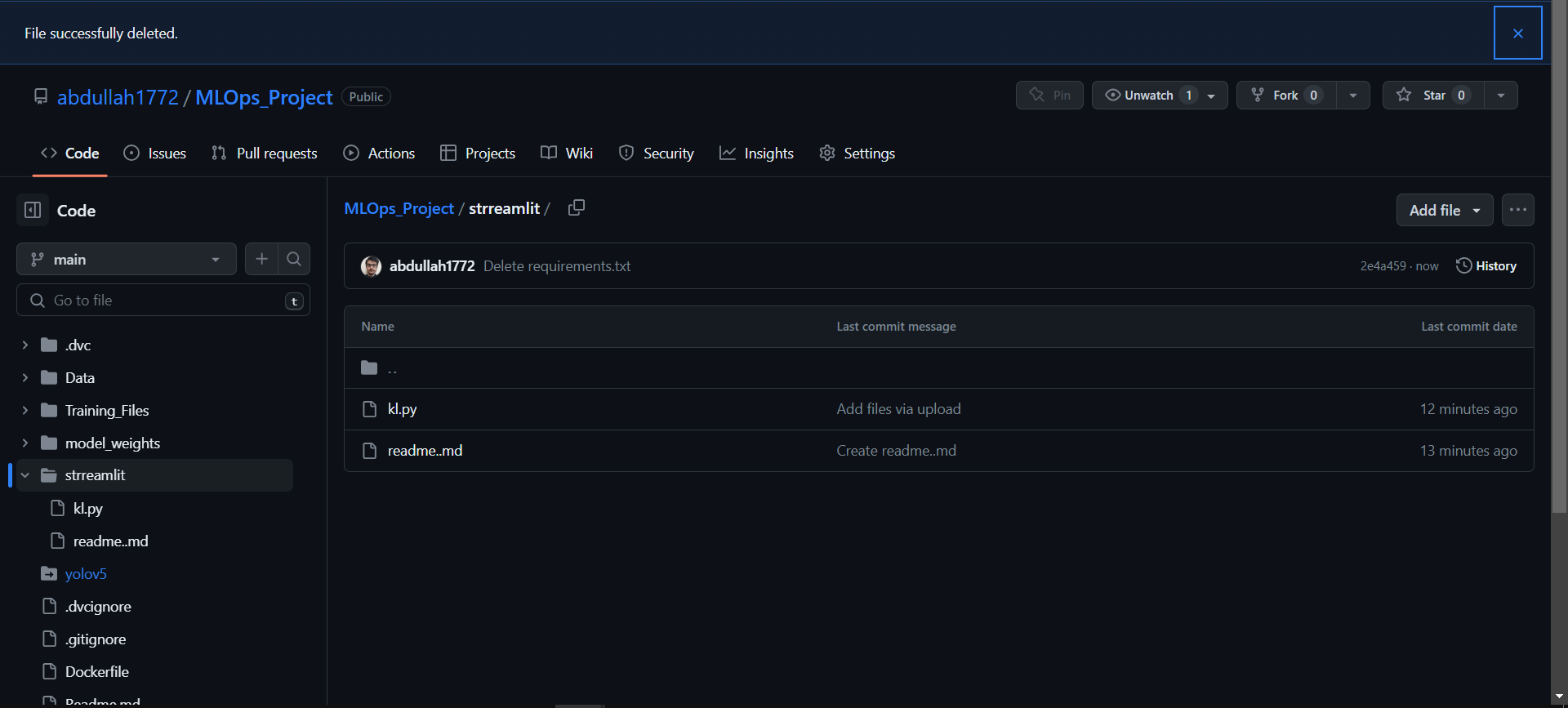
We next used github, dvc and google drive to share these weights.



**4. Creating an inference\prediction app**

Once all the models have been trained, we can create an inference app to get our data digitized.

For creating this inference app we are going to use streamlit application.





This is a simple application, where you can upload an image of handwritten urdu document and it’ll give you the digitized text of that image.

**6.Containerizing and making the app delivery ready**

To run the application locally, you first need to have Docker installed on your machine. After ensuring Docker is running, you can execute the following command in the terminal at the root directory of the application (where the Dockerfile resides):

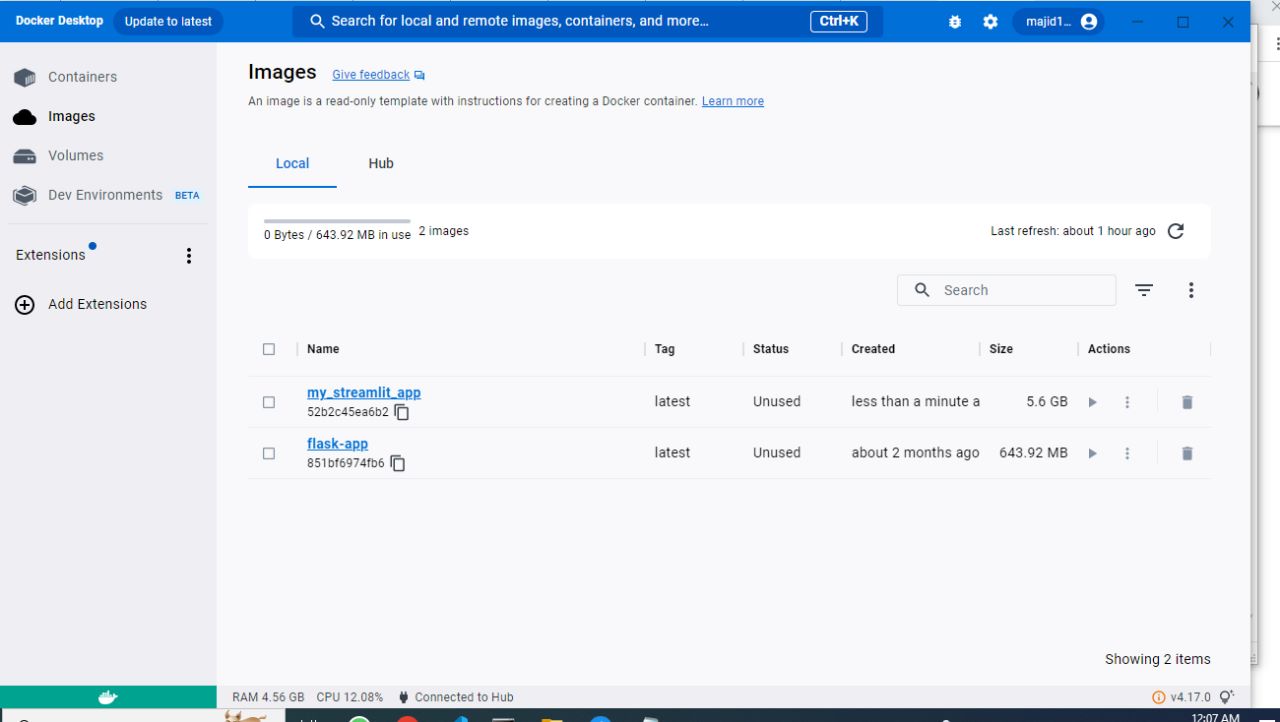
docker build -t urdu-ocr-app:latest .

docker run -p 8501:8501 urdu-ocr-app:latest

This will build the Docker image with the tag urdu-ocr-app:latest and then run the Docker container. The -p 8501:8501 argument maps the port 8501 inside the Docker container to port 8501 on your local machine.

After running the Docker container, open a web browser and navigate to localhost:8501 to see the Streamlit application.

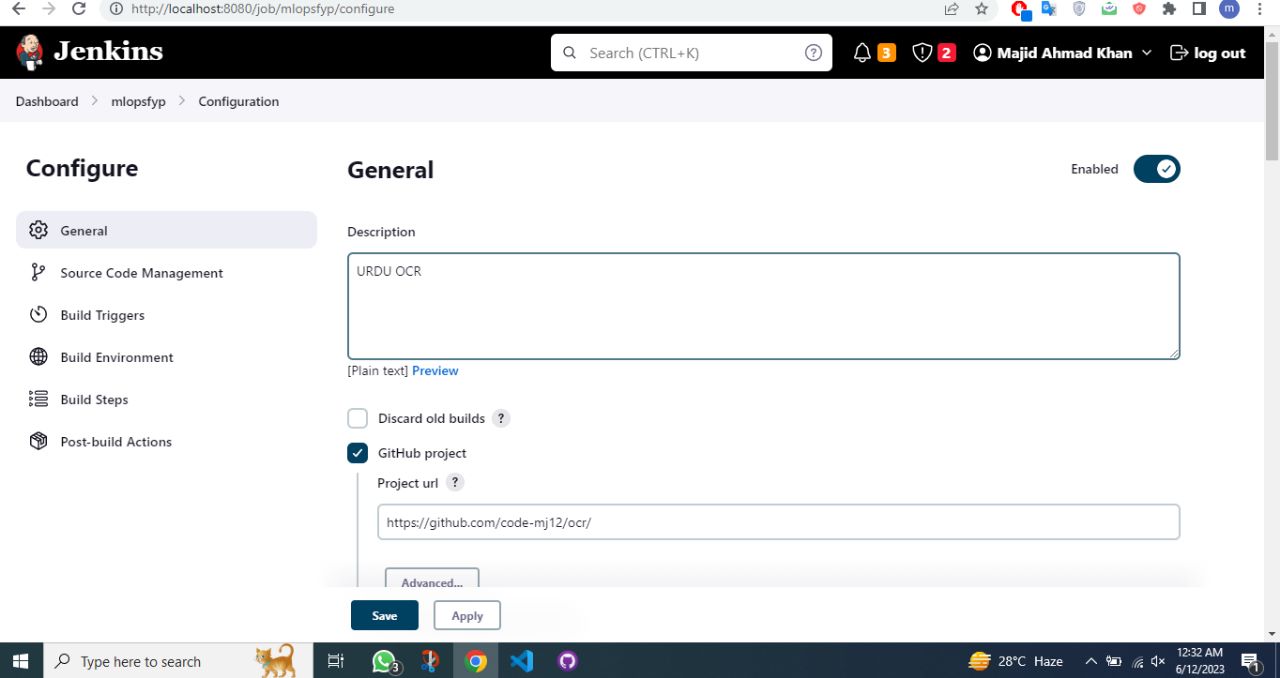
The following screenshot shows the application running in Docker:

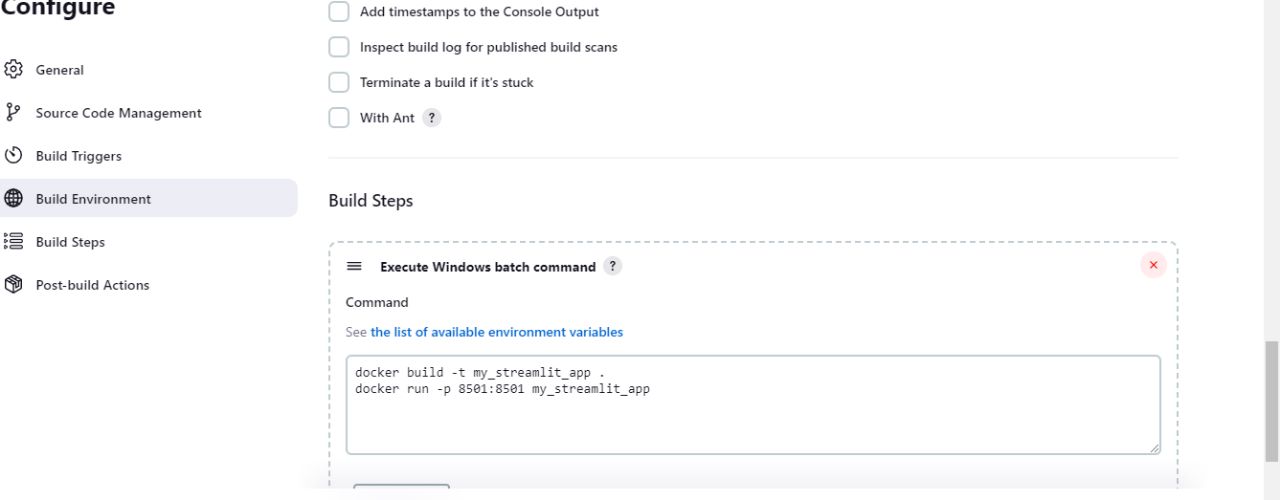


In the application, you can upload an image using the "Choose an image..." file uploader. After an image is uploaded, the application will display the uploaded image and make a prediction of the Urdu text in the image. The result will be displayed under "Prediction".

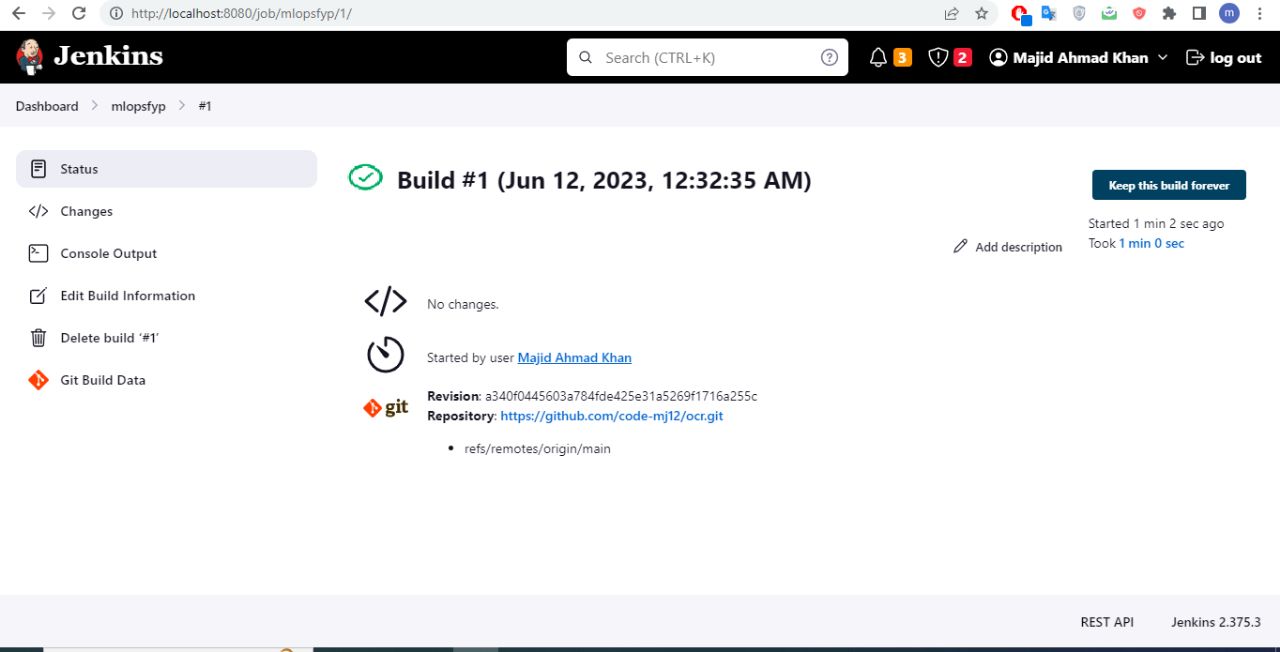


**7.Jenkins Pipeline**





In order to build CI/CD pipeline we chose Jenkins, so we named our workflow mlopsfyp and made freestyle project we attached the github link of our repository and included those 2 docker command so that our application can be containerized automatically. Below you can see our successful build



**8.Conclusion**

This project overall was good source of learning. It helped us learn how to share and version large datafiles, without hassle and ease. We also learned how to create training environments that can run and train on any available environment. This project helped us create a training pipeline of our project. One of the major advantages was that, we are training a Urdu OCR. We lacked the dataset to train ocr. We had established a baseline model but were training it again and again for better predictions and accuracy. Also there is not enough urdu ocr data to go around so, whenever we found or created a new dataset we needed to maintain it on hard drive and train the model step by step from scratch. That consumed a lot of time, but by creating this automated ml pipeline, we can easily version our multiple datsets, create logs against each dataset, train and evaluate different models on different datasets without any hassle. Overall this was an extremely productive project and proved exponentially good for our FYP.

**Learning Outcomes:**

1. Data versioning using dvc
2. Large data sharing using DVC
3. Training multiple YOLO models using Airflow and scheduling them
4. Integrating Wandb with YOLO models in Airflow
5. Using MLflow for model logging and sharing artifacts.
6. Sharing and Versioning model weights using DVC
7. Creating Inference app
8. Creating CI/CD pipeline for multiple machine learning models
9. Containerizing application and making it useable without dependency issues
10. Automate the process of training and testing models on different machine with out dependency issues