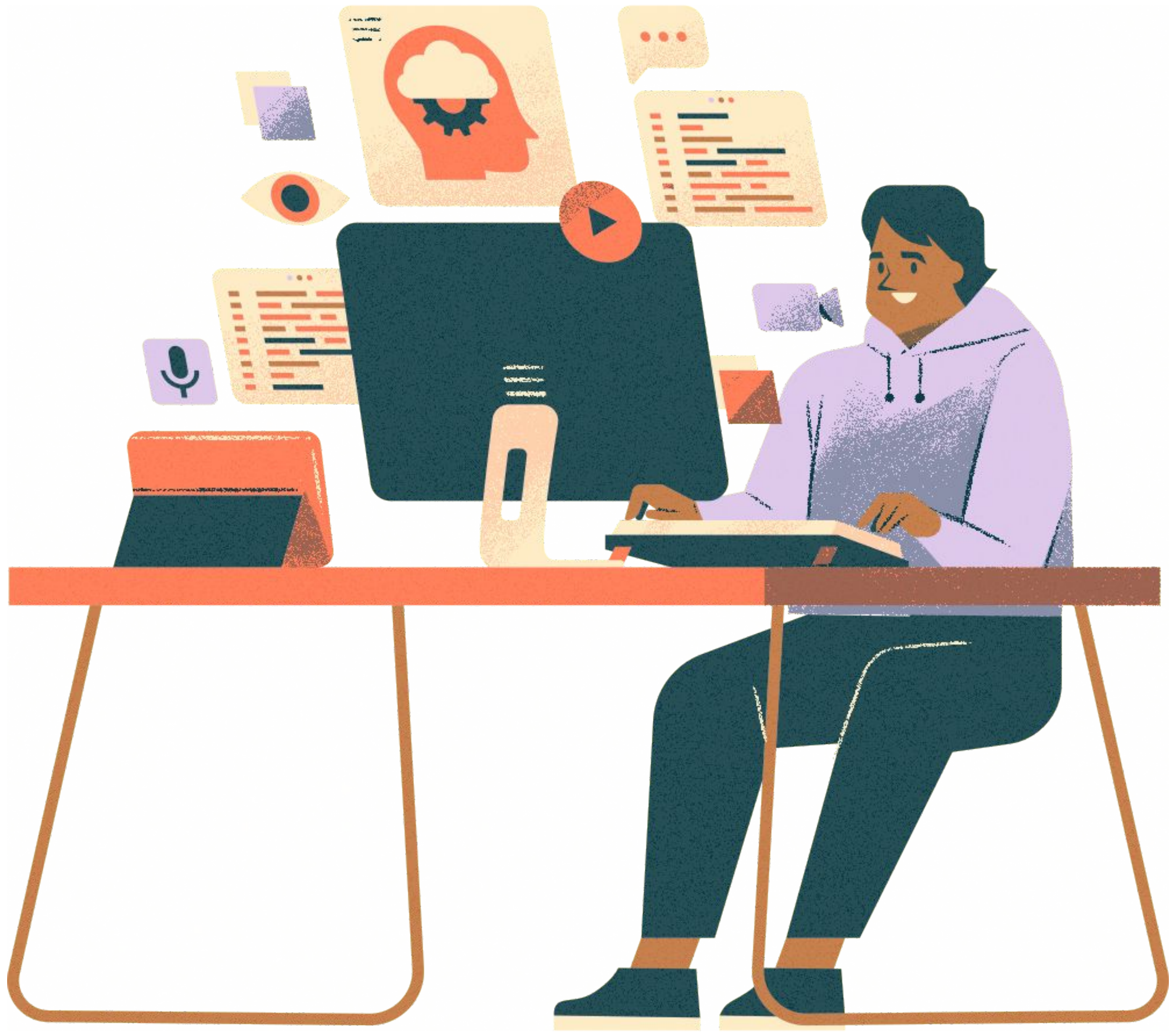
Detection of Defects in Computer Chips



A Project Presentation
CMPE 258- Deep Learning

TEAM MEMBERS

Neel Desai

017436575

Sai Praneeth Konuri

017425876

Mansi Vekariya

016996291

Samarth Sharma

017103437

Overview

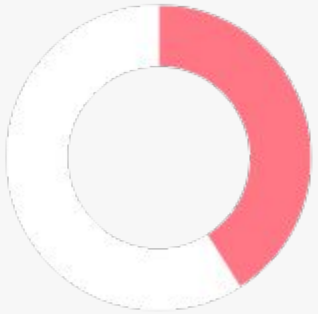
- Build a CNN model capable to detecting if a computer chips has any defects or no

- Building a CNN which has at least 90% accuracy.

- Include MLOps in the training and Inference , we will be focusing: Experiment tracking, model registry, model versioning and model deployment and POC for retraining pipeline

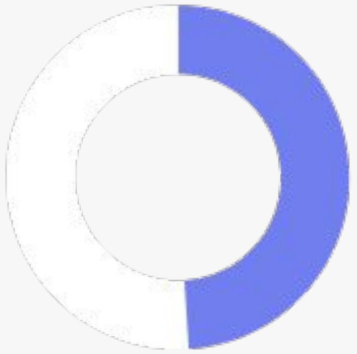
- Link to dataset: https://www.kaggle.com/datasets/qingyi/wm811k-wafer-map

Get to know the data



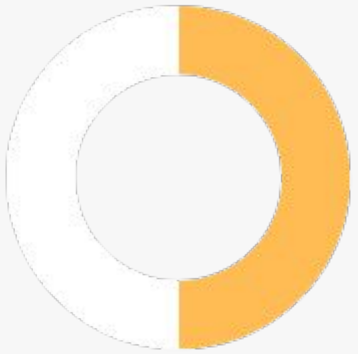
The data which we are using has the following classes for failures:

Center, Donut, Edge-Loc, Edge-Ring, Loc, Random, Scratch, Near-full, none



We have a total of 811457 wafer maps, spread across 632 image size

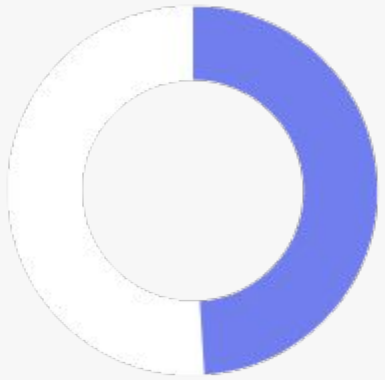
To make this consistent for the purpose of the project we will be looking at images with shape 26x26, we have 30078 such images



On exploring the distribution of labels across our images we found that we had major class imbalance, where the no error class account for more than 95% of the data and rest 8 classes of different defects accounted for remaining 5

Get to know the data (Continued)

To address the imbalance we pivoted from multi-class classification of different defects, to a binary classification problem i.e. identify if an image represents a chip with defect or no defect



Additional we performed downsampling to make sure that both classes are evenly distributed. We ended with having 900 images per each class

Note: we tried Image augmentation but the results were not that great, and we planned on running AutoTraining loop on Low cost cloud resources, so we decided to keep the train computation as light as possible.

Training (Overview)

We used ChatGPT to provide us a good starting point for a CNN architecture from where we can could built forward

First we perform training on colab to find architecture that work well

Then the best performing architectures will be used in Autotraining loop on Azure ML

Additionally we tracked all the experiments using MLFlow (even on colab) so that later we can easily register it in Azure Model registry

Training: Starting Point

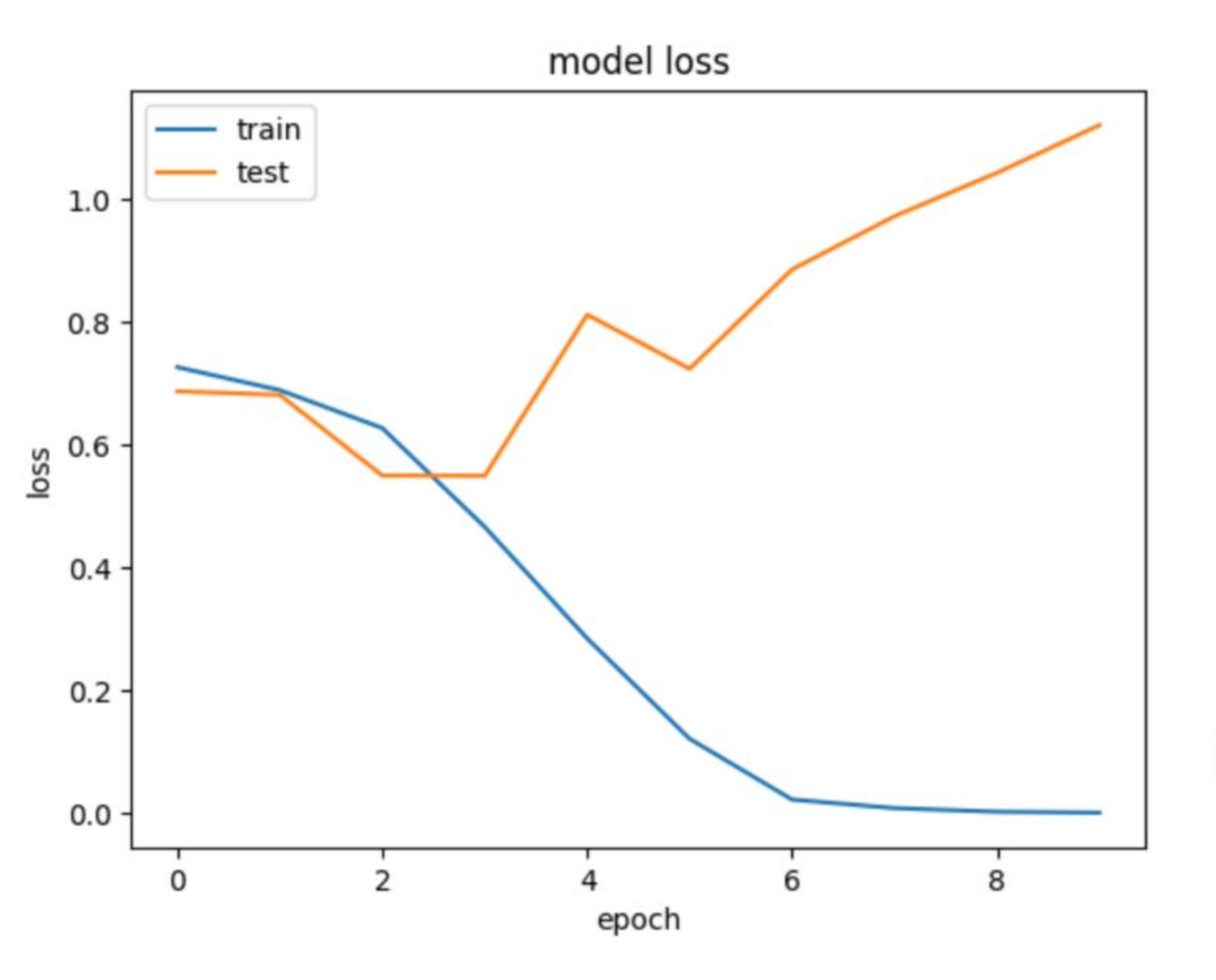
```
model = tf.keras.models.Sequential([
tf.keras.layers.Input(shape=(26, 26, 1)),
tf.keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Starting Architecture

The network is overfitting on train set

It is not able to capture all the variability of the data

Loss Curve



Val_accuracy = 75%

Lets us try a couple of different things to fix it

Further experimentation

For Model 2:

We have added pooling layers and a dropout layer to our model 1.

For Model 3:

We have changed the optimizer from Adam to Adadelta.

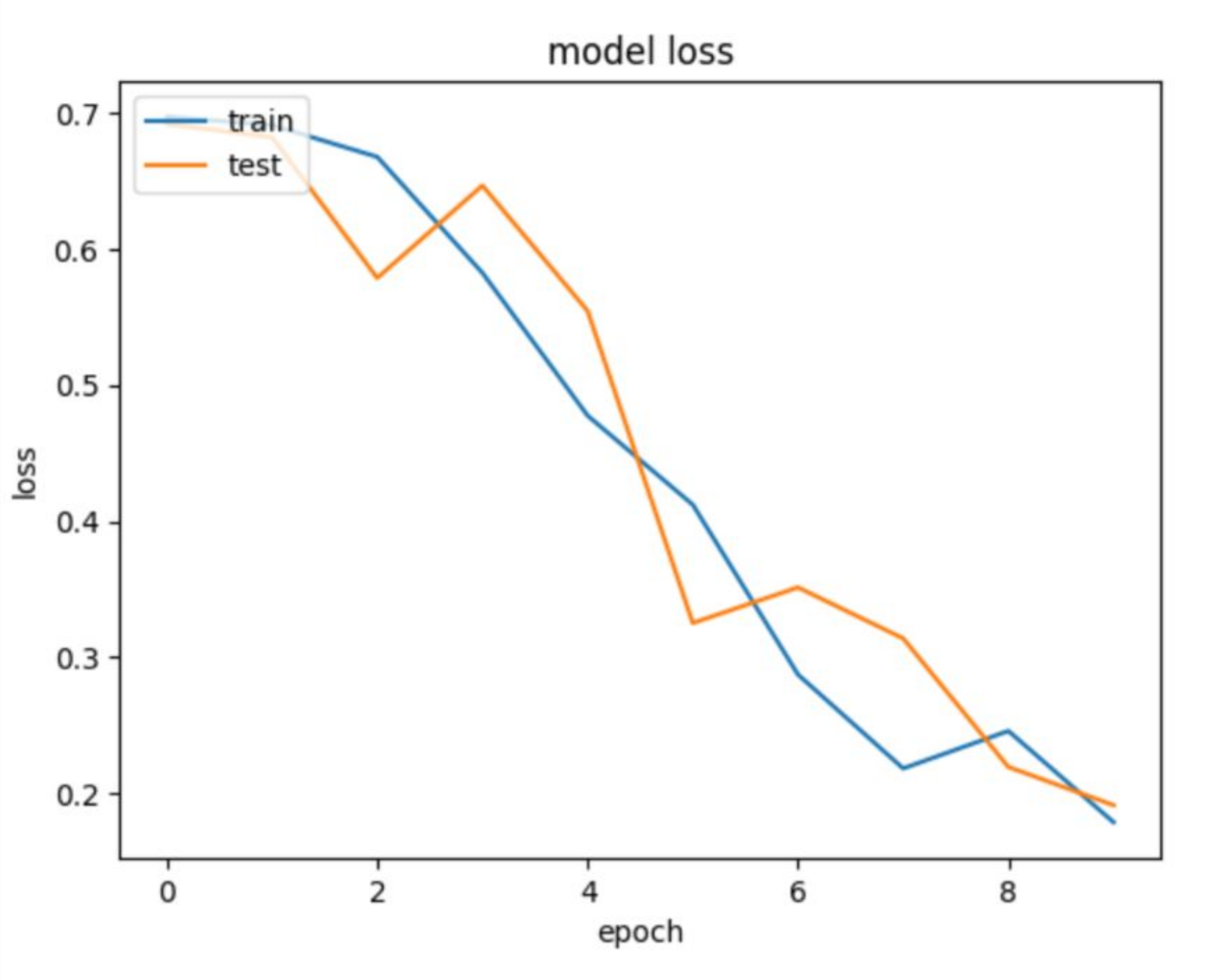
Training: Ending Point, Best Model

```
[ ]  model = tf.keras.models.Sequential([
         tf.keras.layers.Input(shape=(26, 26, 1)),
         tf.keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
         tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
         tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
         tf.keras.layers.Flatten(),
         tf.keras.layers.Dense(128, activation='relu'),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(64, activation='relu'),
         tf.keras.layers.Dense(1, activation='sigmoid')
         ])

     model.compile(optimizer=tf.keras.optimizers.AdamW(), loss='binary_crossentropy', metrics=['accuracy'])
```

There is no more overfitting

Loss is continuing to go down, longer training could result in even better result

Loss Curve



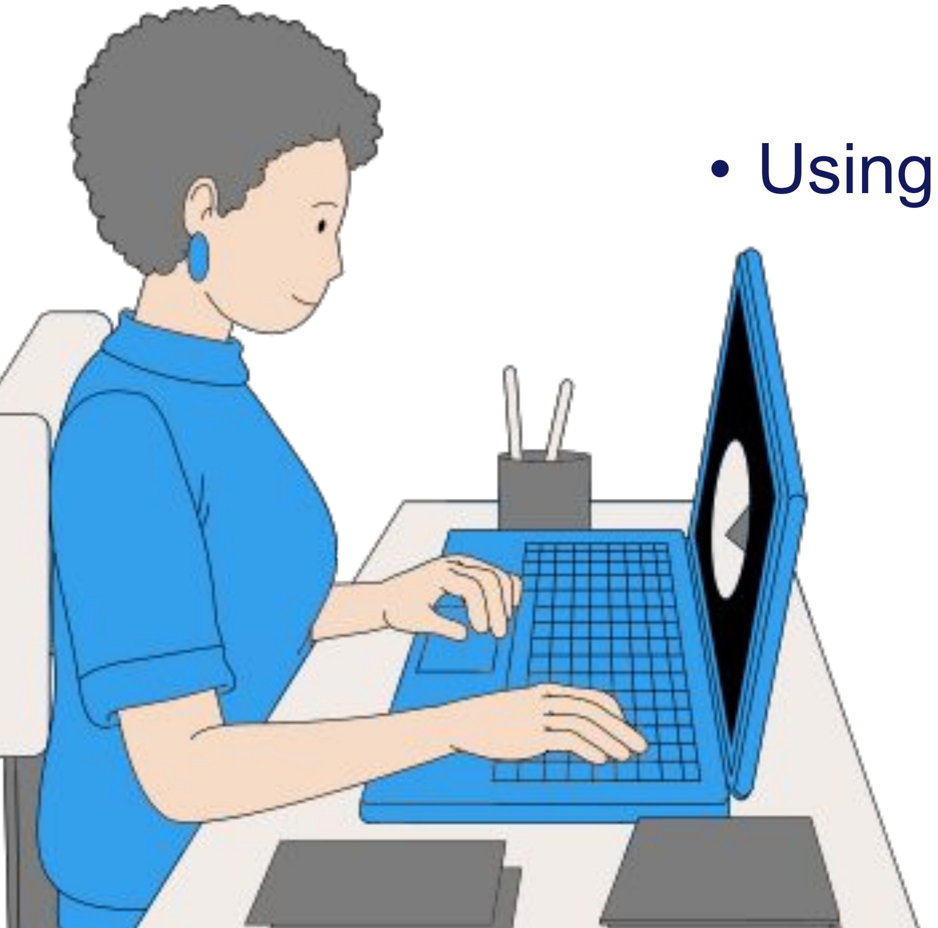
Val_accuracy = 93%

This model is significantly the better than our baseline. And this is where we stop the dev

How did we get there

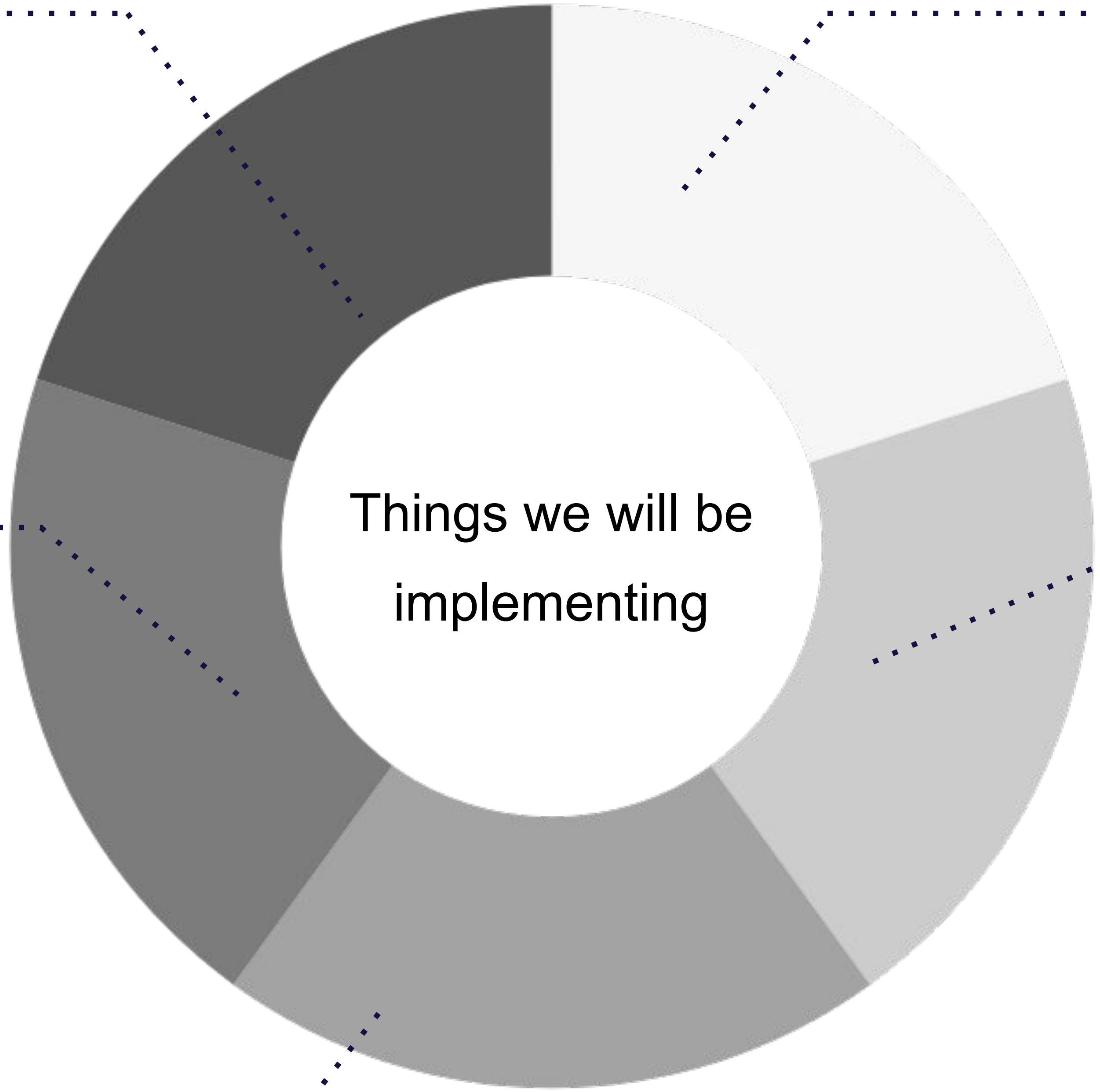
What changes did we make to get from 75% to 93% ?

- Using different regularization techniques such as Batch Normalization and Dropout layer
- Reducing the model complexity - specially after flattening the input
- Using newer more adaptive optimizer AdamW



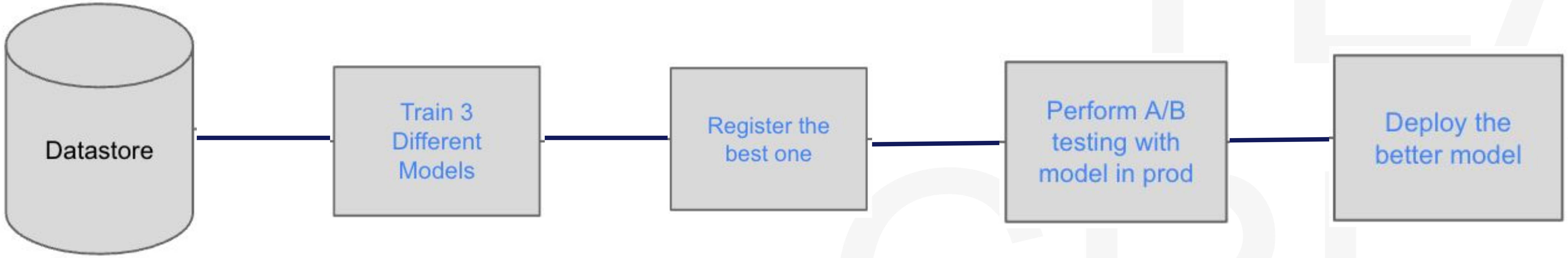
ML Ops on Azure

- Experiment tracking with MLFlow
- Automated Deployment
- Model versioning with model registry
- Scheduling all runs through Azure Designer



- Automated model A/B testing

Training/Retraining Pipeline



- This pipeline focuses in auto training, testing and deployment, cause of which we assume that the data coming in the data store in clean data and doesn't require much data preprocessing.
- This pipeline can be scheduled to run on regular intervals , or run when new data arrives.
- It can configure directly as a pipeline on Azure ML.

Experiment Tracking

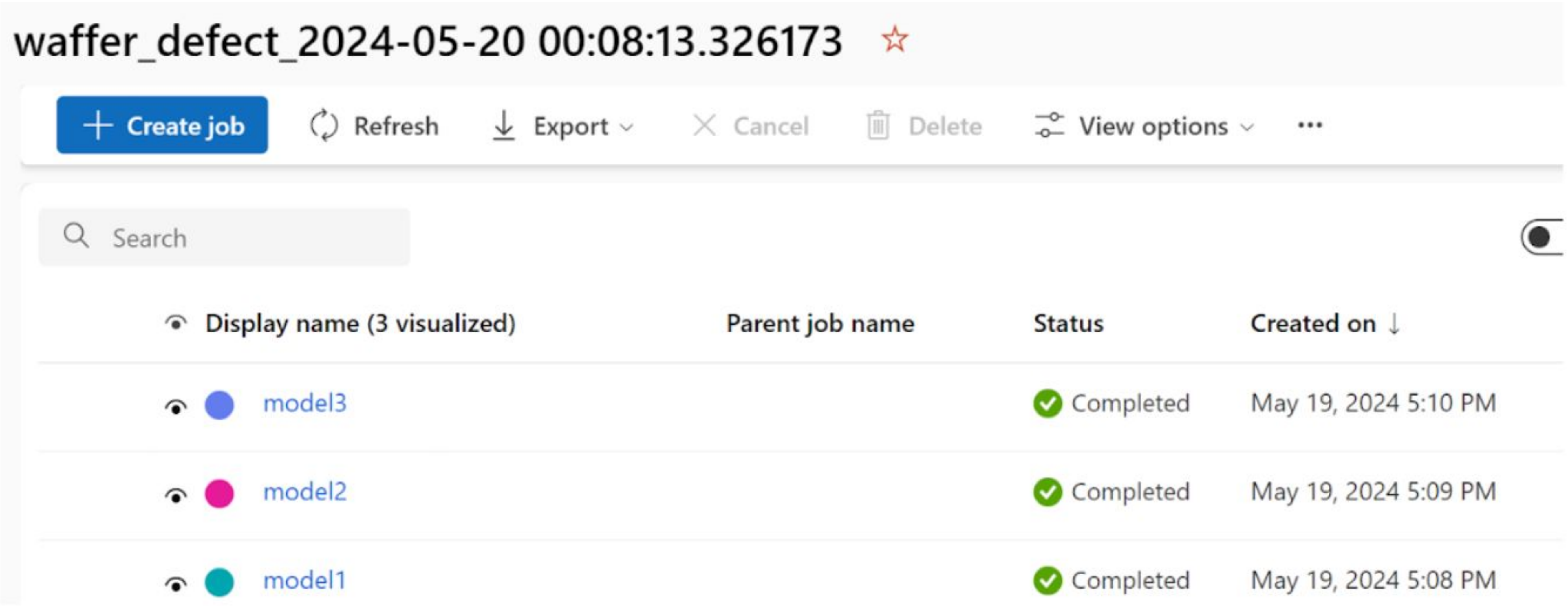
Search

Columns

Experiment	☆	Latest job	Last submitted ↓	Created
automated_training_loop		258Project	May 19, 2024 5:44 PM	May 19, 2024 5
waffer_defect_2024-05-20 00:08:13.326173		model3	May 19, 2024 5:10 PM	May 19, 2024 5
Automated_Pipeline		258Project	May 19, 2024 3:12 PM	May 19, 2024 3
iris_experiment		ivory_ring_lwrwybn0	May 19, 2024 2:34 PM	May 16, 2024 1
waffer_defect_2024-05-19 08:37:14.301825		model3	May 19, 2024 1:39 AM	May 19, 2024 1
iris_experiment_2024-05-18 00:34:41.819518		model3	May 17, 2024 5:41 PM	May 17, 2024 5
iris_experiment_2024-05-17 18:56:50.860243		model2	May 17, 2024 11:57 AM	May 17, 2024 1

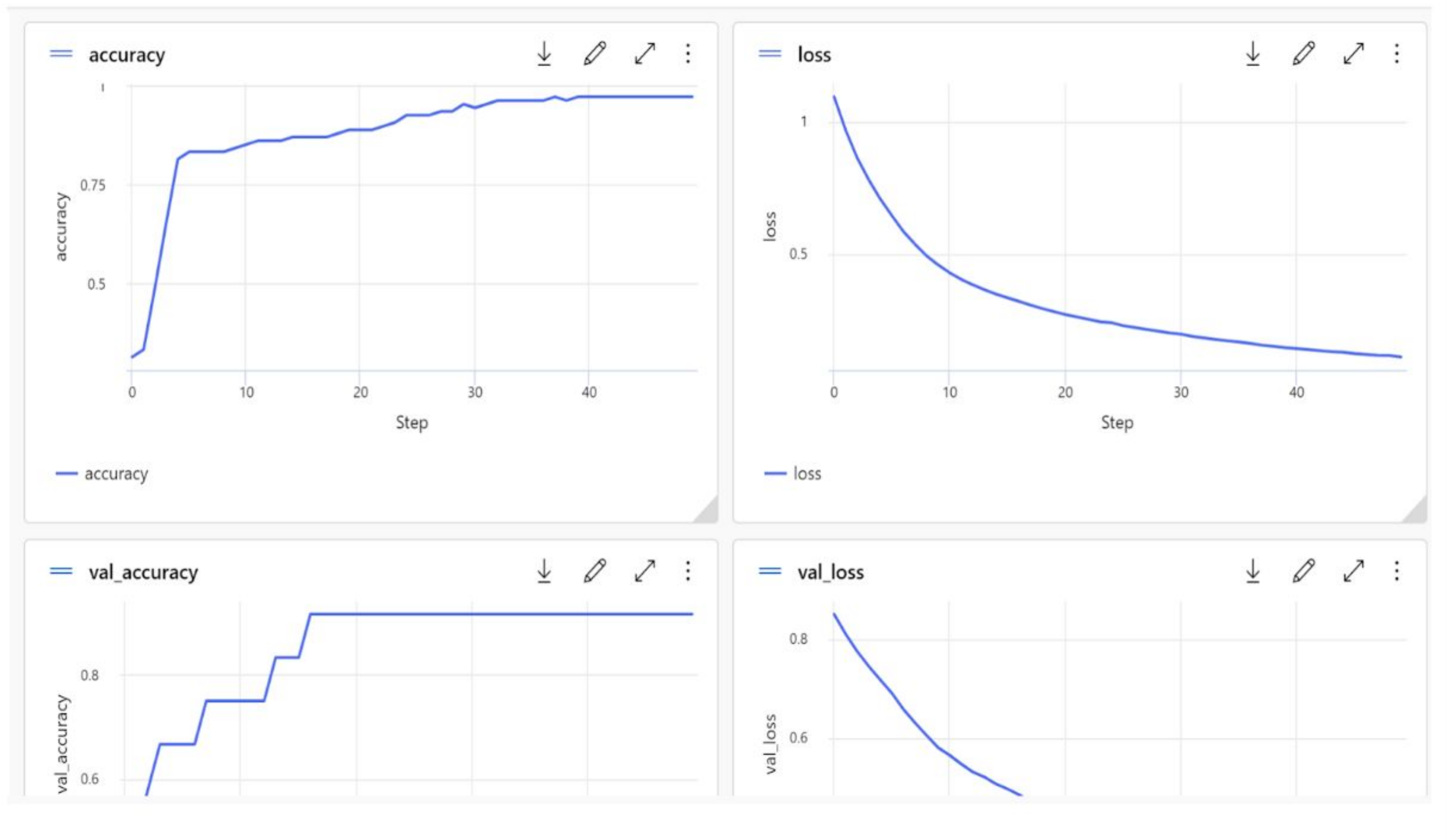
An experiment is nothing but a collection of different models you built when trying to find the best one

MLFlow



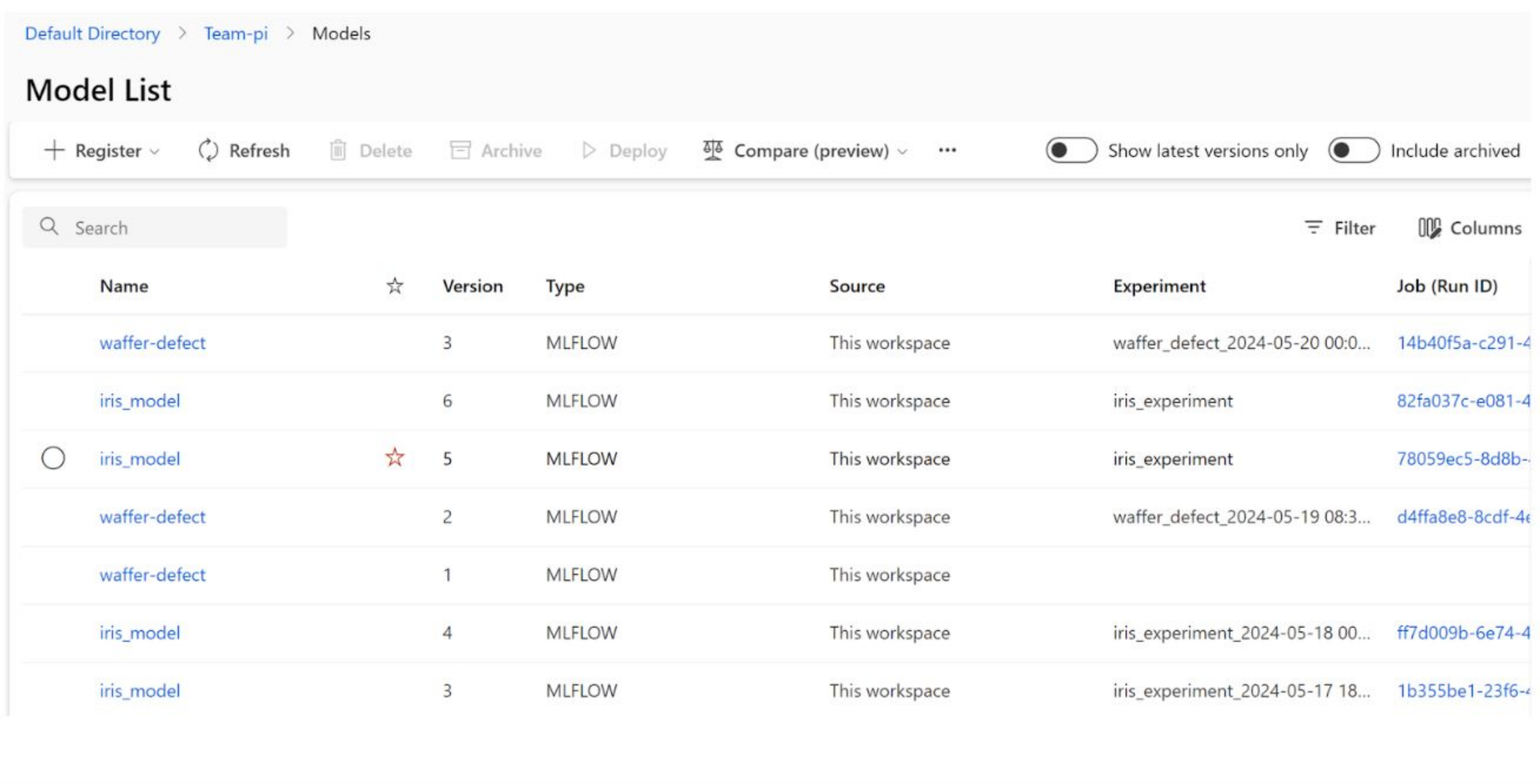
With each experiment , we have the list of all the models trained for it, we use MLFlow for logging models performance and its artifacts

Artifacts and metrics



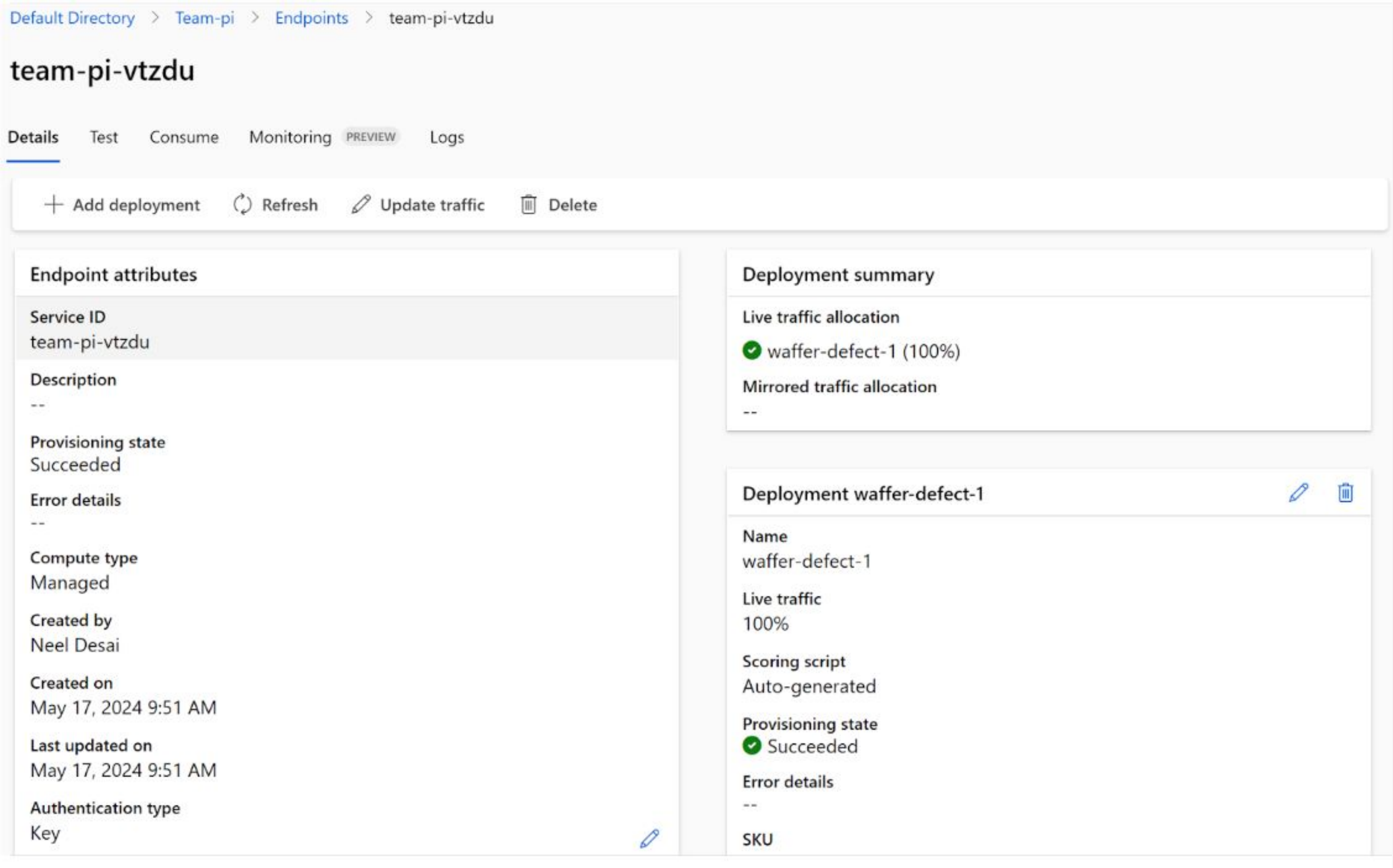
For each logged model we have all the logged artifacts and metrics, along with the hyperparameter used for training it

Model Registry



It's a means through which we can apply a version control on our models, it could be as simple as same model trained on data at 2 different times

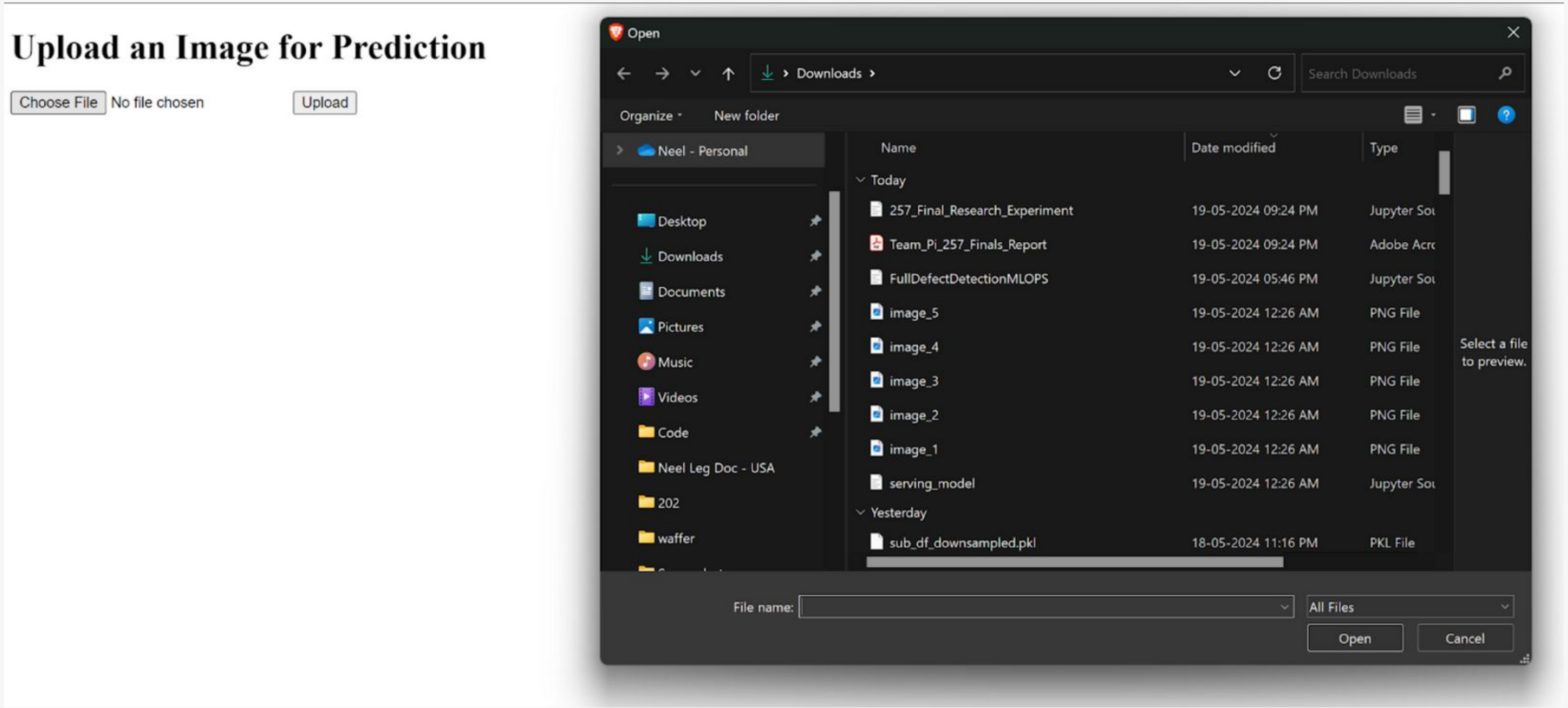
Endpoint Deployment



Allows us to expose our model as an API to the rest of the world.

Inference Pipeline

WebPage Mockup



WebPage Interface

WebPage Interface

Thank You!!