Continuous training with multiple SDKs, Kubeflow, and AI Platform Pipelines

Agenda

Containerized training applications

Containerizing PyTorch, Scikit, and XGBoost applications

Kubeflow and AI Platform Pipelines

Continuous training

Why use containerized training applications?

Google Cloud

Why use containerized training applications?

- You don’t have to worry about dependencies.

Why use containerized training applications?

- You don’t have to worry about dependencies.
- Use them as ops in a Kubeflow pipeline (or other orchestration tools).

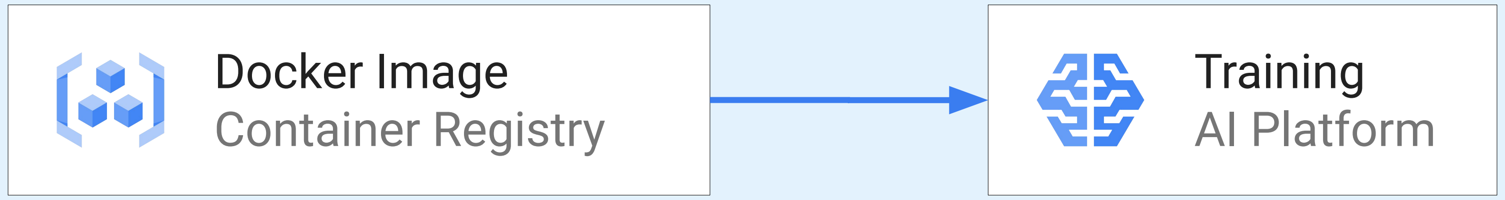
Why use containerized training applications?

- You don’t have to worry about dependencies.
- Use them as ops in a Kubeflow pipeline (or other orchestration tools).
- They are portable across runtime environments.

AI Platform Training with custom containers

```
!gcloud ai-platform jobs submit
training $JOB_ID \
   --region=$REGION \
   --master-image-uri=$IMAGE_URI \
      --training_args
```

Train Models from Images in GCR



AI Platform Training job as an op in a Kubeflow pipeline

```
import kfp.dsl as dsl

mlengine_train_op =
component_store.load_component(
        'ml_engine/train')

@dsl.pipeline(
    name='My Pipeline'
)
def pipeline(pipeline_args):

    train_model = mlengine_train_op(
        project_id=project_id,
        region=region,
        master_image_uri=TRAINER_IMAGE,
        args=training_args)
```

- Load the pre-built AI Platform Training component.
- Use op with training image and args.

Step 1: Create a model training script

```
%%writefile ./tensorflow_trainer_image/train.py
import tensorflow as tf

def train_evaluate(training_args):
    # Data ingestion and model building code here

    history = model.fit(
        trainds,
        validation_data=evalds,
        epochs=num_evals,
        steps_per_epoch=steps_per_epoch
     )

    tf.saved_model.save(
        obj=model, export_dir=EXPORT_PATH)

if __name__ == '__main__':
    fire.Fire(train_evaluate)
```

Step 2: Create a Dockerfile

```
%%writefile ./trainer_image/Dockerfile

FROM
gcr.io/deeplearning-platform-release/b
ase-cpu
RUN pip install -U fire
tensorflow==2.1.1
WORKDIR /app
COPY train.py .

ENTRYPOINT ["python", "train.py"]
```

/ ••• / lab / trainer_image /

Name
Dockerfile
train.py

Step 3: Build the image and push to Container Registry

```
IMAGE_NAME='trainer_image'
IMAGE_TAG='latest'
IMAGE_URI=f'gcr.io/{PROJECT_ID}/{IMAGE_NAME}:{IMAGE_TAG}'

!gcloud builds submit --tag $IMAGE_URI $IMAGE_NAME
```

Agenda

Containerized training applications

Containerizing PyTorch, Scikit, and XGBoost applications

Kubeflow and AI Platform Pipelines

Continuous training

What if you want to develop your models with a different framework?

The process is exactly the same!

1. Develop a training script in the framework of your choice.
2. Package the training script into a Docker image.
3. Build and push the image to Container Registry.

PyTorch example

```
%%writefile ./pytorch_trainer_image/train.py
import torch

def train_evaluate(training_args):
    # Data ingestion and model building code here
       model.train()
    for e in range(1, num_epochs+1):
        for X_batch, y_batch in train_loader:
            optimizer.zero_grad()
            y_pred = model(X_batch)
            loss = criterion(y_pred,y_batch)
            loss.backward()
            optimizer.step()

    torch.save(model.state_dict(), model_filename)

if __name__ == '__main__':
    fire.Fire(train_evaluate)
```

Step 1:

Develop a training script.

PyTorch example

```
%%writefile ./pytorch_trainer_image/Dockerfile

FROM
gcr.io/deeplearning-platform-release/base-cpu
RUN pip install -U fire torch==1.6.0
scikit-learn==0.23.2 pandas==1.1.1
WORKDIR /app
COPY train.py .

ENTRYPOINT ["python", "train.py"]
```

Step 2:
Create a Dockerfile.

PyTorch example

```
IMAGE_NAME='pytorch_trainer_image'
IMAGE_TAG='latest'
IMAGE_URI=f'gcr.io/{PROJECT_ID}/{IMAGE_NAME}:{
IMAGE_TAG}'

!gcloud builds submit --tag $IMAGE_URI
$IMAGE_NAME
```

Step 3:
Build and push the image.

Agenda

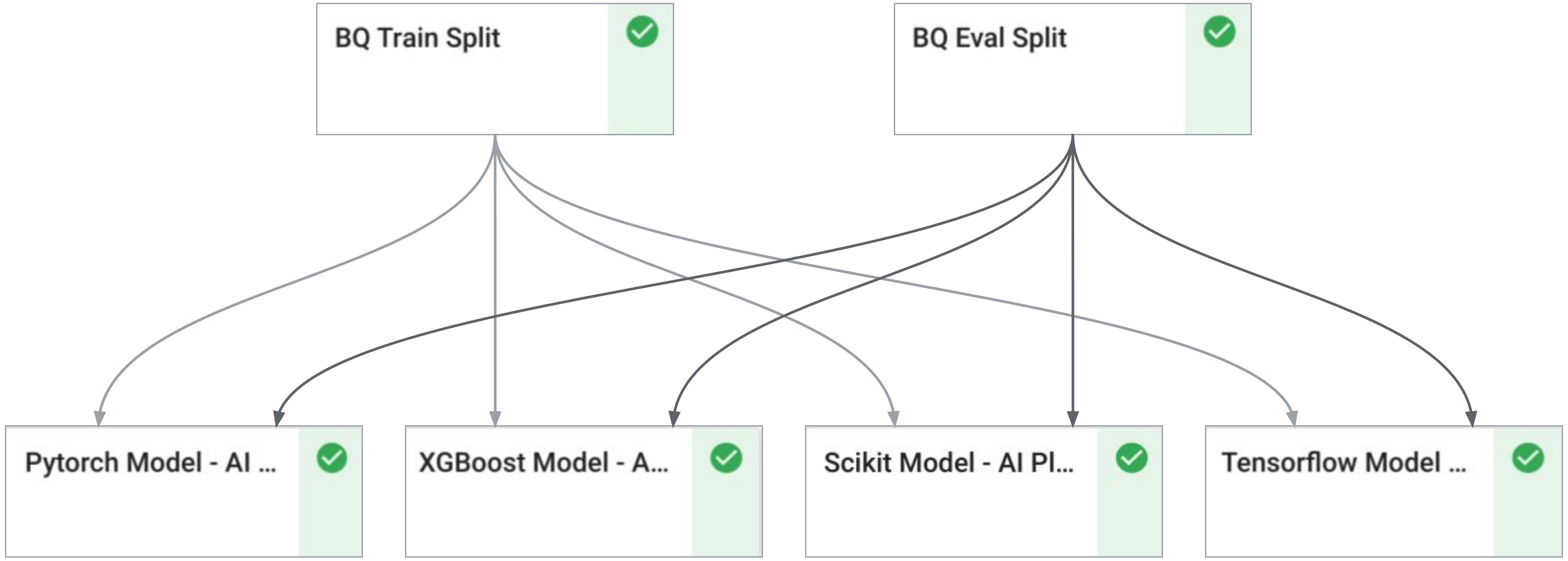
Containerized training applications

Containerizing PyTorch, Scikit, and XGBoost applications

Kubeflow and AI Platform Pipelines

Continuous training

Training multiple models in a Kubeflow pipeline



Create seperate arg lists for each container

```
   torch_args = [
        '--training_dataset_path',
create_training_split.outputs['output_gcs_path'],
        '--validation_dataset_path',
create_validation_split.outputs['output_gcs_path'],
        '--output_dir', torch_output_dir,
        '--batch_size', '32',
        '--num_epochs', '15',
    ]

    xgb_args = [
        '--training_dataset_path',
create_training_split.outputs['output_gcs_path'],
        '--validation_dataset_path',
create_validation_split.outputs['output_gcs_path'],
        '--output_dir', xgb_output_dir,
        '--max_depth', '10',
        '--n_estimators', '100'
    ]
```

Google Cloud

Use multiple mlengine_train ops in pipeline

```
   train_torch = mlengine_train_op(
        project_id=project_id,
        region=region,
        master_image_uri=TORCH_TRAINER_IMAGE,
        args=torch_args).set_display_name('Pytorch
Model - AI Platform Training')

    train_xgb = mlengine_train_op(
        project_id=project_id,
        region=region,
        master_image_uri=XGB_TRAINER_IMAGE,
        args=xgb_args).set_display_name('XGBoost
Model - AI Platform Training')
```

Agenda

Containerized training applications

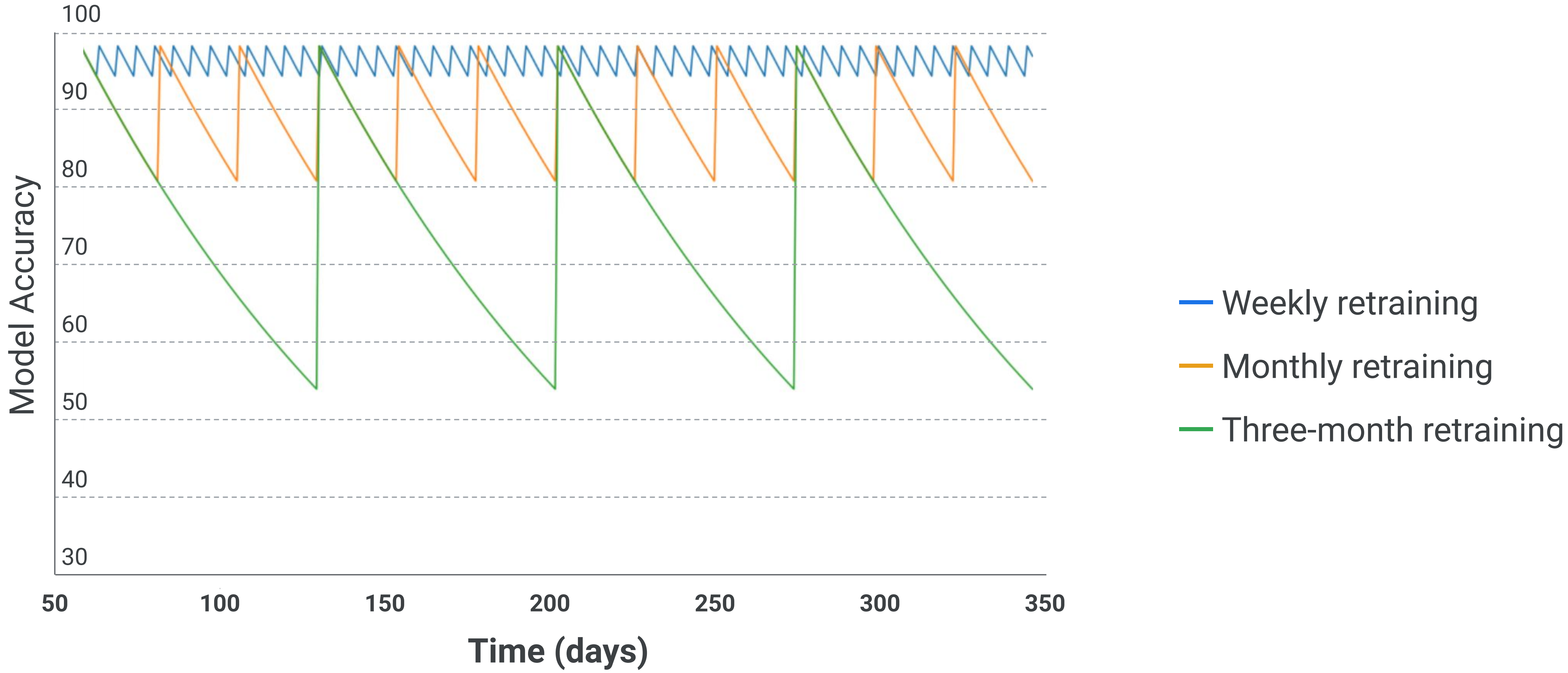
Containerizing PyTorch, Scikit, and XGBoost applications

Kubeflow and AI Platform Pipelines

Continuous training

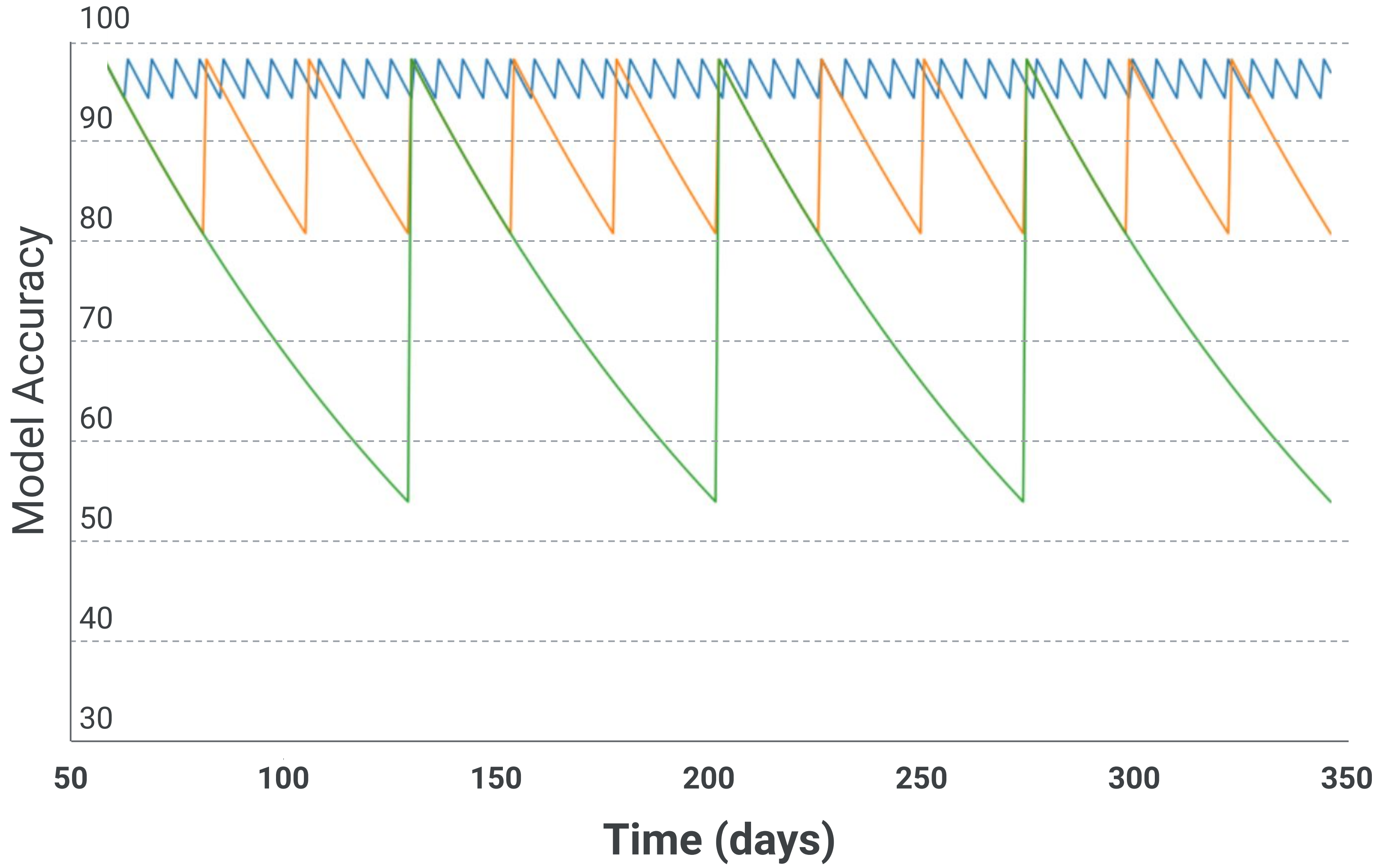
Considerations for continuous training

Model performance with different retraining intervals

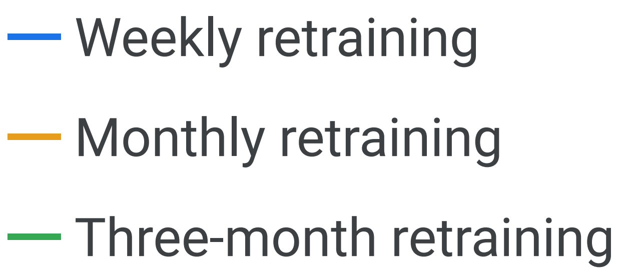


Considerations for continuous training

Model performance with different retraining intervals



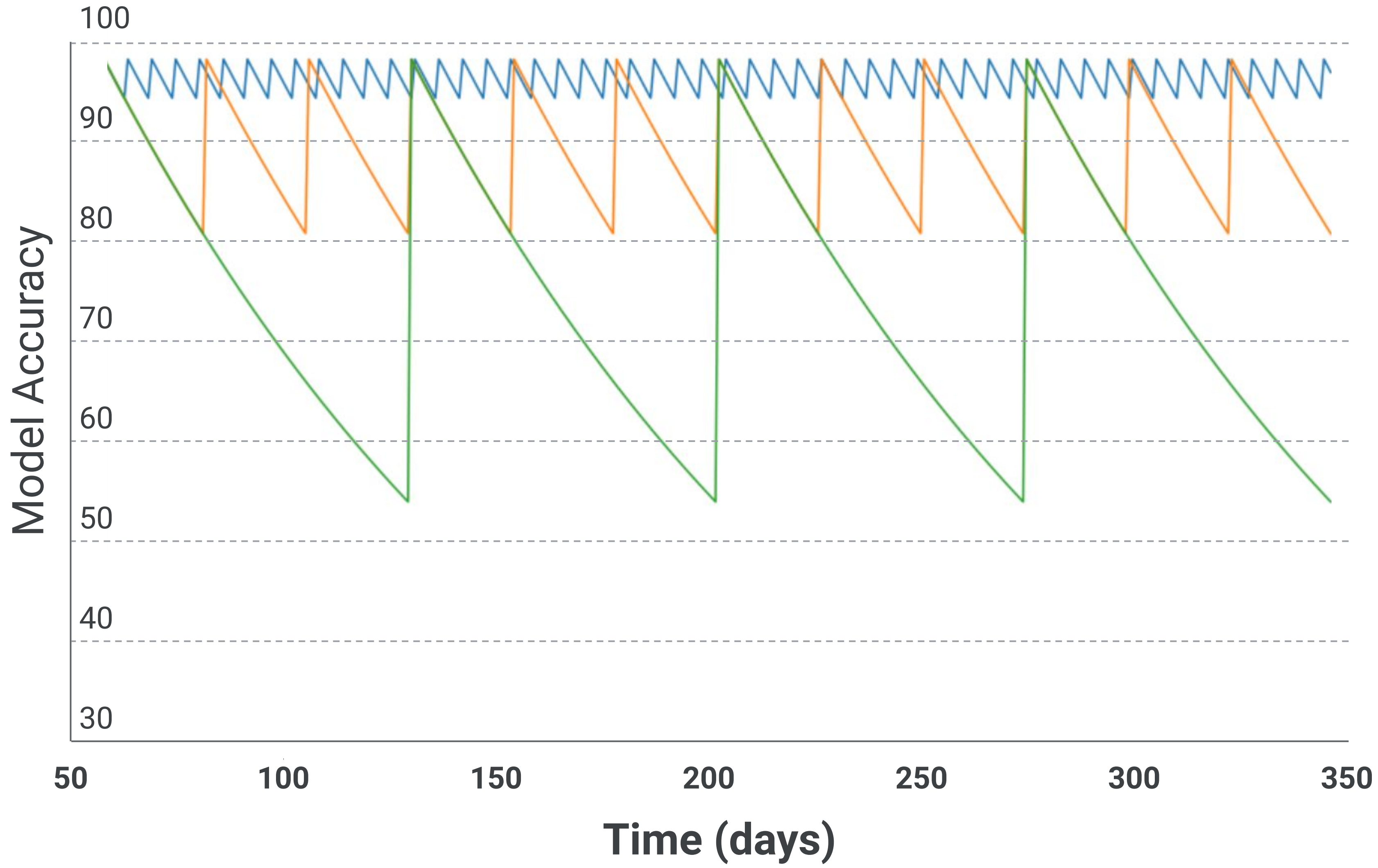
- Deterioration of model performance



Google Cloud

Considerations for continuous training

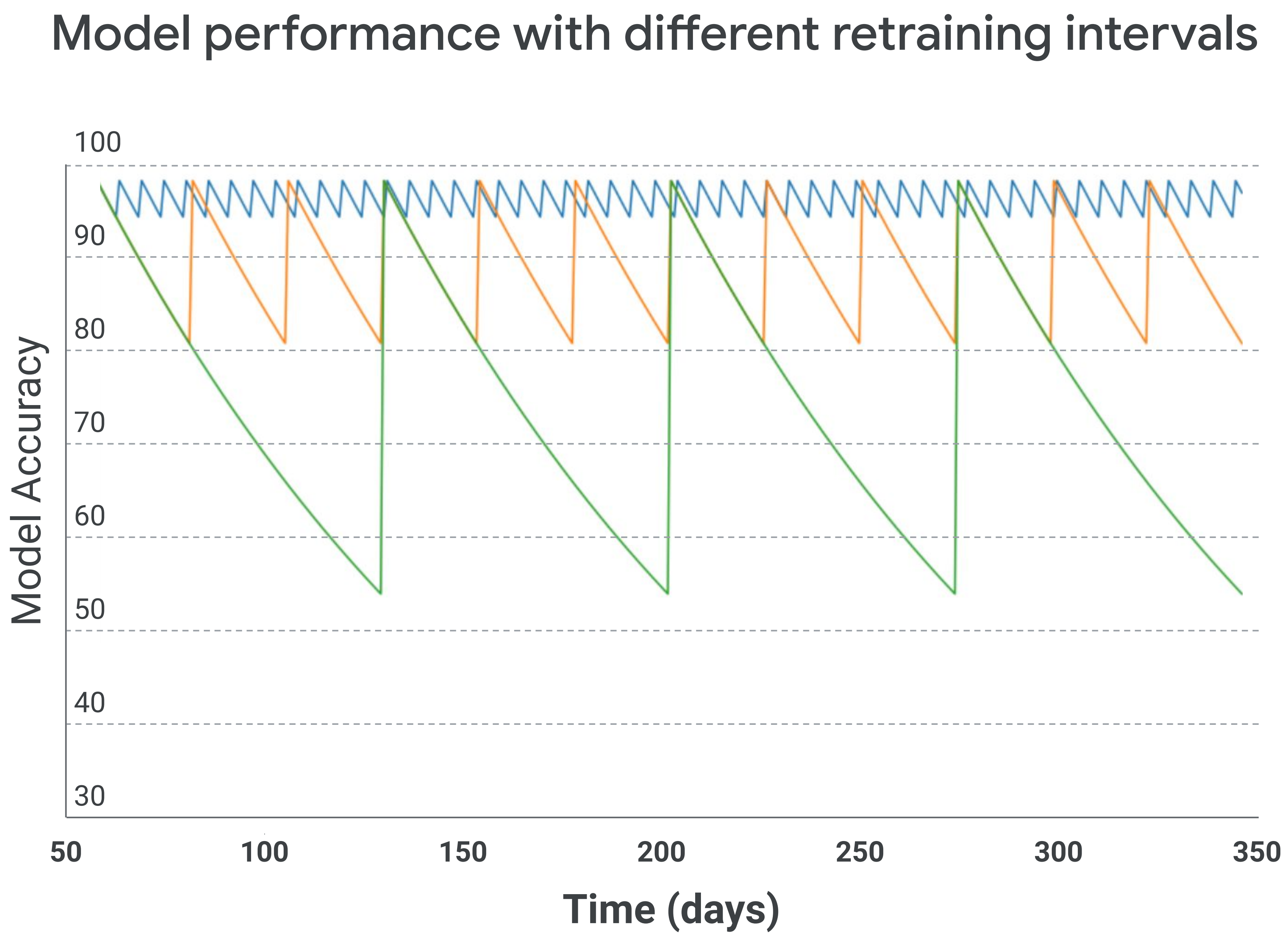
Model performance with different retraining intervals



- Deterioration of model performance
- Changes in the data distributions

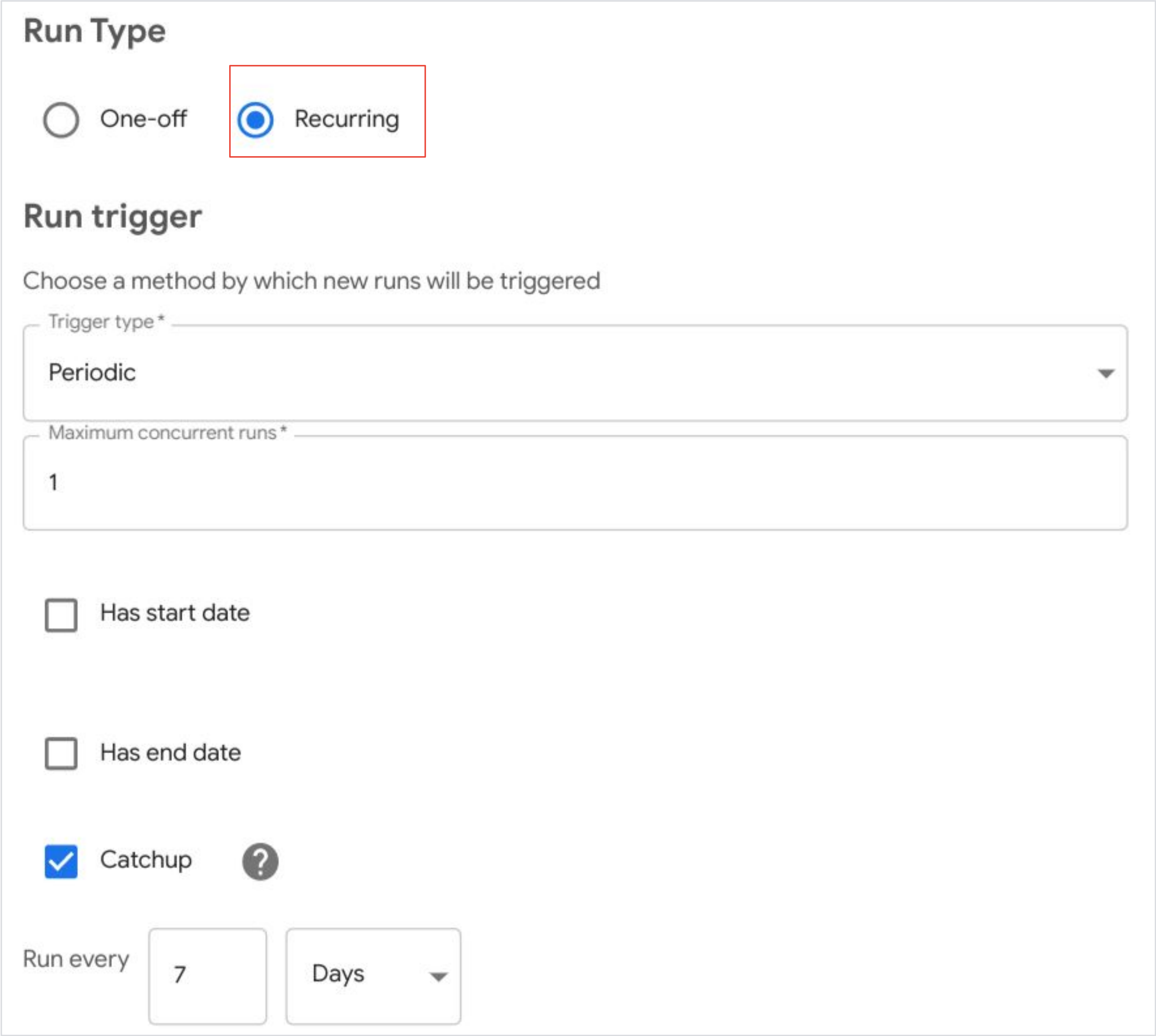
Weekly retraining
Monthly retraining
Three-month retraining

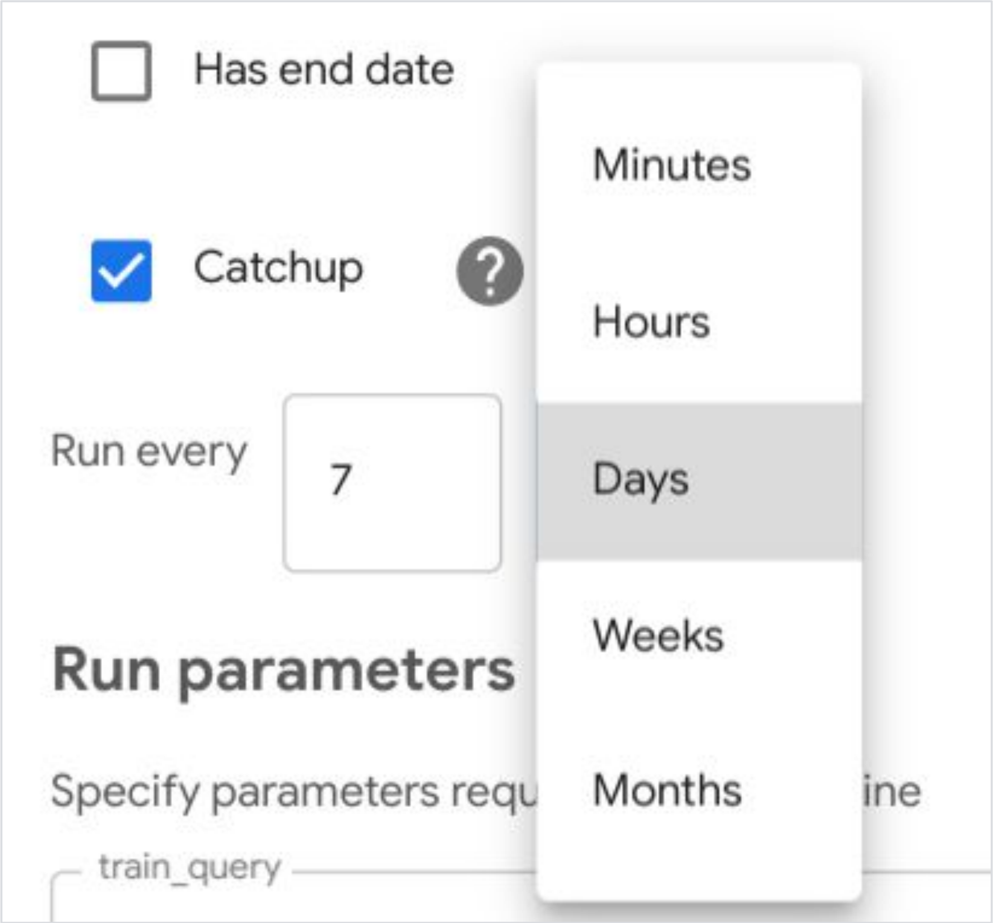
Considerations for continuous training



- Deterioration of model performance
- Changes in the data distributions
- Cost and time to retrain model

Scheduled pipeline runs with AI Platform Pipelines





Lab

Continuous Training with TensorFlow, PyTorch, XGBoost, and Scikit Learn Models with Kubeflow and AI Platform Pipelines

In this lab, you create containerized training applications for ML models in multiple frameworks. You will use these images as ops in a Kubeflow pipeline and train them in parallel. You will then set up recurring runs of your Kubeflow pipeline in the UI.

Google Cloud