Training, Tuning, and Serving on Vertex AI

Agenda

System and Concepts Overview

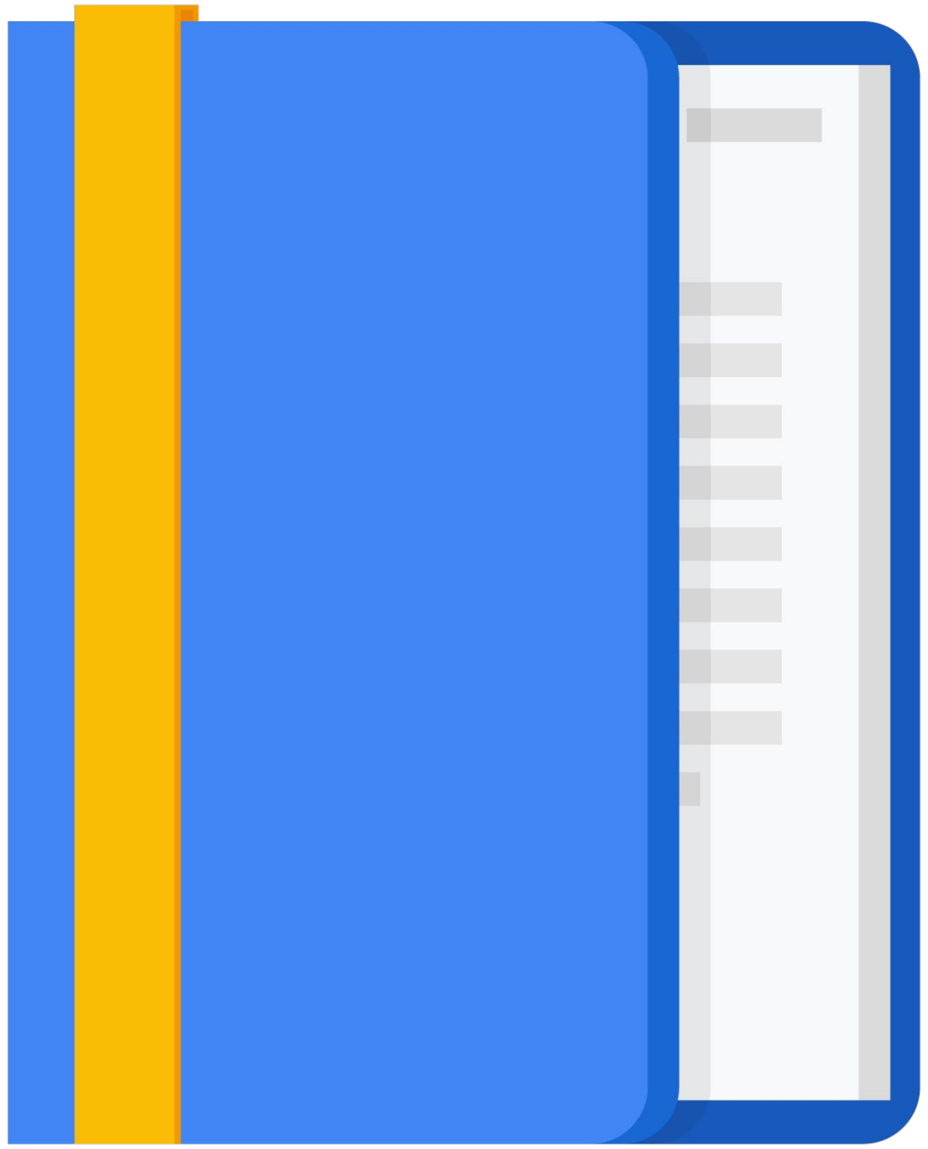
Create a Reproducible Dataset

Implement a Tunable Model

Build and Push a Training Container

Train and Tune a Model

Serve and Query a Model



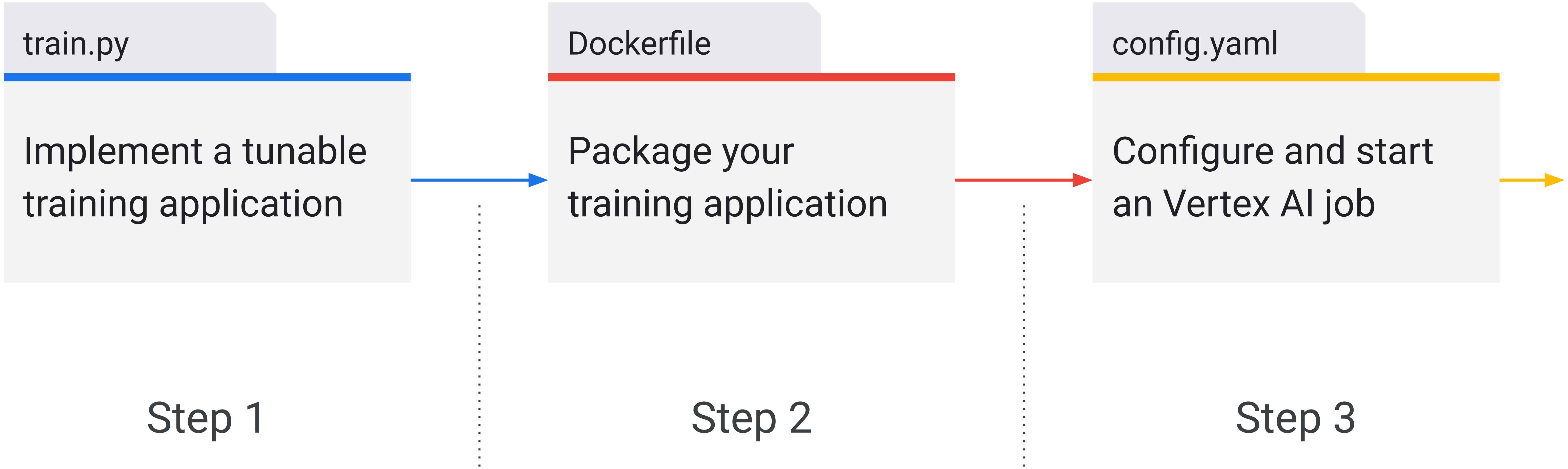
ML model building process

Create
the dataset

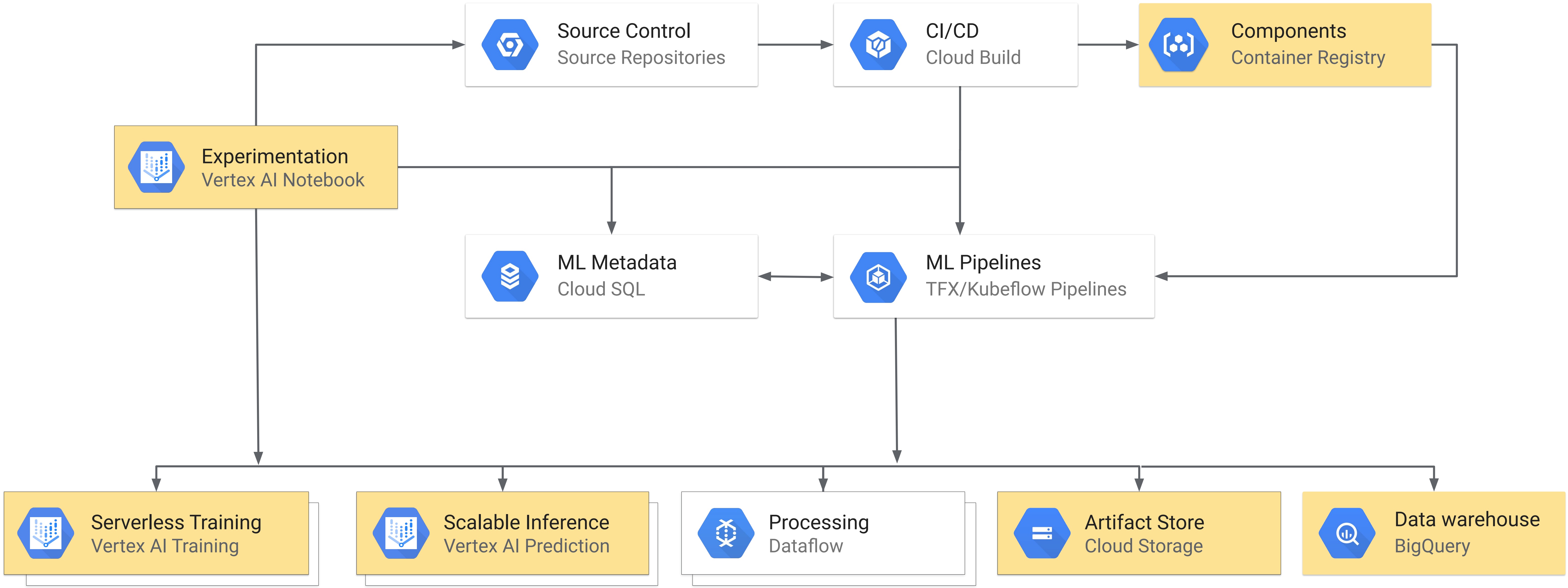
Build
the model

Operationalize
the model

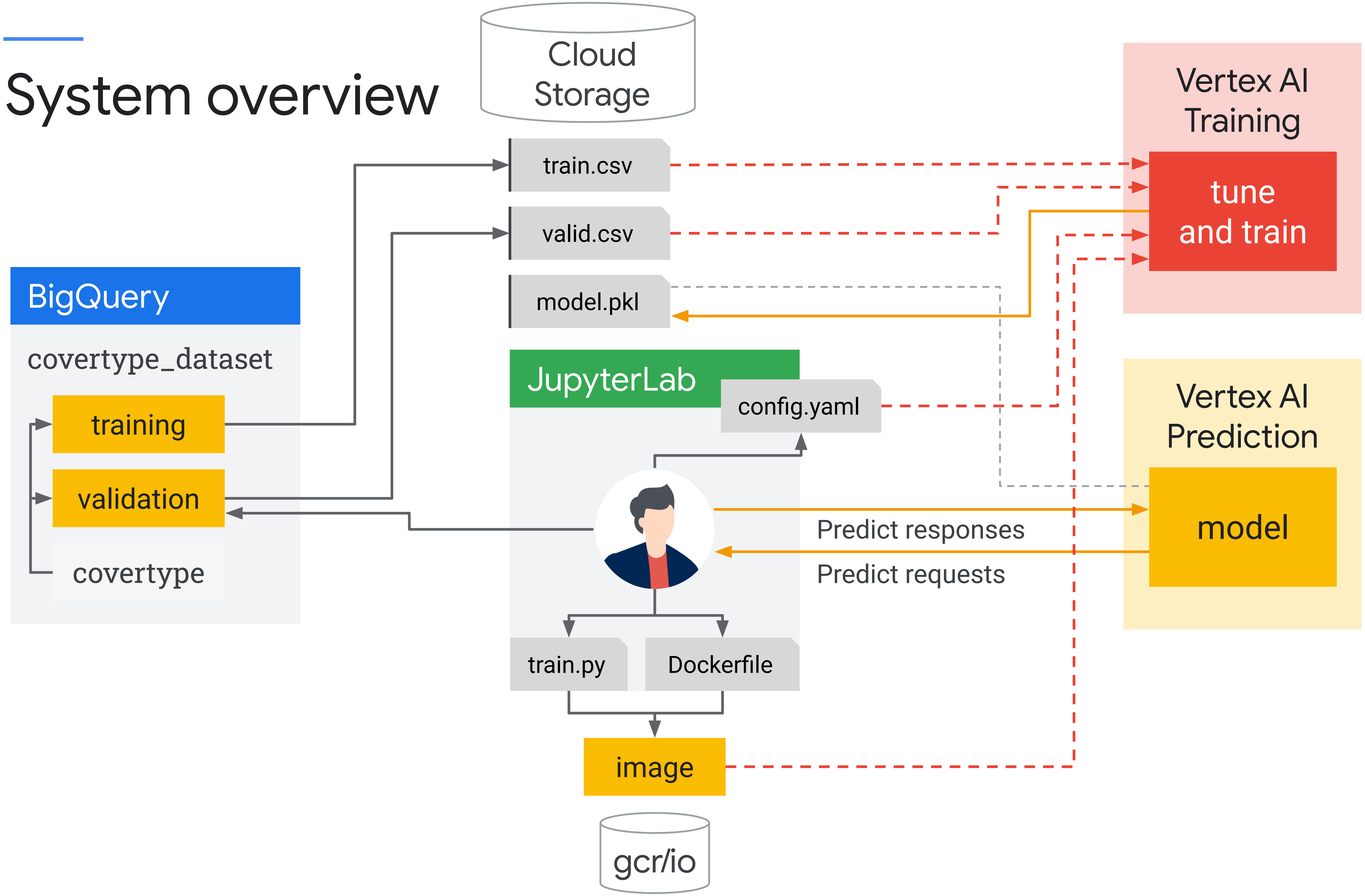
Building and operationalizing the model



MLOps building blocks on Google Cloud in this module



System overview



Agenda

System and Concepts Overview

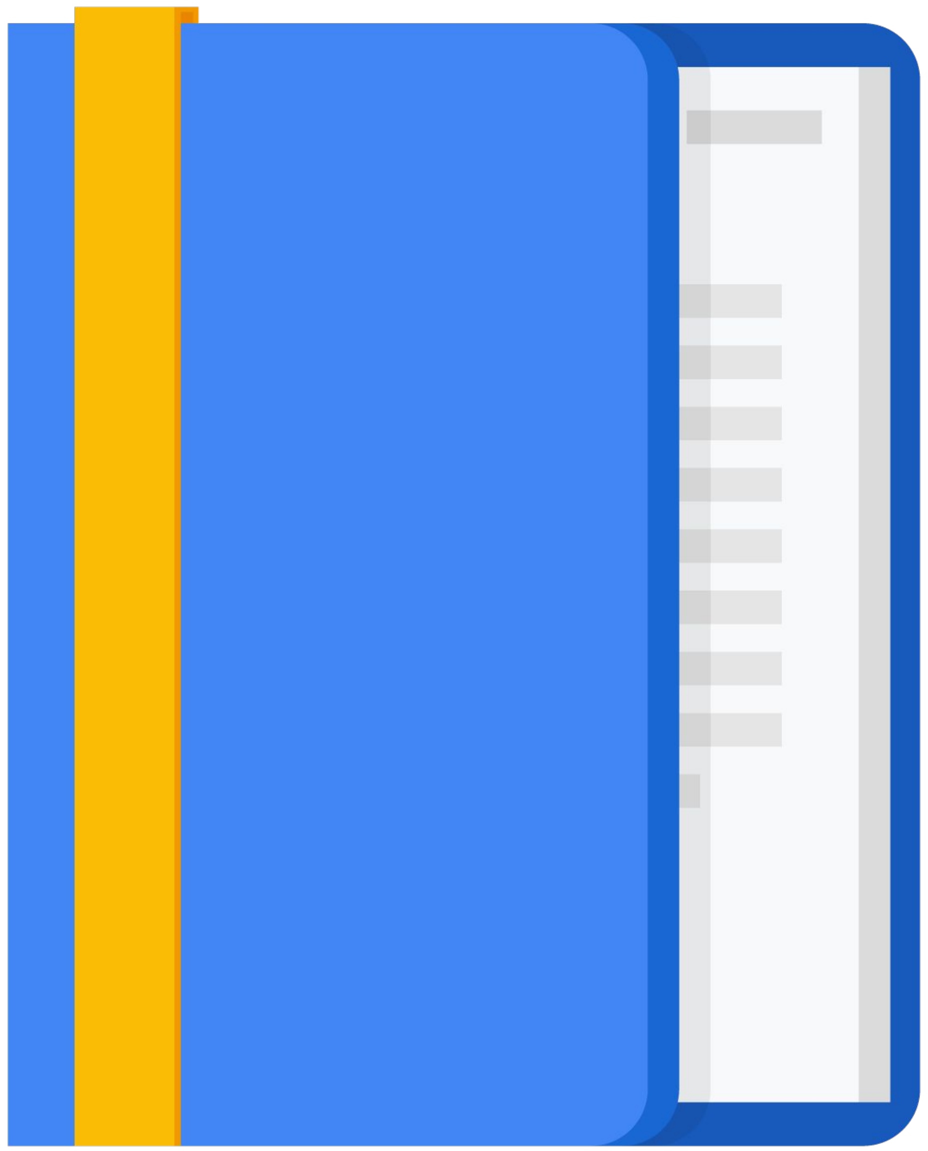
Create a Reproducible Dataset

Implement a Tunable Model

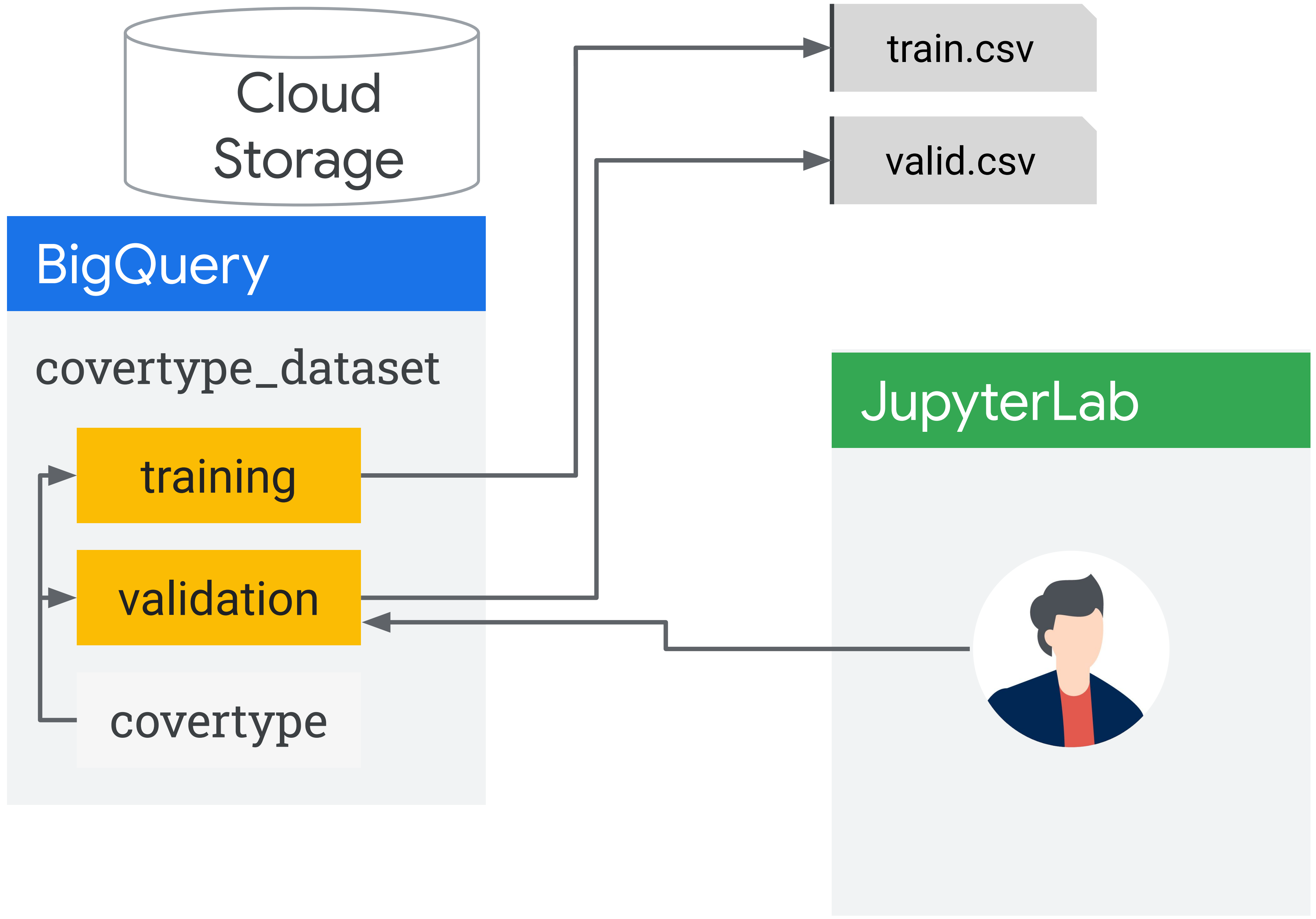
Build and Push a Training Container

Train and Tune a Model

Serve and Query a Model



System overview



Field name	Type
Elevation	INTEGER
Aspect	INTEGER
Slope	INTEGER
Horizontal_Distance_To_Hydrology	INTEGER
Vertical_Distance_To_Hydrology	INTEGER
Horizontal_Distance_To_Roadways	INTEGER
Hillshade_Noon	INTEGER
Hillshade_3pm	INTEGER
Horizontal_Distance_To_Fire_Points	INTEGER
Wilderness_Area	STRING
Soil_Type	STRING
Cover_Type	INTEGER

UCI

Machine Learning Repository

Center for Machine Learning and Intelligent System

Covertype Data Set

Download: Data Folder, Data Set Description

Abstract: Forest CoverType dataset

Data Set Characteristics:	Multivariate	**Number of Instances:**	581012	**Area:**	Life
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	54	**Date Donated**	1998-08-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	289499

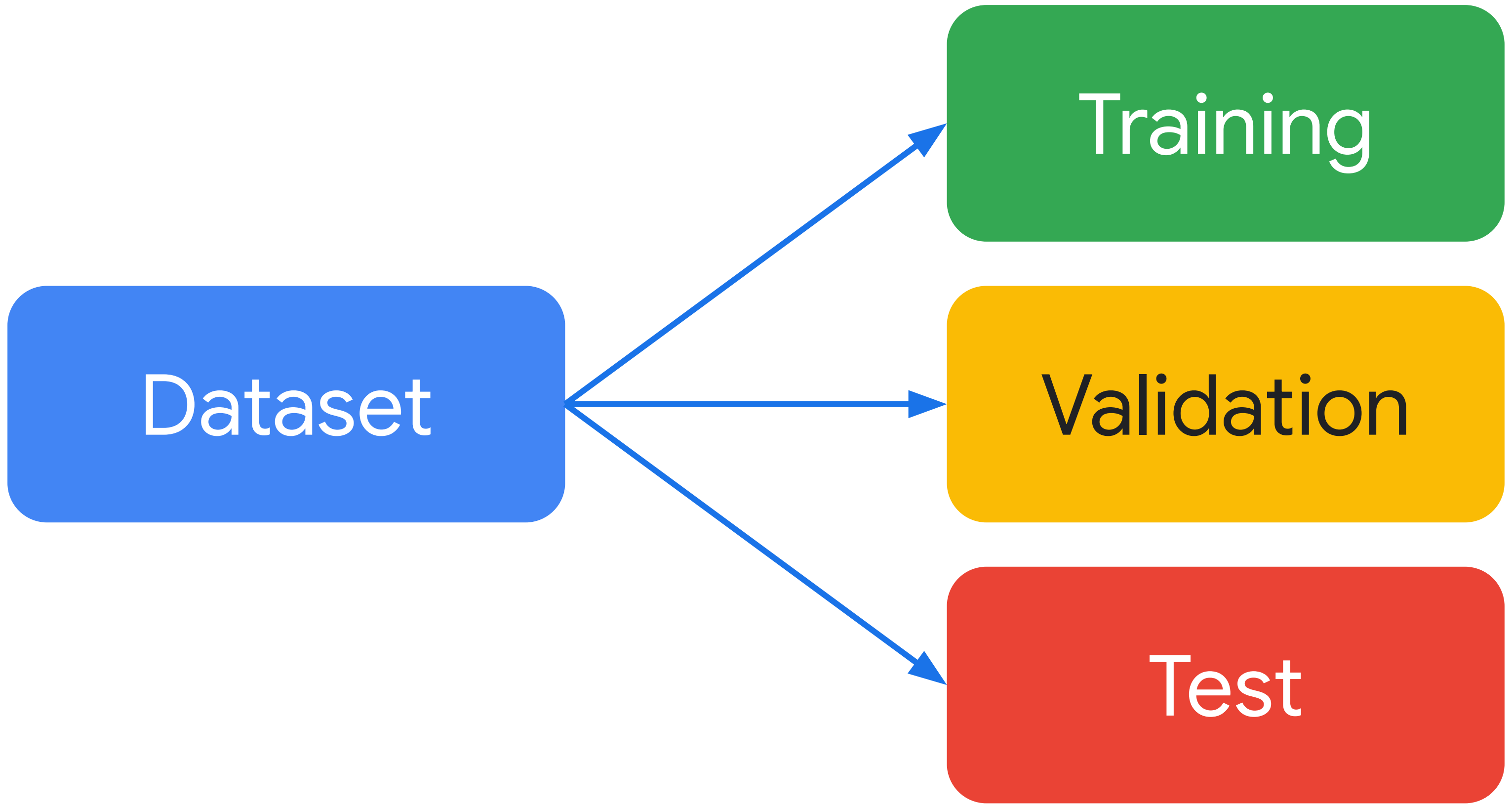
https://archive.ics.uci.edu/ml/datasets/covertype

Features

Target

Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noon	Hillshade_3pm	Horizontal_Distance_To_Fire_Points	Wilderness_Area	Soil_Type	Cover_Type
2067	0	21	270	9	755	184	196	145	900	Cache	C2702	5
2574	0	2	319	20	1419	216	235	156	1595	Commanche	C2703	4
2559	0	0	510	16	1113	218	238	156	1332	Commanche	C2703	2
2647	0	6	402	94	641	212	229	155	1104	Commanche	C2703	2
2651	0	3	335	103	488	215	233	156	1381	Commanche	C2703	2
2647	0	6	417	94	648	212	229	155	1082	Commanche	C2703	2
2639	0	10	366	80	589	206	222	154	1041	Commanche	C2703	2
2590	0	2	201	13	1200	216	235	156	1719	Commanche	C2703	1
2447	0	4	0	0	631	213	232	156	711	Commanche	C2705	5
2501	0	6	228	31	1012	211	228	155	930	Commanche	C2705	1
2500	0	4	30	3	1746	213	232	156	886	Commanche	C2705	5
2641	0	1	90	15	1518	217	236	156	182	Commanche	C2705	2

Split the dataset and experiment with models



Getting a random 80% of your dataset for training is easy

```
#standardSQL
SELECT
  date,
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
  `bigquery-samples.airline_ontime_data.flights`
WHERE
  RAND() < 0.8
```

RAND will return a number between 0 and 1.

However, experimentation requires repeatability

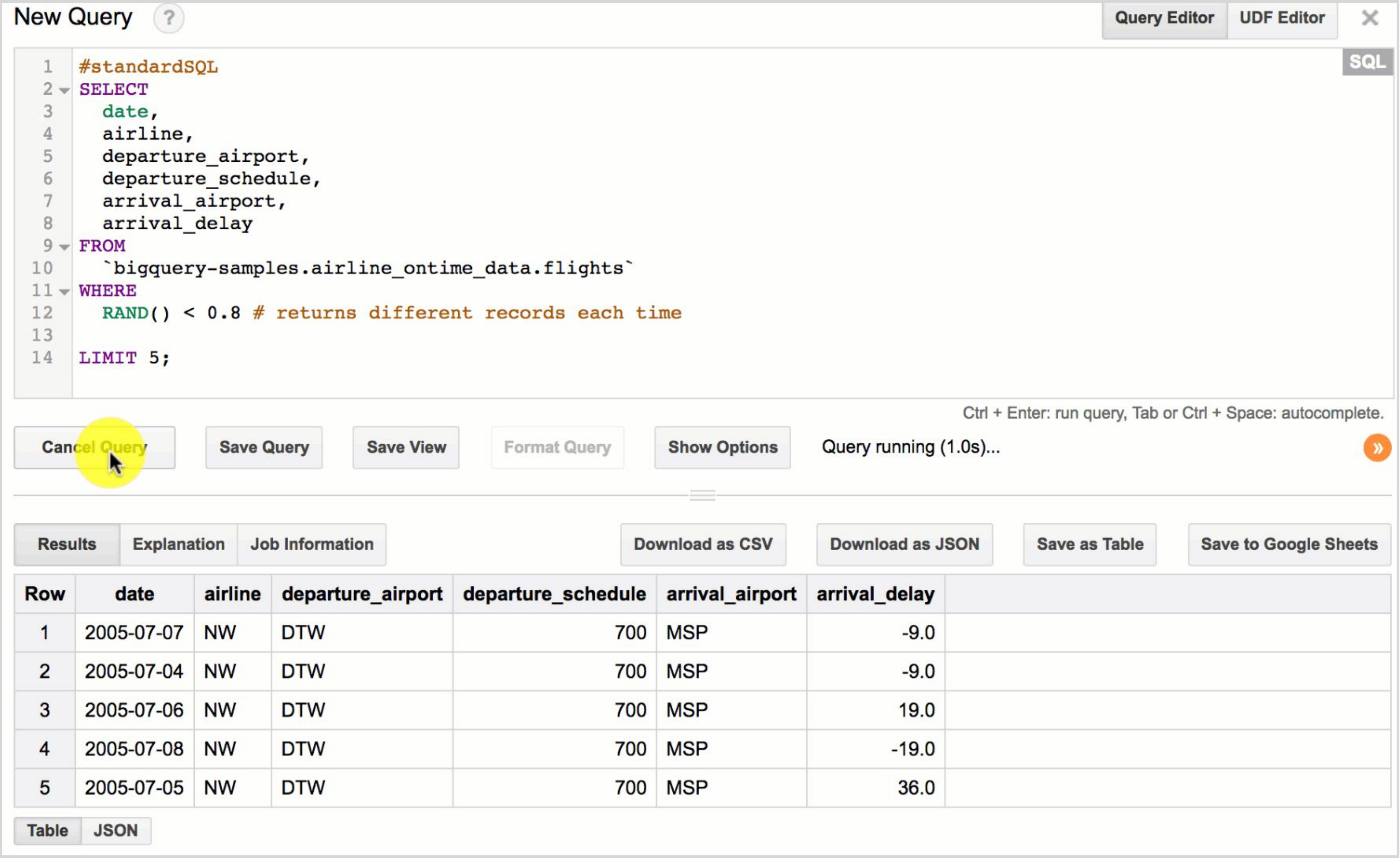
You need to know which specific data was involved in training, validation, and testing.

Naive random splitting is not repeatable

The order of rows in BigQuery is not certain without ORDER BY.

Identifying and splitting the remaining 20% of data for validation and testing is difficult.

RAND() will return different results each time →



Solution: Use hashing and modulo operators to split a dataset into training/validation

Input		Hash Value
Fox	Hash function	56042741
The red fox **runs** across the ice	Hash function	96042831
The red fox **walks** across the ice	Hash function	46042841

Solution: Use hashing and modulo operators to split a dataset into training/validation

```sql
#standardSQL
SELECT
  date,
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
 `bigquery-samples.airline_ontime_data.flights`

WHERE
  MOD(ABS(FARM_FINGERPRINT(date)), 10) < 8
```

Solution: Use hashing and modulo operators to split a dataset into training/validation

```
#standardSQL
SELECT
  date
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
 `bigquery-samples.airline_ontime_data.flights`

WHERE
  MOD(ABS(FARM_FINGERPRINT(date)),10) == 8
```

Note: Even though we select date, our model wouldn't actually use it during training.

Validation

Solution: Use hashing and modulo operators to split a dataset into training/validation

```
#standardSQL
SELECT
  date,
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
 `bigquery-samples.airline_ontime_data.flights`

WHERE
  MOD(ABS(FARM_FINGERPRINT(date)),10) == 9
```

Note: Even though we select date, our model wouldn't actually use it during training.

Testing

Solution: Use hashing and modulo operators to split a dataset into training/validation

```
#standardSQL
SELECT
  date,
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
 `bigquery-samples.airline_ontime_data.flights`

WHERE
  MOD(ABS(FARM_FINGERPRINT(date)),10) == 9
```

Note: Even though we select date, our model wouldn't actually use it during training.

Which field to hash on?

Which field to hash on?

Possible solution: Concatenate all the fields as a JSON string, and hash on that.

```
TO_JSON_STRING(cover)
```

Create a training split

```
bq query \
  -n 0 \
  --destination_table covertype_dataset.training \
  --replace \
  --use_legacy_sql=false \
    'SELECT * \
     FROM `covertype_dataset.covertype` AS cover \
     WHERE \
     MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (1, 2, 3, 4)'
```

Create the training table in BigQuery.

Create a training split

```
bq query \
  -n 0 \
  --destination_table covertype_dataset.training \
  --replace \
  --use_legacy_sql=false \
    'SELECT * \
     FROM `covertype_dataset.covertype` AS cover \
     WHERE \
     MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (1, 2, 3, 4)'
```

Create the training table in BigQuery.

Create a training split

```
bq query \
  -n 0 \
  --destination_table covertype_dataset.training \
  --replace \
  --use_legacy_sql=false \
    'SELECT * \
     FROM `covertype_dataset.covertype` AS cover \
     WHERE \
     MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (1, 2, 3, 4)'
```

Create the training table in BigQuery.

```
bq extract \
  --destination_format CSV \
  covertype_dataset.training \
  $TRAINING_FILE_PATH
```

Export it to Cloud Storage as a CSV file.

Create a training split

```
bq query \
  -n 0 \
  --destination_table covertype_dataset.training \
  --replace \
  --use_legacy_sql=false \
    'SELECT * \
     FROM `covertype_dataset.covertype` AS cover \
     WHERE \
     MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (1, 2, 3, 4)'
```

Create the training table in BigQuery.

```
bq extract \
  --destination_format CSV \
  covertype_dataset.training \
  $TRAINING_FILE_PATH
```

Export it to Cloud Storage as a CSV file.

Do the same for the validation split

```
bq query \
  -n 0 \
  --destination_table covertype_dataset.validation \
  --replace \
  --use_legacy_sql=false \
    'SELECT * \
     FROM `covertype_dataset.covertype` AS cover \
     WHERE \
     MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (8)'

bq extract \
  --destination_format CSV \
  covertype_dataset.validation \
  $VALIDATION_FILE_PATH
```

Do the same for the validation split

```
bq query \
  -n 0 \
  --destination_table covertype_dataset.validation \
  --replace \
  --use_legacy_sql=false \
    'SELECT * \
     FROM `covertype_dataset.covertype` AS cover \
     WHERE \
     MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (8)'

bq extract \
  --destination_format CSV \
  covertype_dataset.validation \
  $VALIDATION_FILE_PATH
```

Agenda

System and Concepts Overview

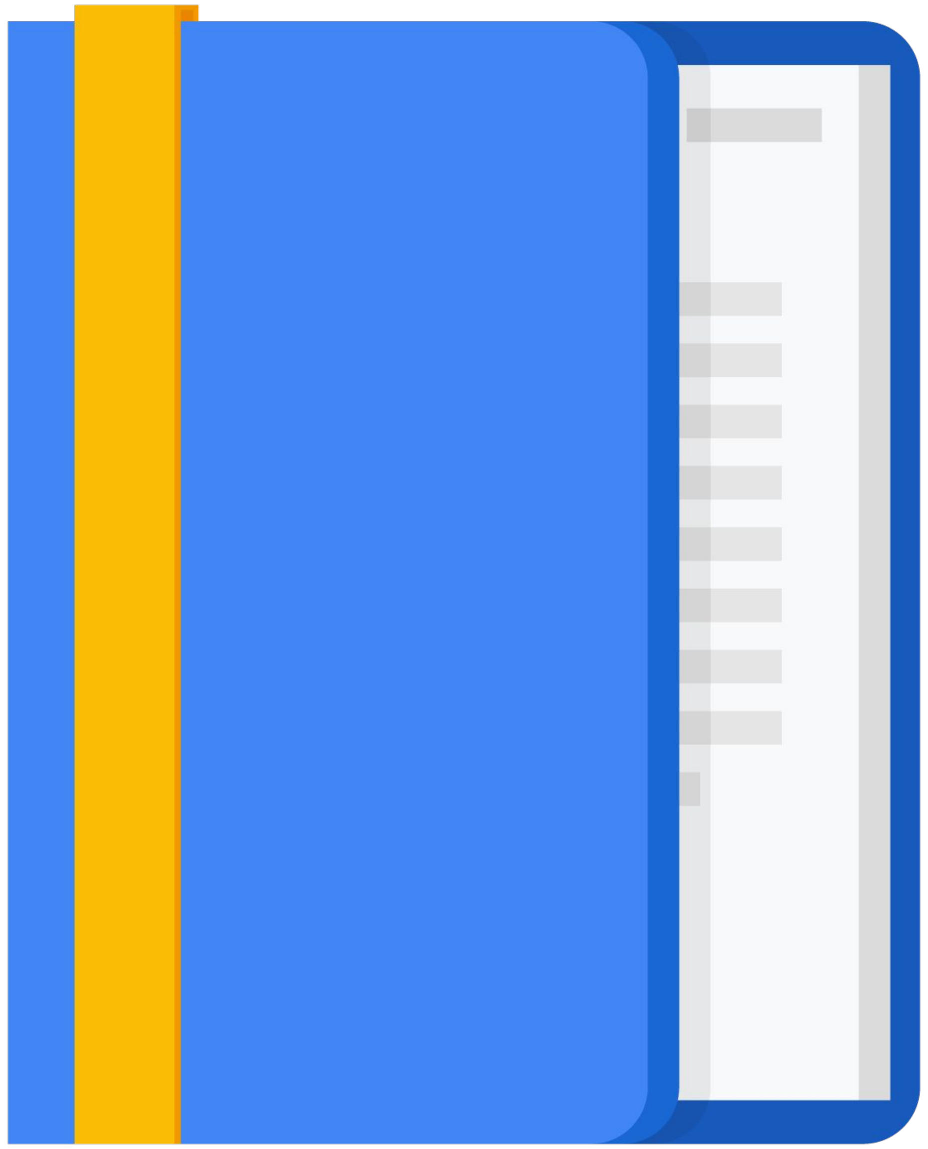
Create a Reproducible Dataset

Implement a Tunable Model

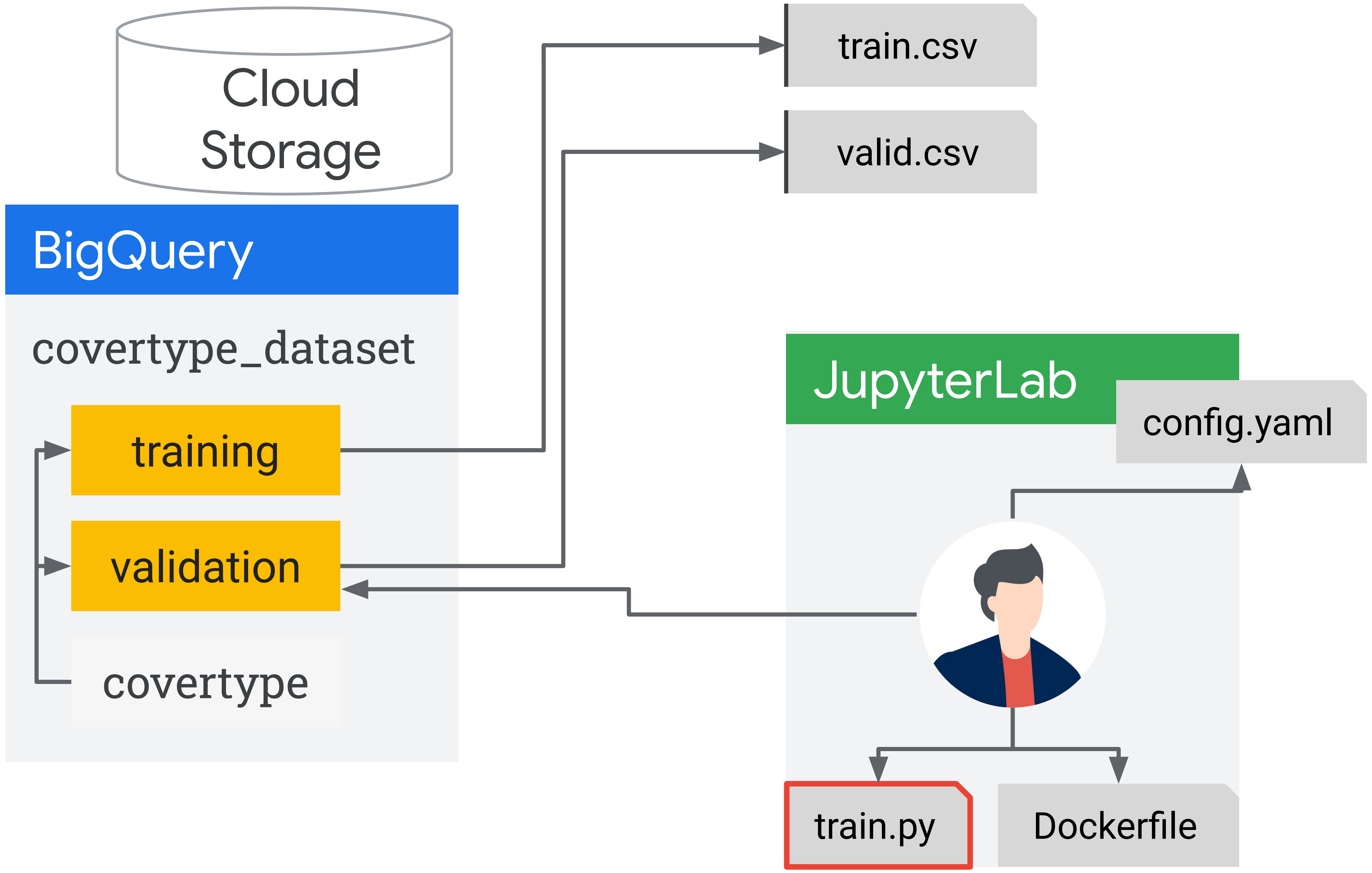
Build and Push a Training Container

Train and Tune a Model

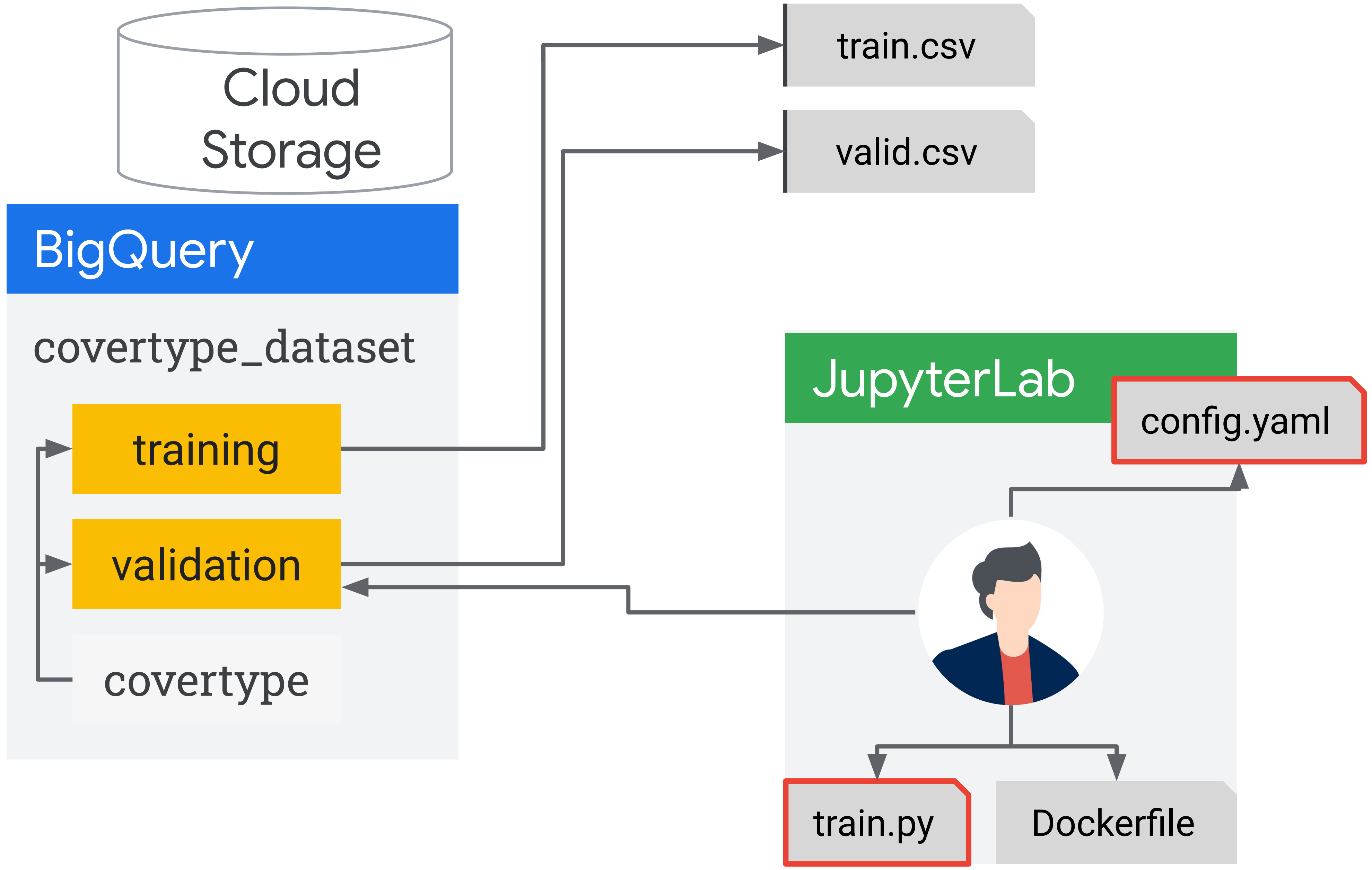
Serve and Query a Model



System overview



System overview



ML model: Sklearn pipeline

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

train.py

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

train.py

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

train.py

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

train.py

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier_alpha=0.001, classifier_max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

train.py

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

train.py

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

train.py

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

ML model: Sklearn pipeline

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)

accuracy = pipeline.score(X_validation, y_validation)
```

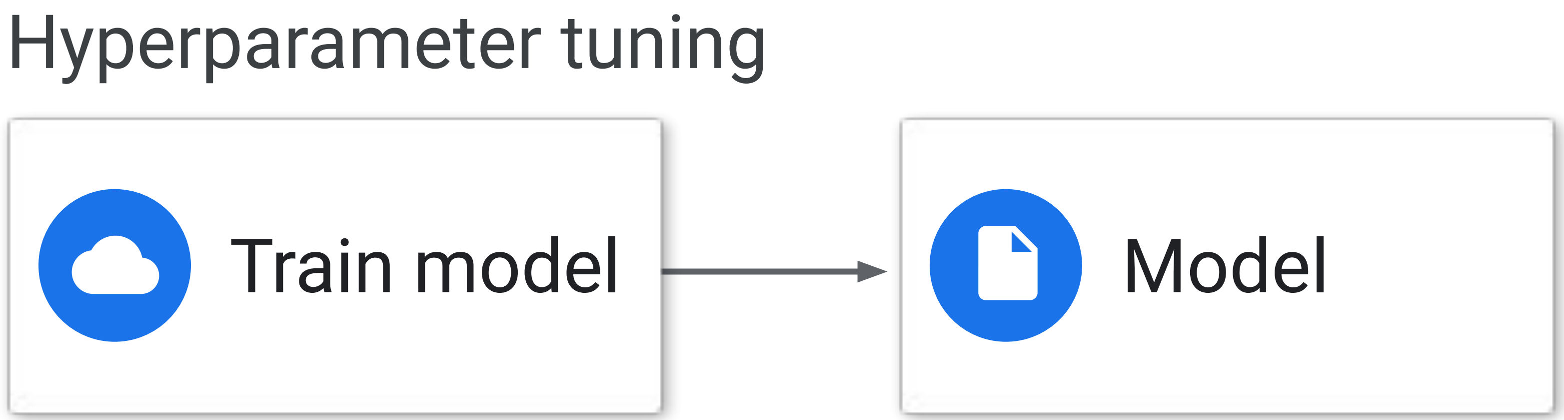
How to use Vertex AI for hyperparameter tuning

1. Make the hyperparameter a command-line argument.

2. Set up `cloudml-hypertune` to record training metrics.

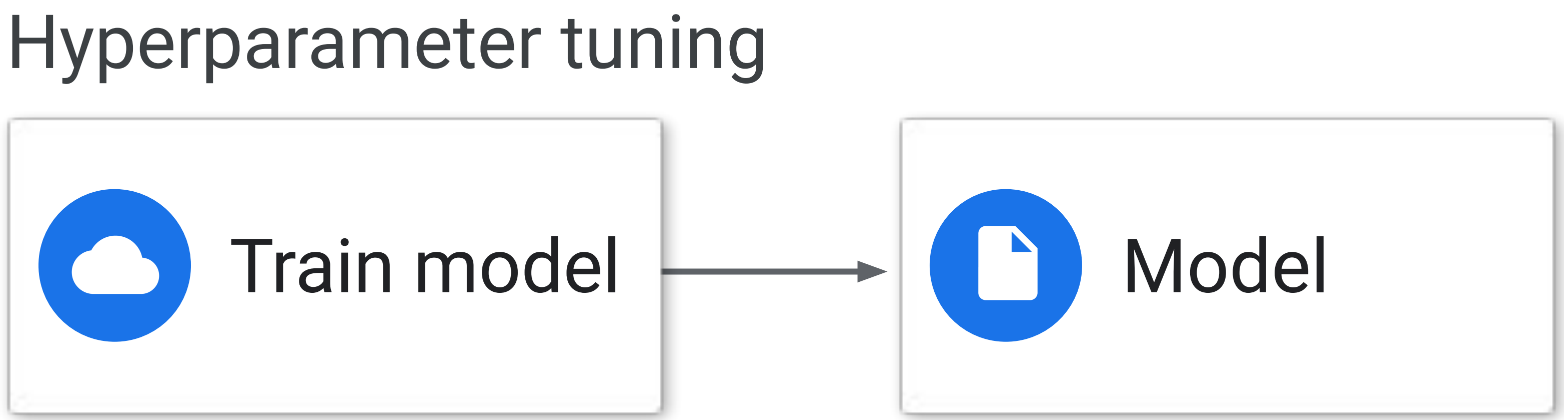
3. Export the final trained model.

4. Supply hyperparameters to the training job.



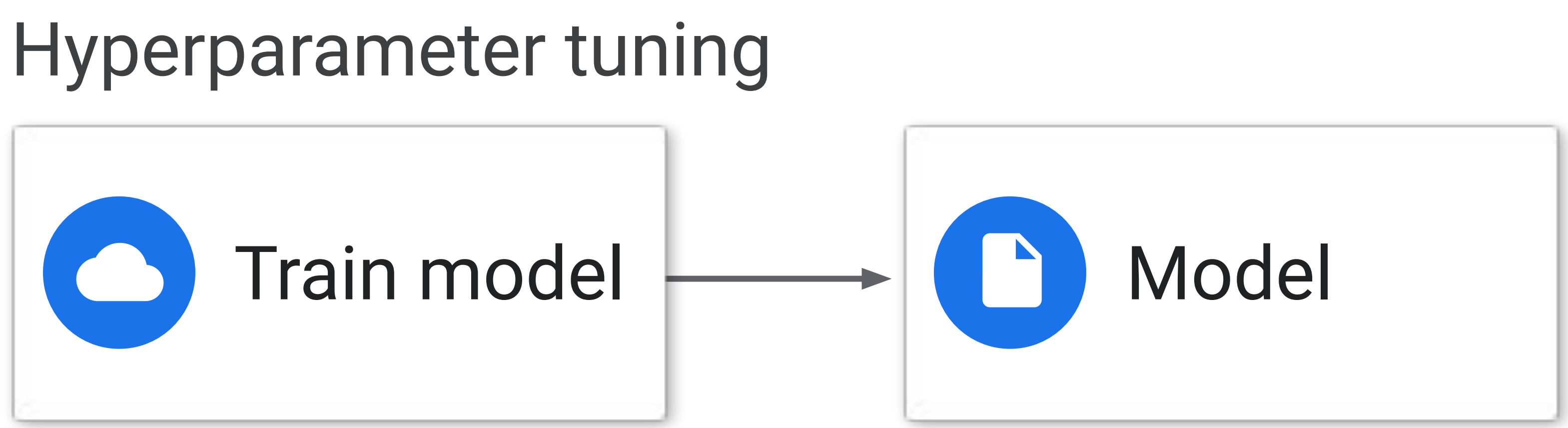
How to use Vertex AI for hyperparameter tuning

1. Make the hyperparameter a command-line argument.
2. Set up `cloudml-hypertune` to record training metrics.
3. Export the final trained model.
4. Supply hyperparameters to the training job.



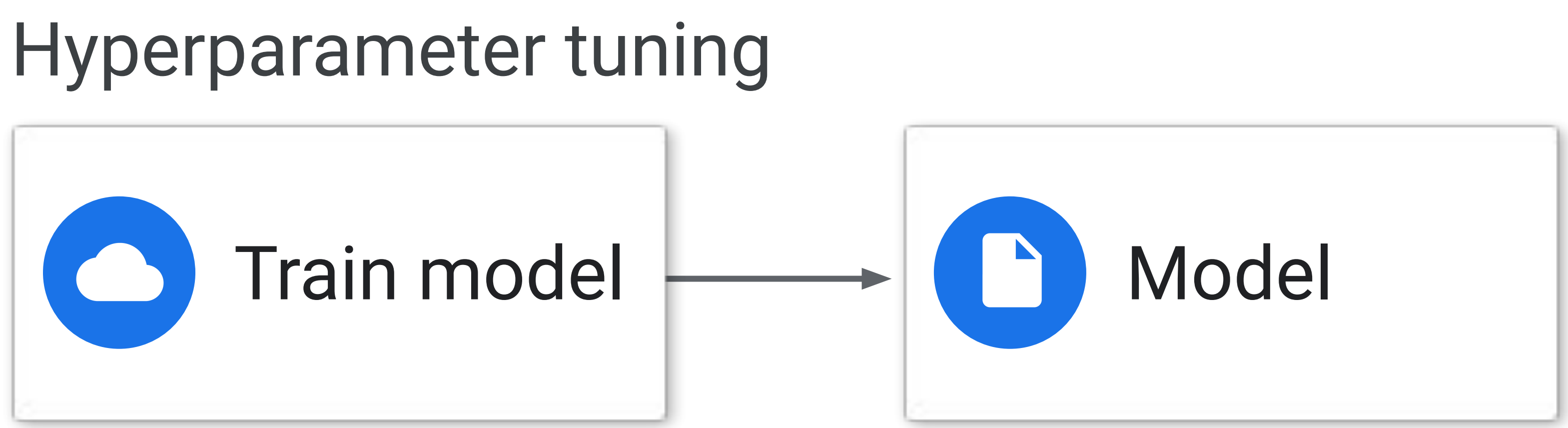
How to use Vertex AI for hyperparameter tuning

1. Make the hyperparameter a command-line argument.
2. Set up `cloudml-hypertune` to record training metrics.
3. Export the final trained model.
4. Supply hyperparameters to the training job.



How to use Vertex AI for hyperparameter tuning

1. Make the hyperparameter a command-line argument.
2. Set up `cloudml-hypertune` to record training metrics.
3. Export the final trained model.
4. Supply hyperparameters to the training job.



1. Make the hyperparameter a command-line argument

train.py

```
import fire

def train_evaluate(job_dir,
                   training_dataset_path,
                   validation_dataset_path,
                   alpha, max_iter, hptune):


    # [...]


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

```
python train.py \
  --job_dir $JOBDIR \
  --training_dataset_path $TRAINING_PATH \
  --validation_dataset_path $VALID_PATH \
  --alpha \
  --max_iter \
  --hptune
```

2. Set up `cloudml-hypertune` to record training metrics

train.py

```
import hypertune

def train_evaluate(job_dir,
                   training_dataset_path,
                   validation_dataset_path,
                   alpha, max_iter, hptune):
    # [...]

    if hptune:
        accuracy = pipeline.score(X_validation, y_validation)

        hpt = hypertune.HyperTune()

        hpt.report_hyperparameter_tuning_metric(
          hyperparameter_metric_tag='accuracy',
          metric_value=accuracy
        )

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

Import cloudml-hypertune.

2. Set up `cloudml-hypertune` to record training metrics

train.py

```
import hypertune

def train_evaluate(job_dir,
                   training_dataset_path,
                   validation_dataset_path,
                   alpha, max_iter, hptune):
    # [...]

    if hptune:
        accuracy = pipeline.score(X_validation, y_validation)

        hpt = hypertune.HyperTune()

        hpt.report_hyperparameter_tuning_metric(
          hyperparameter_metric_tag='accuracy',
          metric_value=accuracy
        )

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

Capture the metrics.

3. Export the final trained model

train.py

```
import pickle

def train_evaluate(job_dir,
                   training_dataset_path,
                   validation_dataset_path,
                   alpha, max_iter, hptune):
    # [...]

    if not hptune:
        model_filename = 'model.pkl'
        with open(model_filename, 'wb') as model_file:
            pickle.dump(pipeline, model_file)
        gcs_model_path = "{}/{}".format(job_dir, model_filename)
        subprocess.check_call(['gsutil', 'cp', model_filename, gcs_model_path],
        stderr=sys.stdout)

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

3. Export the final retrain model when not tuning

train.py

```
import pickle

def train_evaluate(job_dir,
                   training_dataset_path,
                   validation_dataset_path,
                   alpha, max_iter, hptune):
    # [...]

    if not hptune:
        model_filename = 'model.pkl'
        with open(model_filename, 'wb') as model_file:
            pickle.dump(pipeline, model_file)
        gcs_model_path = "{}/{}".format(job_dir, model_filename)
        subprocess.check_call(['gsutil', 'cp', model_filename, gcs_model_path],
        stderr=sys.stdout)

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

3. Export the final retrain model when not tuning

train.py

```
import pickle

def train_evaluate(job_dir,
                   training_dataset_path,
                   validation_dataset_path,
                   alpha, max_iter, hptune):
    # [...]

    if not hptune:
        model_filename = 'model.pkl'
        with open(model_filename, 'wb') as model_file:
            pickle.dump(pipeline, model_file)
        gcs_model_path = "{}/{}".format(job_dir, model_filename)
        subprocess.check_call(['gsutil', 'cp', model_filename, gcs_model_path],
        stderr=sys.stdout)

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

3. Export the final retrain model when not tuning

train.py

```
import pickle

def train_evaluate(job_dir,
                   training_dataset_path,
                   validation_dataset_path,
                   alpha, max_iter, hptune):
    # [...]

    if not hptune:
        model_filename = 'model.pkl'
        with open(model_filename, 'wb') as model_file:
            pickle.dump(pipeline, model_file)
        gcs_model_path = "{}/{}".format(job_dir, model_filename)
        subprocess.check_call(['gsutil', 'cp', model_filename, gcs_model_path],
        stderr=sys.stdout)

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

4. Supply hyperparameters to the training job

config.yaml

```yaml
studySpec:
  metrics:
  - metricId: accuracy
    goal: MAXIMIZE
  parameters:
  - parameterId: max_iter
    discreteValueSpec:
      values:
      - 10
      - 20
  - parameterId: alpha
    doubleValueSpec:
      minValue: 1.0e-4
      maxValue: 1.0e-1
    scaleType: UNIT_LINEAR_SCALE
  algorithm: ALGORITHM_UNSPECIFIED # results in Bayesian optimization
```

4. Supply hyperparameters to the training job

config.yaml

```
trialJobSpec:
  workerPoolSpecs:
  - machineSpec:
      machineType: n1-standard-4
    replicaCount: 1
    containerSpec:
      imageUri:
      args:
      - --job_dir=<JOBDIR>
      - --training_dataset_path=<TRAINING_DATASET_PATH>
      - --validation_dataset_path=<VALIDATION_DATASET_PATH>
      - --hptune
```

Agenda

System and Concepts Overview

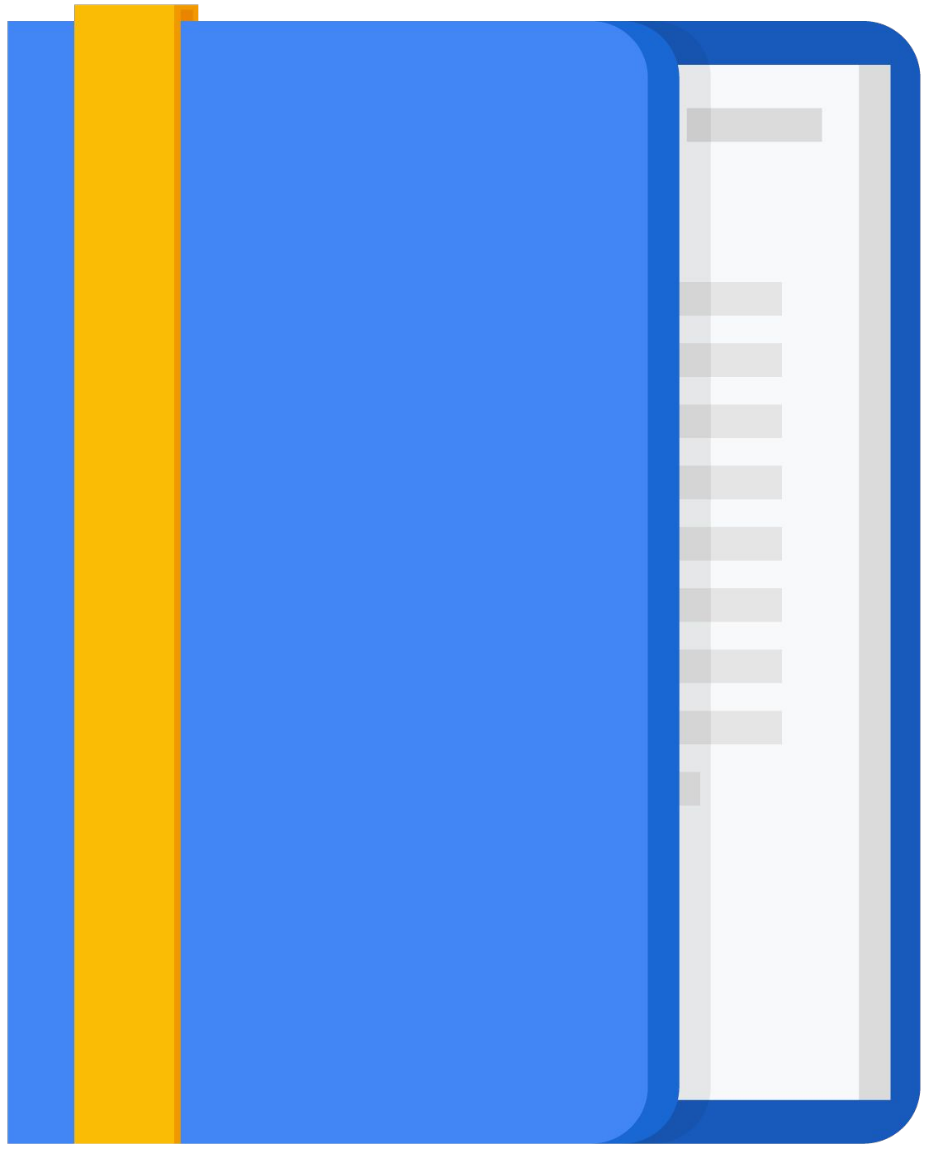
Create a Reproducible Dataset

Implement a Tunable Model

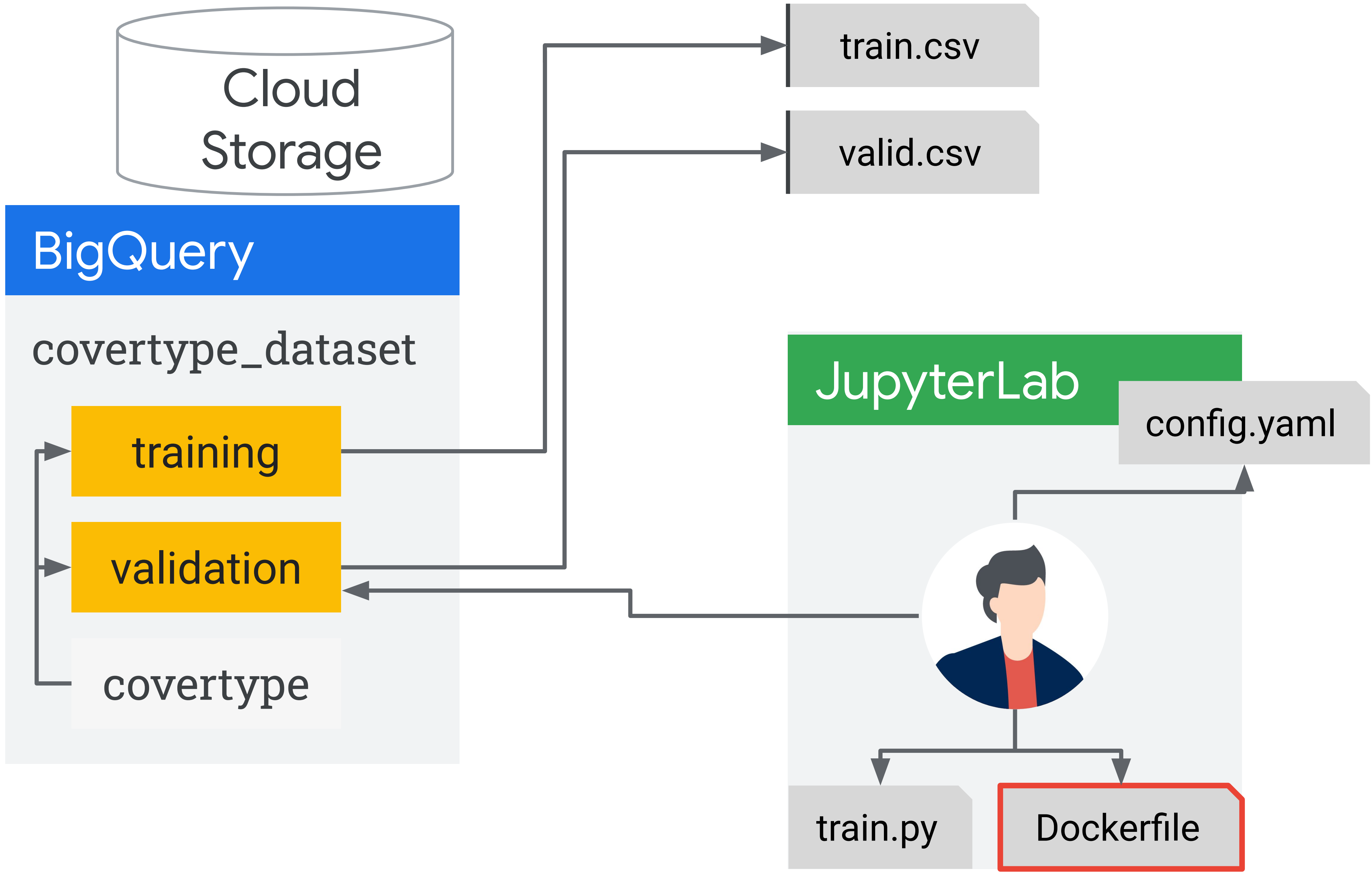
Build and Push a Training Container

Train and Tune a Model

Serve and Query a Model



System overview



Create the training Docker container

Dockerfile

```
FROM gcr.io/deeplearning-platform-release/base-cpu

RUN pip install -U fire cloudml-hypertune scikit-learn==0.20.4 pandas==0.24.2

WORKDIR /app

COPY train.py .

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

```
gcloud builds submit --tag gcr.io/$PROJECT/$IMAGE:$TAG $TRAINING_APP_FOLDER
```

Agenda

System and Concepts Overview

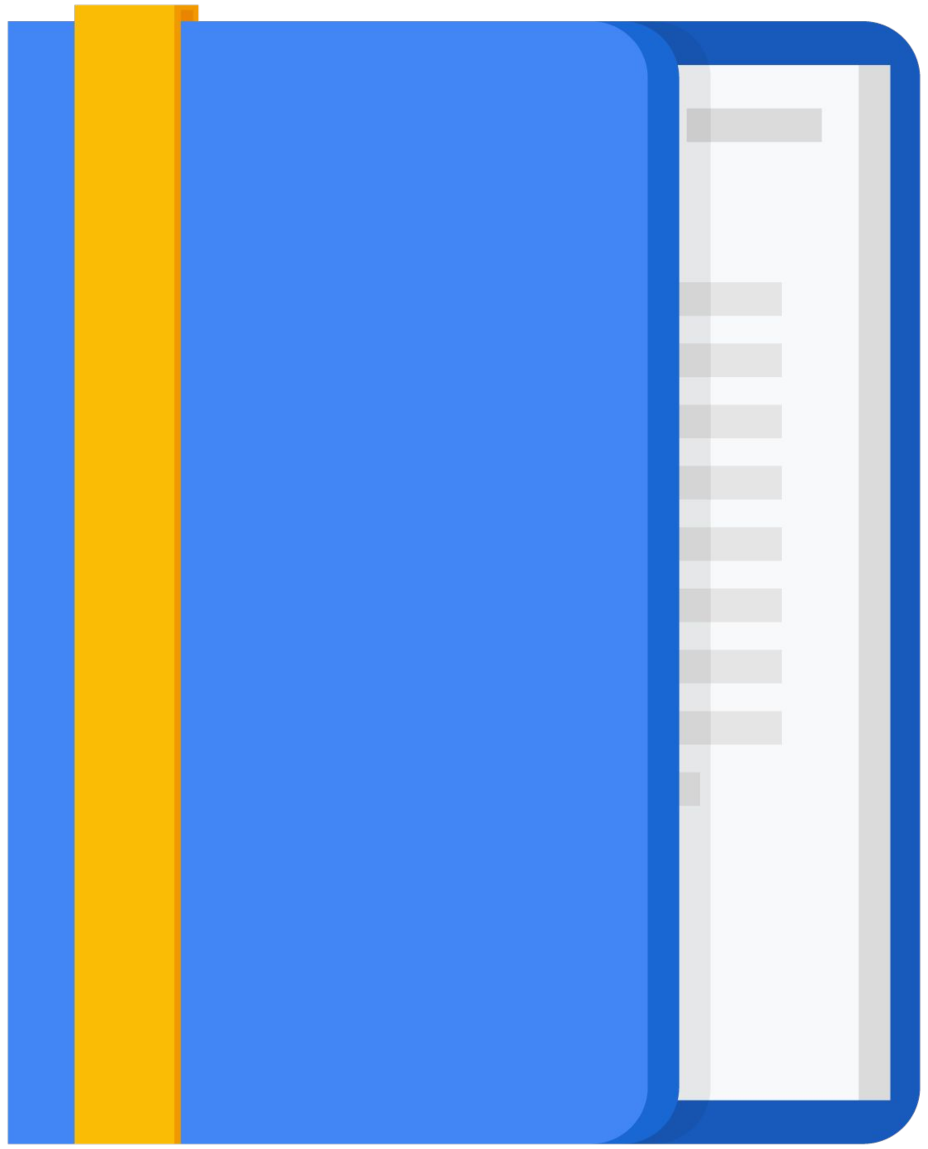
Create a Reproducible Dataset

Implement a Tunable Model

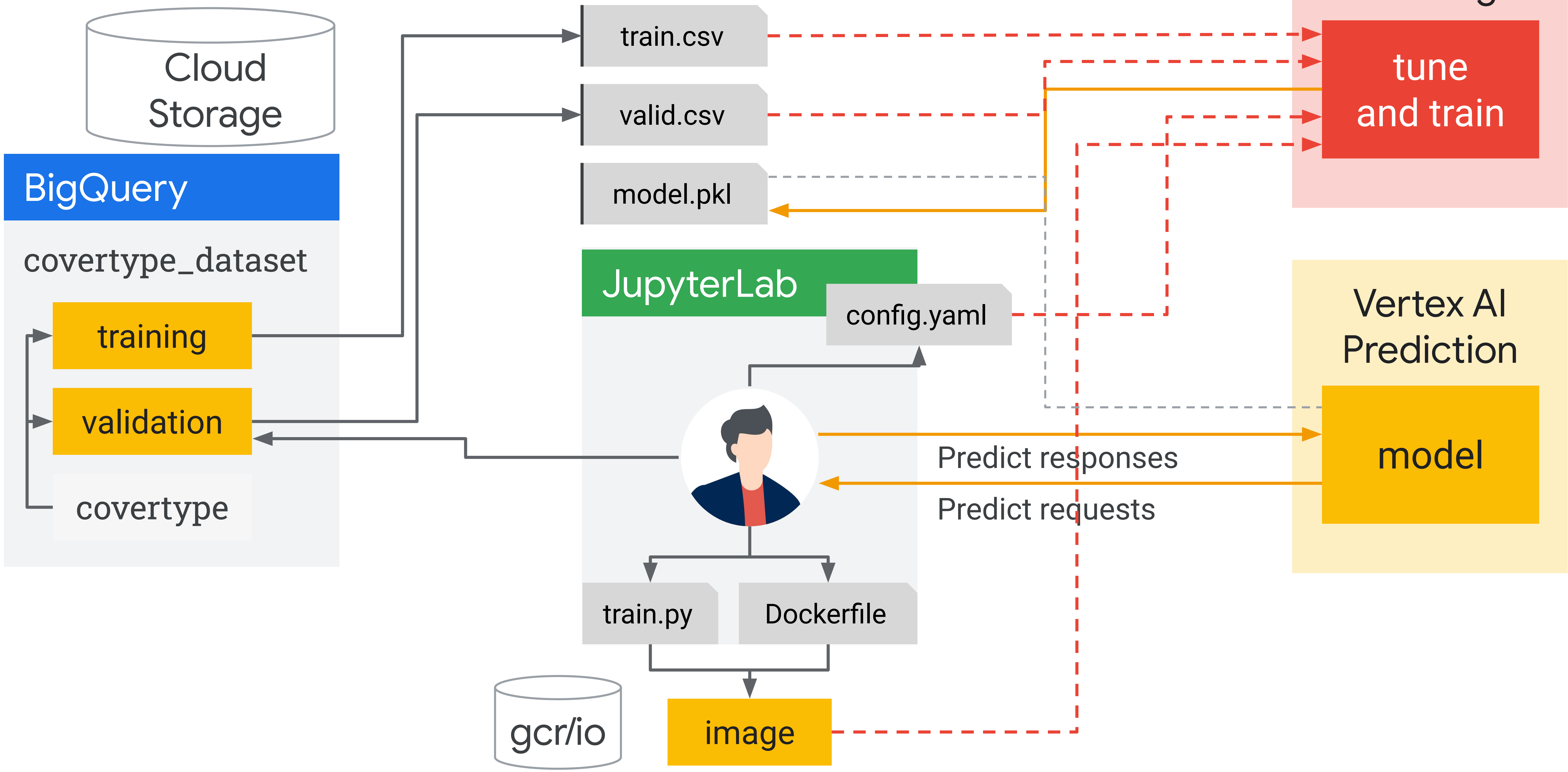
Build and Push a Training Container

Train and Tune a Model

Serve and Query a Model

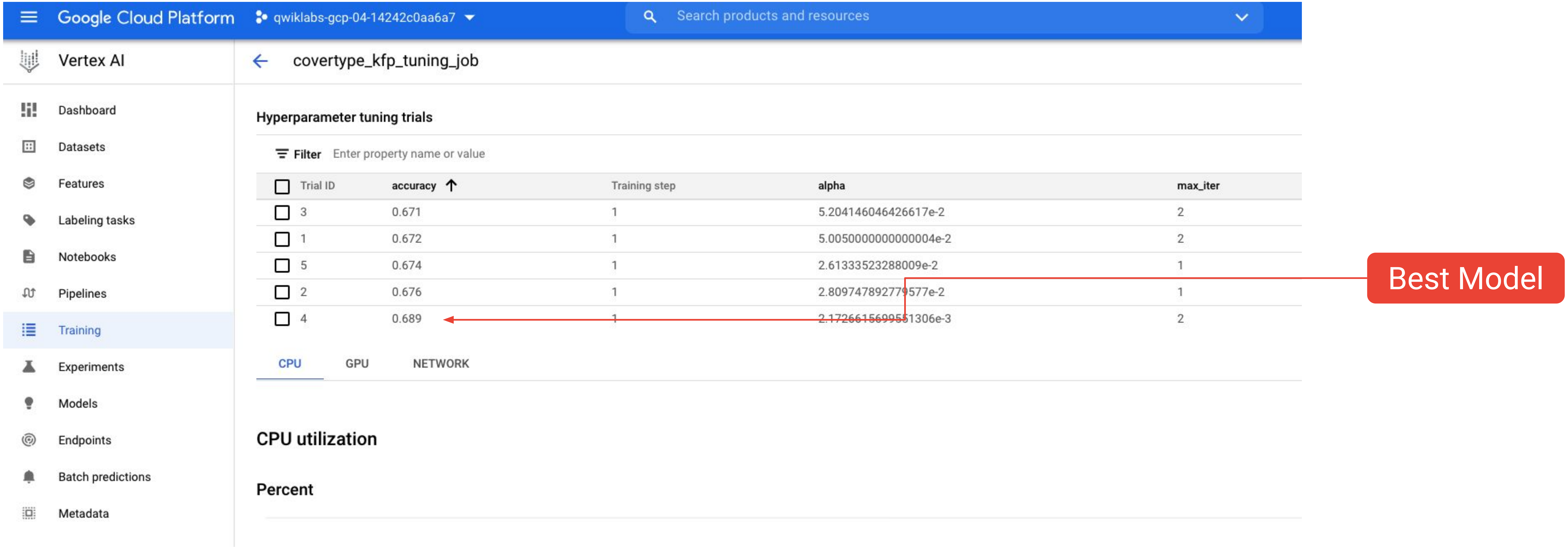


System overview



Start the hyper tuning job on Vertex AI

```
gcloud ai hp-tuning-jobs create \
    --region=$REGION \
    --display-name=$JOB_NAME \
    --config=$CONFIG_YAML \
    --max-trial-count=5 \
    --parallel-trial-count=5
```



Trial ID	accuracy ↑	Training step	alpha	max_iter
3	0.671	1	5.204146046426617e-2	2
1	0.672	1	5.0050000000000004e-2	2
5	0.674	1	2.61333523288009e-2	1
2	0.676	1	2.809747892779577e-2	1
4	0.689	1	2.1726615699551306e-3	2

Query Vertex AI Training for the best hyperparameters

```
from google.cloud import aiplatform

def get_trials(job_name):
    jobs = aiplatform.HyperparameterTuningJob.list()
    match = [job for job in jobs if job.display_name == JOB_NAME]
    tuning_job = match[0] if match else None
    return tuning_job.trials if tuning_job else None

def get_best_trial(trials):
    metrics = [trial.final_measurement.metrics[0].value for trial in trials]
    best_trial = trials[metrics.index(max(metrics))]
    return best_trial

def retrieve_best_trial_from_job_name(jobname):
    trials = get_trials(jobname)
    best_trial = get_best_trial(trials)
    return best_trial
```

Retrain with the best hyperparameters and export

```
gcloud ai custom-jobs create \
        --region={REGION} \
        --display-name={JOB_NAME} \
        --worker-pool-spec={WORKER_POOL_SPEC} \
        --args={ARGS}
```

```
WORKER_POOL_SPEC = f"""\
machine-type={MACHINE_TYPE},\
replica-count={REPLICA_COUNT},\
container-image-uri={IMAGE_URI}\
"""

ARGS = f"""\
--job_dir={JOB_DIR},\
--training_dataset_path={TRAINING_FILE_PATH},\
--validation_dataset_path={VALIDATION_FILE_PATH},\
--alpha={alpha},\
--max_iter={max_iter},\
--nohptune\
"""
```

Agenda

System and Concepts Overview

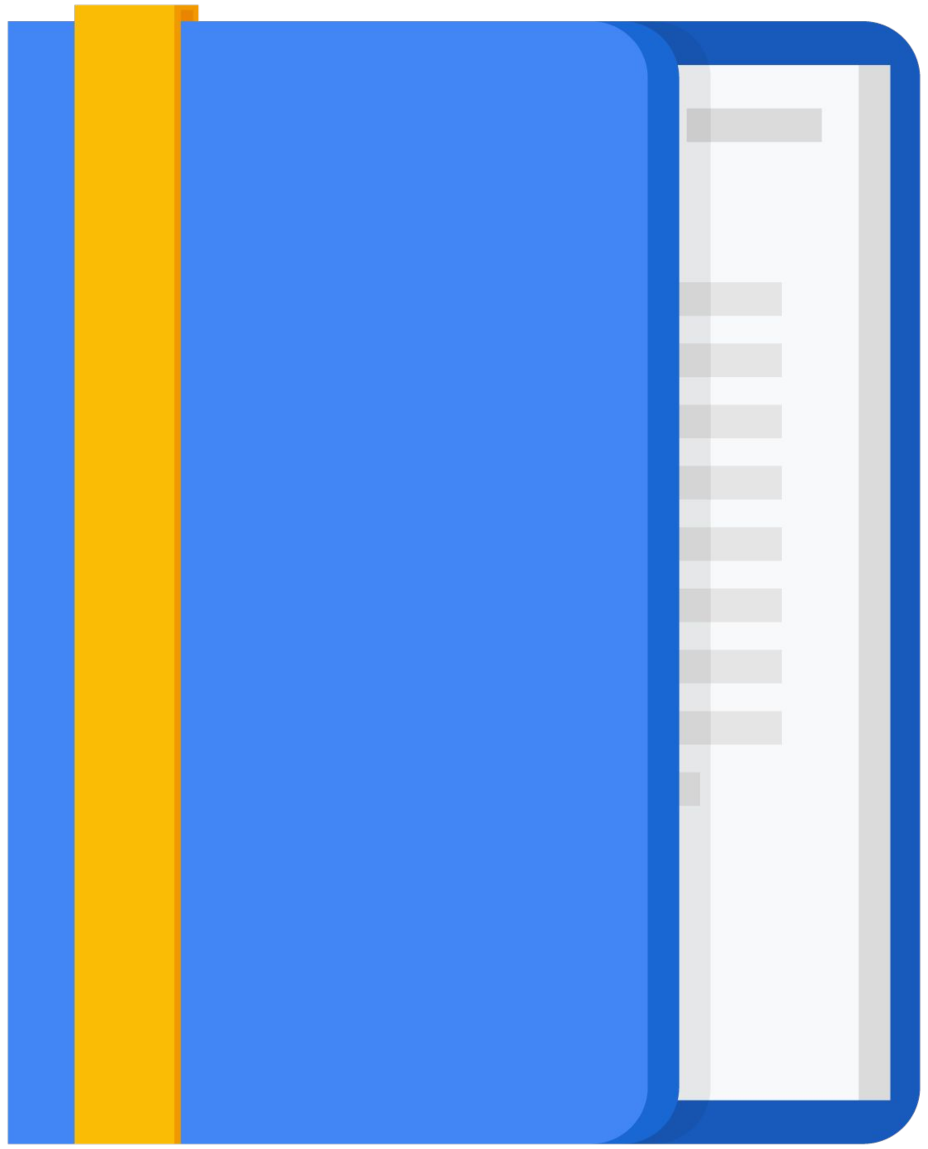
Create a Reproducible Dataset

Implement a Tunable Model

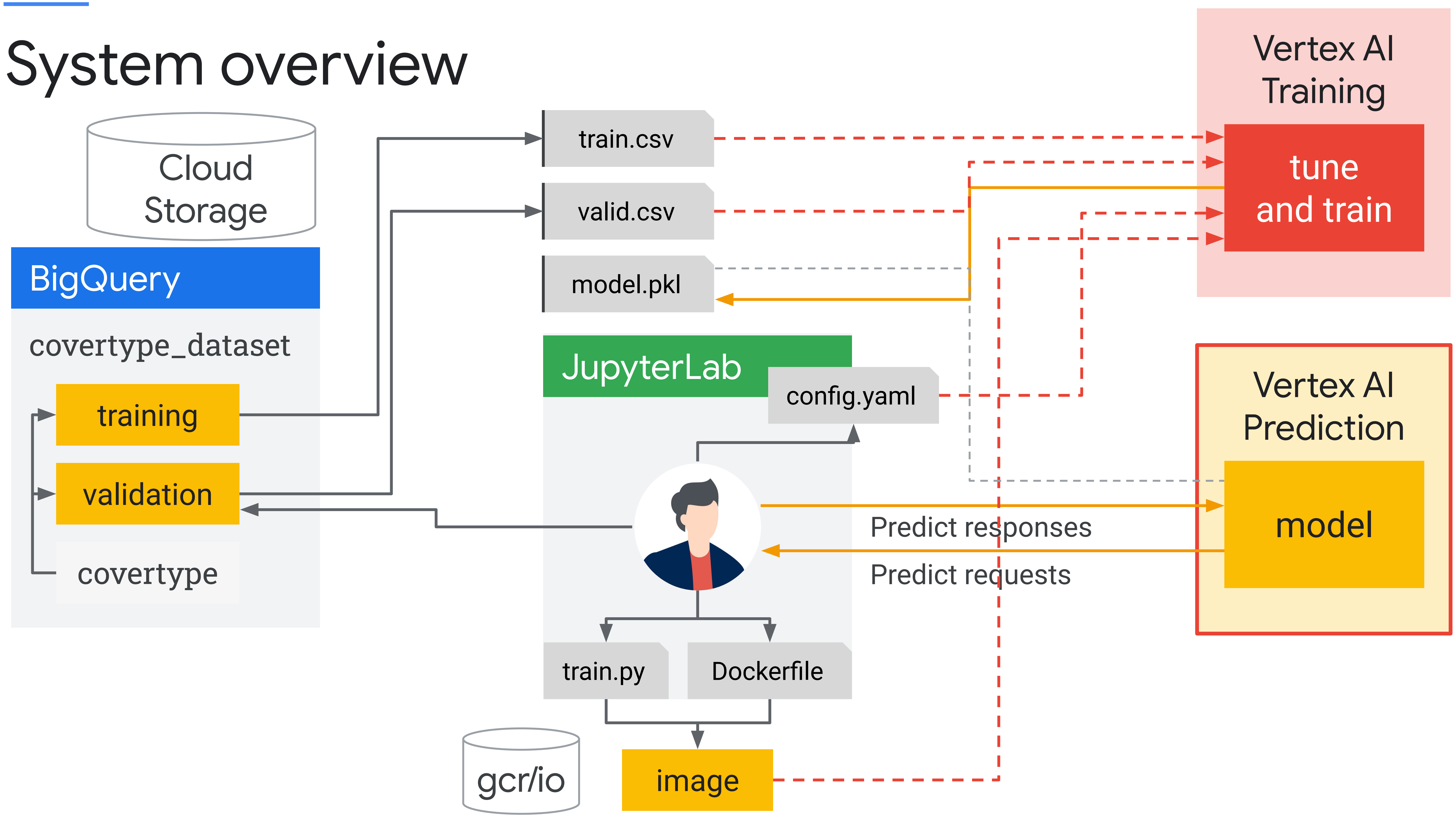
Build and Push a Training Container

Train and Tune a Model

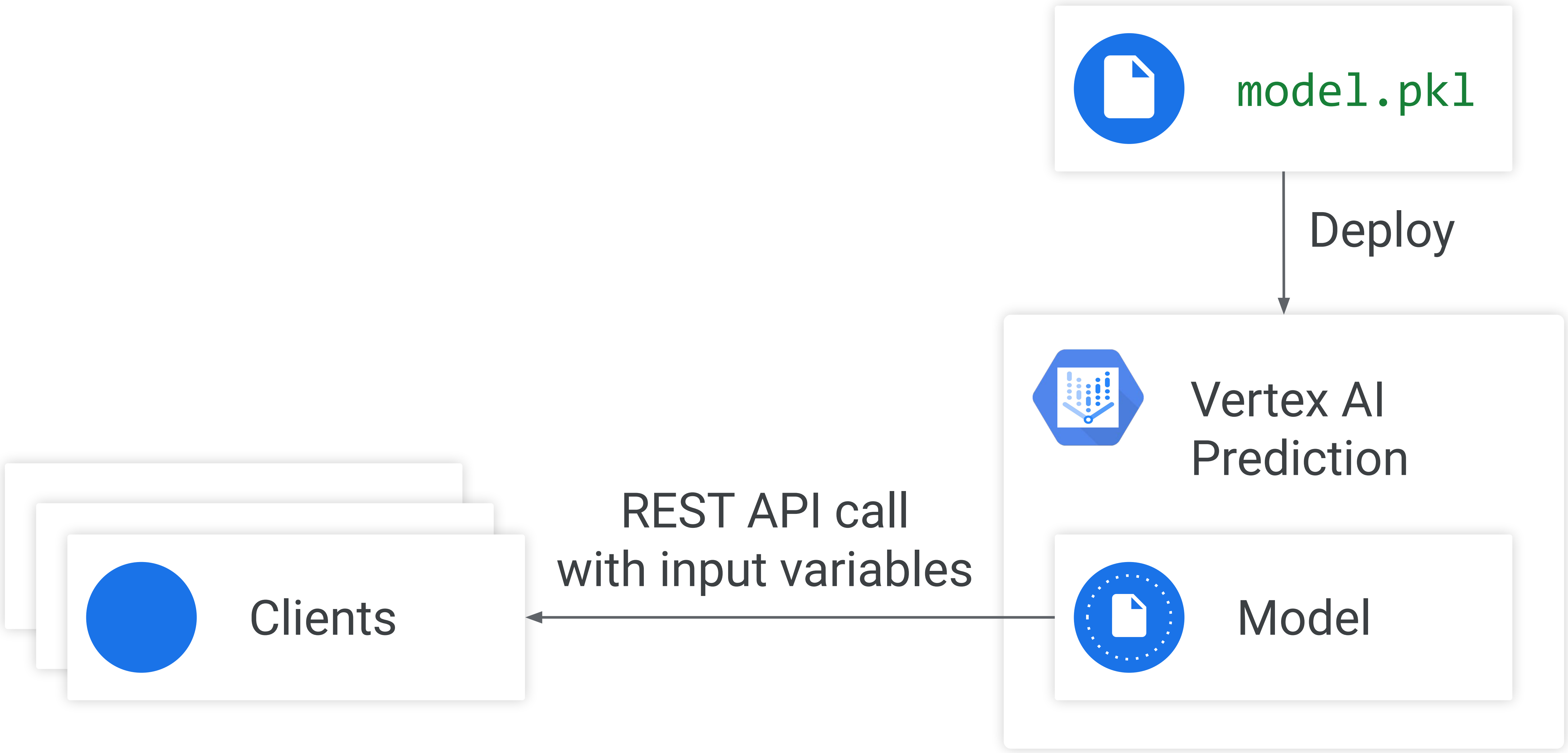
Serve and Query a Model



System overview



Vertex AI Prediction makes deploying models easy



Uload the trained model

```
from google.cloud import aiplatform

uploaded_model = aiplatform.Model.upload(
    display_name=MODEL_NAME,
    artifact_uri=JOB_DIR,
    serving_container_image_uri=SERVING_CONTAINER_IMAGE_URI,
)
```

Deploy the uploaded model

```
endpoint = uploaded_model.deploy(
    machine_type=SERVING_MACHINE_TYPE,
    accelerator_type=None,
    accelerator_count=None,
)
```

Query the model

```
instance = [2841.0, 45.0, 0.0, 644.0, 282.0, 1376.0, 218.0, 237.0,
156.0, 1003.0, "Commanche", "C4758"]

endpoint.predict([instance])
```

Lab

Training, Tuning, and Serving in Vertex AI

kubeflow_pipelines/walkthrough/labs/kfp_walkthrough_vertex.ipynb cloud.google.com