Model Deployment

Cloud vs. Edge Deployment

Our model is entirely deployed in the cloud using **Google Cloud Platform (GCP)**, ensuring scalability, reliability, and seamless integration with other GCP services. The deployment leverages **Vertex AI's AutoML capabilities**, which eliminates the need for custom code for the model. Instead, Vertex AI manages all aspects of model training, deployment, and predictions.

Key Points:

1. **No Model Code**: The project relies fully on **Vertex AI AutoML**, meaning there is no codebase for the model itself.
2. **Data-Driven Triggers**: The workflows are configured to trigger automatically whenever new data is added to the designated **Cloud Storage buckets**.

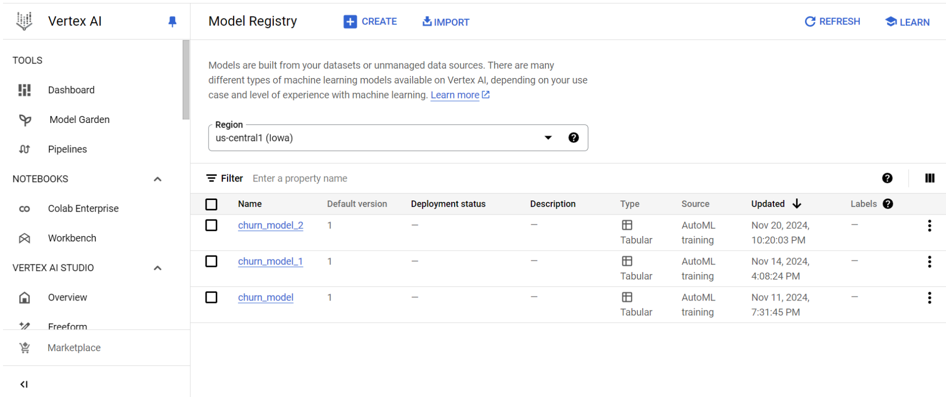
Cloud Deployment Details - Google Cloud Platform (GCP)

1. **Deployment Service: Vertex AI**

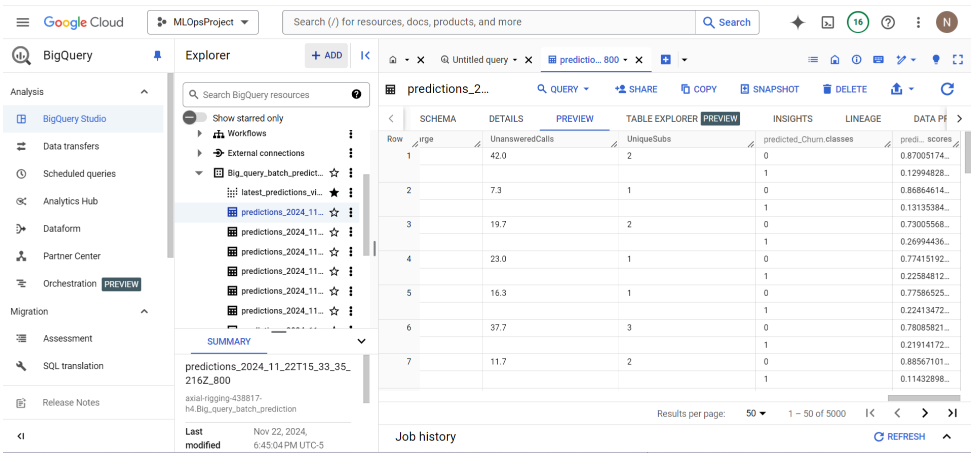
Model Serving:

The trained model is deployed and managed using **Vertex AI**, with workflows designed to:

- **Check for Existing Models**: The DAG first verifies whether the latest model exists in the **Vertex AI Model Registry**.



- **Train on Demand**: If no model exists or retraining is required, the DAG triggers Vertex AI's **AutoML Tabular Training** using data from the training bucket.
- **Batch Predictions**: Once a model is ready, a batch prediction job is created using the deployed model and data stored in a **GCS bucket**.



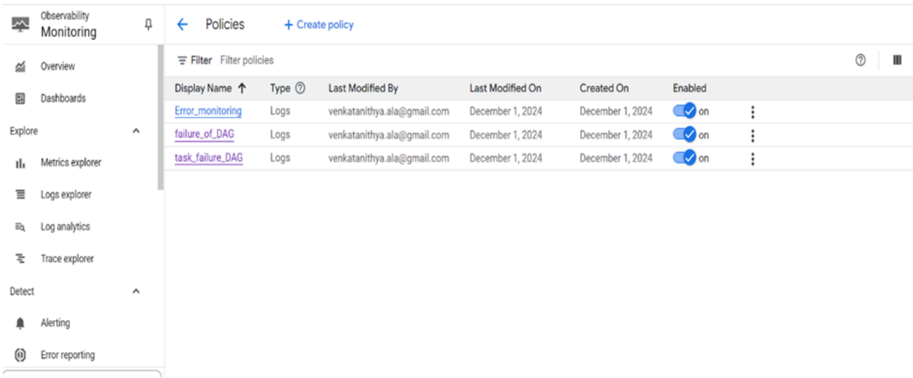
Retraining Architecture:

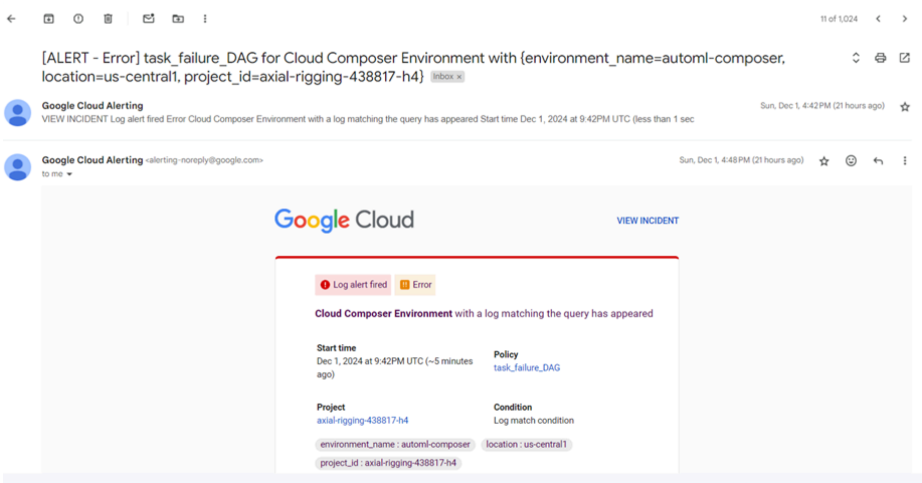
- **On-Demand Retraining**: New data placed in the training data bucket triggers the retraining DAG. Control over what data is added to the bucket provides flexibility to retrain only when necessary.

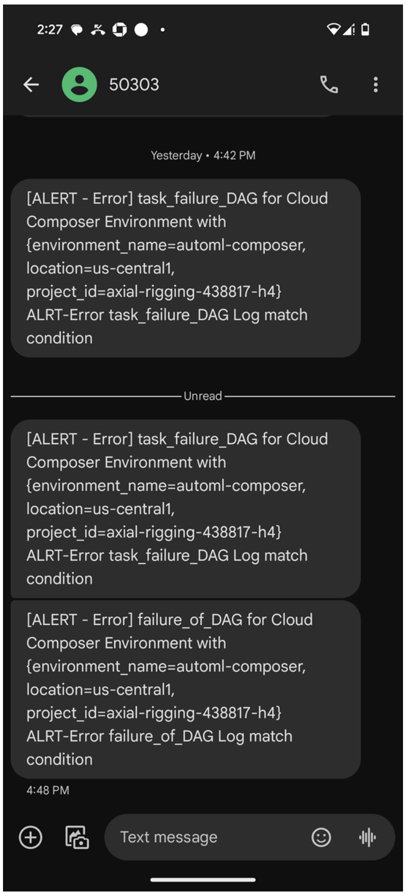
- **Anomaly Detection and Schema Validation**: The DAGs include tasks for validating schemas and detecting data anomalies. If significant anomalies (e.g., data drift) are detected, new data is added to the training bucket, which in turn triggers retraining.

Model Versioning:
Vertex AI handles versioning automatically, ensuring smooth updates and model lifecycle management.

2. **Deployment Automation**
   Workflow Orchestration with Apache Airflow (Cloud Composer)
   DAG Automation:
   - The pipeline is orchestrated using **Apache Airflow DAGs** deployed in **Cloud Composer**.
   - DAGs are triggered automatically by new data in **GCS buckets** rather than code changes, ensuring a hands-free operation. Tasks include:
     - Verifying model availability in **Vertex AI**.
     - Training a new model if required.
     - Fetching the latest data files and running batch predictions.
     - Updates frontend (Looker Studio) after model inference
     - Email and sms alerts set to monitor dag status







Fixed DAG Logic:

- The DAGs have a static, pre-defined logic that governs the entire workflow, making them robust and maintenance-free for routine data updates.

Vertex AI Integration:

- Interaction with Vertex AI is handled through DAG-2 (Vertex_ai_churn_model_training), Python and Airflow operators, allowing seamless orchestration.

3. **Connection to Repository**
   Code Management:
   - While the DAGs are maintained in a GitHub repository, routine connection to Git is not required for deployment. Due to the fixed DAG logic and the reliance on Vertex AI and GCS for automation, traditional CI/CD integration is not necessary for this project.

4. **Detailed Steps for Replication**
   Environment Setup:
   Access GCP and Ensure Required Permissions:
   - Obtain access to the GCP project (axial-rigging-438817-h4) from the project administrator.
   - Ensure the following roles are granted:
     - Vertex AI Admin
     - Cloud Composer User
     - Storage Admin
     - BigQuery Admin

   Preconfigured Resources:
   - Verify the presence of these resources in the GCP project:
     - **Snapshot of the Cloud Composer environment** with deployed Airflow DAGs (which is already saved in the snapshots folder).
     - **GCS buckets** containing ingested data files.

   Running the Workflow:
   1. User has to create a new Cloud Composer environment and load the snapshot to configure the entire environment and its dag dependencies.
   2. Access the **Airflow UI** via the **Cloud Composer console**.

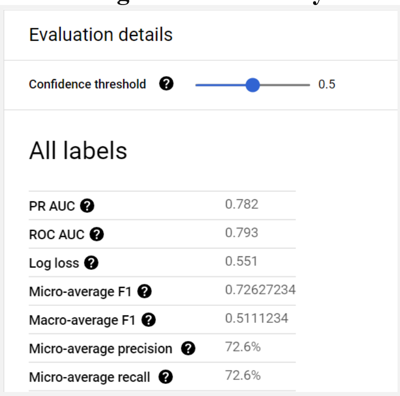
3. As the new data is ingested into the GCS buckets,subsequent DAGs trigger automatically, processing data and making predictions. DAG execution follows this order:
    1. **Telecome_dag_new** – This DAG ensures training data quality, validating schema for incoming data and generating alerts on anomalies.
    2. **Vertex_ai_churn_model_training** – Makes a call to Vertex AI model and generates predictions (inference), triggers retraining with incoming new training data.
    3. **Bigquery_to_pubsub** – Pushes messages to pubsub and updates frontend after model inference.
4. Use GCP Monitoring and Logging to ensure smooth operation and generating alerts.

Verifying Deployment:

- The deployed model is called and batch predictions are stored in a **BigQuery** and reflected in a **Looker Studio dashboard**, which updates automatically to display the latest results.
- Users can directly analyze results via the dashboard.

Model Monitoring and Triggering Retraining

1. Monitoring for Model Decay and Data Shift



We have used CreateAutoMLTabularTrainingJobOperator for creating model monitoring jobs.

```
# Task: Create batch prediction job
def create_batch_prediction_job(**kwargs):
    log = get_task_logger("create_batch_prediction_job")
    try:
        model_name = kwargs['ti'].xcom_pull(task_ids='check_model_existence', key='model_name')
        if not model_name:
            raise ValueError("No model found for batch prediction.")

        log.info("Initializing Vertex AI platform for batch prediction...")
        aiplatform.init(project=PROJECT_ID, location=LOCATION)

        gcs_input = f"gs://{BUCKET_NAME}/latest_best_features_for_churn.jsonl"
        bigquery_output_prefix = "axial-rigging-438817-h4.Big_query_batch_prediction"

        log.info("Creating batch prediction job...")
        batch_prediction_job = aiplatform.BatchPredictionJob.create(
            job_display_name="batch_prediction_job",
            model_name=model_name,
            gcs_source=gcs_input,
            predictions_format="bigquery",
            starting_replica_count=10,
            max_replica_count=20,
            machine_type="c2-standard-30",
            bigquery_destination_prefix=bigquery_output_prefix
```

Features	Baseline	Target	Alerts	Drift score	Threshold	Metric
AdjustmentsToCreditRating				0.58344	0.3	Jensen-Shannon
AgeHH1			–	0.00102	0.3	Jensen-Shannon
AgeHH2			–	0.00532	0.3	Jensen-Shannon
BuysViaMailOrder			–	0.02425	0.3	L-Infinity
CallWaitingCalls			–	0.03946	0.3	Jensen-Shannon
CreditRating				0.49963	0.3	L-Infinity
CurrentEquipmentDays			–	0.02856	0.3	Jensen-Shannon
CustomerCareCalls			–	0.19007	0.3	Jensen-Shannon
CustomerID			–	0.13037	0.3	Jensen-Shannon
DirectorAssistedCalls				0.3237	0.3	Jensen-Shannon
EmailID			–	0.16255	0.3	L-Infinity
HandsetModels			–	0.0295	0.3	Jensen-Shannon
HandsetPrice				0.61517	0.3	Jensen-Shannon
HandsetRefurbished			–	0.01126	0.3	L-Infinity
HandsetWebCapable			–	0.04381	0.3	L-Infinity
Handsets			–	0.05549	0.3	Jensen-Shannon

2. **Detecting Data Shift**

   Input Data Monitoring: Mechanisms monitor and compare input data distributions against the training data distribution to detect signs of data drift. (Anomaly detection in DAG-1)

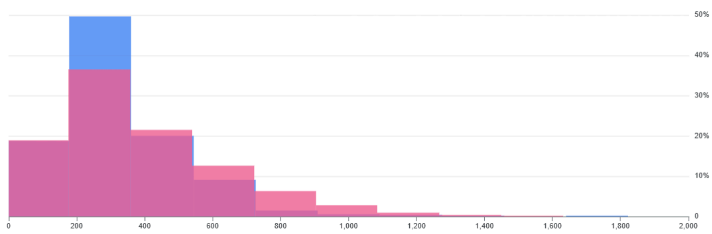
   Once the predictions are done, input data distribution is compared with the batch prediction distribution for important features to analyze feature distributions and identify potential data shifts

INPUT FEATURE DRIFT | OUTPUT PREDICTION DRIFT | FEATURE ATTRIBUTION DRIFT | CONFIGURATION

Run status	Run time	Target	Baseline
Succeeded	Dec 5, 2024, 6:30:01 PM	BigQuery table View details	GCS object View details

CurrentEquipmentDays: Target vs Baseline

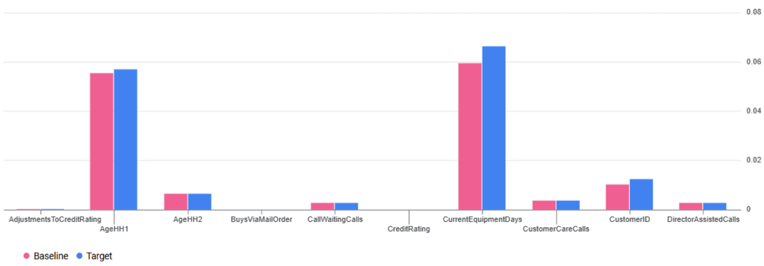
Snapshot of distribution job ran Dec 5, 2024, 6:30:00 PM



Statistics	Baseline	Target
Data points	51047	20015
Null	0%	0%
Mean	380.5444590279546	314.08463652260804
Min	-5	-5
Median	329	287
Max	1812	1823
Std Deviation	253.79720238700855	194.5371424855205
Zeros	0.04%	0.03%

Run status	Run time	Target	Baseline
Succeeded	Dec 5, 2024, 6:30:01 PM	BigQuery table View details	GCS object View details

Target vs Baseline - Feature attribution score



Features	Alerts	Score	Baseline score	Deviation	Threshold
AdjustmentsToCreditRating	–	0.00029	0.00031	-0.00002	0.3
AgeHH1	–	0.057	0.05538	0.00162	0.3
AgeHH2	–	0.00648	0.00665	-0.00017	0.3
BuysViaMailOrder	–	0	0		0.3
CallWaitingCalls	–	0.00269	0.00268	0.00001	0.3
CreditRating	–	0	0		0.3
CurrentEquipmentDays	–	0.06634	0.05942	0.00692	0.3

3. **Threshold for Triggering Retraining**
   Our architecture supports on-demand retraining, triggered by the addition of new data to the 'training data' bucket. The decision to include new data for retraining is informed by schema validation and anomaly detection tasks, which are part of the DAGs processing the incoming data. If excessive anomalies or data drift are detected, we prepare new training data and place it in the designated bucket. This action automatically triggers the training DAG, initiating the retraining process. By centralizing operations on GCP with storage buckets, Cloud Composer, and Vertex AI, our system maintains streamlined and automated workflows, eliminating the need for traditional CI/CD pipelines.

4. **Automating the Retraining Pipeline (CI/CD)**
   We determined that integrating with Git for CI/CD workflows related to code changes is unnecessary for our project as it is fully deployed on Google Cloud Platform (GCP), leveraging Vertex AI's AutoML for model training and predictions. Since Vertex AI AutoML handles the modeling process, no custom model code is required. We have implemented automatic triggers based on incoming data, ensuring that Dataflow pipelines and directed acyclic graphs (DAGs) in Cloud Composer are executed whenever new data is added to the GCP storage buckets. The logic for these DAGs is predefined and fixed, automating the entire workflow without requiring code modifications