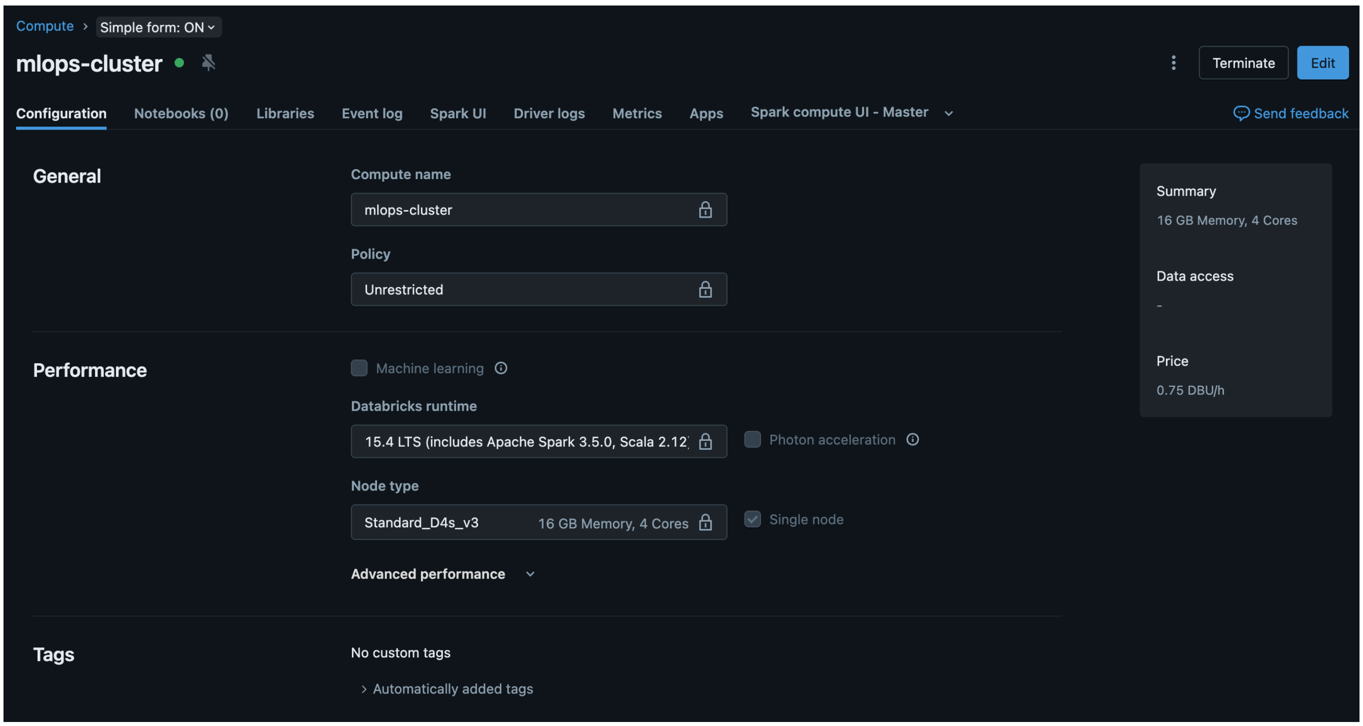
# Databricks. Integration and Workflow.

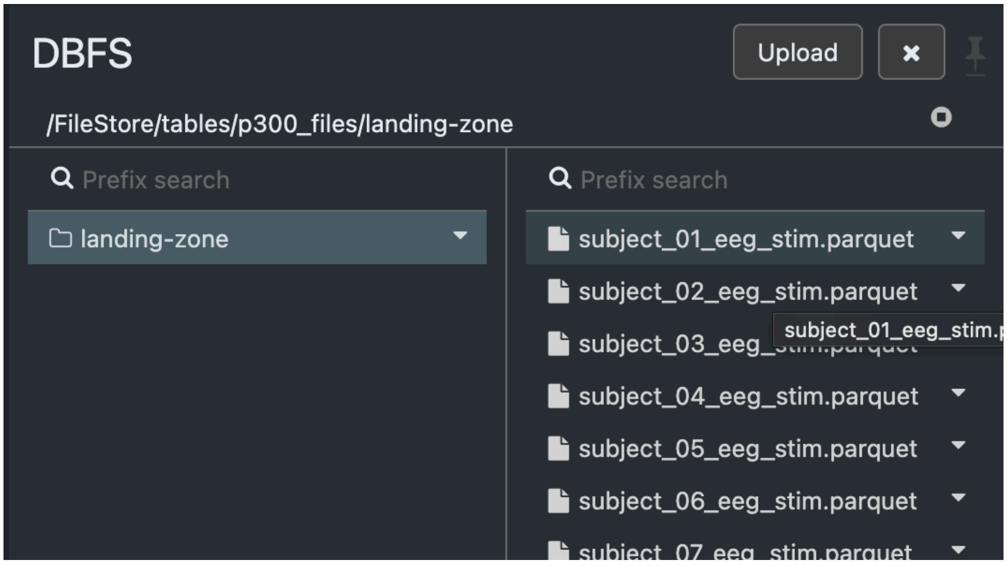
This report documents the execution of the P300 classification pipeline on the Databricks platform. The workflow demonstrates data ingestion, preprocessing, model training, a champion-challenger deployment strategy, and automated model evaluation.

## Initial setup

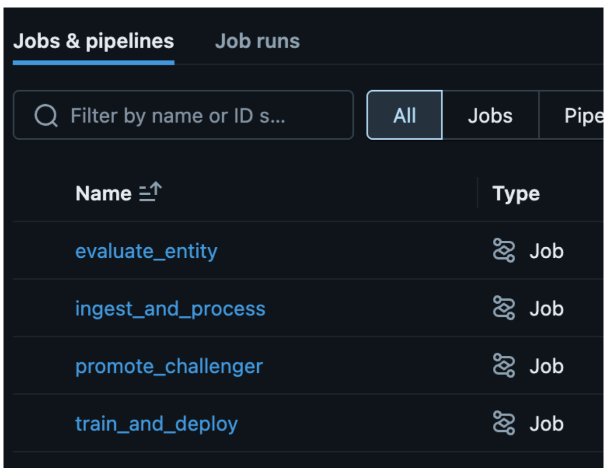
The foundation of the workflow is a Databricks All-Purpose Compute cluster. This managed environment provides the necessary computational resources for all notebook-based tasks, replacing the local CPU and Docker environment.



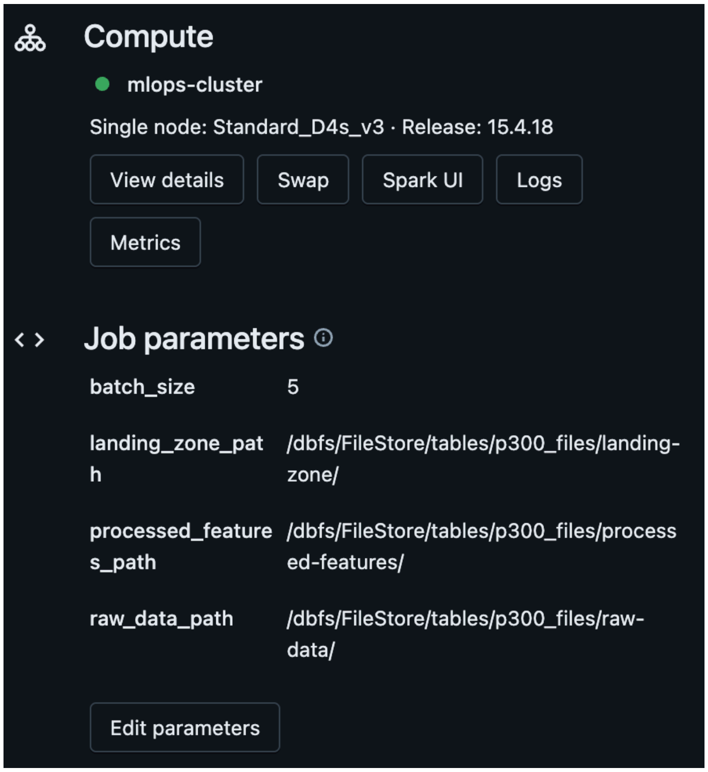

Data is managed using the Databricks File System (DBFS). The workflow begins with raw subject data uploaded to a landing-zone directory within DBFS. It is worth noting that due to permissions on the university-provided Azure account, I was unable to use the modern Unity Catalog Volumes. As a result, this project uses the legacy DBFS for file management, which functions similarly to the MinIO object storage used in the local Docker setup.



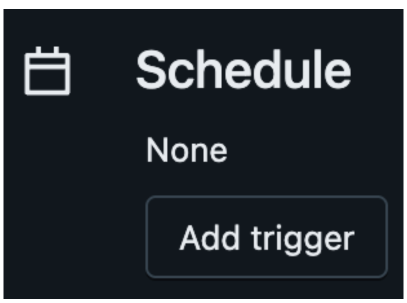

The entire pipeline is orchestrated using Databricks Workflows (Jobs). Separate jobs are configured for data preparation, model training/deployment, evaluation, and promotion. These jobs execute a series of parameterized notebooks, passing configuration like data paths and model names, which makes the pipeline modular and configurable without code changes.




Example of job configuration with attached compute and job parameters. They are like environmental variables for notebook-based tasks.



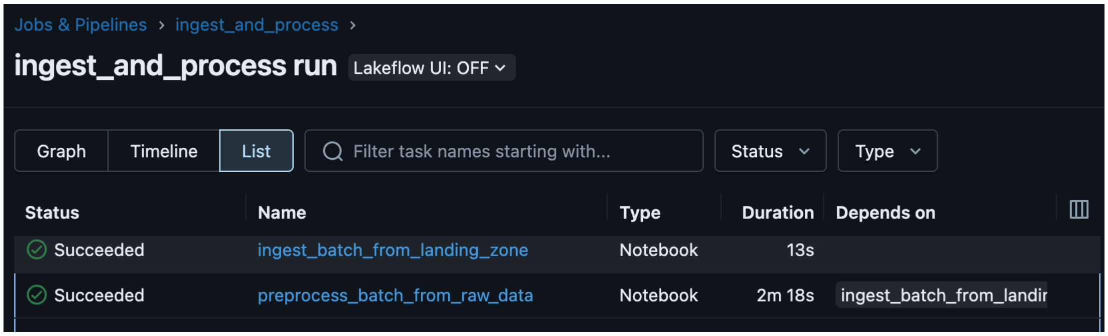

We can also schedule job run with the corresponding section allowing continuous pipeline execution:



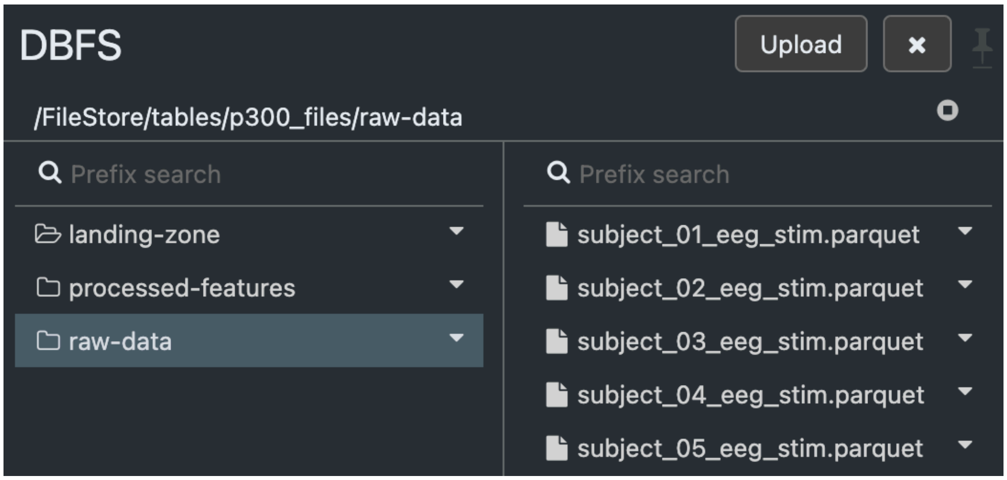

# 1. Data ingestion and preprocessing

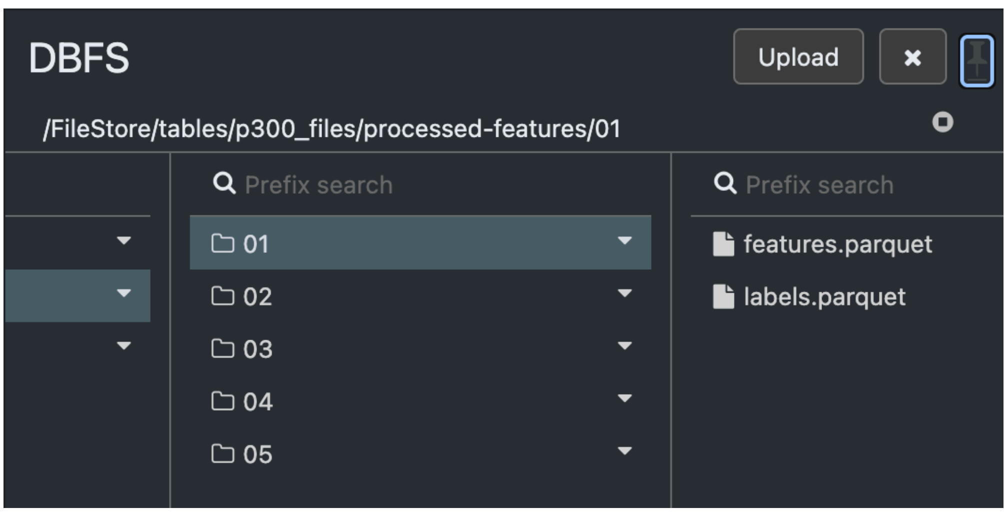
The `ingest_and_process` job is executed first. This job consists of two sequential tasks:

1. `ingest_batch_from_landing_zone` moves a batch of raw data files from the landing zone to the raw-data directory.
2. `preprocess_batch_from_raw_data` scans the raw-data directory, identifies new subjects not present in processed-features, and triggers a separate notebook run for each new subject to perform filtering, epoching, and feature extraction.




Upon completion, the DBFS directories are populated, with the final features ready for model training in processed-features.

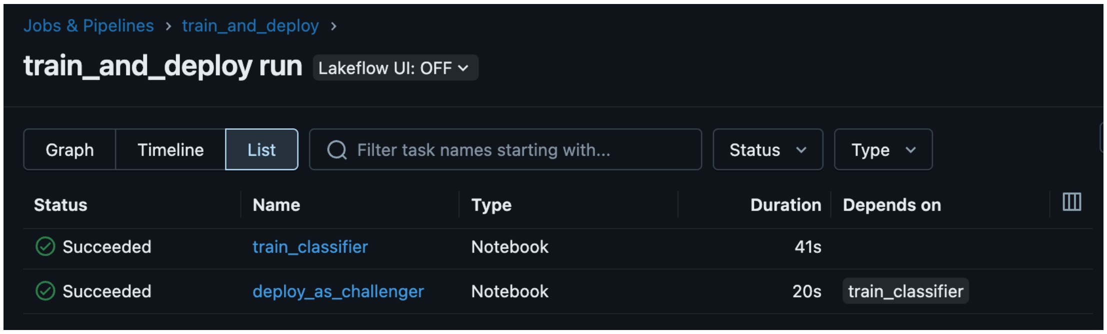


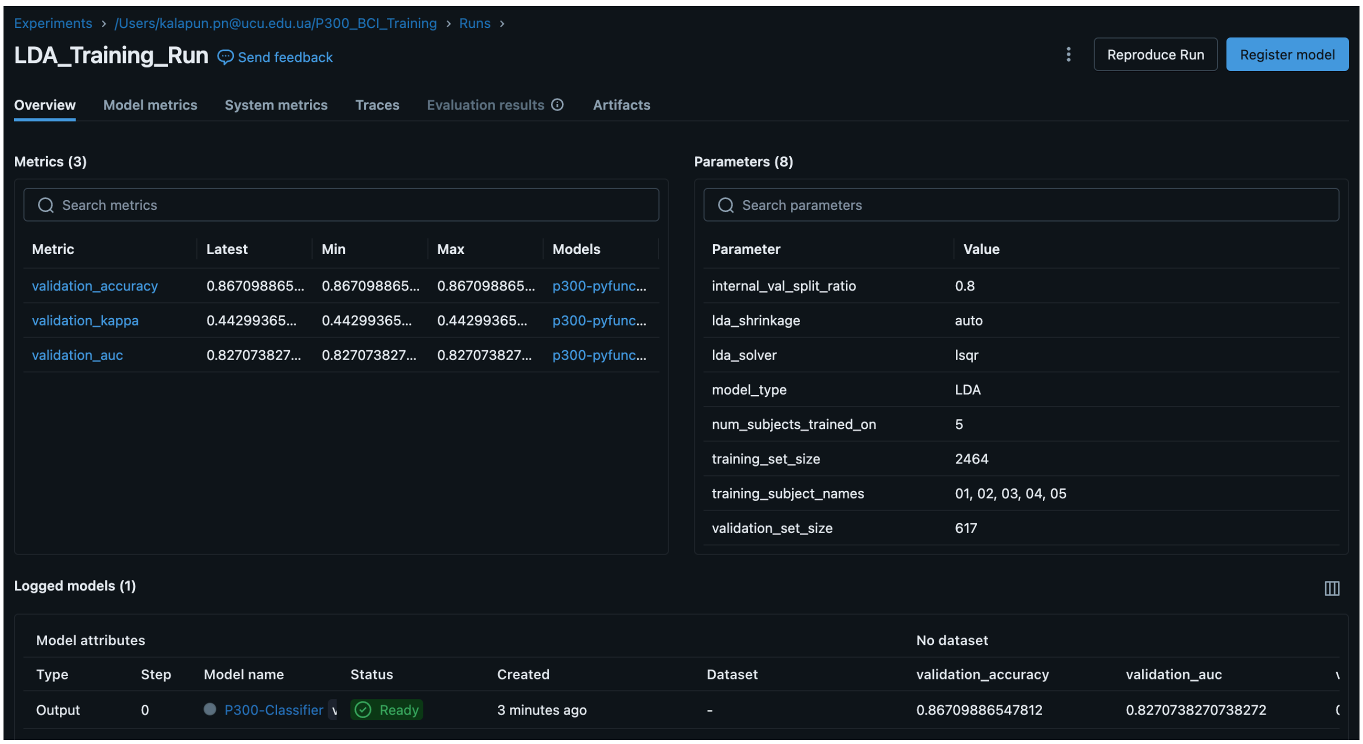

# 2. Training and deploying the first model

The `train_and_deploy` job orchestrates model creation and initial deployment. It consists of two tasks: `train_classifier` and `deploy_as_challenger`.
The training notebook retrains the model from scratch using all available processed data. For the current dataset size and the choice of LinearDiscriminantAnalysis (which does not support

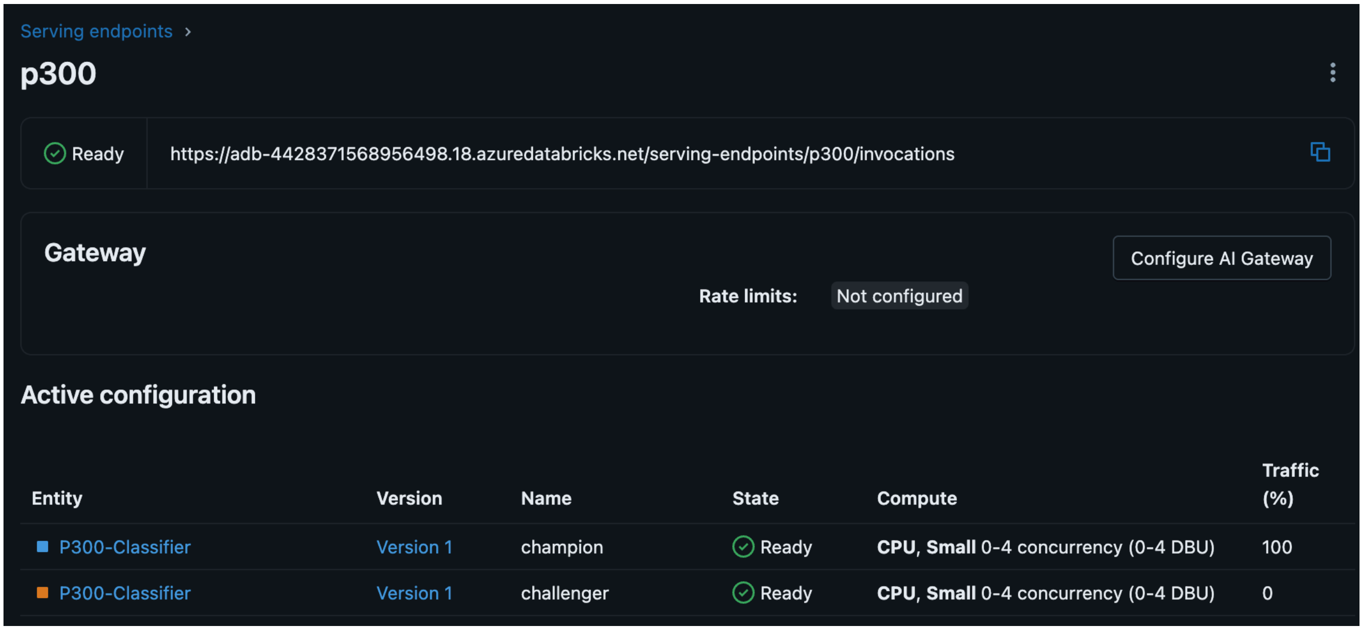
incremental learning), this approach is computationally feasible.




The run logs parameters, metrics, and registers the packaged model and scaler to the MLflow Model Registry. A key feature is that the training run also logs a list of subject names it trained on as an artifact ( `training_subjects.json` ). This ensures that evaluation further on can be performed on a completely unseen holdout set, preventing data leakage.

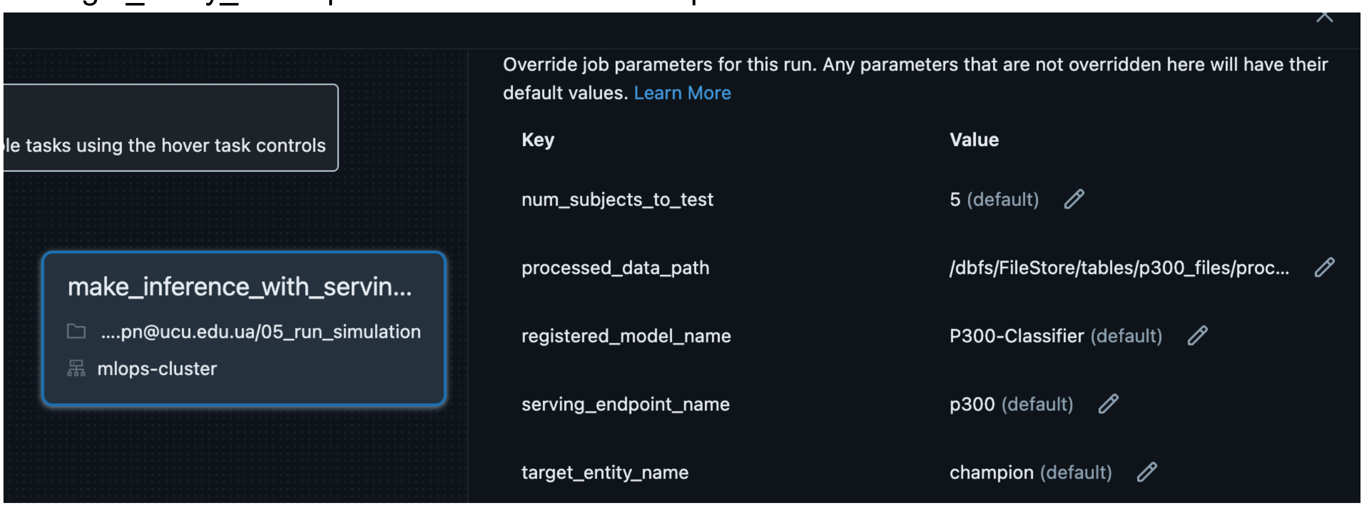



The deploy_as_challenger task then programmatically creates the p300 Model Serving Endpoint if it does not exist, or updates it if it does. On the first run, it seeds the endpoint by assigning the newly trained model (Version 1) to both the champion and challenger slots. Traffic is configured to send 100% of requests to the champion entity by default.



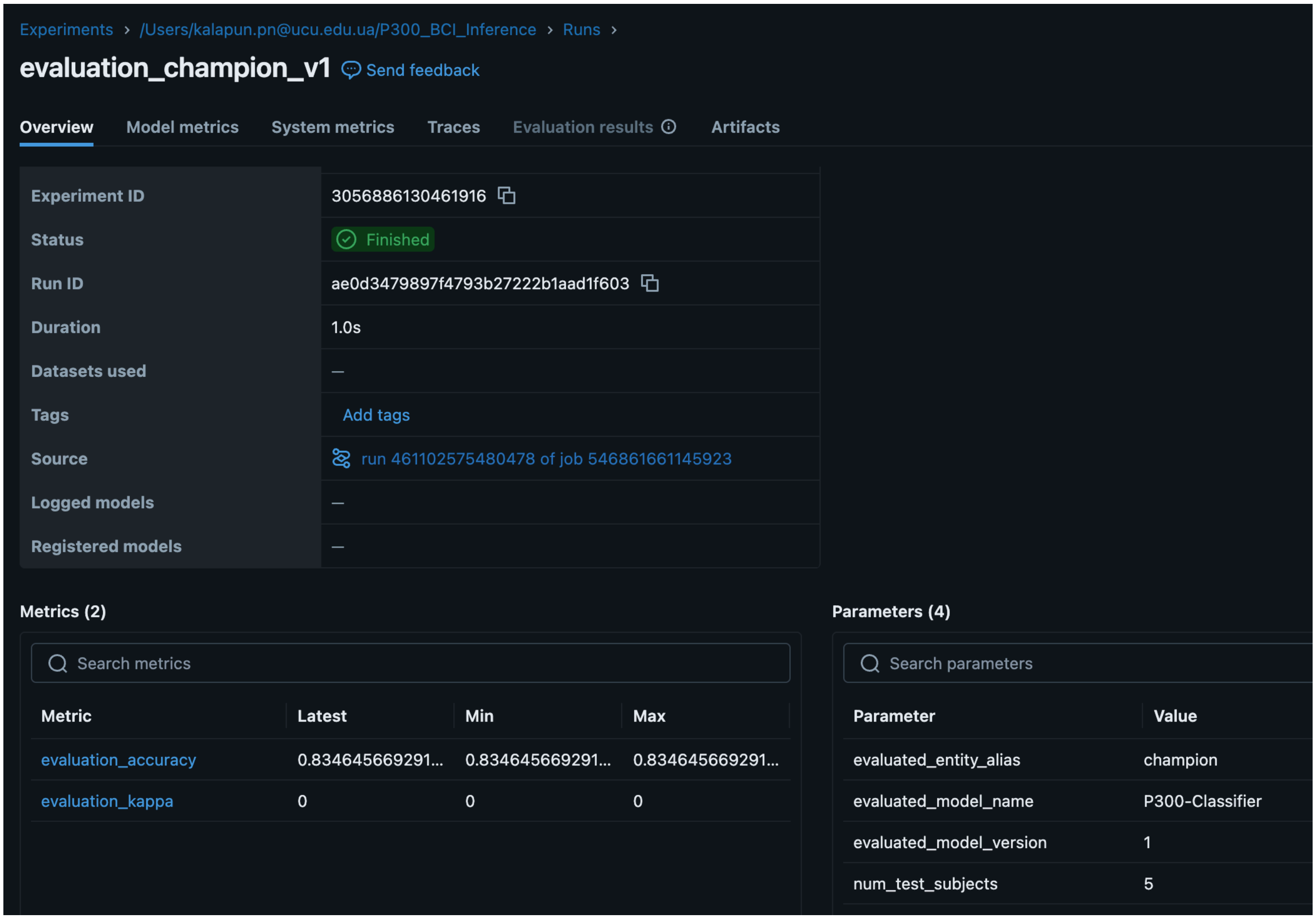

# 3. Evaluating the champion model

After ingesting more data (+5 subjects), the evaluate_entity job is run. By default, its target_entity_name parameter is set to "champion".




The simulation notebook first queries the serving endpoint's API to confirm which model version the champion is serving. It then fetches that version's `training_subjects.json` artifact to create a holdout set of unseen subjects.

The final metrics are logged to a new MLflow run.




## Notes on evaluation

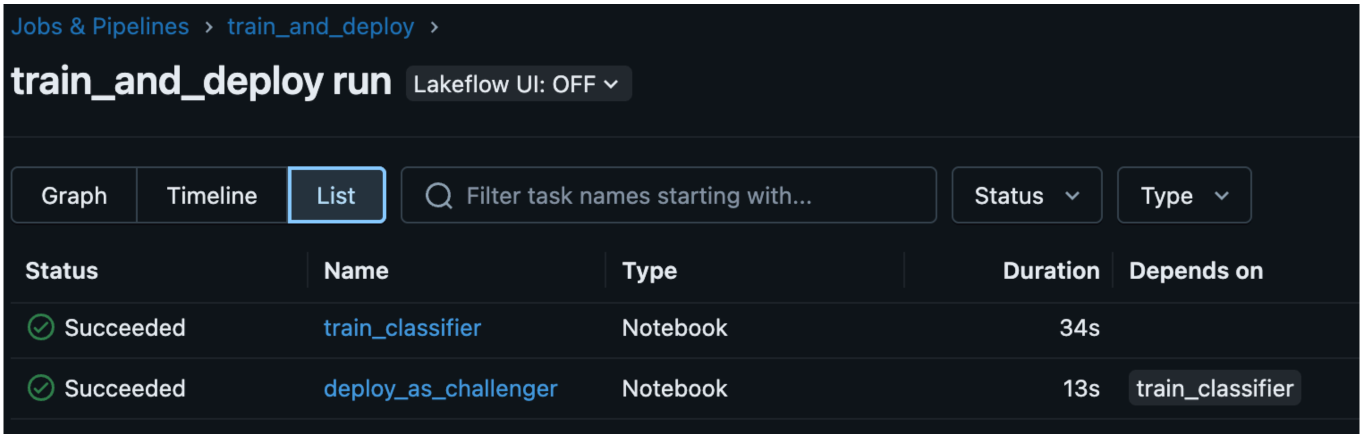
1. A key part of this evaluation is ensuring requests are sent to the correct model. While aliases are a Unity Catalog feature and not available in my workspace, I discovered through documentation that it is still possible to query a specific served entity directly.

   The simulation notebook constructs a direct invocation URL like `/serving-endpoints/{endpoint-name}/served-models/{entity-name}/invocations`. By setting entity-name to "`champion`", it bypasses the main traffic split and guarantees that only the champion model is evaluated.

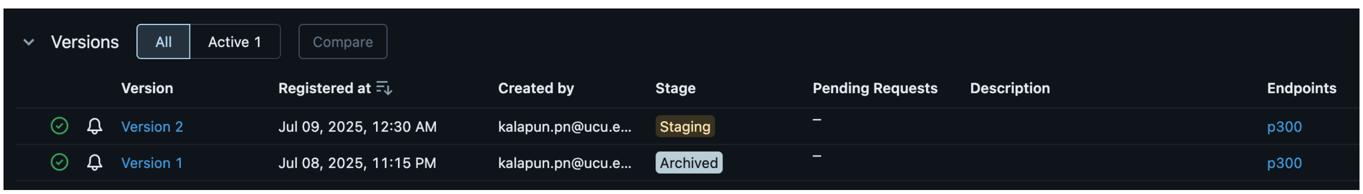
   The notebook first queries the endpoint's management API to confirm which model version the champion entity is serving, then fetches that version's `training_subjects.json` artifact to create a holdout set of unseen subjects.
2. During the simulation, I encountered `SSLError` issues, which indicated the serving endpoint was crashing. This was resolved by implementing mini-batching in the request loop, sending a smaller number of records per API call to fit within the resource limits of the "Small" compute size of the serving endpoint.

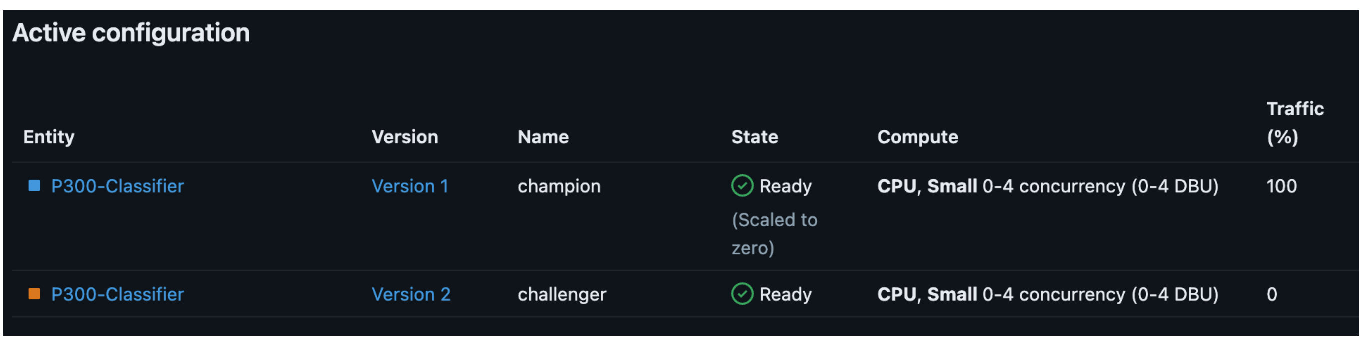
# 4. Training and deploying a new challenger

The `train_and_deploy` job is run again.




This creates a new model (Version 2), transitions it to the Staging stage in the legacy Model Registry, and the `deploy_as_challenger` task updates the serving endpoint to point its challenger entity to this new Version 2.

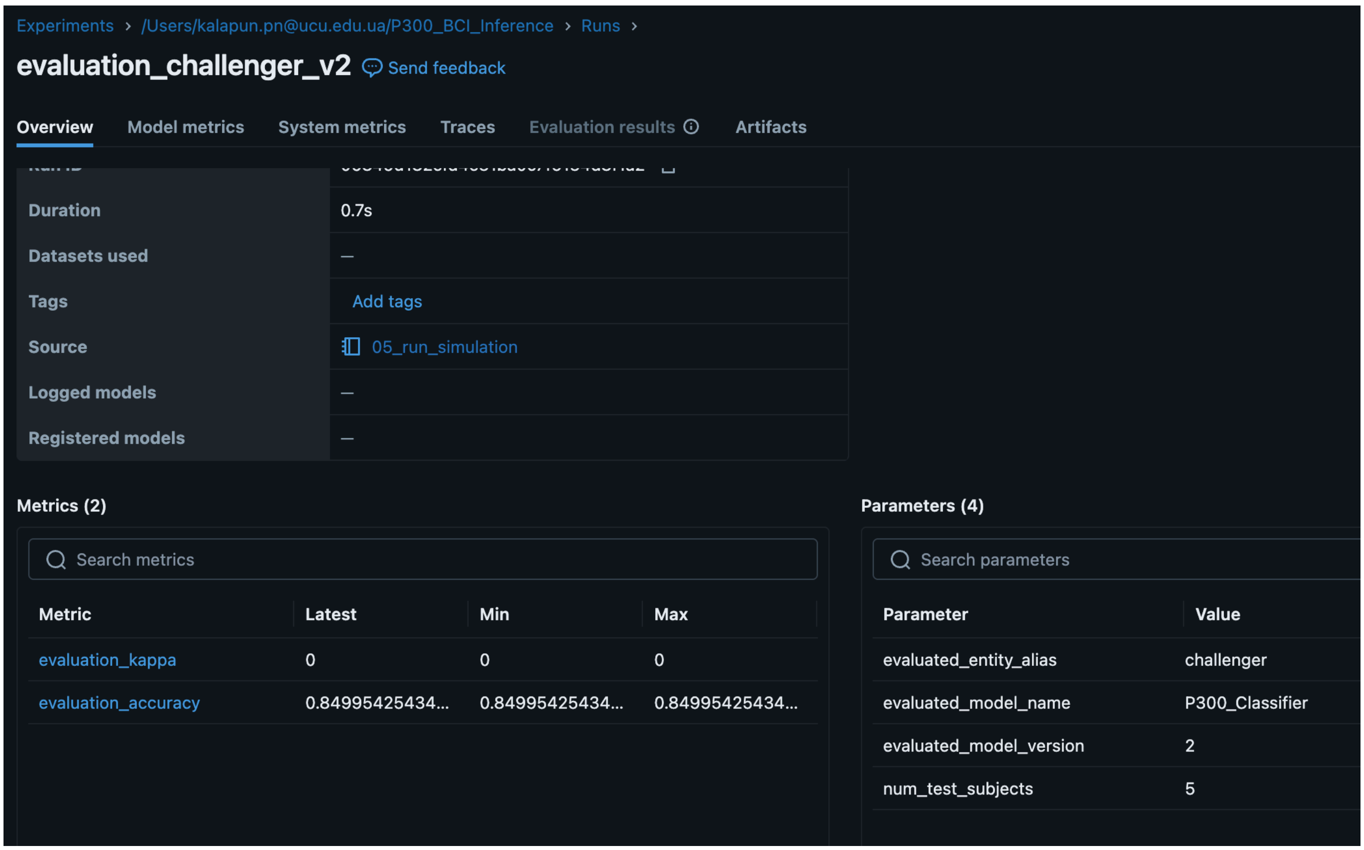


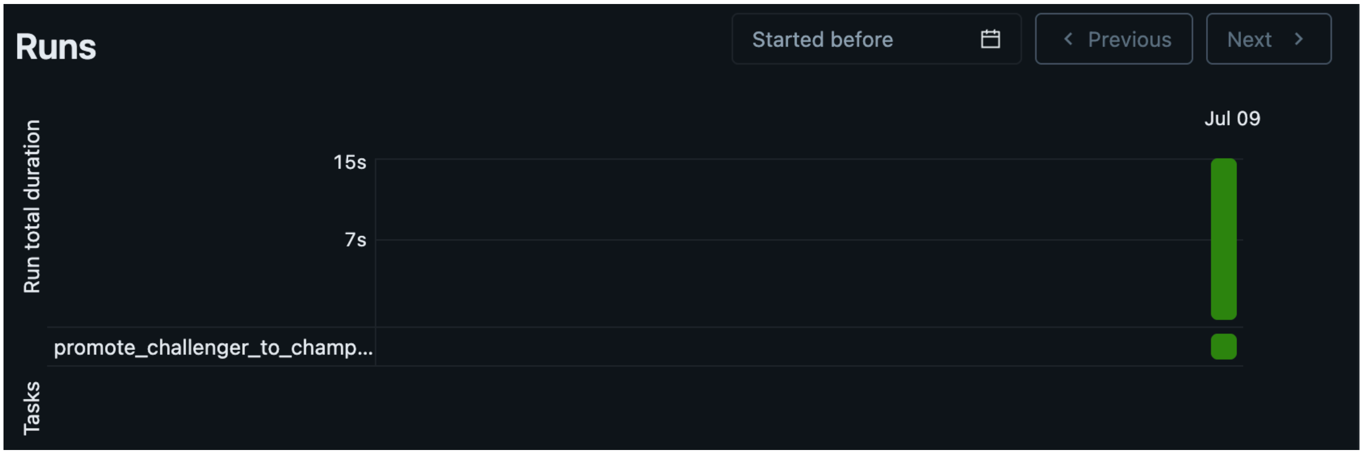

# 5. Evaluating the new challenger and promotion

The `evaluate_entity` job is run again, but this time the `target_entity_name` parameter is explicitly set to `"challenger"`.

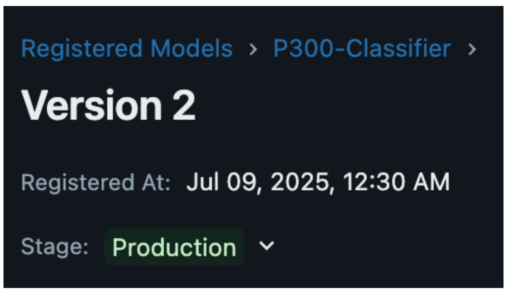

The evaluation metrics show that Version 2 performs slightly better than the current champion.

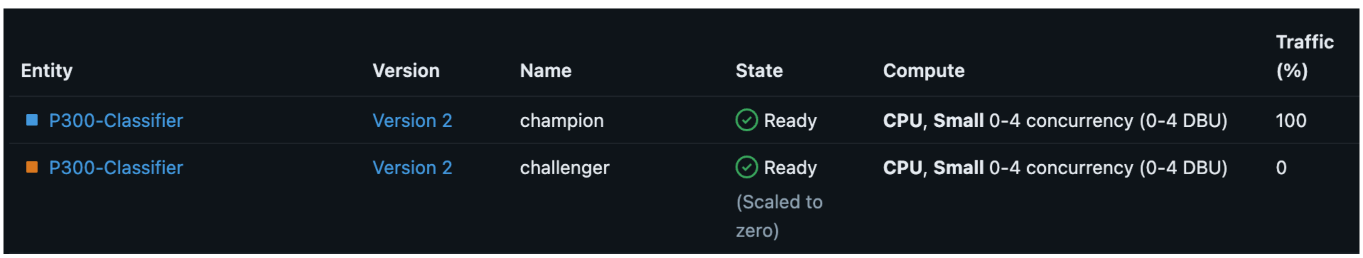


Based on these results, we run the `promote_challenger` job.




This completes the automated promotion cycle, demonstrating a full champion-challenger workflow on the Databricks platform



| Entity | Version | Name | State | Compute | Traffic (%) |
|---|---|---|---|---|---|
| P300-Classifier | Version 2 | champion | Ready | **CPU**, **Small** 0-4 concurrency (0-4 DBU) | 100 |
| P300-Classifier | Version 2 | challenger | Ready (Scaled to zero) | **CPU**, **Small** 0-4 concurrency (0-4 DBU) | 0 |