Using the power of Apache Airflow and Ray for Scalable AI deployments

Venkata Jagannath, Sr ML Engineer, Astronomer

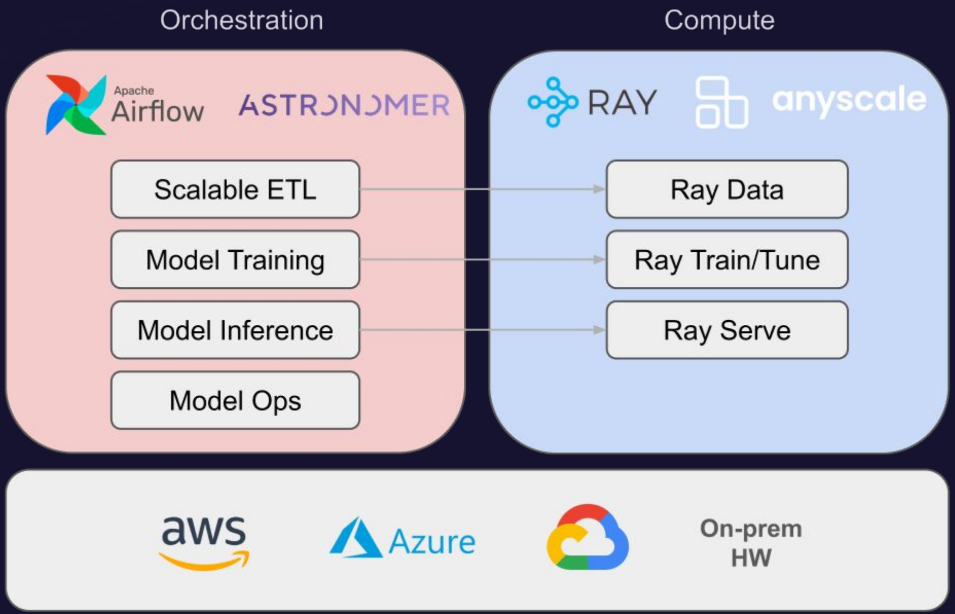
Marwan Sarieddine, AI Engineer, Anyscale

Agenda

- Why Airflow + Ray?
- Architecture
- Continuous data update
- Continuous model update
- Ray/Airflow integration deep dive
- Key takeaways

Why Airflow + Ray?

- Orchestration
  - Manage Data and AI/ML workflows
  - Manage Infrastructure through on-demand scaling
  - Data and Time driven scheduling
- New AI use cases
  - Scalable Python
  - Batch Inference
  - Continuous fine-tuning
  - Rollout real-time deployments



Apache Airflow

The standard for data pipelines in a cloud-native world

25M
Monthly Downloads

3K
Contributors

36K
GitHub Stars

53K
Slack Community

ASTRONOMER

The driving force behind Apache Airflow

5 offices | 237 employees | 24×7 worldwide support

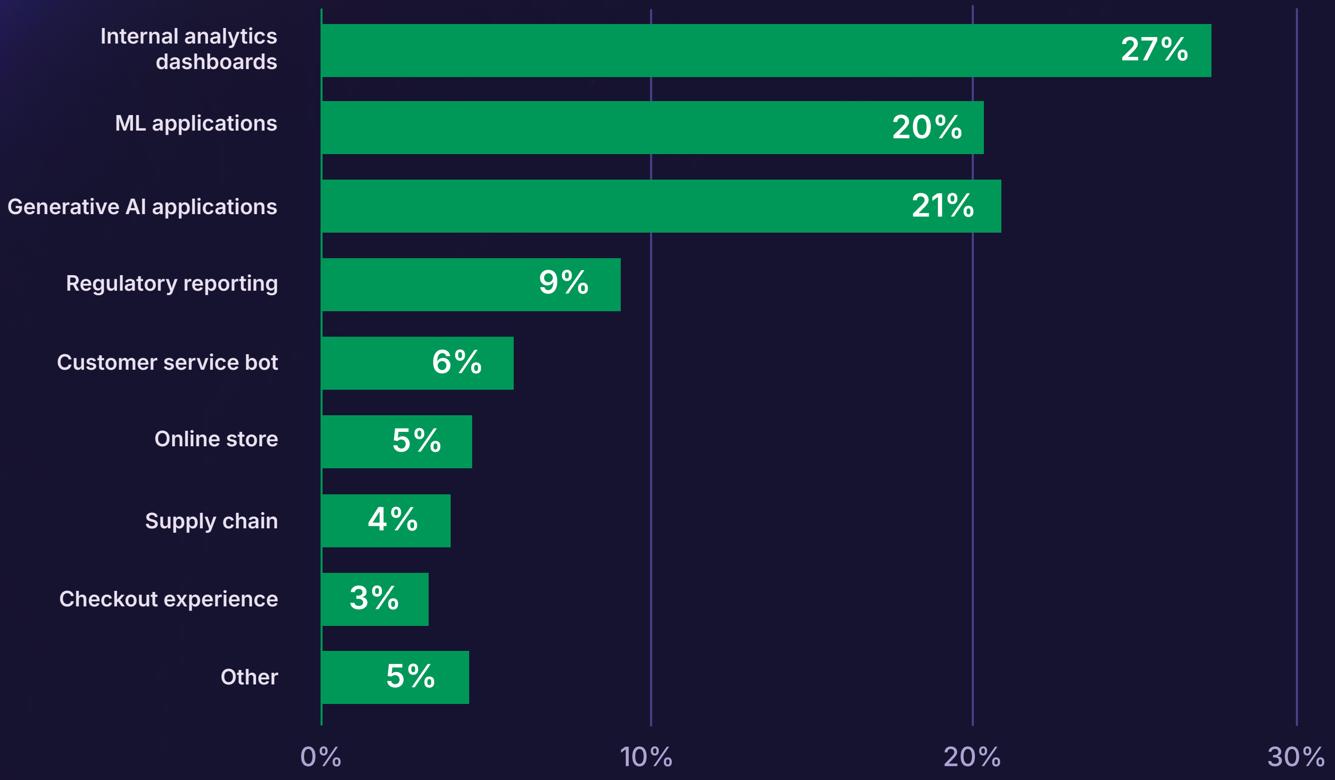
100%
Drives 100% of Airflow releases

55%
Of Airflow code contributed

18 of 25
18 of the top 25 committers on board, 8 PMC members

40K+
40K+ Airflow students in Academy ecosystem

What does, or could, Airflow drive for your organization?



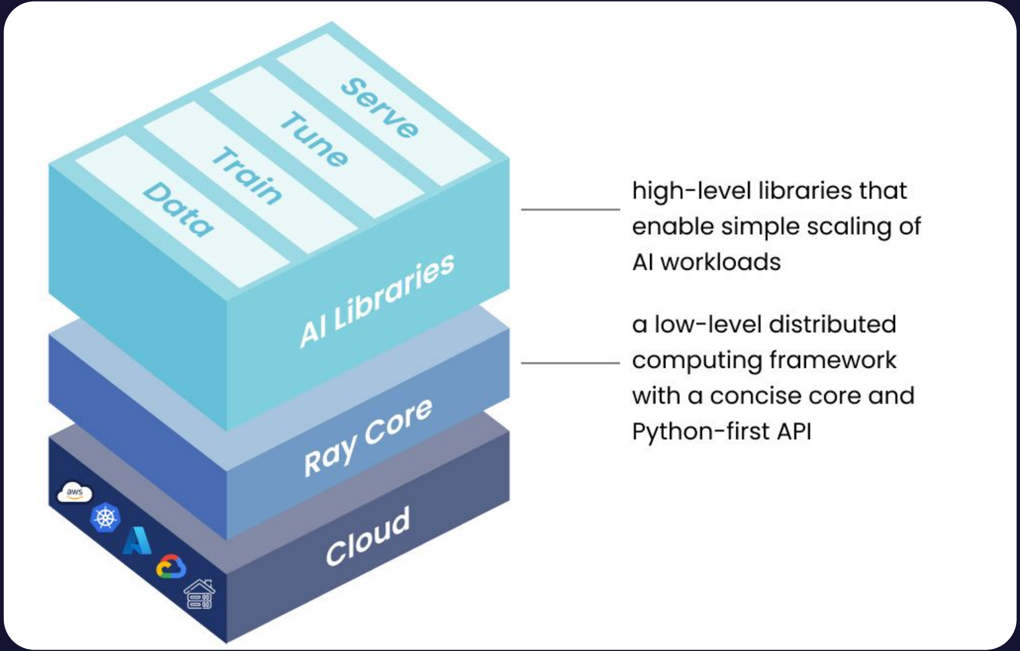
Source: 2023 Gatepoint Research, n=281

24%

AI and ML usage facilitated by Airflow spiked by **24%** year over year.

Why Ray?

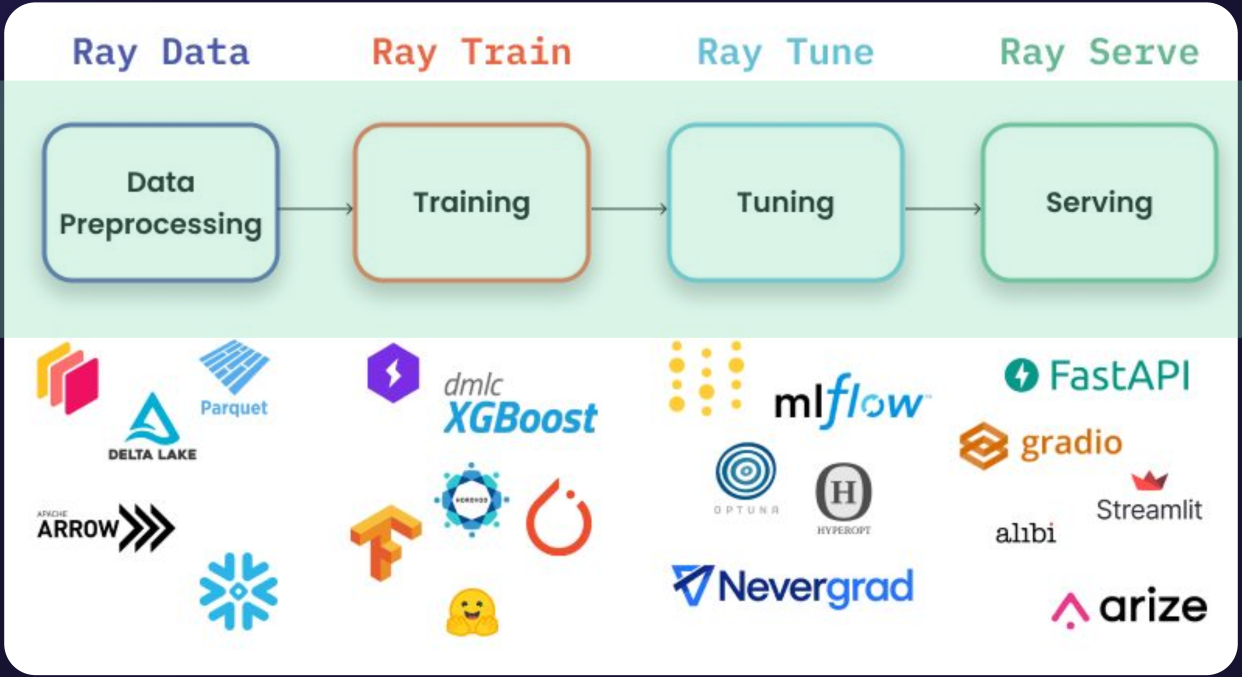
Ray is a **highly scalable** distributed compute framework



high-level libraries that enable simple scaling of AI workloads

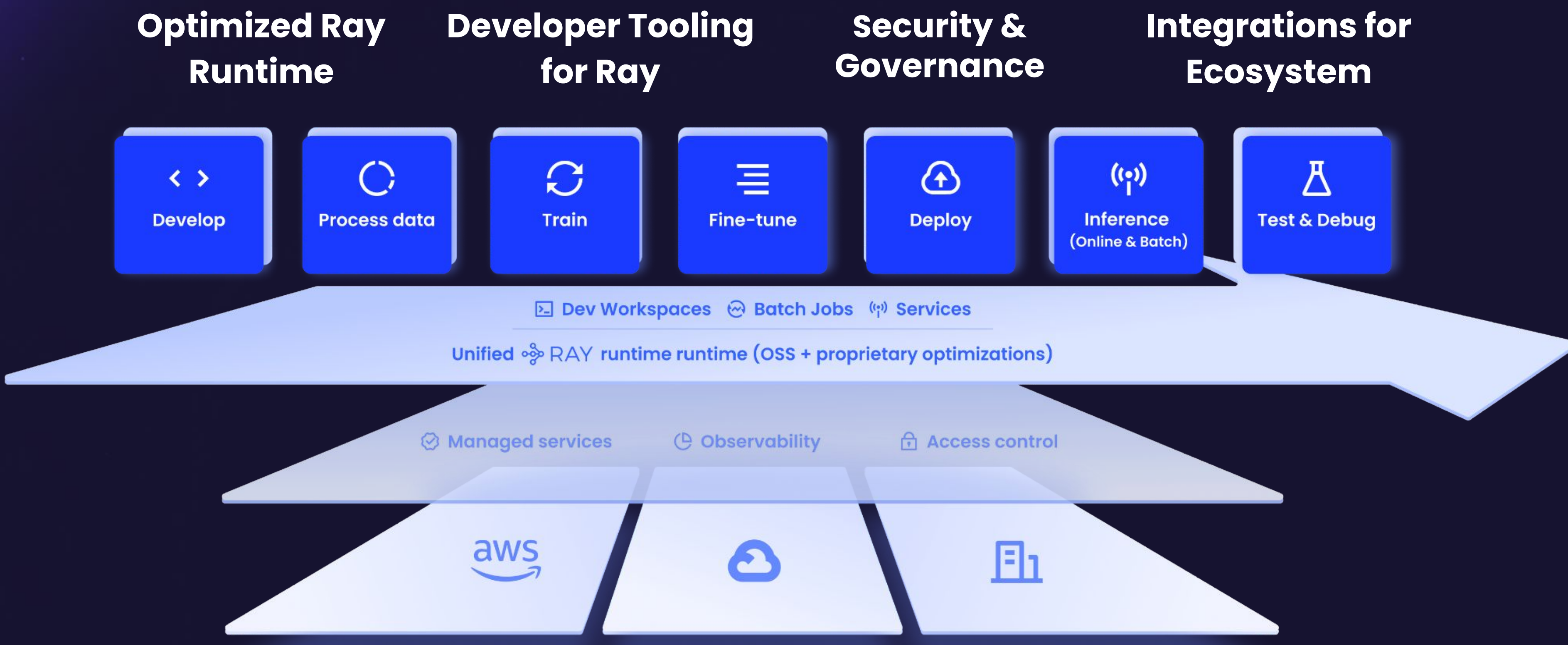
a low-level distributed computing framework with a concise core and Python-first API

End-to-End MLOps Scaling



High-level libraries that make scaling easy for both data scientists and ML engineers.

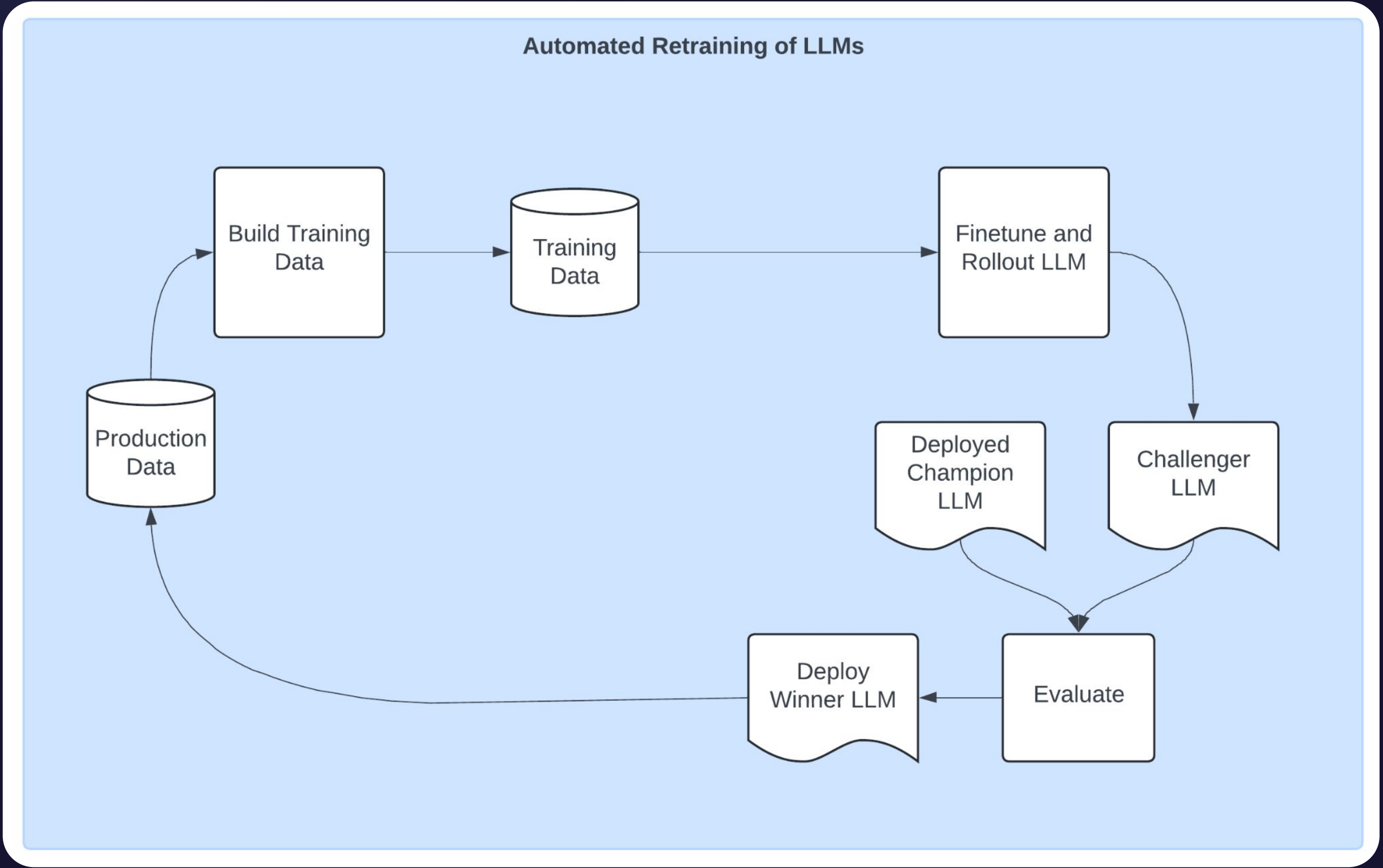
Anyscale: End-to-end AI Platform



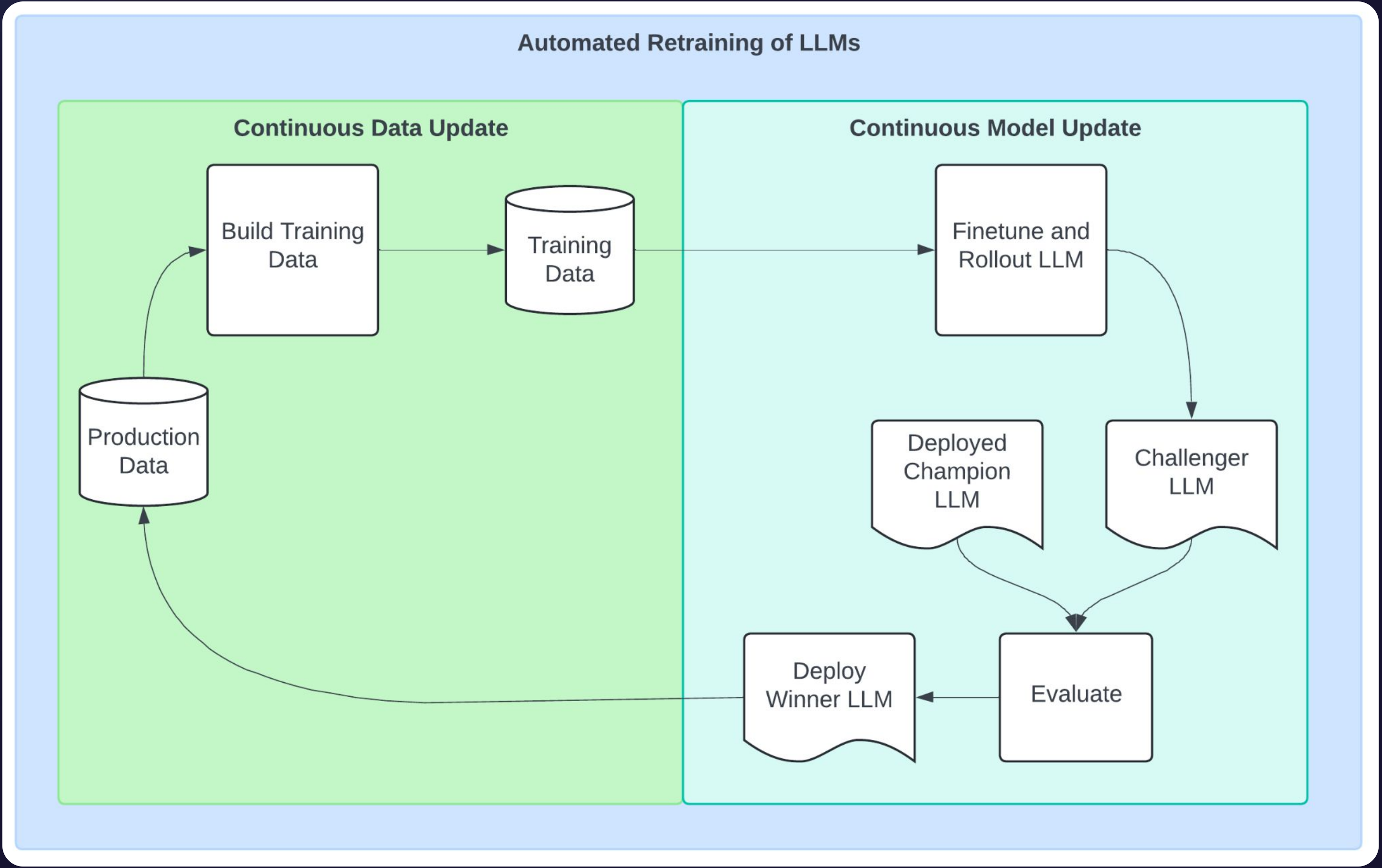
Example: Processing User Feedback

- Users provide feedback on online video games
- Fine-tuned LLMs are used to categorize the feedback by product, platform, etc..
- Product managers receive a summary of feedback relevant for their product
- LLMs are updated to keep track of the latest trends and product releases

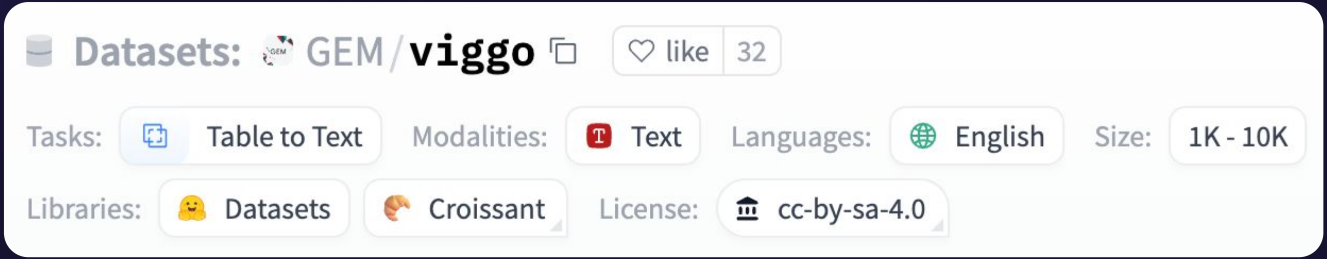
Architecture - 30k ft view



Architecture - 30k ft view



Dataset



Datasets: GEM/viggo ♡ like 32

Tasks: Table to Text Modalities: Text Languages: English Size: 1K - 10K

Libraries: Datasets Croissant License: cc-by-sa-4.0

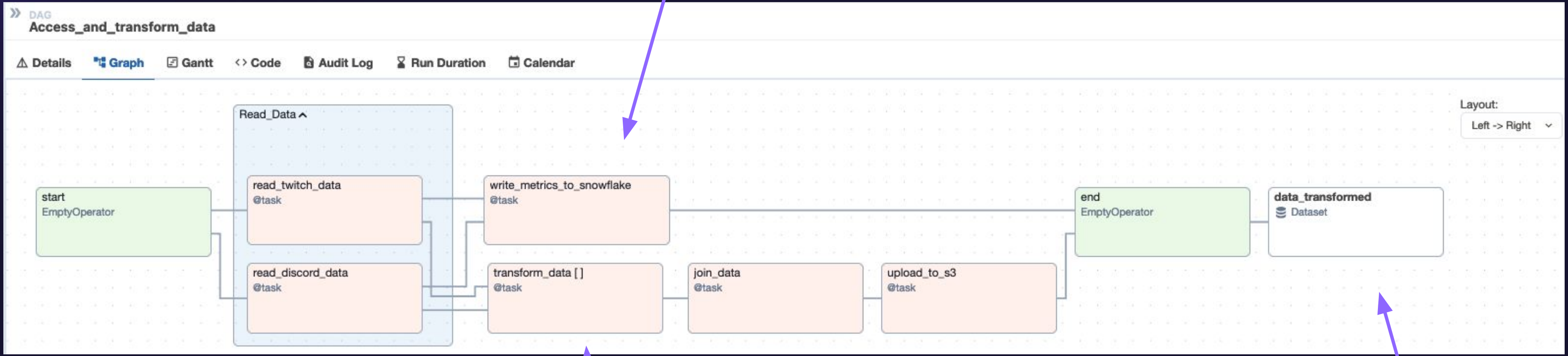
- Example:
- Input: “What is it about games released in 2005 that makes you think it's such a fantastic year for video games?"
- Output: request_explanation(release_year[2005], rating[excellent])

Fine-tuning model specifics

- **Model:** mistralai/Mistral-7B-Instruct-v0.1
- **Technique:** Low Rank Adaption (LoRA)
- **Evaluation**
  - Metric: accuracy
- **Baseline**
  - mistralai/Mistral-7B-Instruct-v0.1 + few-shot with n=20

Airflow DAG (1/2) - Continuous Data Update

Write to DW for dashboards

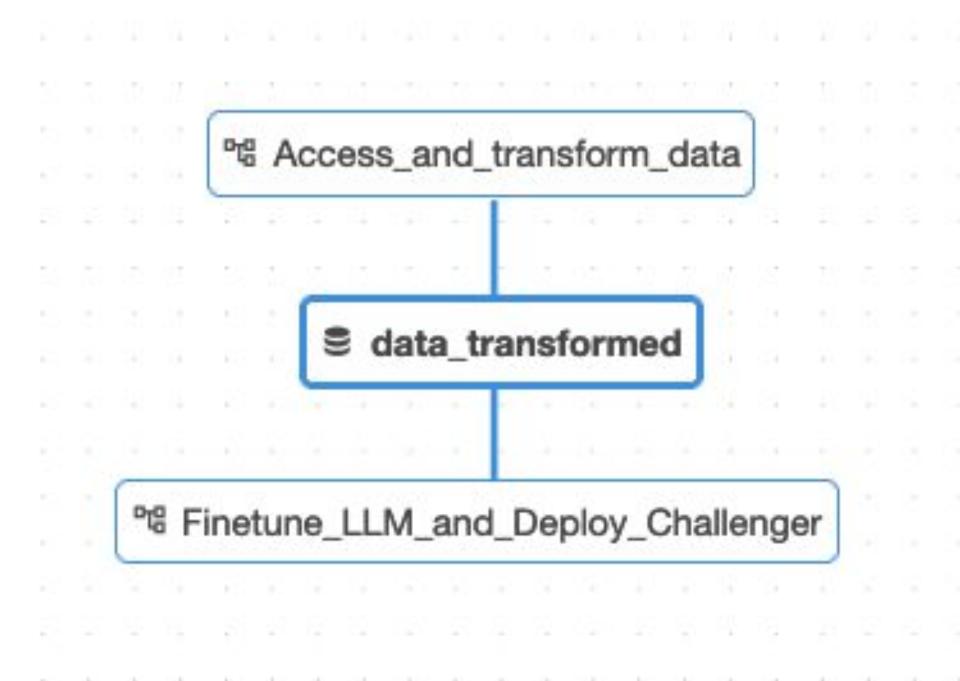


TaskGroup to read from multiple sources

Dynamic Task Mapping to transform each data source

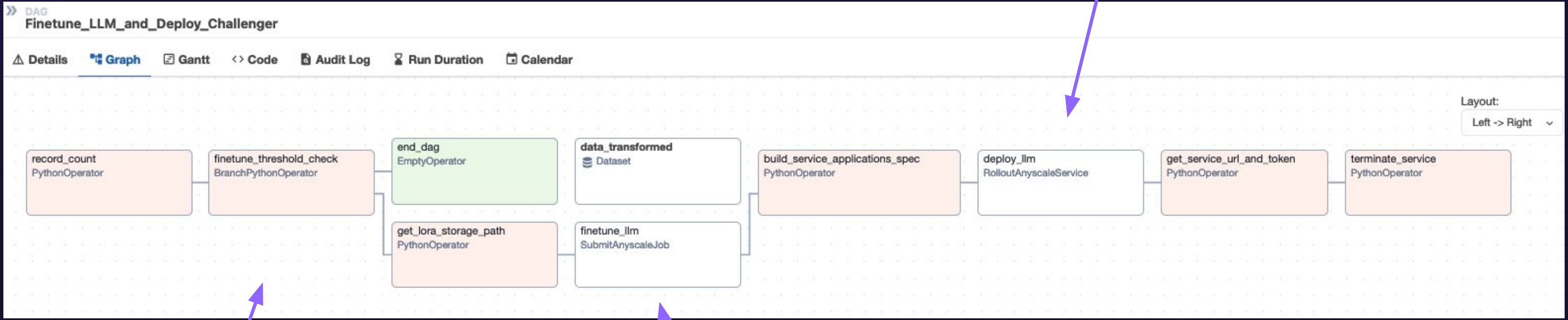
Update Airflow Dataset

Integrating 2 DAGs – Data-Driven Scheduling



Airflow DAG (2/2) – Continuous Model Update

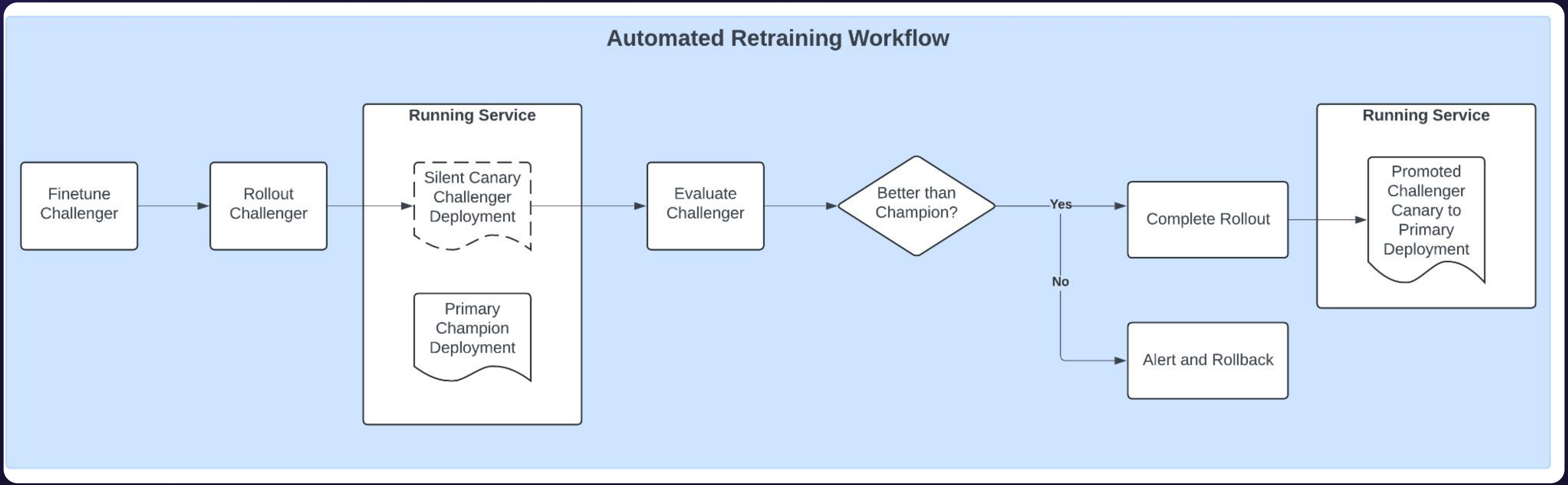
Deploy fine-tuned model



If record count >= 200, start fine-tuning

Start fine-tuning job

Model Fine-Tuning & Update



Automated Retraining Workflow

Finetune Challenger

Rollout Challenger

Running Service

Silent Canary Challenger Deployment

Primary Champion Deployment

Evaluate Challenger

Better than Champion?

Yes

No

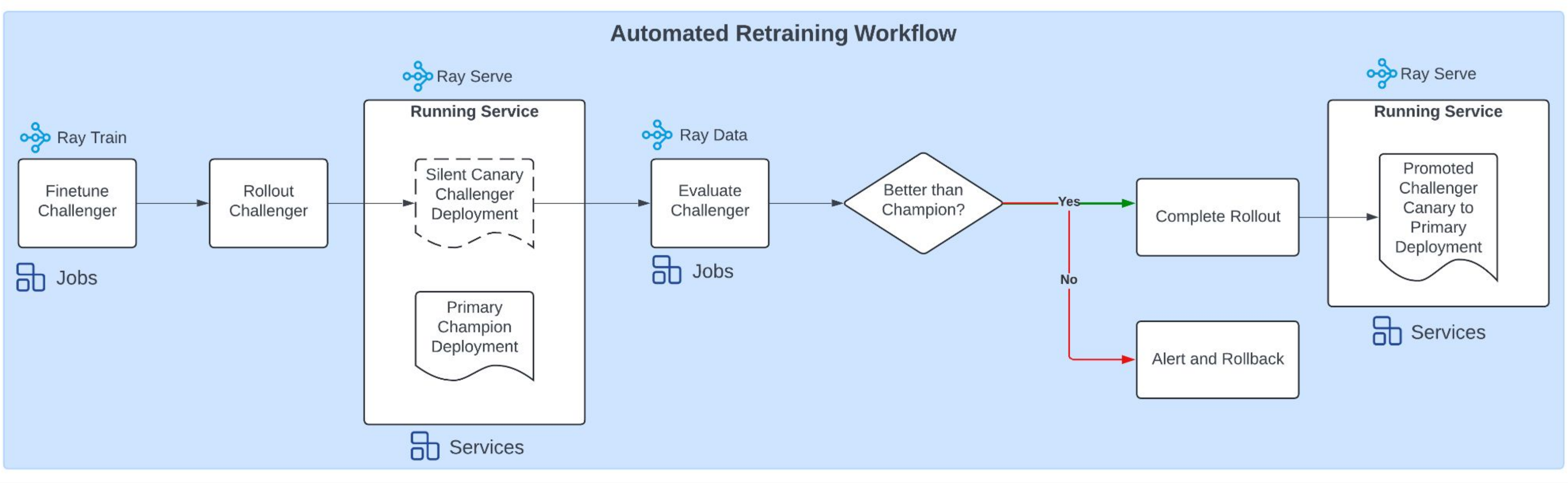
Complete Rollout

Alert and Rollback

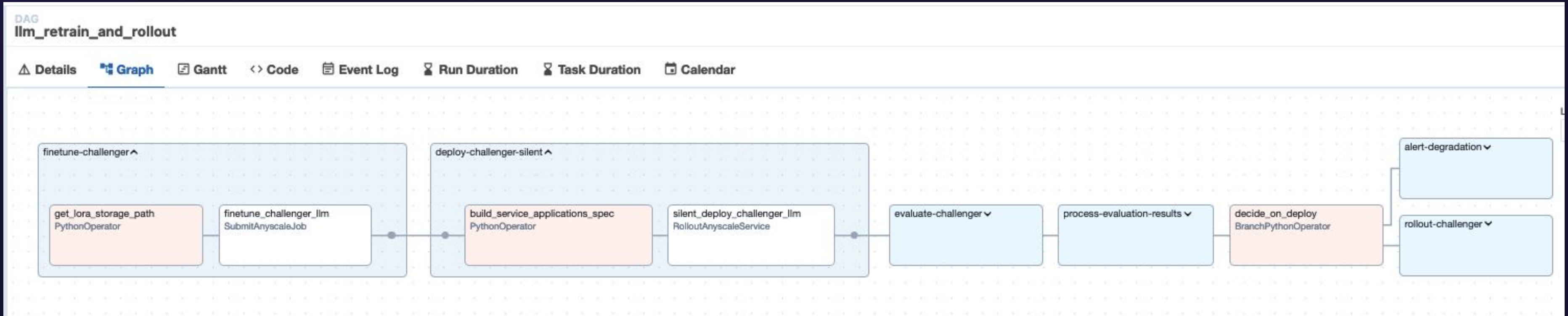
Running Service

Promoted Challenger Canary to Primary Deployment

Model Fine-Tuning & Update



Airflow DAG - Model Fine-Tuning & Update



Integration spotlight: Airflow & Ray

Apache Airflow

ASTRONOMER

+

RAY

anyscale

Easily Run **Ray/Anyscale** Jobs or Services from Airflow DAGs.

Ray Provider

Users can offload batch AI jobs using the Ray operators

```
from ray_provider.operators.ray import SubmitRayJob

RAY_SPEC = './dags/scripts/ray.yaml'

RAY_RUNTIME_ENV={“pip”=[“numpy”],
                 "working_dir": './dags/ray_scripts'}

SubmitRayJob(task_id="SubmitRayJob",
             conn_id=CONN_ID,
             entrypoint='python script.py',
             runtime_env=RAY_RUNTIME_ENV,
             num_cpus=1,
             num_gpus=0,
             xcom_task_key="SubmitRayJob.dashboard",
             ray_cluster_yaml=RAY_SPEC,
             wait_for_completion=True,
             job_timeout_seconds = 600,
             poll_interval=5,
             dag = dag,)
```

```
 SetupRayCluster(...), DeleteRayCluster(...)
```

Ray Provider

Users can offload batch AI jobs using the Ray operators

```
from ray_provider.operators.ray import SubmitRayJob

RAY_SPEC = './dags/scripts/ray.yaml'

RAY_RUNTIME_ENV={“pip”=[“numpy”],
                 "working_dir": './dags/ray_scripts'}

SubmitRayJob(task_id="SubmitRayJob",
             conn_id=CONN_ID,
             entrypoint='python script.py',
             runtime_env=RAY_RUNTIME_ENV,
             num_cpus=1,
             num_gpus=0,
             xcom_task_key="SubmitRayJob.dashboard",
             ray_cluster_yaml=RAY_SPEC,
             wait_for_completion=True,
             job_timeout_seconds = 600,
             poll_interval=5,
             dag = dag,)



 SetupRayCluster(...), DeleteRayCluster(...)
```

```
from ray_provider.decorators.ray import ray

RAY_SPEC = './dags/scripts/ray.yaml'

RAY_TASK_CONFIG = {
   'conn_id': CONN_ID,
   'runtime_env': { "working_dir": './dags/ray_scripts',
                    "pip": ["numpy"]},
   'num_cpus': 1,
   'num_gpus': 0,
   'ray_cluster_yaml': RAY_SPEC,
   'xcom_task_key': "dashboard"
}

@ray.task(config=RAY_TASK_CONFIG)
def sample_script(data):

    import ray

    @ray.remote
    def hello_world():
        return "Hello, World!"

    ray.init()
    result = ray.get(hello_world.remote())
    print(result)
```

Anyscale Provider

Users can offload batch AI jobs using the Anyscale operators

```
from anyscale_provider.operators.anyscale import SubmitAnyscaleJob


# https://docs.anyscale.com/reference/job-api/#job-models
job_config = dict(
        entrypoint=”python finetune.py …”,
        … )

SubmitAnyscaleJob(task_id="llm-finetune",
                  conn_id=ANYSCALE_CONN_ID,
                  name=”llm-finetune”,
                  image_uri="anyscale/ray:2.23.0-py311",
                  compute_config="my-compute-config:1",
                  entrypoint="python ray-job.py",
                  working_dir=str(FOLDER_PATH),
                  requirements=["pandas", "numpy", "torch"],
                  wait_for_completion = True,
                  **job_config
                  dag = dag,)
```

Anyscale Provider

Users can offload batch AI jobs using the Anyscale operators

```
from anyscale_provider.operators.anyscale import SubmitAnyscaleJob

# https://docs.anyscale.com/reference/job-api/#job-models
job_config = dict(
        entrypoint=”python finetune.py …”,
        … )

SubmitAnyscaleJob(task_id="llm-finetune",
                  conn_id=ANYSCALE_CONN_ID,
                  name=”llm-finetune”,
                  image_uri="anyscale/ray:2.23.0-py311",
                  compute_config="my-compute-config:1",
                  entrypoint="python ray-job.py",
                  working_dir=str(FOLDER_PATH),
                  requirements=["pandas", "numpy", "torch"],
                  wait_for_completion = True,
                  **job_config
                  dag = dag,)
```

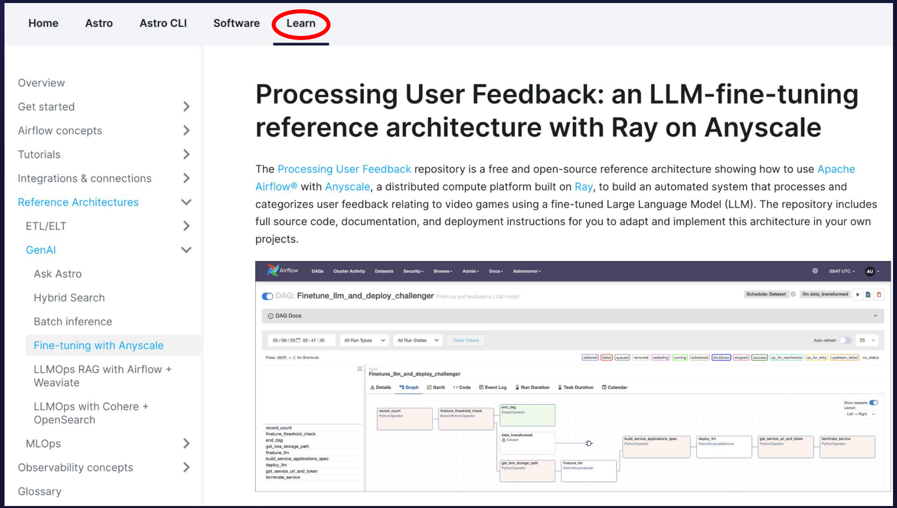
```
from anyscale_provider.operators.anyscale import RolloutAnyscaleService

# https://docs.anyscale.com/reference/service-api/#service-models
service_config = dict(
        name=”finetuned-llm-service”,
        working_dir=”https://github.com/anyscale/docs_examples/archive/refs/heads/main.zip”,
        applications=[{“import_path: “sentiment_analysis.app:model”}],
        requirements=[“transformers”,”requests”,”pandas”,”numpy”,”torch”],
        … )

RolloutAnyscaleService(task_id="llm-finetune",
                  conn_id=ANYSCALE_CONN_ID,
                  name=”finetuned-llm-service”,
                  image_uri="anyscale/ray:2.23.0-py311",
                  compute_config="my-compute-config:1",
                  in_place=False,
                  canary_percent=30,
                  **service_config
                  dag = dag,)
```

Key Takeaways

- **Build a continuous data update flow**
  - Use Airflow's data-driven scheduling to trigger DAGs
- **Build a continuous model update flow**
  - Use a scalable compute framework like Ray to finetune/deploy
- **Adopt evaluation-driven development deployment**
  - Establish a baseline LLM model & define evaluation metrics linked to bottom line
  - Try few-shot learning
  - Evaluate challenger model against baseline model
  - Only deploy a challenger model if its an improvement

Resources

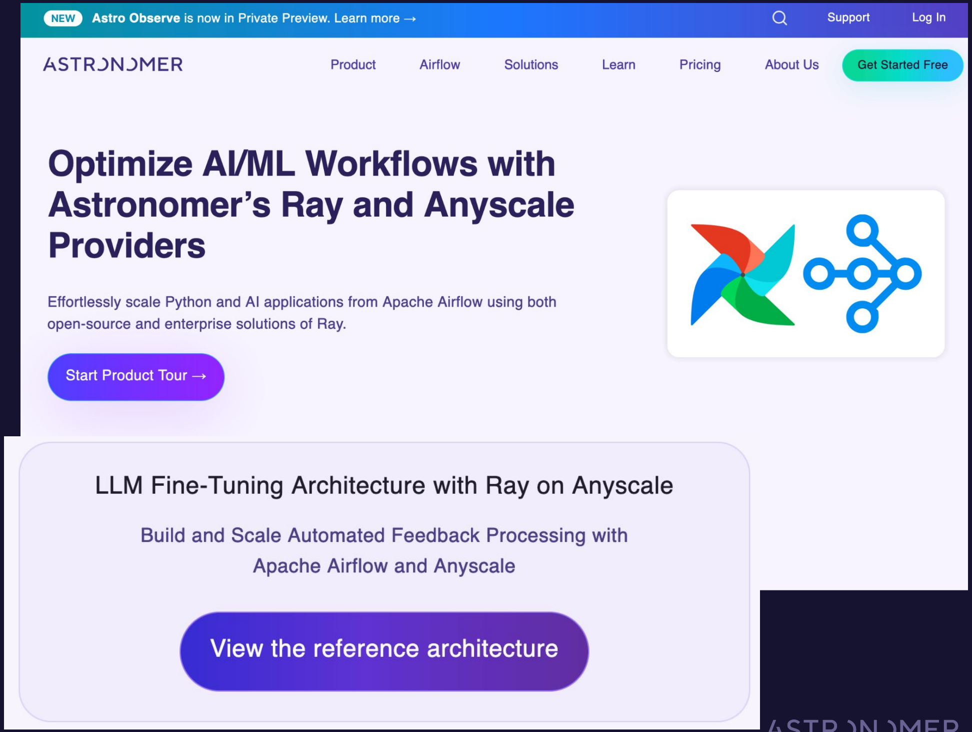
Astronomer docs



Processing User Feedback: an LLM-fine-tuning reference architecture with Ray on Anyscale

The Processing User Feedback repository is a free and open-source reference architecture showing how to use Apache Airflow® with Anyscale, a distributed compute platform built on Ray, to build an automated system that processes and categorizes user feedback relating to video games using a fine-tuned Large Language Model (LLM). The repository includes full source code, documentation, and deployment instructions for you to adapt and implement this architecture in your own projects.

astronomer.io/ray



Optimize AI/ML Workflows with Astronomer's Ray and Anyscale Providers

Effortlessly scale Python and AI applications from Apache Airflow using both open-source and enterprise solutions of Ray.

LLM Fine-Tuning Architecture with Ray on Anyscale

Build and Scale Automated Feedback Processing with Apache Airflow and Anyscale

RAY SUMMIT 2024

Sept 30 - Oct 2nd

Thank you!
Any questions?