

Compositional DAGs in Airflow

Fixing the Hidden Data Bottleneck in Physical AI Training

Saket Milind Karve
Member Of Technical Staff, Engineering, Dyna Robotics

What you'll leave with

AIRFLOW SUMMIT | ONLINE RECONNECT

Per-step resource specs

Every ingestion step declares its own CPU, GPU and memory profile and ships as a versioned container image.

Distribute-wait fan-out

Shard work across Kubernetes workers, then wait on a single barrier before the next stage starts.

Selective re-execution

Gate each step on a fingerprint so a downstream logic change never re-runs unaffected upstream steps.

Orchestrator / compute split

Business logic runs off-orchestrator, so the scheduler's worker pool never saturates under load.

Applies to any heterogeneous, resource-heavy Airflow pipeline, not just robotics.



Robot foundation model training is data-bound, not compute-bound

The GPUs are ready. The data isn't.

Where the training time actually goes

Robot fleet & teleop logs
Raw episodes land continuously

→

Heterogeneous trajectories
Video, IMU, joints, force/torque

→

BOTTLENECK

Transform to tensors
Decode, align, normalize, shard

→

IDLE

GPU training cluster
Waits on the step before it

Tens of TB

of new trajectory data per day in production teams

4+ modalities

per episode, each at its own sample rate

Every change

to preprocessing logic threatens a full rebuild

Anatomy of one robot trajectory

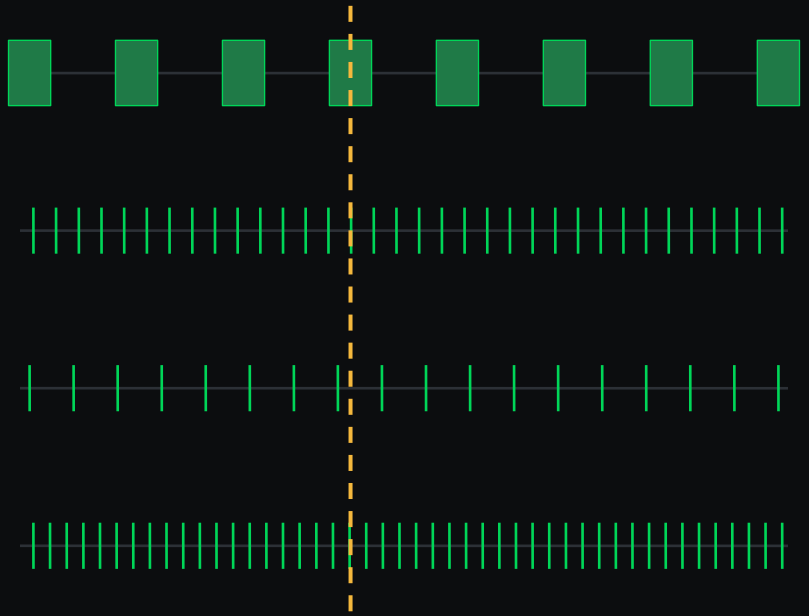
AIRFLOW SUMMIT | ONLINE RECONNECT

Camera streams
multi-view, synchronized

IMU
high-rate inertial

Joint state
positions, velocities

Force / torque
contact signals



t: what does the robot see, feel and do right now?

The hard part

Every stream has its own clock and rate. A training sample is only valid once all of them are aligned to the same instant, decoded and normalized.

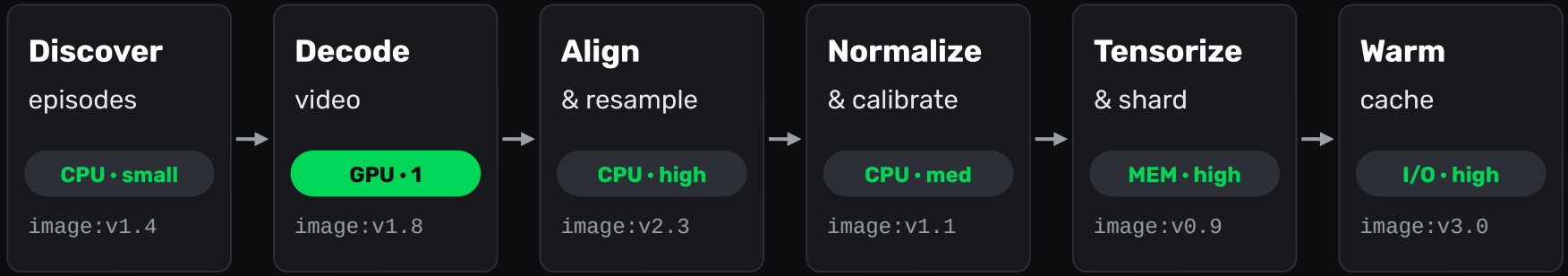
Why generic loaders don't compose

AIRFLOW SUMMIT | ONLINE RECONNECT

A generic loader assumes	Trajectory data needs
Uniform samples, one modality	Multi-modal, multi-rate episodes aligned in time
One resource profile for every transform	GPU decode, CPU alignment, high-memory sharding
Transforms run inside the training loop	Heavy work done upstream and cached ahead of time
Any change means re-running everything	Re-run only the steps a change actually affects

The compositional DAG

AIRFLOW SUMMIT | ONLINE RECONNECT



Resource profile

declared per step, not per DAG

Versioned image

each step ships and rolls back alone

Gate

skips the step if its inputs are unchanged

Step breakdown and versions are illustrative.

Pattern 1: Per-step resources

AIRFLOW SUMMIT | ONLINE RECONNECT

1. **Right-size every step**

   GPU only where decode needs it; high memory only for sharding.

2. **Profiles live with the step**

   One registry entry drives scheduling, not ad-hoc operator args.

3. **Kubernetes does the packing**

   Requests and limits map straight onto node pools.

```
STEPS = {
  "decode": Step("ingest/decode:1.8.2",
                 cpu="8", mem="32Gi", gpu=1),
  "align":  Step("ingest/align:2.3.0",
                 cpu="16", mem="64Gi"),
}

def step_op(name):
    st = STEPS[name]
    return KubernetesPodOperator.partial(
        task_id=name, image=st.image,
        container_resources=V1ResourceRequirements(
            requests={"cpu": st.cpu, "memory": st.mem},
            limits={"nvidia.com/gpu": str(st.gpu)}),
        deferrable=True)
```

Pattern 2: Versioned images



`ingest/decode`

CUDA + video codecs

1.8.2



`ingest/align`

resampling logic changed

2.3.0 → 2.4.0



`ingest/tensorize`

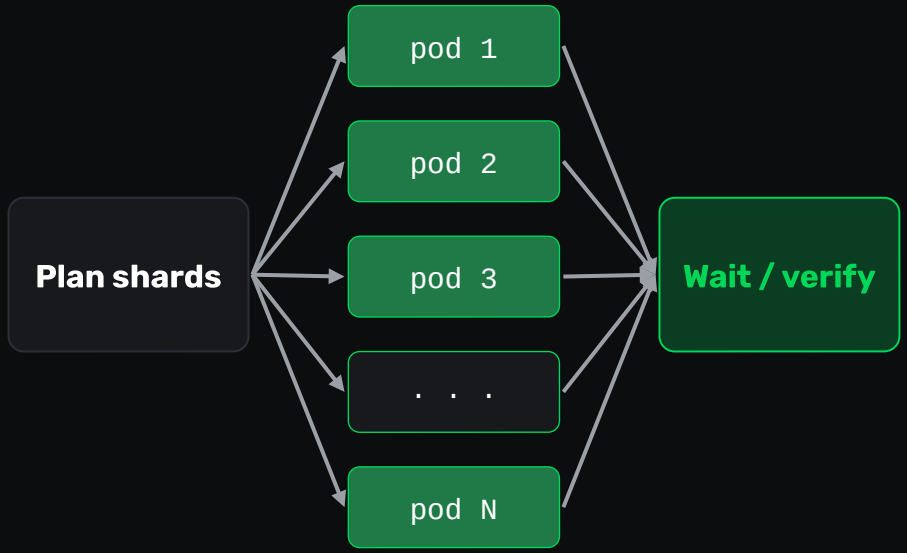
high-memory sharder

0.9.4

Why it matters

- Dependencies are isolated: CUDA and codecs live only in the decode image.
- Immutable tags make every run reproducible.
- One step can be upgraded or rolled back on its own.
- The image version becomes part of the step's cache key (Pattern 4).

Pattern 3: Distribute, then wait




```python
@task
def plan_shards() -> list[list[str]]:
    # one arg list per shard manifest
    ...

shards = plan_shards()
decoded = step_op("decode").expand(
    arguments=shards)

@task
def wait_and_verify(shards):
    # barrier: all shards landed + checksums
    ...

decoded >> wait_and_verify(shards)
```

Pattern 4: Selective re-execution



align changed (2.3.0 → 2.4.0): it re-runs, its downstream fingerprints change with it, upstream outputs are reused.

```python
@task.short_circuit(ignore_downstream_trigger_rules=False)
def gate(step: str, upstream_fp: str) -> bool:
    fp = fingerprint(STEPS[step].image,   # code version
                     upstream_fp,         # input version
                     CONFIG[step])        # parameters
    return not manifest.exists(step, fp) # False -> skip
```

Fingerprint = image + inputs + config

A step re-runs only if one of the three changed. Results are recorded in a manifest so reruns are cheap no-ops.

Pattern 5: Compute off-orchestrator

AIRFLOW SUMMIT | ONLINE RECONNECT



Airflow control plane

- Scheduler
- Triggerer
- Small worker pool

Plans, gates, submits, waits. Never transforms data.

pod spec →

← status (deferred)



Kubernetes compute plane

- **GPU pool** video decode pods
- **High-CPU pool** alignment, normalization
- **High-memory pool** tensorize & shard
- **Cache nodes** warm the training cache

What it did in production

AIRFLOW SUMMIT | ONLINE RECONNECT

>30%

of training slowdown eliminated

88 TB

training cache coordinated

16×H100

nodes kept fed with data

The same GPUs, the same model code. The change was how the data pipeline was composed and scheduled.

Practical guidelines



Make fingerprints explicit

Code version, inputs and config, recorded in a manifest you can query.



Size fan-out to the cluster

Cap concurrency with pools and max active tasks, not dataset size.



Keep XCom tiny

Pass shard manifests and paths between tasks, never the data.



Treat the cache as a product

Version it, verify it at the barrier, and warm it before training starts.

Beyond robotics

Use these patterns wherever an Airflow pipeline is heterogeneous and resource-heavy.



Per-step resources

and versioned images



Distribute-wait

fan-out with one barrier



Selective gating

re-run only what changed



Separate compute

off-orchestrator, deferrable

Also fits:

Video & multimodal ML

Autonomous driving logs

Geospatial imagery

Genomics



Questions?

saketmkarve@gmail.com
linkedin.com/in/saket-karve-43930511b