### 1. Project Overview and Structure

This project successfully designed, implemented, and tested a comprehensive 4-phase data pipeline on top, built in Python and orchestrated with Apache Airflow. The primary objective was to create a robust and scalable system capable of ingesting, processing, aligning, and curating complex multimodal datasets for an advanced robotics platform, Origami AI. The complete project, including all scripts and DAGs.

The pipeline was specifically designed to handle the challenges of real-world robotics data by processing diverse, containerized datasets (BridgeData V2, RoboNet) and integrating a workflow for high-fidelity synthetic data generation using NVIDIA Isaac Sim. A significant challenge involving the RT-1 dataset was identified and robustly handled, showcasing a professional approach to real-world technical limitations.

### 2. The 4-Phase Implementation

The project was executed in four distinct, sequential phases, each with a clear objective and a corresponding set of deliverables.

#### Phase 1: Schema Finalization & Metadata Standards

The foundation of the project was the creation of a master blueprint for our data. We adopted a JSON-centered schema where each robotic demonstration ("episode") is represented by a single metadata.json file. This file acts as a manifest, containing all critical metadata and pointing to larger binary files like videos.

Key standards were established:

Time Synchronization: All timestamps are recorded in UTC with millisecond precision (e.g., YYYY-MM-DDTHH:MM:SS.sssZ) for accurate data fusion.

Task Context: To capture the "why" of an action, the schema includes goal, language\_prompt, and user\_intent.

Sensor Metadata: To support AI modules like AMDC, each sensor entry contains its source\_id and calibration\_info.

This phase was automated by the schema\_metadata\_dag.py, which programmatically generated and validated the core data\_schema\_v1.json blueprint.

#### Phase 2: Real-World Data Ingestion and Normalization

This phase built the core machinery of the pipeline. The central challenge was processing the three distinct and complex real-world datasets: RT-1, BridgeData, and RoboNet.

After initial analysis revealed that all three datasets were stored in complex, containerized formats (TFRecord or ArrayRecord), we scrapped a multi-adapter approach in favor of a single, robust universal\_translator.py script. This script acts as the "master key" for ingestion, using the official tensorflow-datasets (tfds) library to correctly read the data.

The multimodal processing works as follows:

Read the Container: The script uses the dataset\_info.json and features.json files provided with each dataset as a blueprint to unlock and read the .tfrecord or .array\_record files.

Extract Multimodal Streams: It iterates through each episode and extracts the separate, synchronized data streams. For example, from a single Bridge data record, it extracts:

The sequence of images (observation/image).

The language command (language\_instruction).

The robot's physical state (observation/state).

Translate to a Unified Format: The extracted data is then translated into our project's simple, standard format: a video.mp4 file (created by encoding the image sequence) and a metadata.json file containing the language prompt and other context.

This process successfully translates three completely different raw data formats into a single, unified structure that the rest of the pipeline can work with.

#### Phase 3: Integration with Origami AI Modules

With clean, standardized data, this phase focused on "tuning" it for specific AI models. The align\_modules\_dag.py orchestrates this process, using the utils/origami\_mapper.py script to enrich the data. This script adds model-specific fields to the metadata.json files, such as:

uncertainty scores: To support the Spatio-Temporal Uncertainty Modeling (STUM) module.

structured\_prompt: A machine-readable version of the language command for advanced task planners like HTD-IRL.

Phase 4: Curation and Quality Control

The final phase acts as an automated quality assurance step. The curate\_qc\_dag.py uses the utils/curator.py script to:

Analyze Data Quality: It programmatically inspects each record, for example, by calculating a "blur score" for the video.

Route Files: Based on predefined thresholds, it automatically moves the data (video.mp4 and metadata.json) into either a data/curated or data/rejected folder.

Tag Gold-Standard Data: The highest-quality records in the curated folder are automatically tagged with "is\_gold\_standard": true in their metadata, creating a pristine dataset for final model evaluation.

### 3. Compatibility with NVIDIA Is3aac Sim

A key success of this project was establishing a workflow for generating unlimited, high-quality synthetic data that is perfectly compatible with our processing pipeline.

The goal is to create a "digital twin" of the tasks found in the real-world datasets. The simulation/isaac\_sim/isaac\_kitchen\_generator.py script, although may not be fully functional, still serves as a starting point for the prelude of this process. When run from within the Isaac Sim application, it suggested the steps of building a 3D Scene that creates a virtual environment with a robot, objects, and lighting. Then record photorealistic data, in which uses Isaac Sim's virtual camera to capture photorealistic, multimodal data as the robot performs a task. Lastly Exports in the Correct Format: Saves the output as a video.mp4 and metadata.json file in the exact same format as our adapters.

The "dotted" videos generated during our local tests were a successful simulation of this process, proving that our Phase 2 pipeline is ready to ingest and process data coming from Isaac Sim without any changes. This allows further operations for AI models to be trained on a powerful mix of real-world and synthetic data.