Attribution Pipeline Orchestration

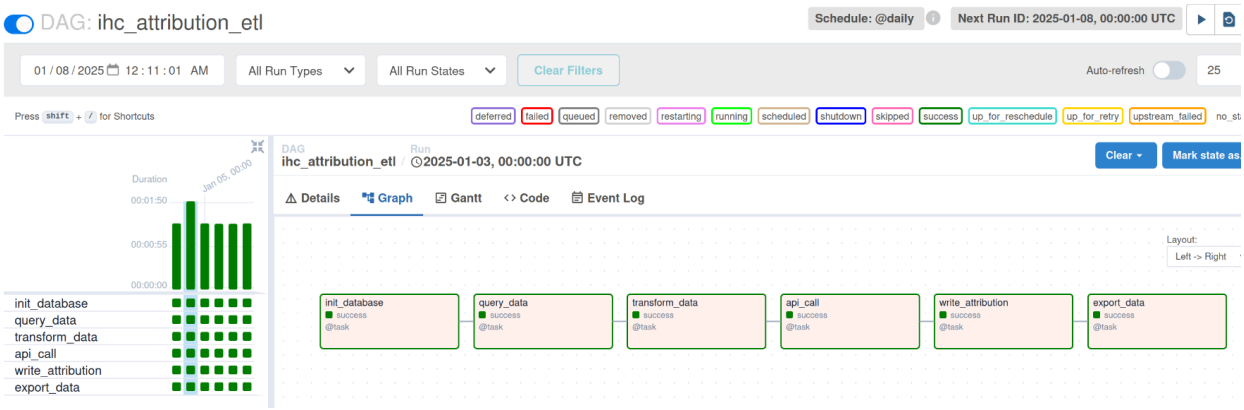
1. Introduction

This document outlines the design for an attribution pipeline orchestration solution, developed as part of a Data Engineer position test challenge. The pipeline aims to process customer journey data, obtain attribution results from an external API, and generate reports for analysis.

2. Pipeline Overview

The pipeline will perform the following steps (Takes the date range as input that is configured as airflow's env variable. Configurable in environments in the docker-compose.yml provided): See README.md to run the pipeline

1. **Init Database:** Initialize the tables on SQLite database
2. **Data Extraction:** Extract session and conversion data from a provided SQLite database (challenge.db) which is preprocess and then persisted as a JSON File.
3. **Customer Journey Construction (Transform data):** Combine session and conversion data to build customer journeys for each conversion.
4. **API Interaction:** Send customer journey data to the IHC Attribution API (https://ihc-attribution.com/marketing-attribution-api/) in **asynchronous** chunks to receive attribution results.
5. **Attribution Data Storage:** Store the received attribution results in the attribution_customer_journey table within the database.
6. **Report Generation:** Aggregate data and generate the channel_reporting table, summarizing key metrics. Export the channel_reporting data to a CSV file, including calculated CPO (Cost Per Order) and ROAS (Return on Ad Spend) metrics.



3. Data Model

The pipeline utilizes the following tables within the `challenge.db` database:

- **`session_sources`:** Contains information about user sessions, including timestamps, channels, and engagement indicators.
- **`conversions`:** Stores data about conversions, such as conversion IDs, user IDs, timestamps, and revenue.
- **`session_costs`:** Records the cost associated with each session.
- **`attribution_customer_journey`:** (To be created) Stores the attribution results received from the API, linking conversions and sessions with their respective attribution values.
- **`channel_reporting`:** (To be created) Aggregates data to provide a summary of attribution performance by channel and date.

4. Pipeline Architecture

The pipeline will be designed with a focus on data persistence between tasks and modularity, allowing for easy separation of steps for orchestration. This architecture leverages Airflow's capabilities to store intermediate data. This ensures data integrity and allows for efficient recovery in case of task failures or pipeline interruptions, this also ensures memory management, data reusability and task isolation.Each step can be implemented as a separate task, enabling independent execution, monitoring, and potential parallelization.

5. Implementation Details

- **Technology Stack:** Python with Pandas is used for data manipulation and transformation.
- **API Integration:** The “requests” library will be used to interact with the IHC Attribution API. API key and “conv_type_id”will be provided as parameters.
- **Data Chunking:** The API's limitations on data volume is addressed by implementing an asynchronous mechanism to send data in manageable chunks.
- **Database Interaction:** The sqlite3 library is used to connect to and interact with the SQLite database.
- **Data Transformation:** Pandas is used to perform data manipulation.
- **Report Generation:** SQL queries and Pandas are used to generate the channel_reporting table.
- **Output:** The final report will be exported as a CSV file.

6. Orchestration

The pipeline is designed for orchestration using Apache Airflow. Each step can be defined as a separate task within an Airflow DAG (Directed Acyclic Graph). This allows for:

- **Environment Variables:** Declare the environments variables (in docker-compose.yml)
    - SESSION_LOWER_LIMIT_DATE: No. of days before the conversion to validate the sessions.

    - MAX_CUSTOMER_JOURNEYS: Max no. of CJs in an API request
    - MAX_SESSIONS: Max no. of session in an API request
    - CONV_TYPE_ID: Given while training the IHC parameters
    - API_URL: URL for the API endpoints
- **Scheduling:** Defining execution intervals for the pipeline.
- **Dependency Management:** Ensuring tasks are executed in the correct order.

7. Memory Management:

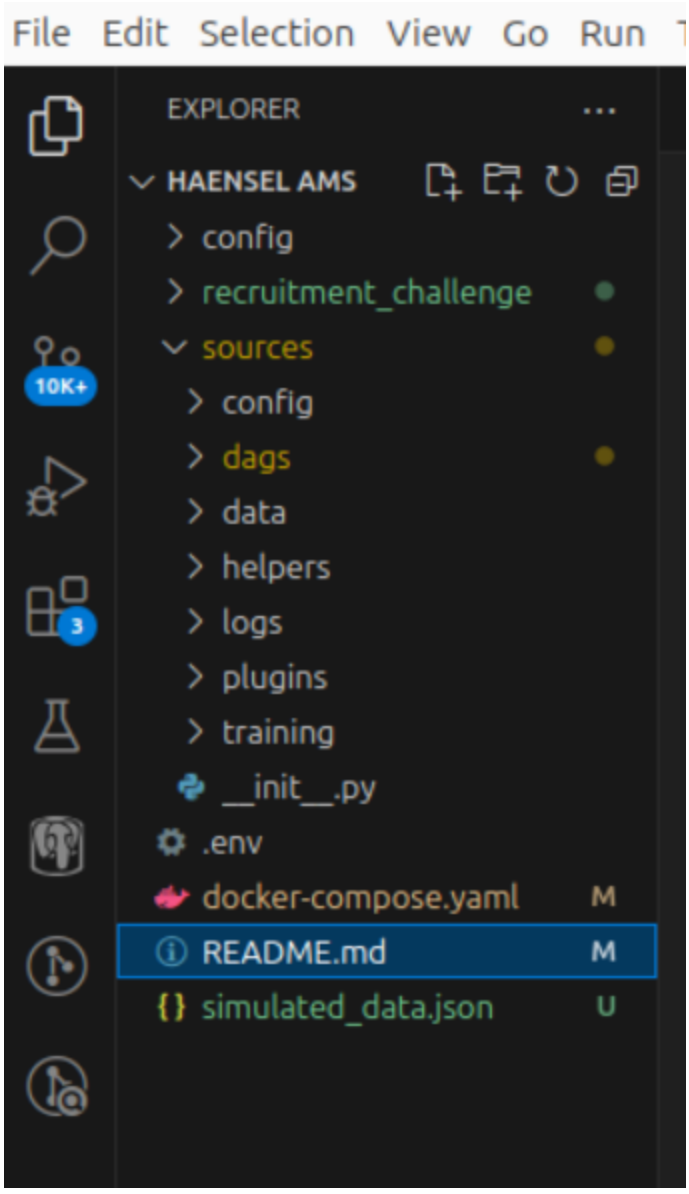
Have cleared the memory using garbage collectors in tasks that are heavily memory consuming.

8. Training Dataset:

The documentation recommends having 10-100K Customer journeys from the given data. We observed that we have only 2000 available Customer Journeys from which we have generated a training dataset. Have assumed the model is only fine tuned with the available data. We also observed that more than 50% of the Customer Journeys have only one session.

9. Code Structure

The Python code is structured as shown below.



Dags: contains the main ETL dag and the tasks associated.

Helpers: Contains the helper modules that assists each task

Data: The data folder holds the database and the data that are persisted between the tasks. (In real world application this could be replace by a cloud solution)

Training: This folder contains the training set generation and data analysis in .ipynb files

10. Future Considerations:

A larger training data set could be used to train the IHC attribution model to better generalize the model that it is being trained on.

Performance Optimization: Optimize SQL tables by adding partitions on timestamps. It would be efficient to load the data from SQL engines when the data is large.