ResuMatrix: An Agentic-AI powered Resume Analyzer

IE 7374 Spring 2025: Team 20

Data Pipeline Document

Team Members:

- Ashish Annappa Magadum
- Kishan Sathish Babu
- Nishchith Rao Palimar Raghupathi
- Pranay Saggar
- Shailendra Tripathi

Introduction

Our resume-job matching pipeline is designed to automate the end-to-end process of extracting, processing, and analyzing job descriptions and resumes. Built using Apache Airflow, our pipeline orchestrates various tasks such as loading data, cleaning, preprocessing using natural language processing (NLP) techniques, and applying machine learning models to determine the suitability of a resume for a given job description.

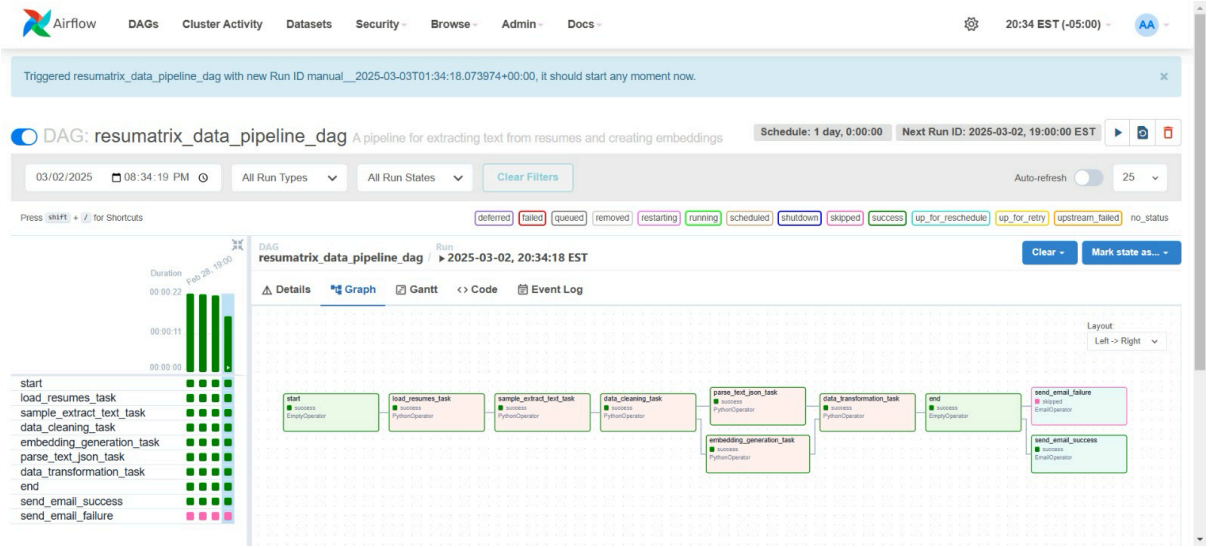
This document describes the data pipeline built using Apache Airflow, following a structured DAG format. The pipeline is designed to handle the initial phase of the resume-job matching workflow, i.e. uploading of resumes, preprocessing, versioning, and logging. By leveraging tools such as DVC for version control and Airflow for orchestration, this pipeline ensures reproducibility, modularity, and efficient automation of the resume-job matching process. The report will outline each stage of the pipeline with detailed explanations, configurations, and tools used.

Folder structure:

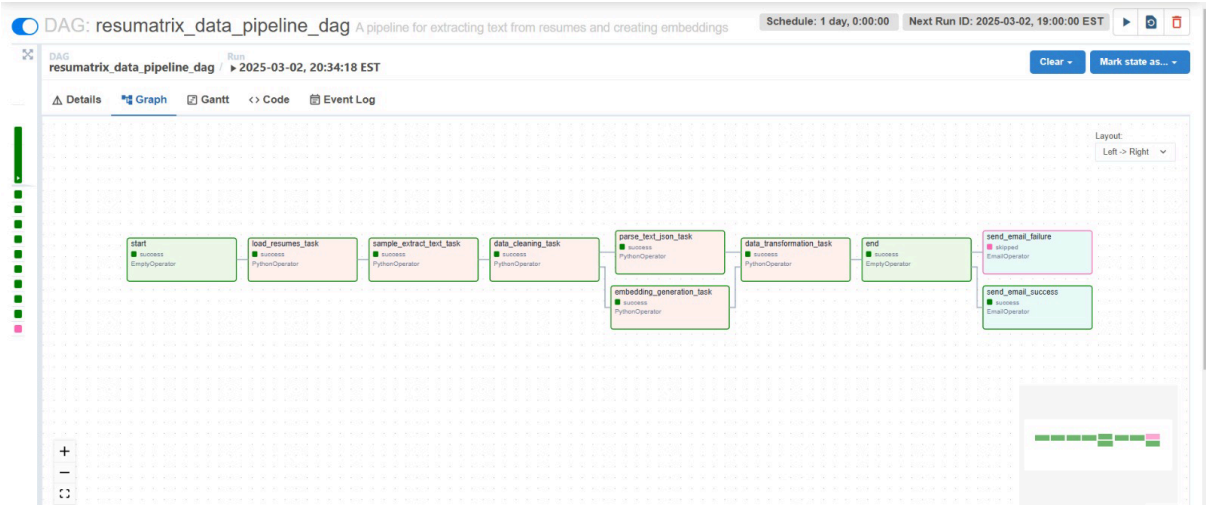
```
/                          # Root
-- src/                    # Source code (Frontend, backend WIP)
-- data_pipeline/          # Airflow data pipeline directory
        -- dags/           # Airflow DAGs
        -- data/           # Data
        -- logs/           # Logs
        -- res/            # Git readme meta files (images)
        -- tests/          # Unit tests
        -- dvc.yaml        # dvc config
        -- docker-compose.yml  # Entry point
-- README.md
```

Data Pipeline

Airflow Dag Dashboard:



Directed Acyclic Graph:



DAG File: resumatrix_dag.py in the dags/ folder

Data Acquisition

The pipeline begins with a data acquisition step where the resume and job description dataset is downloaded from the Hugging Face dataset repository. The data is stored locally within the data/ directory and tracked with DVC (Data Version Control) to maintain version history and ensure reproducibility. This step ensures that the dataset remains versioned, enabling seamless tracking of changes and facilitating consistent experimentation across different iterations of the pipeline.

Data preprocessing

The data preprocessing stage focuses on cleaning and structuring the resume and job description text to improve the accuracy of the fit/no-fit classification. This step includes removing HTML tags,

filtering out URLs, eliminating non-text elements, and standardizing text formats. Additionally, NLP techniques such as tokenization, stopword removal, and lowercasing are applied to enhance text consistency. By refining the dataset, preprocessing ensures that the model can effectively analyze and determine whether a resume is a good fit for a given job description.

We use openAI's embeddings to represent our data in the form of vectors. Our core requirements that led us to choose an LLM's embeddings instead of a SLM/MLM embeddings are:

1. Large vocabulary size to cover named-entities, skills, technical jargons in resumes
2. Better contextual understanding of the candidate's skills given their career level
3. Easy integration with other features of such LLMs. (e.g. RAG)

Data Schema & Statistics Generation

In parallel to the vector embeddings task, a separate task uses an LLM in structured output mode to parse the pre-processed data and identify all the major sections. It essentially extracts details such as skills, work experience and collects relevant information to produce a JSON output.

As part of the data quality assurance process, schema validation is performed to verify the format, column names, and data types of the incoming dataset. This step ensures that the data adheres to the expected structure, preventing potential errors in downstream processing. This is done using the **Pydantic library** when the LLM generates the output.

schema_check_task: Validates the dataset schema to ensure it aligns with the expected format and structure, identifying any inconsistencies before proceeding with data processing.

Tracking and logging

To ensure seamless monitoring and troubleshooting, the pipeline is equipped with a structured logging framework that tracks each stage of execution and captures errors as they occur. This facilitates efficient debugging and maintains a detailed operational history for auditing purposes.

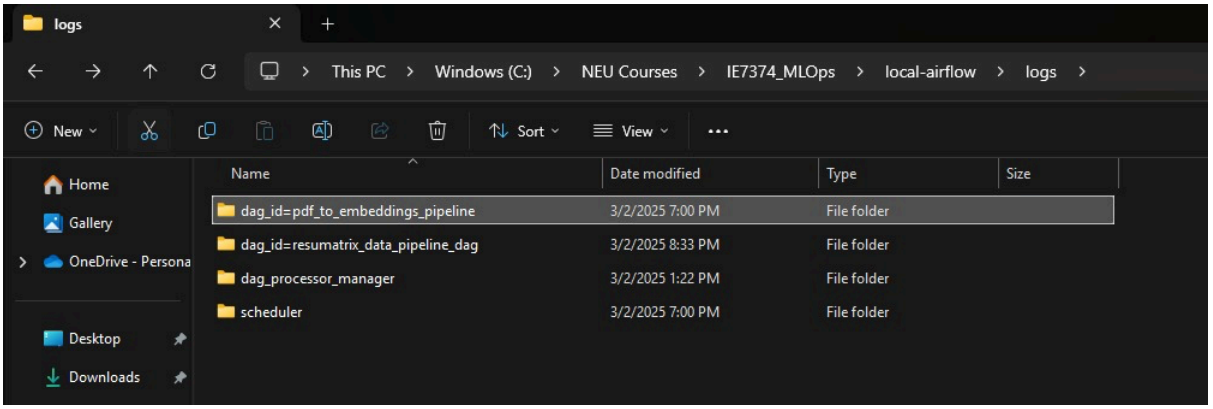
Key Logging Mechanisms:

Execution Tracking: Every task logs its initiation and completion, along with any encountered errors, enabling systematic issue resolution. The Python logging module is used to organize and store log data.

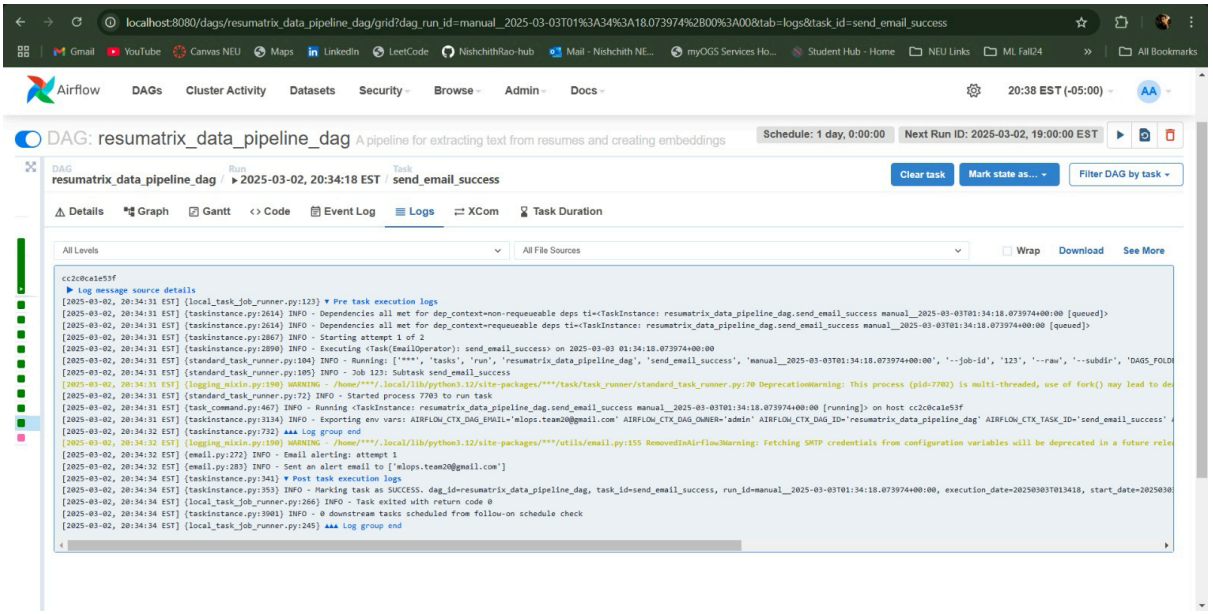
Automated Alerts: In the event of a failure, email notifications are sent to promptly inform relevant stakeholders, ensuring quick intervention.

With a well-integrated logging and error-handling system, the pipeline provides real-time visibility into its workflow, promptly detects failures, and simplifies debugging, ultimately enhancing overall reliability and performance.

Log files are generated for each task in the pipeline and stored in the logs folder.

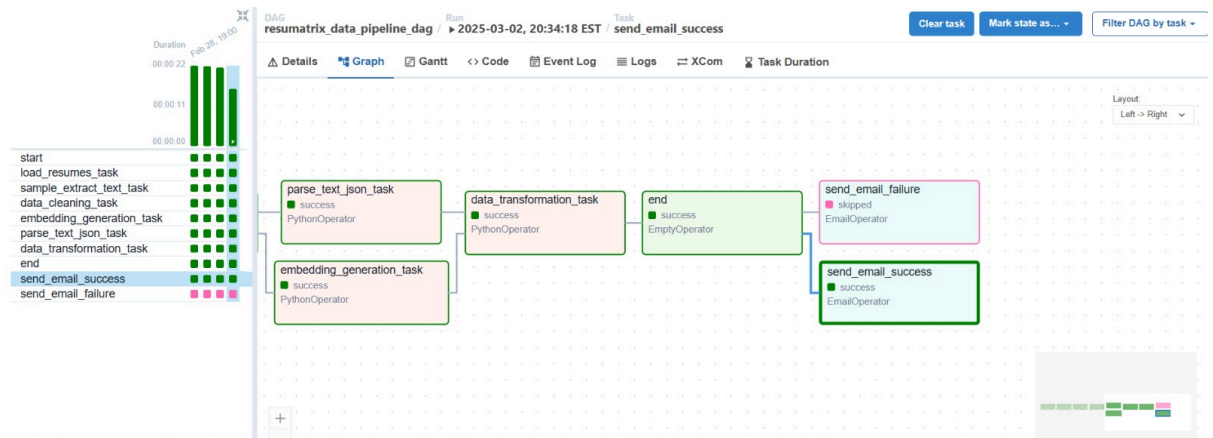


Sample log file:

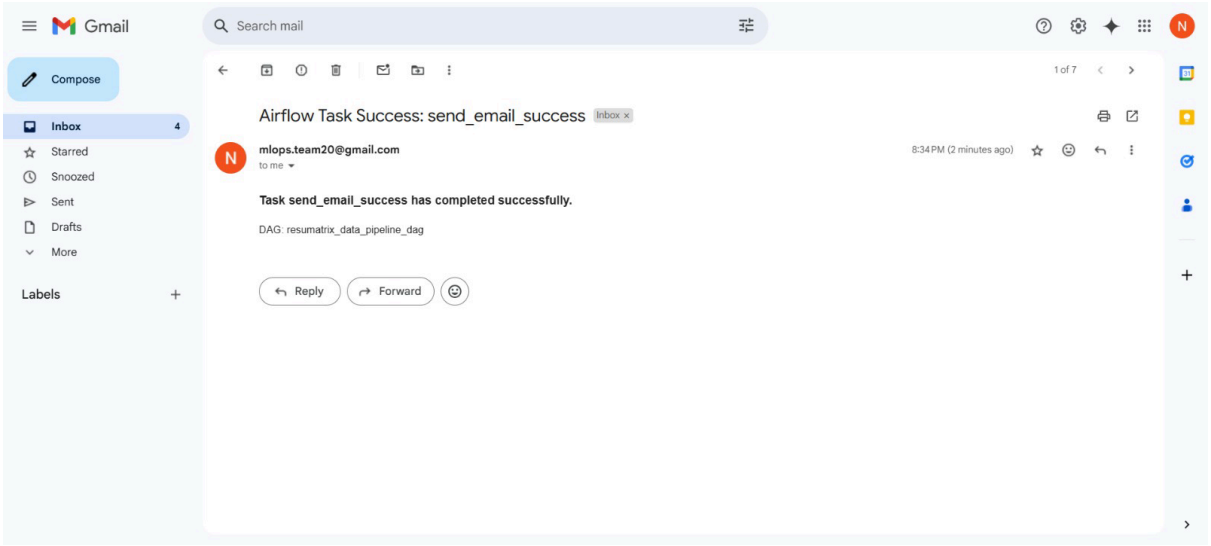


Anomaly detection and alerts

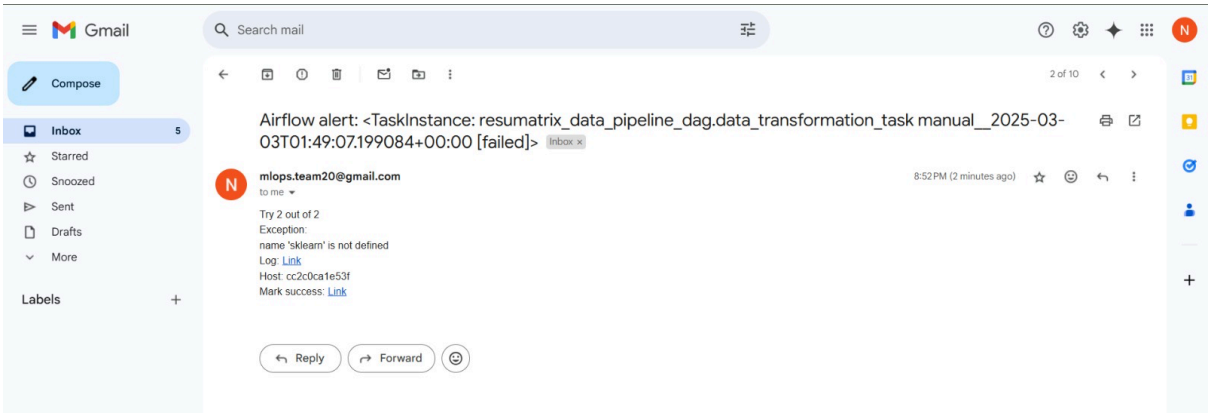
Email alerts are configured to notify the provided email of any anomalies or errors detected within the pipeline, ensuring swift action can be taken. These automated notifications provide critical information to address issues promptly and maintain pipeline integrity.



Success email:



Failure email:



Data Versioning with DVC:

DVC is employed to version-control the dataset, allowing the team to monitor and manage different versions of the data throughout the pipeline's lifecycle. The configuration includes remote storage on Google Cloud Storage (GCS) for efficient and scalable data management.

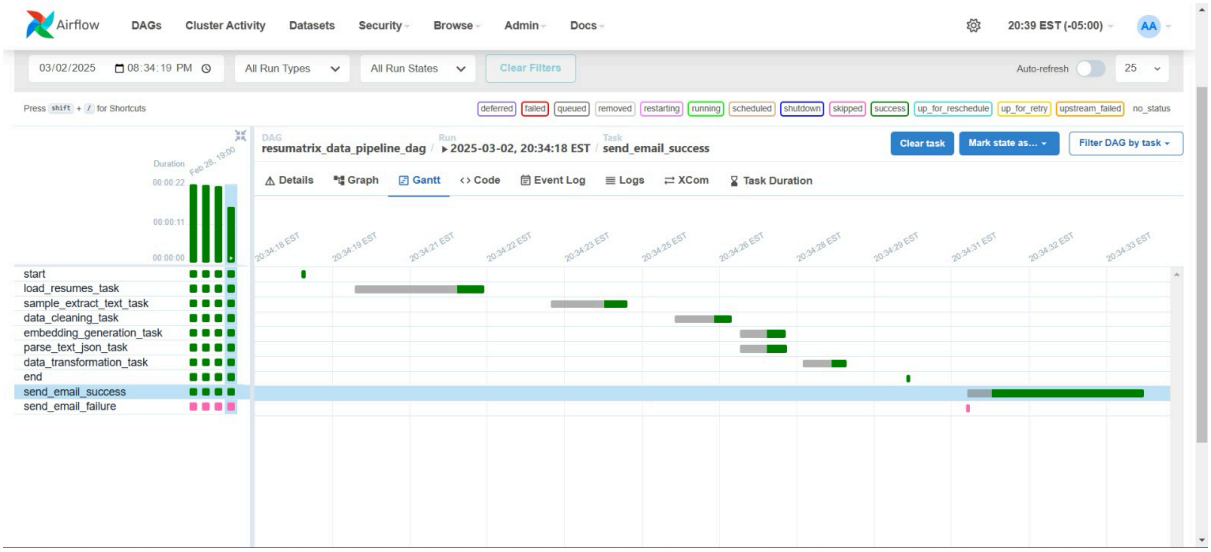
DVC Details:

Configuration: GCS bucket set up as the DVC remote storage for seamless integration and access.
Data Tracking: Tracks the resume-job dataset and related files in the data/ folder.
Reproducibility: Allows team members to retrieve specific versions of the data, ensuring the ability to replicate pipeline results and maintain consistency across experiments.

Pipeline flow optimization

To optimize the pipeline flow and improve efficiency, Apache Airflow's Gantt chart is leveraged for visualizing task execution timelines and resource allocation. By using this feature, the pipeline is designed to enable parallel execution of tasks wherever possible, reducing overall runtime and improving resource utilization. The Gantt chart provides a clear view of task dependencies, start and

end times, and resource usage, allowing for better scheduling and management of resources. This visualization helps identify bottlenecks and areas for potential parallelization, ensuring that computational resources are allocated efficiently and that tasks are executed in the most optimal order. Additionally, parallelization improves the speed of data processing, making the pipeline more scalable and responsive to changing workloads.



Testing

To optimize the pipeline flow and improve efficiency, Apache Airflow's Gantt chart is leveraged for visualizing task execution timelines and resource allocation. By using this feature, the pipeline is designed to enable parallel execution of tasks wherever possible, reducing overall runtime and improving resource utilization. The Gantt chart provides a clear view of task dependencies, start and end times, and resource usage, allowing for better scheduling and management of resources. This visualization helps identify bottlenecks and areas for potential parallelization, ensuring that computational resources are allocated efficiently and that tasks are executed in the most optimal order. Additionally, parallelization improves the speed of data processing, making the pipeline more scalable and responsive to changing workloads.

Data Bias

During model development, we may uncover additional biases, particularly related to PII and demographic factors, as our data consists of textual information, such as resumes and job descriptions. To address these concerns, in mitigate_bias.py we remove all personally identifiable information (PII) and eliminate stopwords, such as pronouns, to mitigate demographic bias. Other demographic features like age, associated organizations become important features to absolutely remove, but to improve the fairness of the model while maintaining feature quality, we categorize the values into various bins.

Additionally, we aim to remove simple but potentially misleading parametric features, like resume length, by converting the data into embeddings, which provide a more nuanced representation. This approach helps us avoid overfitting to trivial features that might skew the model's fairness.

However, it is important to note that biases could still emerge throughout the development process, and continuous evaluation will be necessary to identify and address them as we move forward.