Apache Jira Scraping & LLM Data Pipeline

Assignment Completion Report

Author: Ayush Kumar
Date: November 2, 2025
GitHub Repository: https://github.com/Ayush2554work/AYUSH-Apache-jira-scraper

1. Project Overview

This project is a **robust, fault-tolerant data engineering pipeline** built to extract public issue data from Apache's Jira instance. The primary goal was to design a system that can efficiently scrape, clean, and transform vast amounts of real-world data into a high-quality, structured JSONL corpus, specifically formatted for training Large Language Models (LLMs).

The pipeline successfully processed three major projects—**SPARK**, **KAFKA**, and **HADOOP**—demonstrating resilience to network failures, API limits, and real-world data inconsistencies.

2. Technical Architecture & Design Decisions

To meet the requirements for a professional-grade scraping system, several key design decisions were made.

Architecture Choice: REST API over HTML Scraping

While HTML scraping was an option, the decision was made to use the **official Jira REST API**. This approach is fundamentally more robust, faster, and more reliable:

- **Stability:** The API provides structured JSON, eliminating the need for brittle HTML/CSS selectors that break with any UI change.
- **Efficiency:** It allows for precise data fetching, reducing network overhead.
- **Data Integrity:** It delivers data directly from the source, ensuring accuracy.

Core System Components

The system is a **Python-based pipeline** composed of three main modules:

- scraper.py (The Scraper): A stateful, class-based scraper responsible for all API communication.
- transformer.py (The Transformer): A set of functions that clean and restructure the raw API data.
- main.py (The Orchestrator): The main entry point that manages the flow of data from scraping to transformation to final file output.

3. Key Features: Fault Tolerance & Reliability

This system was designed to handle the unpredictable nature of scraping large, real-world datasets.

Resumability from Point-of-Failure

This is the pipeline's **most critical feature**.

Mechanism: The scraper uses a `scraper_state.json` file. After every successfully fetched page, it saves the `startAt` index for that specific project.

Benefit: If the script is interrupted for any reason (network loss, server crash, manual stop), it can be restarted and will automatically resume from the exact page it left off for each project. This prevents data loss and avoids re-scraping thousands of issues.

Graceful Error Handling

The system implements multiple layers of error handling:

- **Network & Server Errors (5xx, Timeouts):** Implemented the `tenacity` library to apply exponential backoff. The system automatically retries failed requests with increasing delays, preventing server overload and gracefully handling temporary outages.
- **API Rate Limits (HTTP 429):** The scraper explicitly detects the 429 status code, logs it, and enters a 60-second cooldown before retrying the same request.
- **Malformed Data Handling:** The `transformer.py` module wraps data extraction in a try...except block. This ensures that a single malformed issue (e.g., with null fields where data was expected) is logged and skipped, preventing one bad record from crashing the entire multi-hour scraping job.

4. Data Transformation Pipeline

The "raw" data from the API is not suitable for LLM training. It was processed as follows:

Input: Raw JSON objects from the Jira API.

Transformation Process:

- **Structuring:** Data is parsed into a clean Python dictionary.
- **Cleaning:** A `clean_text` function strips Jira's wiki markup (e.g., `{code}`) from descriptions and comments.
- **Consolidation:** The issue title, description, and all comments are combined into a single, clean text field, providing the LLM with full context.
- **LLM Task Generation:** For each issue, a `derived_tasks` JSON object is created. This formats the data into the standard (instruction, input, output) format required for fine-tuning.

Example Task (Classification):

```
Instruction: "Based on the issue description, classify its priority..."
Input: "Title: [Title]\nDescription: [Desc]"
Output: "[Priority]"
```

Output: The final data is appended to `jira_corpus.jsonl`. Using the JSONL format ensures that data is written line-by-line, which is highly efficient for both writing and reading into LLM training scripts.

5. Technology Stack

Language: Python 3

Libraries:

- **requests:** For efficient API communication and connection pooling (`requests.Session`).

- **tenacity:** For robust, automated, exponential-backoff retries.

Data Format: JSON & JSONL

Version Control System: Git & GitHub

Conclusion

This Apache Jira scraping pipeline demonstrates professional-grade data engineering practices with emphasis on reliability, scalability, and fault tolerance. The system successfully combines robust error handling, intelligent state management, and comprehensive data transformation to produce high-quality training data for Large Language Models.