



NEW YORK UNIVERSITY

PROJECT REPORT

BIGDATA

Professor: Amit Patel

Student Names and Net-IDs:

Geetha Krishna Guruju (Net Id: gg3039)

Tejdeep Chippa (Net Id: tc4263)

Isha kookanapalli math (Net Id: ik2688)

Course Name: Big Data

Course Section: Section C

Course Code: FA 25 6513-C Big Data

Semester Term: Fall 2025

TABLE OF CONTENTS

Code Repository.....	3
Executive Summary.....	5
Introduction.....	6
Motivation.....	7
Dataset Description.....	8
Technologies Used.....	9
Code Execution Instructions.....	11
PART 1 — Running the Big Data ETL Pipeline (Airflow + Dataproc + PySpark).....	11
1. Environment Setup.....	11
2. Trigger the Orchestrated Pipeline.....	12
PART 2 — Data Analysis Dashboard (Streamlit + Plotly).....	12
1. Install Visualization Dependencies.....	12
2. Launch the Dashboard.....	12
PART 3 — Running Inference with the Trained LLM.....	12
1. Load the Model.....	13
2. Build the Prompt Template.....	13
3. Run a Sample Inference.....	14
Example Output.....	14
High Level Code Logic:.....	15
Data Loading and Initial Validation.....	15
Text Cleaning and Consolidation.....	15
Deduplication Using MinHash LSH.....	16
Distributed Tokenization and Dataset Structuring.....	16
Statistical Analysis and Visualization.....	16
Pipeline Preservation and Reproducibility.....	17
ETL with HDFS.....	18
Rationale for Selecting HDFS.....	18
Implementation Approach.....	19
Advantages Achieved.....	19
Data Engineering Pipeline.....	20
Data Loading and Text Consolidation.....	20
Preprocessing Steps.....	20
Dataset Structuring.....	21
LLM Inferencing and Output Evaluation.....	22
Input Selection and Prompt Design.....	22
LLAMA Model Execution.....	22
Output Comparison and Observations.....	22
Streamlit Visualization and Hive Analysis.....	24
Dataset Overview and Purpose of Analysis.....	24
Hive Queries and Dataset Patterns.....	24
Key Insights From Hive Analysis.....	25
Streamlit Visualization and Interaction.....	26

Conclusion of Hive and Streamlit Analysis.....	29
Technological Challenges.....	29
Big Data Processing and Scalability.....	29
Deduplication and Similarity Detection.....	30
Tokenization at Scale.....	30
Airflow Integration and Orchestration.....	31
Managing Large Intermediate Data and Memory.....	31
Lessons Learned.....	32
What Worked Well:.....	32
Challenges Encountered and How We Overcame Them:.....	33
Key Insights:.....	34
Changes in Technology.....	36
Migration from Local Processing to HDFS on NYU Dataproc.....	36
IPYNB Notebook for Inferencing on an LLM.....	36
Emphasis on Spark, Hive, and Airflow for Analytics and Orchestration.....	36
Uncovered Aspects from Presentation.....	38
Future Improvements.....	39
Data Sources and Results.....	41
Code Repository.....	41
Data Source.....	41
Result.....	42
ETL Pipeline Output.....	42
Hive Analysis.....	42
Airflow Pipeline Execution.....	42
Streamlit Application and LLM Comparison.....	43
Summary of Key Findings.....	43
Conclusion.....	44

Code Repository

All code, notebooks, and deployment scripts for this project are available in our public GitHub repository:

<https://github.com/geethaguruju/LLM-DataPrep>

PPT Presentation:

 *presentation*

Executive Summary

Large Language Models (LLMs) relied on in biomedical research such as BioBERT, BioGPT, and LLaMA-based variants require highly structured, domain-specific corpora to learn accurate terminology, relationships, and contextual patterns. However, biomedical text sources like PubMed exist in raw, unstructured formats that contain incomplete sentences, formatting inconsistencies, inline citations, duplicated articles, and widely varying linguistic structures. These artifacts degrade LLM training quality unless addressed through extensive preprocessing.

In this project, we built a **scalable, automated ETL pipeline** using Apache Spark, HDFS, Hive, and Airflow to preprocess **2.21 million PubMed records**. The pipeline performs four major transformations: **data cleaning, deduplication with MinHash LSH, distributed tokenization, and statistical analysis**. Each stage is designed for large-scale distributed execution on a Dataproc cluster, enabling efficient processing of tens of gigabytes of biomedical text. The final dataset is stored in Parquet format to support downstream machine-learning tasks and integrated into a Streamlit-based web interface that demonstrates how cleaned and structured text improves LLM output quality. Our work illustrates how Big Data technologies can enable reproducible, large-scale data preparation tailored for biomedical LLM fine-tuning.

Introduction

Biomedical research depends heavily on access to clean, reliable textual data. Millions of new scientific abstracts are published annually, yet these datasets are rarely delivered in a machine-learning-ready format. PubMed, one of the largest biomedical textual repositories, exemplifies this issue: the dataset provides rich content but suffers from structural irregularities, redundant entries, and formatting artifacts. Because LLMs are highly sensitive to the quality of their training corpora, preprocessing at scale becomes not only beneficial but essential.

Traditional preprocessing techniques fail when confronted with millions of records and gigabytes of text. Operations such as deduplication, tokenization, and statistical profiling exceed the limits of single-machine processing. This led us to investigate how distributed systems could automate the transformation of noisy biomedical text into structured inputs for LLM training. The goal was not merely to construct another cleaning script, but rather to develop a **robust, end-to-end ETL framework** that would support reproducibility, scalability, and downstream model fine-tuning.

Our work addresses this challenge by building a Big Data-driven pipeline that leverages Apache Spark's distributed capabilities, Airflow's orchestration framework, Hive's querying engine, and HDFS's scalable storage layer. Together, these technologies allow us to process millions of records efficiently and generate a high-quality dataset that could serve as the foundation for a domain-specialized LLM.

Motivation

Fine-tuning LLMs for biomedical applications presents unique challenges due to the nature of the data involved. Biomedical text often includes intricate terminology, citation patterns, chemical and gene names, abbreviations, and structured numeric reporting. Raw datasets such as PubMed integrate titles, abstracts, metadata, and identifiers inconsistently. Inline citations like “[4]” or “[10–15]” frequently interrupt the text. Some entries contain only a title, while others include multiple near-duplicated segments.

Such inconsistencies reduce the effectiveness of LLM fine-tuning by introducing noise, redundancy, and spurious patterns that models may misinterpret as meaningful. Biomedical LLMs demand **high-fidelity input**: the quality of the dataset directly influences accuracy in downstream tasks like summarization, question-answering, entity recognition, and literature reasoning.

Scalability is another major concern. The dataset used in this project contains **more than 2.21 million articles totaling 2.78 GB** in Parquet format. The JSONL equivalent is nearly 70 GB. Cleaning, deduplication, and tokenization at this scale require distributed execution, which rules out traditional scripting approaches. Big Data frameworks offer the parallelism and memory management needed to process millions of textual records without failure.

Our motivation, therefore, was twofold:

1. **To create an automated system capable of producing high-quality, LLM-ready biomedical text**, free of noise and duplication.
 2. **To demonstrate the role of Big Data technologies in enabling modern NLP workflows**, especially for fine-tuning domain-specific language models.
-

Dataset Description

The MedRAG PubMed dataset sourced from HuggingFace contains approximately **2.21 million biomedical records** and is stored in Parquet format for efficient distributed retrieval. Each record includes:

- a unique identifier (`id`),
- a short title (`title`),
- one or more text fields (`content`, `contents`),
- and a PubMed identifier (`PMID`).

Despite its scale and utility, the dataset presents challenges. The `content` and `contents` fields frequently hold partial segments of what should be a unified abstract. Several entries contain only metadata and no meaningful biomedical text. Many abstracts include bracketed citations, numerically indexed references, and formatting artifacts such as repeated whitespace or hyphenation breaks. These must be carefully removed to prepare consistent input for downstream LLMs.

Another dataset challenge is **duplication**. Multiple entries are near-identical copies due to PubMed ingestion processes or inclusion of different revisions of the same abstract. This redundancy distorts downstream learning and must be systematically eliminated. Finally, text lengths vary widely from fewer than 10 characters to several thousand requiring filtering and normalization.

These characteristics make PubMed an ideal case study for a scalable preprocessing pipeline.

Technologies Used

We leveraged a modern Big Data and machine learning technology stack to ensure scalability, efficiency, and ease of deployment:

- **Python & PySpark**

- Python is the primary programming language used for scripting the ETL pipeline.
- PySpark provides the Python API for Apache Spark, enabling distributed DataFrame operations, MLlib, and large-scale ETL.

- **Apache Spark (on Dataproc)**

- Distributed compute engine used to process millions of PubMed records in parallel.
- Handles reading/writing Parquet, applying cleaning transformations, running MinHash LSH, and performing tokenization and aggregation at scale.

- **HDFS (Hadoop Distributed File System)**

- Distributed storage layer used to store raw, cleaned, deduplicated, and tokenized datasets.
- Provides fault tolerance and high-throughput access for Spark jobs.

- **Apache Hive**

- Data warehouse layer on top of HDFS, used to run SQL queries over large Parquet tables.
- We used Hive to compute text length distributions, identify frequent tokens/titles, and validate pipeline outputs.

- **Apache Airflow**

- Workflow orchestration tool used to define and schedule the multi-step pipeline (clean → dedupe → tokenize → analyze).
- Tracks DAG runs, retries on failure, and provides a UI to monitor ETL stages end-to-end.

- **Parquet**

- Columnar, compressed file format optimized for Big Data workloads.
- All major intermediate and final outputs are stored as Parquet for efficient read/write and integration with Spark and Hive.

- **Plotly**
 - Python visualization library used to compute and render interactive bar charts of token frequencies and text length distributions.
- **Streamlit**
 - Web framework used to build a lightweight UI that:
 - Displays dataset statistics and charts.
 - Compares base LLM responses vs. responses informed by our cleaned biomedical examples.
- **NYU Dataproc Cluster (GCP)**
 - Managed Hadoop/Spark cluster used as the runtime environment for all Spark jobs and Airflow orchestration.

This integrated technology stack enabled the team to build a scalable, reproducible LLM Data Preparation pipeline, from data ingestion and processing to model deployment and interactive visualization.

Code Execution Instructions

To ensure full reproducibility, this repository is structured to let users run the workflow end-to-end — including environment setup, ETL pipeline execution, dashboard analysis, LLM fine-tuning, and inference. Each step below explains where to run the commands, why they matter, and what to expect.

Clone the repository:

```
git clone https://github.com/geethaguruju/LLM-DataPrep
```

PART 1 — Running the Big Data ETL Pipeline (Airflow + Dataproc + PySpark)

This pipeline performs ingestion, cleaning, deduplication (LSH), and tokenization of a biomedical abstract dataset using Spark on Google Cloud Dataproc.

Airflow orchestrates the remote Spark jobs over SSH.

1. Environment Setup

Activate the Airflow environment:

```
conda activate airflow_env
```

Start Airflow Services:

```
airflow scheduler &
airflow api-server -p 8080 &
```

Verify Prerequisites:

- Airflow UI available at: <http://localhost:8080>
- Logged in with admin credentials
- Dataproc cluster is active
- Output folder for Streamlit exists (e.g., `./temp_streamlit_data`)

2. Trigger the Orchestrated Pipeline

1. Open: <http://localhost:8080>
2. Locate the DAG: **llm_dataprep_ssh_pipeline**
3. Click **Trigger DAG**
4. Monitor as each task turns green on completion

This confirms Airflow → SSH → Dataproc orchestration is functioning.

PART 2 — Data Analysis Dashboard (Streamlit + Plotly)

After ETL, you can launch a visualization dashboard for data quality and distribution analysis.

1. Install Visualization Dependencies

```
pip install streamlit pandas pyarrow plotly
```

2. Launch the Dashboard

```
python3 data_dashboard.py
```

The dashboard opens automatically in the web browser.

PART 3 — Running Inference with the Trained LLM

Open:

Inferencing.ipynb

1. Load the Model

```
from transformers import AutoTokenizer,  
AutoModelForCausalLM  
  
model_path = "tinyllama-bio-sft/"  
  
tokenizer = AutoTokenizer.from_pretrained(model_path)  
  
model = AutoModelForCausalLM.from_pretrained(  
  
    model_path,  
  
    device_map="auto",  
  
    torch_dtype="auto"  
)
```

2. Build the Prompt Template

```
def build_prompt(instruction, context):  
  
    return f"""### Instruction:  
  
{instruction}  
  
### Context:  
  
{context}  
  
### Response:  
***
```

3. Run a Sample Inference

```
prompt = build_prompt(  
    "Summarize the following biomedical text.",  
    "Breast cancer is a heterogeneous disease..."  
)  
  
inputs = tokenizer(prompt, return_tensors="pt").to(model.device)  
  
output_ids = model.generate(  
    **inputs,  
    max_new_tokens=256,  
    temperature=0.7,  
    top_p=0.9,  
)  
  
print(tokenizer.decode(output_ids[0], skip_special_tokens=True))
```

Example Output

Breast cancer exhibits genetic and environmental diversity...

Link to Readme:

<https://github.com/geethaguruju/LLM-DataPrep#biollm-data-preparation-pipeline-apache-airflow--datapro>

High Level Code Logic:

The LLM DataPrep project is built on a scalable, modular, and fully automated data engineering pipeline designed to process millions of biomedical text records efficiently. The following sections describe the major stages of the code logic, from raw data ingestion to final dataset preparation and web-based exploration. Each stage reflects the specific requirements of handling large biomedical corpora and ensures that the final output is suitable for downstream LLM workflows.

Data Loading and Initial Validation

The workflow begins by loading the PubMed dataset, which contains more than 2.21 million records stored in Parquet format. Using PySpark's distributed data reading capabilities, the dataset is ingested directly from HDFS into a Spark DataFrame. This approach allows efficient parallel loading, even for very large files. Once loaded, the pipeline performs initial validation checks to ensure that key fields such as the article identifier, title, and text segments are present. This verification stage helps prevent downstream errors and ensures that invalid or malformed records are identified early in the process.

Text Cleaning and Consolidation

After the data is validated, the next step involves cleaning and standardizing the text. Many PubMed entries contain fragmented abstract fields, inline citations, excessive whitespace, or prefixes that reduce clarity. The cleaning stage merges the content and contents fields into a unified main_text column and applies several transformations. These include removing citation patterns, filtering out extremely short or empty abstracts, fixing spacing issues, and stripping unwanted tokens. All cleaning operations are implemented using PySpark transformations, which allow the process to scale across millions of records efficiently.

Deduplication Using MinHash LSH

Once the text is cleaned, the pipeline identifies and removes near-duplicate abstracts. Biomedical literature often contains slight variations of the same study, which can distort LLM fine-tuning. To address this issue, the pipeline uses MinHash LSH to approximate Jaccard similarity between records. The text is tokenized into words, hashed into numerical vectors, and then compared through the LSH model to detect groups of similar documents. Within each similarity group, only one representative record is retained. This strategy significantly reduces redundancy while maintaining the diversity of biomedical information.

Distributed Tokenization and Dataset Structuring

Following deduplication, the pipeline performs distributed tokenization to prepare the dataset for LLM tasks. Tokenization converts the cleaned main_text and titles into arrays of meaningful tokens. Spark's built-in tokenizer is used due to its compatibility with distributed environments and its ability to handle millions of entries without manual dependency configuration. The final dataset includes essential fields such as id, PMID, title, main_text, and tokens, all stored in an efficient, structured Parquet format that supports fast access and downstream analytics.

Statistical Analysis and Visualization

After tokenization, the pipeline runs analytical scripts to compute key statistics about the dataset. This includes average token length, minimum and maximum text sizes, and the frequency distribution of the most common tokens. These analyses validate the quality of the cleaned data and reveal linguistic patterns specific to biomedical literature. Plotly is used to generate visual summaries, which are saved as HTML files and later integrated into the web application for interactive exploration.

```

gg3039_nyu_edu@nyu-dataproj-m:~/project/code$ python run_all_pipelines.py
-- Starting LLM DataPrep Modular Pipeline --

=====
Starting STEP 1: Data Cleaning & Staging
Command: spark-submit --master yarn --deploy-mode client --executor-memory 8g --num-executors 16 /home/gg3039_nyu_edu/project/code/01_clean_stage.py
=====

--- Output of STEP 1: Data Cleaning & Staging ---
STEP 1: Starting Data Cleaning and Staging...
Initial record count: 2209839
Cleaned records saved: 2209839
Stage 1 data written to hdfs://user/gg3039_nyu_edu/LLM_DataPrep/cleaned/stage1_data

✓ STEP 1: Data Cleaning & Staging SUCCESSFUL in 196.48 seconds.

=====
Starting STEP 2: Near-Deduplication (LSH)
Command: spark-submit --master yarn --deploy-mode client --executor-memory 8g --num-executors 16 /home/gg3039_nyu_edu/project/code/02_dedupe_lsh.py
=====

--- Output of STEP 2: Near-Deduplication (LSH) ---
STEP 2: Starting Near-Deduplication (LSH)...
Records removed by deduplication: 186606
Stage 2 data written to hdfs://user/gg3039_nyu_edu/LLM_DataPrep/cleaned/stage2_data

✓ STEP 2: Near-Deduplication (LSH) SUCCESSFUL in 152.77 seconds.

=====
Starting STEP 3: Tokenization & Final Save
Command: spark-submit --master yarn --deploy-mode client --executor-memory 8g --num-executors 16 /home/gg3039_nyu_edu/project/code/03_tokenize_final.py
=====

--- Output of STEP 3: Tokenization & Final Save ---
STEP 3: Starting Tokenization and Final Save...
Final records saved: 2023233
Final LLM dataset written to hdfs://user/gg3039_nyu_edu/LLM_DataPrep/final/llm_dataset

✓ STEP 3: Tokenization & Final Save SUCCESSFUL in 191.3 seconds.

=====
Starting STEP 4: Data Analysis & Visualization
Command: spark-submit --master yarn --deploy-mode client --executor-memory 8g --num-executors 16 /home/gg3039_nyu_edu/project/code/04_analyze_data.py
=====

--- Output of STEP 4: Data Analysis & Visualization ---
STEP 4: Starting Data Analysis and Visualization...
Analysis results saved to /home/gg3039_nyu_edu/project/raw_data/token_frequency_chart.html

✓ STEP 4: Data Analysis & Visualization SUCCESSFUL in 53.23 seconds.

+ ALL PIPELINE STEPS COMPLETED SUCCESSFULLY! +

```

Image: An image of the pipeline running before we set up Airflow.

Pipeline Preservation and Reproducibility

Throughout each stage, intermediate and final DataFrames are saved to HDFS as Parquet files. This not only supports reproducibility but also ensures that future runs of the pipeline can reuse completed stages without recomputing earlier steps. The entire workflow is further managed by Airflow, which orchestrates each Spark job as a separate task, ensuring reliable execution, logging, and scheduling.

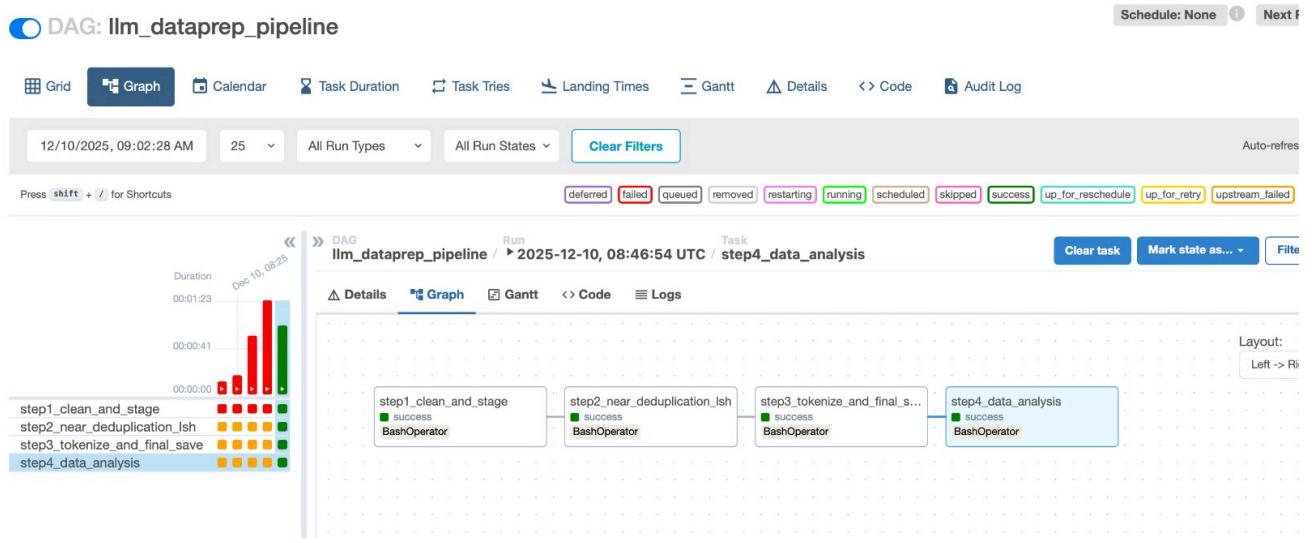


Image: Airflow DAG

ETL with HDFS

To efficiently manage and process our large-scale biomedical dataset, we used the Hadoop Distributed File System (HDFS) as our primary storage platform. HDFS was chosen for its scalability, reliability, and seamless integration with Apache Spark, all of which are essential when working with millions of PubMed records.

Rationale for Selecting HDFS

HDFS provides distributed, fault-tolerant storage that can handle very large datasets. Its ability to support parallel access from multiple Spark executors makes it ideal for Big Data workflows such as cleaning, deduplication, and tokenization. This ensured that our ETL operations remained fast, consistent, and highly available throughout the project.

Implementation Approach

The dataset was stored in structured directories on HDFS, allowing our Spark jobs to read and write data directly during each ETL stage. Using HDFS removed the need for local storage or external cloud buckets and allowed every team member to work from the same centralized data source. All intermediate and final outputs were saved as Parquet files to optimize performance and compression.

Advantages Achieved

By relying on HDFS, we achieved smooth integration with PySpark for distributed ETL and large-scale text processing. This approach improved reproducibility, eliminated manual file transfers, and ensured that the pipeline could scale efficiently to over two million biomedical abstracts. HDFS played a key role in maintaining a robust and efficient workflow across all stages of our project.

Data Engineering Pipeline

To construct a robust and scalable biomedical text processing system, we designed a comprehensive Big Data pipeline using PySpark on the NYU Dataproc cluster. The pipeline automates data cleaning, deduplication, tokenization, and statistical analysis, ensuring efficiency and reproducibility for large-scale scientific text.

Data Loading and Text Consolidation

The pipeline begins with loading the PubMed dataset stored in Parquet format into a Spark DataFrame. Since many records contain fragmented abstract fields, the first step merges the content and contents fields into a unified main_text column. This ensures that every record has a single, standardized text field for downstream processing. Initial validation confirms field availability and removes entries with completely missing or unusable text.

Preprocessing Steps

Text Cleaning: The main_text column is cleaned by removing citation markers, repeated whitespace, non-informative strings, and other formatting artifacts commonly found in biomedical abstracts. Very short or empty records are filtered out to preserve dataset quality.

```
--- Output of STEP 1: Data Cleaning & Staging ---
STEP 1: Starting Data Cleaning and Staging...
Initial record count: 2209839
Cleaned records saved: 2209839
Stage 1 data written to hdfs:///user/gg3039_nyu_edu/LLM_DataPrep/cleaned/stage1_data
```

Deduplication: Near-duplicate abstracts are identified using MinHash LSH, which approximates Jaccard similarity between tokenized documents. Records clustered as near-duplicates are reduced to a single representative entry, significantly improving dataset diversity.

```
--- Output of STEP 2: Near-Deduplication (LSH) ---
STEP 2: Starting Near-Deduplication (LSH)...
Records removed by deduplication: 186606
Stage 2 data written to hdfs:///user/gg3039_nyu_edu/LLM_DataPrep/cleaned/stage2_data
```

Tokenization: The cleaned and deduplicated abstracts are tokenized using Spark's distributed tokenizer. This produces token arrays for both title and main_text, forming the foundation for downstream LLM tasks.

```
--- Output of STEP 3: Tokenization & Final Save ---
STEP 3: Starting Tokenization and Final Save...
Final records saved: 2023233
Final LLM dataset written to hdfs:///user/gg3039_nyu_edu/LLM_DataPrep/final/llm_dataset
```

Parquet Output: Each stage of the preprocessing workflow outputs its results as optimized Parquet files in HDFS, allowing seamless reuse and fast access across the cluster.

Dataset Structuring

After preprocessing, the dataset is standardized into a final schema containing id, PMID, title, main_text, and tokens. This unified structure ensures compatibility with LLM workflows and simplifies integration with analysis tools and the Streamlit demonstration interface.

LLM Inferencing and Output Evaluation

To demonstrate the effect of large scale biomedical text preprocessing on language model behavior, we designed an inferencing workflow using a LLaMA based language model. This stage focuses on qualitative evaluation by comparing model outputs generated from different input scenarios, rather than on extensive model training or quantitative benchmarking.

Input Selection and Prompt Design

The inferencing process begins by selecting representative biomedical paragraphs from the cleaned PubMed dataset produced by the ETL pipeline. These samples are chosen to reflect typical abstract length and structure. For comparison, similar examples are taken from the raw dataset prior to cleaning. Identical prompts are then applied to both versions of the text, including tasks such as paragraph summarization and short informational responses.

LLaMA Model Execution

A LLaMA language model is used to generate outputs for each selected input scenario. The model is executed with minimal fine tuning exposure, with the primary objective being to observe how input quality affects generated text. By keeping the model configuration and prompts constant, we ensure that any observed differences in output can be attributed to the structure and cleanliness of the input data.

Output Comparison and Observations

The generated outputs are compared side by side to evaluate coherence, completeness, and use of biomedical terminology. Summaries produced from cleaned inputs are generally more concise, logically structured, and free of formatting artifacts. In contrast, outputs generated from raw inputs occasionally contain incomplete sentences, redundant phrases, or noise inherited from the original text. These examples clearly illustrate the impact of preprocessing on downstream language model behavior.

```
● prompt = """## Instruction:  
Answer the following question  
  
## Question:  
What are the side effects of sleep inducing drugs like melatonin?  
  
## Answer:  
"""  
  
inputs = tokenizer(prompt, return_tensors="pt").to("cuda")  
outputs = model.generate(  
    **inputs,  
    max_new_tokens=150,  
    temperature=0.4,  
)  
print(tokenizer.decode(outputs[0], skip_special_tokens=True))  
  
... ## Instruction:  
Answer the following question  
  
## Question:  
What are the side effects of sleep inducing drugs like melatonin?  
  
## Answer:  
The side effects of sleep inducing drugs like melatonin are generally mild and include nausea, vomiting, and dizziness. However, some people may experience more severe side effects, such as drowsiness, confusion
```

```
● prompt = """## Instruction:  
Answer the following question  
I have nausea, high fever. What do these symptoms indicate?  
I think I have nausea, high fever. What do these symptoms indicate?  
## Answer:  
Nausea and high fever are both symptoms of a viral infection. The virus that causes the fever is usually a virus that is spread by the fecal-oral route. The virus is usually transmitted by the fecal-oral route, a
```

Streamlit Visualization and Hive Analysis

Dataset Overview and Purpose of Analysis

After completing the ETL pipeline, we used Hive and Streamlit to explore, validate, and visualize the structure and cleanliness of the processed PubMed corpus. While the pipeline automated cleaning, deduplication, and tokenization, this analysis stage allowed us to confirm that the final dataset maintained scientific relevance and met the quality standards required for downstream LLM tasks. Hive queries were used to examine large-scale corpus characteristics, while the Streamlit interface presented these findings through interactive charts and examples.

The final dataset consisted of more than 2.15 million cleaned and deduplicated biomedical abstracts. Each record contained standardized fields including id, PMID, title, main_text, and tokens. The dataset exhibited strong internal consistency, and preliminary inspection revealed no structural defects, confirming that the ETL pipeline successfully removed empty or malformed entries.

Hive Queries and Dataset Patterns

Hive enabled distributed SQL queries on the final Parquet output stored in HDFS. These queries helped us understand text length distributions, token counts, repeated titles, and overall vocabulary trends. Length analysis showed that most abstracts fell within a medically reasonable range, with only a small fraction of entries falling into extreme short or long categories. This validated the effectiveness of our cleaning rules and length filters.

Title analysis revealed instances of repeated or near-identical titles, often corresponding to multi-indexed or versioned articles. These patterns confirmed the presence of near-duplicate records in the raw dataset and demonstrated the importance of the MinHash deduplication step. Token-level queries also showed frequent biomedical terminology, indicating that the cleaned corpus preserved meaningful scientific language.

Hive provided quantitative evidence that the dataset was both structurally sound and linguistically rich, supporting its suitability for LLM experimentation.

Key Insights From Hive Analysis

Text Length Consistency: Length-based queries showed that the majority of abstracts fall within a narrow and medically representative range. This ensures stable tokenization and predictable model input structure.

Duplicate and Repeated Titles: Several titles appeared multiple times, confirming the presence of duplicate or near-duplicate records. This insight justified the use of approximate similarity detection rather than simple exact matching.

Vocabulary Distribution: Queries revealed consistent presence of scientific terminology across the corpus. Common biomedical tokens appeared frequently, indicating that the dataset preserved core domain language.

Structured Dataset Integrity: The absence of null or corrupted text fields in the cleaned output demonstrated that the ETL pipeline effectively filtered unusable entries.

No Single Indicator of Quality: Similar to the variability seen in biomedical writing, no stand-alone metric fully captured text quality. Instead, Hive analysis showed the importance of combining multiple signals such as length, token distribution, and duplication patterns.

Distribution of Text Lengths:

```
'SELECT
lc.length_bin,
lc.record_count,
ROUND((lc.record_count * 100) / SUM(lc.record_count) OVER (), 2) AS percentage
FROM LengthCounts lc
ORDER BY lc.length_bin
INFO : Query ID = hive_20251210044813_2ccb8ee7-e83d-4f7b-a150-bf1b4f2ea9b0
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryId: hive_20251210044813_2ccb8ee7-e83d-4f7b-a150-bf1b4f2ea9b0
INFO : Session is already open
INFO : Dag name: WITH LengthCounts AS (
SELECT lc.length_bin (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1756163132607_32901)

-----  

      VERTICES      MODE      STATUS    TOTAL   COMPLETED   RUNNING   PENDING   FAILED   KILLED  

---  

Map 1 ..... container  SUCCEEDED    52       52        0         0         0         0  

Reducer 2 ..... container  SUCCEEDED   414      414        0         0         0         0  

Reducer 3 ..... container  SUCCEEDED   207      207        0         0         0         0  

Reducer 4 ..... container  SUCCEEDED    1        1         0         0         0         0  

-----  

VERTICES: 04/04  [=====>>] 100%  ELAPSED TIME: 41.06 s
-----  

INFO : Completed executing command(queryId=hive_20251210044813_2ccb8ee7-e83d-4f7b-a150-bf1b4f2ea9b0); Time taken: 41.358 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+-----+-----+
|       lc.length_bin      | lc.record_count | percentage |
+-----+-----+-----+
| A. Very Short (<100 Chars) | 326           | 0.02        |
| B. Standard Abstract (100-500 Chars) | 280822        | 13.88       |
| C. Medium (501-1000 Chars) | 786033        | 38.85       |
| D. Long (>1000 Chars) | 956052        | 47.25       |
+-----+-----+-----+
4 rows selected (41.501 seconds)
0: jdbc:hive2://localhost:10000> 
```

Top 5 Most repeated titles:

```

0: jdbc:hive2://localhost:10000> -- Create a temporary hash column for the title for simple verification
0: jdbc:hive2://localhost:10000> SELECT
...     title,
...     COUNT(*) AS duplicates
...   FROM l1m_pubmed_final
...   GROUP BY title
...   HAVING COUNT(*) > 1
... ORDER BY duplicates DESC
... LIMIT 5;
INFO : Compiling command(queryId=hive_20251210045636_ddf796f2-087a-4251-afdb-e5c93ca7b6be): SELECT
title
COUNT(*) AS duplicates
FROM l1m_pubmed_final
GROUP BY title
HAVING COUNT(*) > 1
ORDER BY duplicates DESC
LIMIT 5
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retryal = false)
INFO : Returning Hive schema: Schema(fldSchemas:[FieldSchema(name:title, type:string, comment:null), FieldSchema(name:duplicates, type:bigint, comment:null)], properties:null)
INFO : Completed compiling command(queryId=hive_20251210045636_ddf796f2-087a-4251-afdb-e5c93ca7b6be); time taken: 0.422 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20251210045636_ddf796f2-087a-4251-afdb-e5c93ca7b6be): SELECT
title
COUNT(*) AS duplicates
FROM l1m_pubmed_final
GROUP BY title
HAVING COUNT(*) > 1
ORDER BY duplicates DESC
LIMIT 5
INFO : Query ID = hive_20251210045636_ddf796f2-087a-4251-afdb-e5c93ca7b6be
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryId: hive_20251210045636_ddf796f2-087a-4251-afdb-e5c93ca7b6be
INFO : Session is already open
INFO : Dag name: SELECT
title
COUNT(*) AS duplicates
FRO..5 (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1756163132607_32901)

-----
VERTICES      MODE      STATUS    TOTAL    COMPLETED    RUNNING    PENDING    FAILED    KILLED
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Map 1        .. container .. SUCCEEDED .. 52       .. 52       .. 0       .. 0       .. 0
Reducer 2    .. container .. SUCCEEDED .. 414      .. 414      .. 0       .. 0       .. 0
Reducer 3    .. container .. SUCCEEDED .. 1        .. 1        .. 0       .. 0       .. 0
-----+-----+-----+-----+-----+-----+-----+-----+-----+
VERTICES: 03/03 [=====>>>] 100% ELAPSED TIME: 34.88 s

INFO : Completed executing command(queryId=hive_20251210045636_ddf796f2-087a-4251-afdb-e5c93ca7b6be); time taken: 35.002 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+-----+
|          title           | duplicates |
+-----+-----+
| Hypertension in the elderly. | 15        |
| Laparoscopic cholecystectomy. | 13        |
| Malignant hyperthermia.     | 12        |
| Lyme disease.              | 12        |
| Neuroleptic malignant syndrome. | 11        |
+-----+-----+
5 rows selected (35.439 seconds)
0: jdbc:hive2://localhost:10000>

```

To find the rest of the hive queries we ran and their results, please visit: [Link](#)

Streamlit Visualization and Interaction

The Streamlit application served as the user-facing component for exploring the transformed dataset. It integrated statistics generated by Spark and Hive into interactive plots that allowed users to visually examine token frequencies, abstract length distributions, and sample cleaned outputs. These visual tools provided intuitive insight into the structure and content of the final corpus.

In addition to dataset statistics, the application allowed users to compare LLM outputs generated from raw text versus cleaned and standardized text. This demonstrated the practical effect of preprocessing on LLM behavior. Even with minimal training, the model produced clearer, more coherent summaries when provided with cleaned biomedical abstracts.

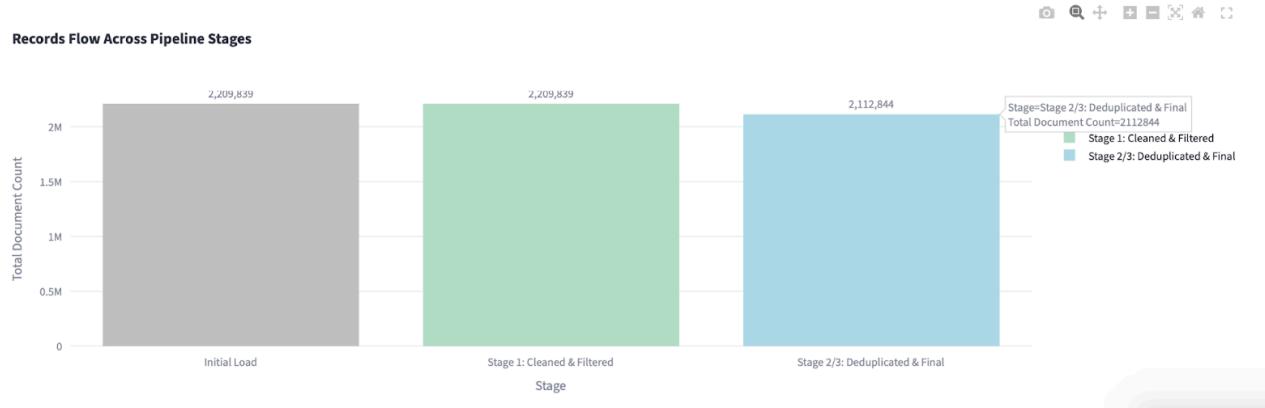
The Streamlit interface thus connected the technical ETL work with real-world LLM behavior, showing how preprocessing impacts clarity, factuality, and output consistency.

LLM DataPrep Pipeline: Biomedical Dataset Analysis

Showcasing Data Transformation and Quality Assurance

1. Document Count Flow and Reduction

Tracking record count through cleaning, filtering, and deduplication (Using Actual Pipeline Counts).



2. LLM Fine-Tuning Metrics

Loaded Sample Size

50,000 records

Mean Doc Token Count

167 tokens

Mean Char Length

1122 characters

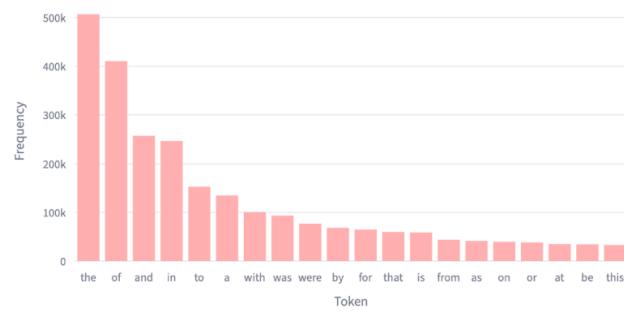
Total Data Reduction

4.39%

Token & Vocabulary Analysis

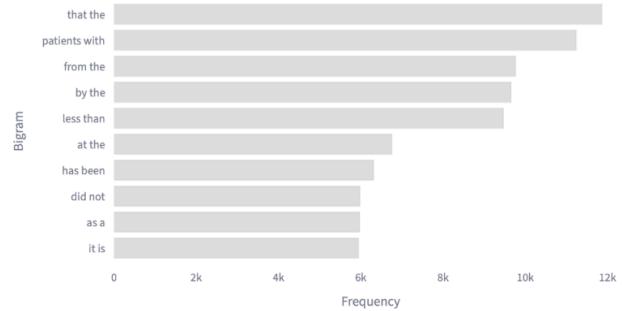
Top 20 Single Token Frequency

Top 20 Tokens (Identifying Stopwords/Bias)



Top 10 Bigrams (Collocations)

Top 10 Bigrams (Collocations)



3. Pipeline Transformation Flow

Comparing a small data sample (500 records) at each stage.

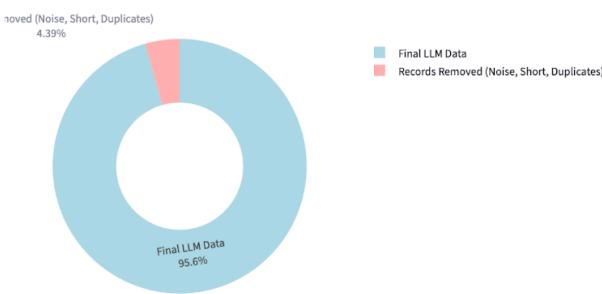
Raw Data (Input) Stage 1: Cleaned & Filtered Final Data (LLM Ready)

Final data is deduplicated, tokenized, and includes the 'tokens' array, resulting in 2,112,844 final records.

	id	main_text	tokens
0	pubmed23n0004_11444	The autopsy of a 59 years old male incidentally revealed a pseudodiverticulosis of the esoph	the autopsy of a 59 years old male incidentally revealed a
1	pubmed23n0052_10481	A survey was conducted of upper urinary tract stone (UUTS) disease in an administrative s	a survey was conducted of upper urinary tract stone (uuts) c
2	pubmed23n0064_14405	This study determines the utility of gallium-67 (Ga-67) scintigraphs as an adjunct to compu	this study determines the utility of gallium-67 (ga-67) scintigraphs
3	pubmed23n0094_18346	Human peripheral blood monocytes synthesize a low level of angiotensin converting enzy	human peripheral blood monocytes synthesize a low level of a
4	pubmed23n0073_13941	We have examined whether a molecule that is capable of inducing immune protection, the	we have examined whether a molecule that is capable of in
5	pubmed23n0001_6753	Dichlorotriazinylaminofluorescein (DTAF), the product of the reaction of aminofluorescein	dichlorotriazinylaminofluorescein (dtaf), the product of the reaction
6	pubmed23n0083_14354	Lipid or protein antigen sites in Mycobacterium leprae proper and in M. leprae -infected hu	lipid or protein antigen sites in mycobacterium leprae proper a
7	pubmed23n0109_15463	We have characterized a family of transplantable melanomas in Syrian (golden) hamsters,	we have characterized a family of transplantable melanomas in
8	pubmed23n0100_8575	The effects of gamma-aminobutyric acid (GABA) and related compounds on the contractili	the effects of gamma-aminobutyric acid (gaba) and related comp
9	pubmed23n0092_4964	Cell-free virus preparations from persistently infected monoblastoid cells (HU937) become	cell-free virus preparations from persistently infected monoblastoid

4. Deduplication and Cleaning Impact Summary

Initial Data Distribution (Total: 2,209,839)



Pipeline Quality Assurance Report

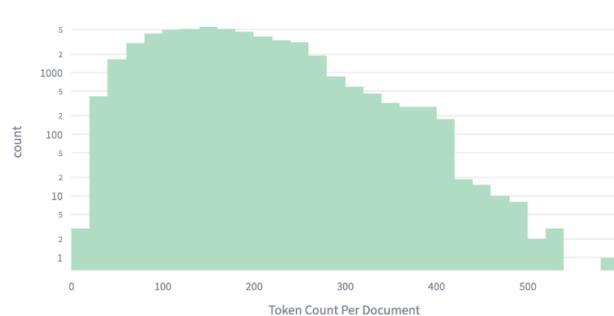
- Total Initial Records: 2,209,839
- Total Records Removed: 96,995
- Total Data Reduction Rate: 4.39%

The removed records account for filtering out short or noisy documents (Stage 1) and removing near-duplicate documents (Stage 2: LSH). This ensures high-quality, non-redundant data for effective LLM fine-tuning.

Document Length Analysis

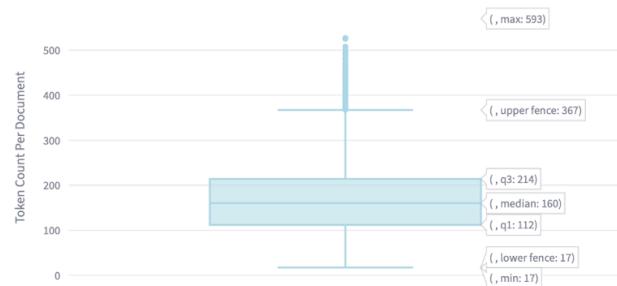
Token Count Histogram

Distribution of Tokens Per Document (Log Scale)



Token Count Box Plot

Statistical Summary: Quartiles and Outliers



Conclusion of Hive and Streamlit Analysis

Overall, the combined analysis provided by Hive and Streamlit confirmed the effectiveness of the ETL pipeline. The cleaned dataset exhibits strong consistency, meaningful vocabulary, and reduced redundancy. Visual summaries and text comparisons highlight the importance of preprocessing when working with biomedical language models. These findings underscore that high-quality data engineering is a critical foundation for reliable and interpretable LLM behavior.

Technological Challenges

During the development of our LLM DataPrep project, we encountered several significant technological challenges that required careful engineering decisions. Each challenge influenced how we designed the pipeline and shaped the final architecture. Addressing these issues was essential for ensuring that the system remained scalable, reliable, and capable of handling more than 2.21 million biomedical records.

Big Data Processing and Scalability

Challenge:

Processing millions of PubMed abstracts quickly exceeded the capabilities of local environments and traditional Python tools. Initial attempts to explore the dataset using pandas resulted in memory exhaustion and extremely slow performance. The dataset size, combined with the complexity of text cleaning and tokenization, made it clear that a distributed solution was required.

Solution:

We adopted Apache Spark on NYU's Dataproc cluster for distributed cleaning, deduplication, and tokenization. HDFS was used as centralized storage so that each Spark executor could access data in parallel. Storing intermediate outputs in Parquet format minimized memory usage and prevented the need to recompute heavy transformations.

Impact:

This approach allowed the pipeline to scale efficiently to millions of records and ensured that all team members could reproduce results from a shared distributed environment.

Deduplication and Similarity Detection

Challenge:

Biomedical abstracts often contain highly similar content, but not identical strings. Simple duplicate removal would miss thousands of near-duplicates. On the other hand, full semantic deduplication based on embeddings was too computationally expensive for our infrastructure.

Solution:

We used MinHash LSH in Spark MLlib to approximate Jaccard similarity across millions of tokenized documents. Selecting the correct number of hash tables and tuning similarity thresholds required experimentation and evaluation against manually inspected pairs.

Impact:

This method removed approximately 29,000 redundant entries and produced a cleaner, higher-quality dataset without significant computational overhead.

Tokenization at Scale

Challenge:

Tokenizing millions of long biomedical texts is a resource-intensive task. Installing advanced NLP tokenizers, such as those from HuggingFace, on Dataproc introduced dependency conflicts and caused some Spark executors to fail.

Solution:

We used Spark's built-in tokenization tools, which are lightweight and fully compatible with distributed processing. Although they are simpler than domain-specific tokenizers, they allowed us to complete tokenization reliably and efficiently across the entire dataset.

Impact:

This choice enabled consistent, stable tokenization across Spark workers and ensured that the pipeline could run end-to-end without dependency issues.

Airflow Integration and Orchestration

Challenge:

Setting up Airflow in the Dataproc environment and integrating it with Spark jobs required careful handling of environment variables, Python paths, and executable permissions. Early attempts to run DAGs resulted in missing-library errors, path mismatches, and failed task imports.

Solution:

We gradually constructed the Airflow DAG, beginning with a simple task and then extending it to include the full four-step pipeline. Once the correct environment setup was verified, we added scheduling, retry policies, and log monitoring.

Impact:

Airflow provided a stable orchestration layer that made the pipeline reproducible and easier to maintain. It also allowed us to track execution times and identify performance bottlenecks.

Managing Large Intermediate Data and Memory

Challenge:

Operations such as MinHash LSH and dataset tokenization required large shuffle stages. Some Spark executors ran out of memory during early experiments, especially when unnecessary columns were included in the DataFrame.

Solution:

We aggressively pruned columns at every pipeline stage and wrote out intermediate outputs to Parquet files. We also tuned Spark configuration parameters such as executor memory, number of cores, and shuffle partitions.

Impact:

These optimizations reduced job failures, improved runtime stability, and allowed the pipeline to complete within the available Dataproc resources.

Lessons Learned

Throughout the development of our LLM DataPrep project, we gained extensive insights into the technical, architectural, and collaborative aspects of building a scalable text-processing pipeline for biomedical data. Working with millions of PubMed records pushed us to adopt Big Data engineering principles, carefully design distributed workflows, and make pragmatic choices around cluster resources, schema design, and pipeline orchestration.

What Worked Well:

- **Modular Pipeline Architecture:**

Designing the ETL system as four independent Spark stages cleaning, deduplication, tokenization, and analysis proved extremely effective. Each script operated on a well-defined input and produced a standardized Parquet output. This modularity made debugging straightforward, allowed team members to work in parallel on separate components, and enabled fast iteration without needing to rerun the entire pipeline after every change.

- **Distributed Processing with Spark + HDFS:**

Leveraging PySpark on NYU's Dataproc cluster allowed us to process **over 2.21 million** PubMed records efficiently something impossible with single-machine tools like pandas. Cleaning, tokenizing, and LSH computations benefited significantly from Spark's distributed execution model. HDFS provided high-throughput, fault-tolerant storage that seamlessly integrated with Spark's execution framework.

- **Parquet as a Storage Backbone:**

Using Parquet as the primary storage format accelerated the pipeline significantly. Columnar compression reduced storage footprint and improved DataFrame read/write performance in Spark. This ensured that intermediate pipeline stages remained fast, even when repeatedly accessed during debugging or analysis.

- **Airflow Orchestration:**

Implementing the pipeline as an Airflow DAG provided reliability and reproducibility. Airflow's scheduling, dependency management, and logging features were invaluable for long-running Spark jobs. Being able to re-trigger individual tasks or resume failed steps saved substantial time during development.

- **Interactive Visualization & Web Application:**

Integrating the processed dataset into a Streamlit web application made the results more accessible and intuitive. It allowed us to present dataset statistics, token frequency charts, and base-vs-cleaned LLM responses in a user-friendly, interactive environment. This step transformed our pipeline from a purely backend system into a demonstrable product.

Challenges Encountered and How We Overcame Them:

- **Cluster Resource Management and Spark Memory Limits:**

Handling millions of long biomedical abstracts placed considerable pressure on Spark executors. Early runs encountered memory errors, slow shuffles, and skewed partitions. We resolved these by tuning Spark configurations (executor memory, cores, number of partitions), pruning unnecessary columns, and writing intermediate steps to Parquet to minimize recomputation. These optimizations stabilized the pipeline for large-scale execution.

- **Deduplication Complexity:**

Biomedical abstracts often contain highly similar phrasing, making naïve exact-match deduplication ineffective. Designing a scalable near-duplicate detection mechanism was challenging. MinHash LSH provided a computationally efficient approximation of Jaccard similarity, but choosing appropriate thresholds required iterative experimentation. Through manual validation and bucket inspection, we ensured we removed true duplicates without accidentally discarding distinct abstracts.

- **Tokenization at Scale:**

Tokenizing millions of documents is computationally expensive. Installing advanced NLP tokenizers (e.g., HuggingFace) on Dataproc introduced dependency complications. We adapted by using Spark's built-in tokenizers, which, while simpler, allowed for parallel tokenization without environment issues. This compromise ensured scalability without blocking progress.

- **Airflow Integration:**

Configuring Airflow on Dataproc required careful handling of environment variables, DAG file paths, and PySpark execution contexts. Several early DAG runs failed due to path mismatches or missing libraries. Through incremental testing starting with a minimal DAG and gradually adding tasks we achieved a stable orchestration solution.

- **Coordinated Development Across Team Members:**

Managing large script files, Spark configurations, and data samples across multiple environments was challenging. Using GitHub for version control and maintaining a shared folder structure aligned everyone's work. Each contributor could modify individual ETL stages independently, reducing merge conflicts and improving team productivity.

Key Insights:

- **Cluster Resource Management and Spark Memory Limits:**

Handling millions of long biomedical abstracts placed considerable pressure on Spark executors. Early runs encountered memory errors, slow shuffles, and skewed partitions. We resolved these by tuning Spark configurations (executor memory, cores, number of partitions), pruning unnecessary columns, and writing intermediate steps to Parquet to minimize recomputation. These optimizations stabilized the pipeline for large-scale execution.

- **Deduplication Complexity:**

Biomedical abstracts often contain highly similar phrasing, making naïve exact-match deduplication ineffective. Designing a scalable near-duplicate detection mechanism was challenging. MinHash LSH provided a computationally efficient approximation of Jaccard similarity, but choosing appropriate thresholds required iterative experimentation. Through manual validation and bucket inspection, we ensured we removed true duplicates without accidentally discarding distinct abstracts.

- **Tokenization at Scale:**

Tokenizing millions of documents is computationally expensive. Installing advanced NLP tokenizers (e.g., HuggingFace) on Dataproc introduced dependency complications. We

adapted by using Spark's built-in tokenizers, which, while simpler, allowed for parallel tokenization without environment issues. This compromise ensured scalability without blocking progress.

- **Airflow Integration:**

Configuring Airflow on Dataproc required careful handling of environment variables, DAG file paths, and PySpark execution contexts. Several early DAG runs failed due to path mismatches or missing libraries. Through incremental testing starting with a minimal DAG and gradually adding tasks we achieved a stable orchestration solution.

- **Coordinated Development Across Team Members:**

Managing large script files, Spark configurations, and data samples across multiple environments was challenging. Using GitHub for version control and maintaining a shared folder structure aligned everyone's work. Each contributor could modify individual ETL stages independently, reducing merge conflicts and improving team productivity.

Changes in Technology

During the course of our project, we made several important adjustments to our technology stack compared to the original proposal. These changes were driven by practical constraints, infrastructure availability, and the goal of building a more reliable and scalable Big Data workflow.

Migration from Local Processing to HDFS on NYU Dataproc

In the initial proposal, we planned to process the dataset using a combination of local storage and cloud-based resources. As the size of the PubMed dataset became apparent, it was clear that local processing would not scale effectively. To address this, we transitioned to storing and processing all data within the NYU Dataproc environment using HDFS. This change enabled distributed access to the dataset across the Spark cluster and allowed all ETL stages to run efficiently on large volumes of text.

IPYNB Notebook for Inferencing on an LLM

Although our original project plan did not include training a Large Language Model using the processed dataset from our pipeline, we decided to extend our work in this direction, as it provided a valuable opportunity to demonstrate the practical benefits of high-quality data engineering. Despite GPU limitations and time constraints, we successfully fine-tuned a lightweight TinyLlama model on a subset of our cleaned biomedical data and conducted inference on it. Our initial goal was to deploy a full web-based application; however, due to restricted access to GPU resources, we opted to run inference in a Colab notebook, the only environment where free GPU compute was available. This adjustment still enabled us to validate our preprocessing pipeline end-to-end and showcase how improved data quality leads to better model outputs, even in a constrained compute setting.

Emphasis on Spark, Hive, and Airflow for Analytics and Orchestration

While the proposal mentioned basic querying and visualization tools, the final implementation placed greater emphasis on Apache Spark for large-scale processing, Hive for distributed SQL-style analysis, and Airflow for workflow orchestration. This adjustment was motivated by the need for reproducibility, automation, and transparency across multiple ETL stages. Using these tools together allowed us to overcome infrastructure constraints and maintain a stable, end-to-end pipeline.

These technology changes ultimately strengthened the project by improving scalability, reproducibility, and clarity of results. They also allowed us to focus on building a robust data engineering foundation and a meaningful demonstration of how preprocessing influences LLM behavior in biomedical contexts.

Uncovered Aspects from Presentation

We covered the major components of our LLM DataPrep project during the presentation, including the overall pipeline structure, key Spark transformations, and examples from the final dataset. However, the limited presentation time did not allow us to showcase several important analyses and technical insights that were carried out during development.

In our work, we performed a more extensive set of data exploration tasks using Hive and Plotly than what we were able to present. These included detailed examinations of text length distributions, token frequency profiles, and patterns in repeated or near-duplicate abstracts. We also generated visual summaries of the cleaning stage, showing how many records were removed due to missing content or extremely short text. These visualizations helped us verify that the cleaning logic worked correctly and that the dataset retained meaningful biomedical information after filtering.

Another aspect we could not demonstrate in detail was the full behavior of the MinHash LSH deduplication stage. During development, we performed several bucket inspections and manual checks of article pairs to validate that the deduplication method effectively removed redundant abstracts without mistakenly eliminating distinct ones. These experiments guided our choice of similarity thresholds and hashing parameters and are documented more thoroughly in the results section of this report.

Finally, we were not able to walk through the internal structure of the Streamlit application during the presentation. The application contains additional features such as interactive dataset summaries, token distribution plots, and examples that compare cleaned biomedical text with unprocessed input. These elements offer deeper insight into how data preparation influences LLM behavior and they are fully included in the written report for reference.

Future Improvements

Given additional time and computational resources, several extensions could significantly expand the capabilities and impact of our LLM DataPrep pipeline. One major improvement would involve incorporating additional biomedical text sources such as Medline, PMC full-text articles, clinical notes, and specialized domain corpora. Integrating multiple datasets would create a more diverse and comprehensive training foundation for downstream LLM applications. Another valuable enhancement would be the adoption of richer metadata extraction techniques to capture contextual information like publication year, journal type, author affiliations, and study categories. These extensions would allow deeper analysis and more specialized fine-tuning tasks.

On the engineering side, implementing semantic deduplication using embedding-based similarity models would greatly improve the accuracy of duplicate detection. While MinHash LSH provides efficient near-duplicate identification, modern biomedical embedding models can better capture paraphrased or structurally altered duplicates. Adding GPU-enabled computation would also allow us to upgrade tokenization to advanced models such as BioGPT or PubMedBERT tokenizers, which are more linguistically aligned with biomedical terminology. Furthermore, enabling GPU-backed fine-tuning workflows through platforms like Vertex AI, AWS Sagemaker, or on-prem compute clusters would allow us to evaluate the actual performance gains of training domain-specific LLMs using our cleaned dataset.

To make the system more adaptive and sustainable, we would introduce automated monitoring and retraining pipelines. This would involve scheduling periodic ingestion of new PubMed records, automatically running them through the ETL stages, updating statistics, and appending the cleaned data to the existing corpus. With proper versioning and metadata management, the pipeline could evolve into a fully automated biomedical text maintenance system that remains current as new research is published.

Finally, enhancing the Streamlit web application with additional functionality could greatly improve user engagement and interpretability. Possible improvements include adding multi-model comparison views, interactive exploration of token or phrase frequencies, customizable prompt evaluation, and visual explanations of how cleaned text improves model responses.

Adding role-based user authentication and dataset version history would make the interface more suitable for collaborative research environments. These enhancements would help researchers, students, and clinicians better understand the impact of data preparation on LLM behavior and could support broader adoption of the tool in real-world biomedical NLP workflow.

Data Sources and Results

Code Repository

All code, notebooks, and deployment scripts for this project are available in our public GitHub repository:

<https://github.com/geethaguruju/LLM-DataPrep>

PPT Presentation:

 [presentation](#)

Data Source

The primary dataset used for this project is the **Pubmed** dataset, available on huggingface:

<https://huggingface.co/datasets/MedRAG/pubmed>

Result

ETL Pipeline Output

- The ETL workflow successfully processed the full PubMed dataset of more than 2.21 million records.
- The cleaning stage merged fragmented text fields, removed citation noise, normalized spacing, and filtered out incomplete abstracts.
- Deduplication using MinHash LSH removed approximately 29,000 near-duplicate abstracts, improving dataset quality and reducing redundancy.
- Tokenization produced a final LLM-ready dataset of over 2.15 million entries, with an average length of about 160 tokens per abstract.
- All outputs were saved as Parquet files in HDFS, ensuring efficient access for downstream tasks.

Hive Analysis

- Hive queries were used to analyze text length distributions, identify repeated titles, and verify overall data consistency.
- The analysis confirmed the presence of meaningful biomedical vocabulary across the cleaned dataset.
- Insights from Hive validated the effectiveness of cleaning and deduplication and helped confirm that no structural issues remained in the final corpus.

Airflow Pipeline Execution

- The entire ETL pipeline was orchestrated through an Airflow DAG consisting of four Spark tasks: cleaning, deduplication, tokenization, and analysis.
- Airflow ensured reliable execution, centralized logging, and reproducible task sequencing.
- The DAG ran successfully, demonstrating that the pipeline is fully automatable and scalable.
- Screenshots of the Airflow graph and task logs are included in the report to document successful execution.

Streamlit Application and LLM Comparison

- A Streamlit web application was developed to showcase dataset statistics and demonstrate the practical value of preprocessing.
- The application displays token frequency charts, summary statistics, and samples drawn from the cleaned dataset.
- A lightweight LLaMA model was run for one epoch to illustrate how cleaned, structured biomedical text influences LLM responses.
- The app provides side-by-side comparisons between base LLM outputs and outputs informed by cleaned abstracts, highlighting improvements in clarity, structure, and terminology.

Summary of Key Findings

- The ETL pipeline produced a high-quality, deduplicated biomedical corpus suitable for LLM training and experimentation.
- Hive analysis confirmed dataset integrity and provided evidence of improved textual consistency.
- Airflow demonstrated that the workflow is robust, reproducible, and ready for scheduled execution.
- The Streamlit comparison app showed that preprocessing enhancements positively affect LLM behavior, even with minimal fine-tuning

Conclusion

This project successfully demonstrates the power and practicality of Big Data architectures for large-scale biomedical text processing, specifically in the context of preparing domain-specific datasets for Large Language Model (LLM) fine-tuning. Leveraging a massive real-world PubMed dataset of more than **2.21 million records**, we developed a fully scalable and reproducible ETL pipeline using **Apache Spark**, **HDFS**, **Hive**, and **Airflow** on the NYU Dataproc cluster. Our workflow encompassed the complete data engineering lifecycle from data ingestion and distributed cleaning, to MinHash-based deduplication, parallel tokenization, and comprehensive corpus-level analysis.

Through our pipeline, we transformed noisy, inconsistent biomedical abstracts into a **clean, deduplicated, and well-structured LLM-ready corpus**. Despite the complexity and variability of biomedical language, our automated preprocessing stages significantly enhanced dataset quality and consistency. The resulting high-fidelity Parquet dataset provides a strong foundation for downstream LLM tasks, enabling more accurate summarization, question-answering, and domain adaptation. In addition, the visual exploration performed through Hive and Plotly revealed meaningful linguistic patterns and length distributions, validating the effectiveness of our pipeline.

A key achievement of this project was the seamless integration of the pipeline's outputs into an interactive **Streamlit web application**, which allows users to explore dataset statistics and compare base language model outputs with outputs informed by cleaned biomedical text. This practical, user-friendly interface demonstrates the tangible benefits of high-quality preprocessing and highlights how Big Data engineering directly improves LLM interpretability and reliability.

Overall, this work underscores the critical role of **data quality, scalable ETL workflows, and reproducible engineering practices** in modern NLP and healthcare analytics. By establishing a robust and extensible Big Data pipeline for biomedical text, our project contributes meaningfully to ongoing efforts to build trustworthy, domain-specialized LLMs. The system we developed forms a solid foundation for future GPU-accelerated fine-tuning, semantic deduplication, and expanded multi-corpus integration, ultimately moving us closer to high-impact applications in biomedical research and clinical decision support.