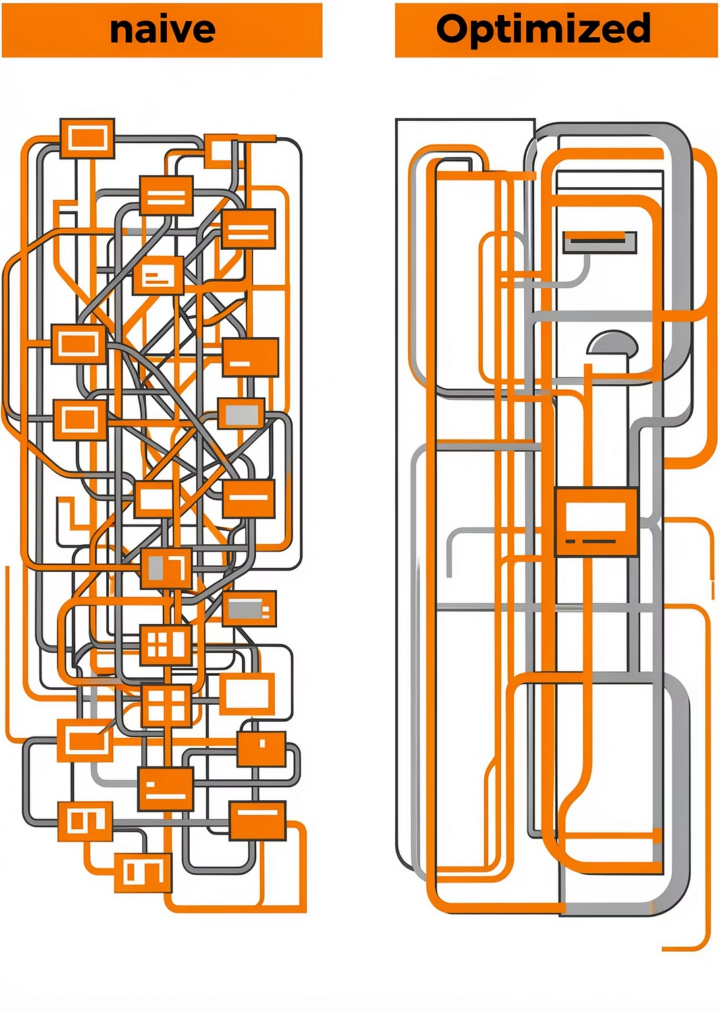
Scalable Distributed Pipeline for Clinical Risk Assessment of Skin Lesions

Master project—University of Bologna, M.Sc.
Computer Science and Engineering
(Intelligent Embedded Systems). Course: Big Data (81932).

Student: Daniel Khayatian.

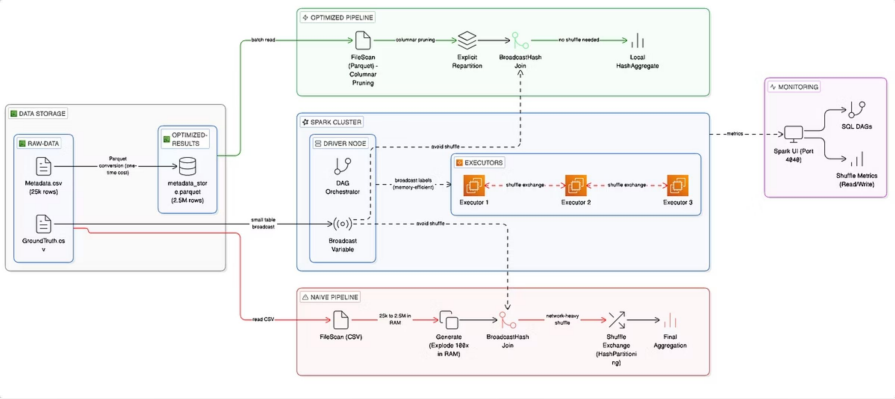
Teacher: Enrico Gallinucci.

Academic Year 2025/2026.



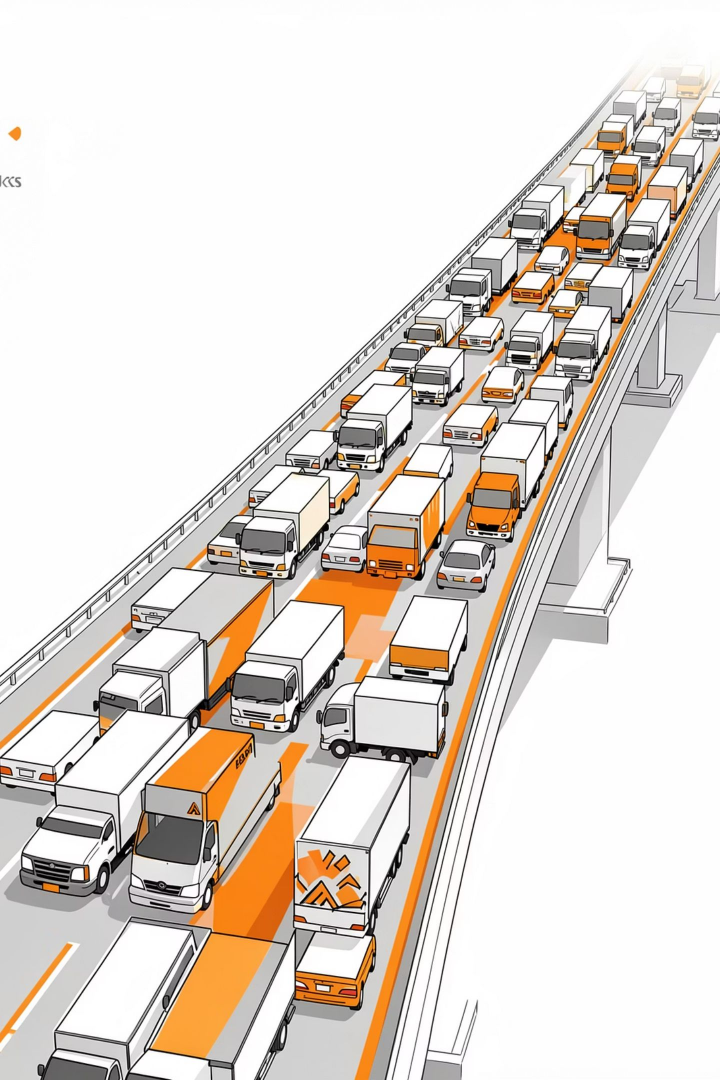
Project Abstract—Purpose & Approach

This project engineered a distributed Apache Spark pipeline on an AWS cluster that scaled 25,000 clinical records into a ~2.5M-row Parquet store to perform high-performance melanoma risk analysis. We compared a baseline "non-optimized" shuffle-heavy pipeline against an "optimized" pipeline using Parquet columnar storage, broadcast joins, and explicit repartitioning. Results were comparable: baseline 5.21s vs optimized 5.65s—showing upfront Parquet costs can offset shuffle savings at this scale.



Architecture Diagram—Distributed Clinical Risk Analysis

The diagram shows three layers: Data Sources (ISIC metadata, ground truth), Spark Cluster (driver + executors performing read, shuffleExchange, and writeParquet), and Data Processing (naive vs. optimized pipelines leading to ML modeling). The naive pipeline generates 2.5M records in RAM; the optimized pipeline reads persistent Parquet with columnar pruning.



Why Scale to 2.5 Million Records?

Trigger Real Shuffles

Small datasets often avoid distributed shuffles. Scaling forces network movement so optimizations are measurable.

Infrastructure Evaluation

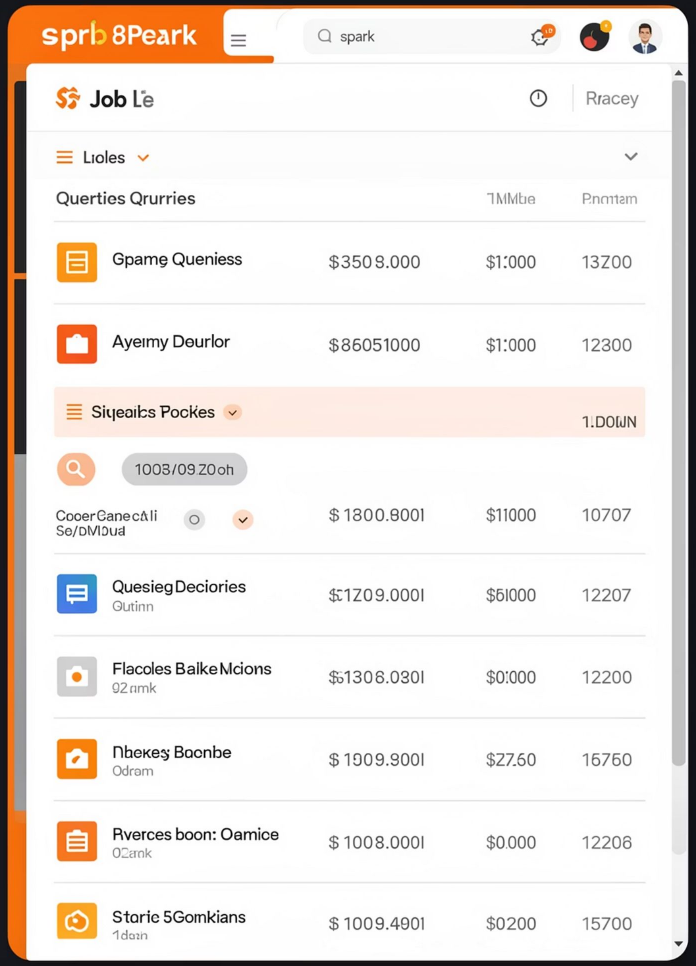
Simulates heavy AWS cluster load to validate stability under production-like traffic.

Performance Benchmarking

Creates a clear before/after comparison between on-the-fly augmentation and pre-materialized Parquet workflows.

Parquet Materialization & File Overview

We used Parquet columnar storage (Snappy compression) to persist the synthesized 2,533,100 rows. Converting CSV → Parquet incurs an upfront write penalty but enables batched, column-pruned reads that drastically reduce I/O for future analyses. Tool: Tad (Parquet viewer)—the screenshot shows a part file and 2,533,100 rows.



Key Spark Job Findings

Completed Queries: 5. Notable jobs: Non-Optimized Shuffle Run (5 s), Broadcast & Partition runs (6 s, 13 s). Job-level analysis shows the non-optimized plan contains multiple uncontrolled exchanges; the optimized plan adds a repartition step but eliminates repeated expensive shuffles during aggregation.

- Schema discovery: FileScan CSV (inferred types).
- Parquet write: materialization of 2.5M rows.
- Optimized: REPARTITION_BY_COL on anatom_site_general for controlled shuffling.

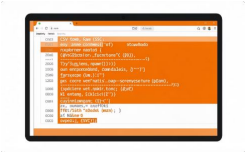
Executor's Summary

A single active node with 8 cores processed 20 tasks. Shuffle written ≈2.2 MiB, read ≈4.3 MiB. Total task time 38 min (cumulative), efficient wall-clock execution.

Why Shuffles Don’t Lose Data

Spark persists shuffle output to local files with index metadata during Shuffle Write, enabling retries if network or node failures occur. Lineage and deterministic partitioning let Spark re-execute only necessary parent tasks to regenerate lost partitions. In this pipeline, broadcasting the small ground truth prevents a full sort-merge join and preserves correctness across distributed transformations.

Optimization Trade-offs—Non-Optimized vs Optimized



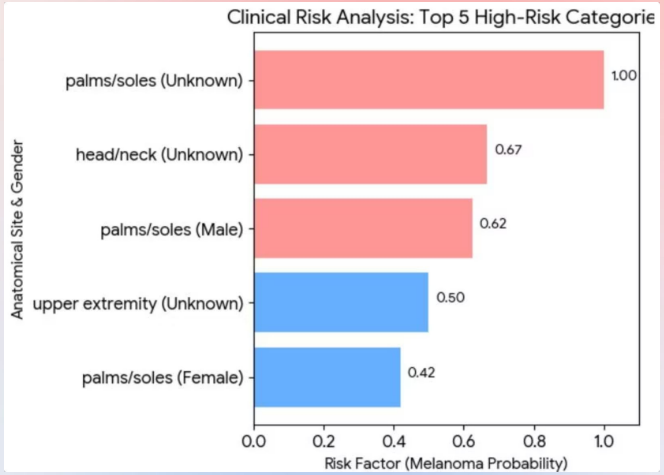
Non-Optimized (On-the-Fly)

Reads CSV (Batched: false), generates exploded rows in RAM, and has multiple uncontrolled exchanges. CPU & memory heavy during generation and shuffle.

Optimized (Structured)

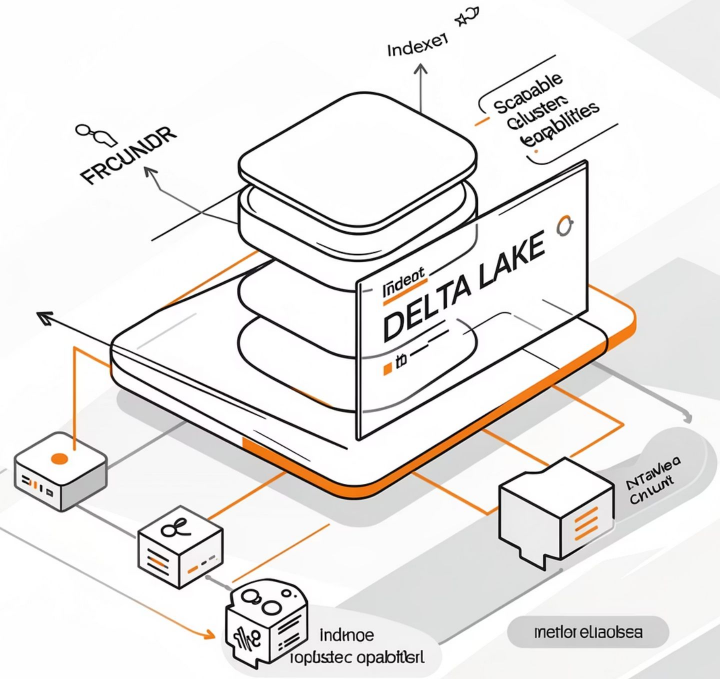
Writes Parquet (Batched:true) with schema stored, uses broadcast joins, and . repartition("anatom_site_general") for controlled shuffling. Upfront write cost and downstream query speed improved.

The optimized script is "heavier" initially due to Parquet materialization, broadcast overhead, and an explicit repartition shuffle. But this up-front work reduces future shuffle complexity and improves scalability.



Clinical Risk Analysis—Method & Results

We computed a risk factor = total_melanoma / total_cases for each anatomical site × sex grouping. Two shuffles: Shuffle 1 (identity join—broadcast GroundTruth) assigns MEL labels; Shuffle 2 (aggregation—repartition by anatom_site_general) co-locates groups for reliable hash aggregation. Key finding: palms/soles (unknown) risk = 1.00; head/neck (unknown) risk = 0.67. These ranked results produce a prioritized screening list for clinicians.



Conclusions, Scalability & Future Work

Conclusions

The pipeline demonstrates that manual architectural tuning matters. Though raw wall-clock times were similar (5.21s vs 5.65s), the optimized approach yields a production-ready architecture that reduces long-term I/O and improves scalability.

Scalability

The approach is linear: repartitioning and broadcast join patterns scale to larger datasets and more nodes. With 250M rows, upfront Parquet and controlled partitioning become essential.

Future Infrastructure Enhancements

Proposals: Delta Lake for incremental processing, Z-order indexing (anatom_site + age_approx), and automated skew handling (skew join hints). These reduce full rewrites and mitigate straggler tasks on skewed clinical data.

Final takeaway: trade upfront compute (Parquet conversion, repartition) for predictable, fault-tolerant, and cost-effective clinical risk analytics as datasets and usage grow.