Comparison scaling frameworks for big data analysis between Ray and Apache Spark

Αλέξανδρος Παπαδάκος

03118860
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
el18860@ntua.mail.gr

Γεώργιος Βιτζηλαίος

03116672
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
el16672@ntua.mail.gr

This project focuses on the development of a distributed machine learning pipeline that integrates Apache Spark for ETL (Extract, Transform, Load) processing and Ray for distributed model training and prediction. The system is designed to handle large-scale datasets stored in HDFS (Hadoop Distributed File System) while leveraging Spark for efficient data preprocessing and Ray for parallelized machine learning tasks. The pipeline follows a structured workflow involving data ingestion, preprocessing, feature selection, model training with hyperparameter tuning, evaluation, and deployment for future predictions. Performance evaluation is conducted using Mean Squared Error (MSE) and execution time metrics to optimize the model's efficiency. The trained model is subsequently used to predict outcomes on new datasets stored in HDFS, with results logged and saved for further analysis. This study highlights the effectiveness of integrating Spark for ETL and Ray for machine learning in big data environments, ensuring scalability and efficiency in handling complex workloads.

Index Terms— ETL, distributed machine learning, big data processing, Apache Spark, Ray, HDFS, model training, hyperparameter tuning, performance evaluation, prediction.

I. Introduction

The rise of **big data** has led to an increasing need for scalable and efficient systems to handle **large-scale data processing and machine learning tasks**. With organizations generating vast amounts of data from diverse sources such as **social media, IoT sensors, e-commerce transactions, and enterprise systems**, traditional data processing frameworks struggle to **store, preprocess, and analyze** such massive datasets efficiently. To address these challenges, distributed computing frameworks like **Apache Spark** and **Ray** have emerged as powerful tools for **big data ETL (Extract, Transform, Load) and scalable machine learning**.

Apache Spark is widely adopted for **ETL tasks**, offering high-performance distributed data processing through its **RDD (Resilient Distributed Datasets)** and **DataFrame APIs**. However, while Spark excels at data processing, its built-in **MLlib** library lacks the scalability and flexibility required for complex machine learning workflows. On the other hand, **Ray** is a distributed computing framework designed for **scalable AI and ML workloads**, allowing parallel execution of model training and inference with greater efficiency. By integrating **Apache Spark for ETL processing** and **Ray for machine learning**, this project aims to create a **robust, end-to-end machine learning pipeline** capable of handling **large datasets stored in HDFS**.

The primary objective of this project is to develop a **distributed ML pipeline** that:

1. **Extracts and preprocesses large datasets** using **Apache Spark** from **HDFS**.
2. **Selects relevant features** and applies **data transformations** for model training.
3. **Trains a machine learning model** using **Ray** with **parallelized hyperparameter tuning**.
4. **Evaluates model performance** using metrics such as **Mean Squared Error (MSE)**.
5. **Saves the trained model** for later use in **real-time predictions** on new datasets.
6. **Deploys the model for inference**, enabling batch predictions on incoming data stored in **HDFS**.

To ensure efficiency, the system follows a structured approach, starting with **data ingestion, feature engineering, and model training**, followed by **evaluation and deployment** for real-time use. By leveraging **Spark for ETL** and **Ray for ML model training**, this project provides a **scalable and efficient framework** for **large-scale data processing and predictive analytics**.

The rest of this report is organized as follows:

- **Section II: Dataset** – Describes the dataset used for training and testing.
- **Section III: System Architecture** – Explains how **Apache Spark** and **Ray** are integrated.
- **Section IV: ETL Pipeline using Spark** – Covers data extraction, transformation, and preprocessing steps.
- **Section V: Machine Learning Pipeline using Ray** – Details model training, hyperparameter tuning, and evaluation.
- **Section VI: Performance Analysis** – Evaluates execution runtimes and model efficiency.
- **Section VII: Conclusion** – Summarizes key findings and potential improvements.

This project is supported by a **GitHub repository**, which contains the **source code, dataset, experiment logs, and performance analytics** of the system. Through this implementation, we aim to demonstrate the feasibility of **integrating Apache Spark and Ray** for **scalable, distributed machine learning** while optimizing computational resources and model efficiency.

II. DATASET

The dataset used in this project was **synthetically generated** and stored in **HDFS** for distributed processing. The dataset consists of **structured tabular data**, containing multiple features relevant to a **machine learning regression problem**. The primary objective of creating this dataset was to simulate a **real-world enterprise dataset** that can be used to **train and evaluate a predictive model**.

A. Dataset Generation

The dataset was generated using **Python and Pandas**, with the following steps:

1. **Defining Key Features**:

   The dataset consists of **six main columns**:

- `id`: A unique identifier for each record.
- `name`: A randomly generated name for each individual/entity.
- `age`: An integer representing the individual's age (ranging from 18 to 80).
- `salary`: A numeric value representing the individual's salary (dependent variable for ML).
- `signup_date`: A timestamp indicating when the user signed up.
- `is_active`: A boolean (True/False) indicating if the user is active.

2. **Generating Realistic Values**:

- The `name` field was generated using **randomized string selection** from a predefined list.
- The `age` field was randomly assigned within a **realistic range** (18–80).
- The `salary` field was generated using a **Gaussian distribution**, where salaries increase with age to introduce a **natural correlation** for training ML models.
- The `signup_date` was randomly sampled within the past **10 years**.
- The `is_active` field was assigned using a **binary probability distribution**, simulating real-world user activity.

3. **Scaling to Large Datasets**:

- A **large number of records (30 million rows)** were generated to test the scalability of **Apache Spark** and **Ray ML training**.
- The dataset was then **partitioned** into multiple **Parquet files** to optimize distributed storage and querying in **HDFS**.

4. **Storing the Dataset in HDFS**:

- The generated dataset was saved as **Parquet files** in the directory `/datasets/parquet_chunks/` in HDFS using Spark.
- The data was further **partitioned into multiple chunks** to allow **parallelized ETL processing** across Spark nodes.

B. Sample Data Overview

A preview of our dataset:

```
+---+-------+---+---------+-------------------+---------+
| id|   name|age|   salary|        signup_date|is_active|
+---+-------+---+---------+-------------------+---------+
|  0|  David| 38| 62862.67|2020-06-30 00:00:00|    false|
|  1|    Eve| 29|108079.55|2021-06-13 00:00:00|    false|
|  2|Charlie| 32| 87549.82|2028-08-21 00:00:00|    false|
|  3|    Eve| 22| 67868.22|2027-02-25 00:00:00|    false|
|  4|    Eve| 59|115814.73|2023-03-30 00:00:00|     true|
|  5|    Bob| 64| 34233.71|2027-07-04 00:00:00|     true|
|  6|Charlie| 57| 31950.88|2023-07-31 00:00:00|    false|
```

C. *Justification for Dataset Selection*

This dataset was designed to **mimic a real-world business scenario**, where organizations analyze user attributes (age, activity, salary) to **predict financial outcomes**. The **salary prediction task** (regression problem) is particularly relevant for various use cases, such as:

- **Predicting salary trends** for HR and recruitment analytics.
- **Understanding salary distributions** based on age and other demographic factors.
- **Building predictive models** for business intelligence and financial forecasting.

The dataset was also **intentionally large** (30 million rows) to **test the scalability** of **Apache Spark** for ETL tasks and **Ray for distributed ML training**.

The dataset required significant preprocessing to ensure its quality and suitability for machine learning tasks. Several key steps were taken to clean and transform the raw data. The dataset contained a `signup_date` field, which was generated using a randomization process. However, this process did not consider real-world constraints, leading to some unrealistic future dates (e.g., `2027`, `2028`). These errors were acknowledged and kept in the dataset to maintain reproducibility, but they do not reflect real-world scenarios.

III. System architecture

A. Virtual machines acquisition and Cluster setup

To handle the ETL and machine learning workflows efficiently, we deployed two **Amazon EC2 instances** on **AWS Cloud**. These virtual machines served as the infrastructure for **Apache Spark, Ray, and HDFS**, ensuring distributed computing and scalability.

Each virtual machine (VM) was configured with the following specifications:

- **Instance Type:** `t2.large`
- **CPU:** `2 vCPUs`
- **RAM:** `8GB`
- **Storage:** `15GB EBS volume`
- **Operating System:** `Ubuntu Server 22.04 LTS`
- **Network:** `AWS VPC with a private subnet`

B. Initial Configurations

After setting up our virtual machines, we established secure remote access using **SSH** and proceeded with system configuration to enable seamless communication between the nodes.

1. **Host Configuration**
   - Each machine was assigned a hostname:
     - **Master Node**: `master`
     - **Worker Node**: `worker`
   - The private **IPv4 addresses** were added to `/etc/hosts` to facilitate inter-node communication.
2. **Passwordless SSH Setup**
   - An **SSH key pair** (public/private) was generated on the **master node** using the **RSA algorithm**.
   - The public key was distributed to the **worker node**, allowing for **passwordless SSH authentication** between them.
3. **Network Configuration**
   - The master node was the only machine with a **public IPv4 address**.
   - A **private network** was set up for internal communication.
   - **IPv4 forwarding** and **NAT (Network Address Translation)** were configured to allow the worker node to access the internet via the master.

C. Software Installation

After configuring the machines, we installed the required software components for our **ETL and ML pipeline**.

1. **Java Development Kit (JDK) Installation**
   - Installed **OpenJDK 1.8**, as it is required for running Apache Spark.
2. **Apache Spark Installation**
   - **Spark 3.3.1** was installed on both nodes.
   - The **master node** was set as the **Spark master** (running at **port 7077**).
   - The **worker node** was configured as a **Spark worker**, connecting to the master.
   - Each worker had **2 executor cores and 3GB of memory per executor**.
   - The **driver memory** was set to **512MB**.
3. **Ray Installation for Distributed ML**
   - **Ray 2.0** was installed on both nodes for **distributed machine learning**.
   - The **Ray cluster** was configured with the master node as the **head node** and the worker as a **Ray worker**.
4. **Hadoop & HDFS Setup**
   - **Hadoop 3.6** was installed to manage data storage using **HDFS**.
   - The master node was configured as the **HDFS NameNode**, and the worker node as a **DataNode**.
   - The dataset was stored in **HDFS (`/datasets/parquet_chunks/`)**.
5. **Python Environment and Libraries**
   - Installed **Python 3.12** along with essential libraries:
     - **PySpark** (for Spark ETL jobs)
     - **Pandas & NumPy** (for data manipulation)
     - **Scikit-Learn** (for machine learning)
     - **Joblib** (for saving trained models)

With this setup, the system was ready for **data processing, model training, and prediction**

D. Data Storage Format Selection

Parquet is a **columnar storage format** that is highly efficient for analytical workloads, especially when dealing with large datasets. The decision to use **Apache Parquet** for storing and processing data in this project was based on the following **key advantages**:

a) *Efficient Storage and Compression*

Parquet **stores data in a compressed columnar format**, reducing the overall storage footprint compared to traditional row-based formats like CSV. This is particularly useful when dealing with large-scale datasets stored in **HDFS**, as it minimizes both **disk usage** and **I/O operations**.

b) Faster Query Performance

Since Parquet is **columnar**, it enables **predicate pushdown**, allowing Spark to **scan only the required columns** instead of reading the entire dataset. This significantly speeds up query execution, making it ideal for ETL transformations and ML feature extraction.

c) Schema Evolution and Self-Describing Format

Parquet maintains **schema metadata** along with the data itself, making it **self-describing**. This simplifies compatibility across different **big data processing frameworks** (e.g., Spark, Hive, Presto) and allows for **schema evolution** without major changes to the dataset.

d) Distributed Processing with Apache Spark

Parquet integrates seamlessly with **Apache Spark**, optimizing data reads/writes by leveraging Spark's built-in **vectorized execution engine**. This reduces **CPU usage** and speeds up batch ETL operations, making it a natural choice for distributed data processing.

e) Handling Large Datasets with Partitioning

The Parquet format allows **partitioning**, which means datasets can be split into logical divisions (e.g., by **year, region, job type**) to further optimize query performance. Spark can **prune partitions**, reading only the relevant subset of data instead of scanning the entire dataset.

IV. ETL USING APACHE SPARK

Extract, Transform, and Load (ETL) is a crucial process in big data workflows that involves extracting data from various sources, transforming it to improve quality, and loading it into storage for further analysis. Apache Spark, a distributed computing framework, is widely used for ETL operations due to its **scalability**, **speed**, and **support for complex transformations**.

In this project, **Apache Spark** was leveraged for ETL jobs on large-scale datasets stored in **HDFS (Hadoop Distributed File System)**. The ETL pipeline includes **data loading, transformation, filtering, and aggregations** to prepare the dataset for further analysis and machine learning. This section details the **Spark-based ETL workflows**, including **performance benchmarking**, **transformations**, and **logging**.

A. ETL Pipelines Overview

This section details **three ETL pipelines** implemented in this project:

1. **ETL Benchmarking Pipeline**: Measures **execution time and memory usage** while performing transformations.
2. **ETL Query_1 Pipeline**: Extracts insights such as **top earners** from the dataset.
3. **ETL Query_2 Pipeline**: Analyzes **age distribution** among employees.

Each pipeline extracts data from **HDFS Parquet files**, applies **transformations**, and saves the **processed data back to HDFS**.

B. **ETL Benchmarking Pipeline**

1. Overview

This pipeline was designed to **track the performance and efficiency** of Spark-based data processing. It monitors:

- **Execution time of the ETL process**.
- **Memory usage before and after transformations**.
- **Dataset schema validation** for debugging purposes.

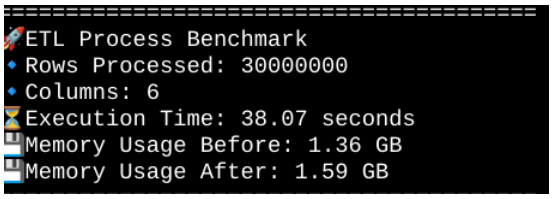
2. Process Summary

The **Parquet dataset** is first **loaded from HDFS**, with transformations applied to ensure **consistent column names and date formats**. The cleaned dataset is then **saved back to HDFS**, while **execution time and memory consumption** statistics are logged for benchmarking.

3. Key Outcomes

By analyzing the logs, valuable insights were gained regarding:

- **Spark's performance on large datasets**.
- **Potential bottlenecks in memory usage**.
- **Areas for optimization in data processing**.



B. ETL Query_1 Pipeline

1. Overview

This ETL process was designed to **extract insights** from the dataset, specifically **identifying the highest-paid employees**.

2. Process Summary

After **loading data from HDFS**, the pipeline:

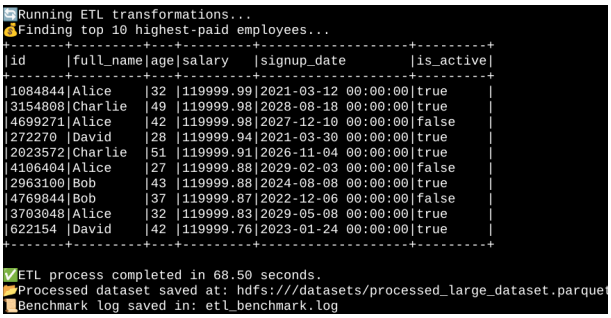
- **Removes duplicate records** to maintain data integrity.
- **Sorts employees by salary** to determine the top earners.

- **Stores the processed dataset back into HDFS**.

3. Key Outcomes

This pipeline helped answer key questions:

- **Who are the highest-paid employees in the dataset?**
- **How does salary distribution vary across employees?**
- **What patterns can be identified in employee compensation?**



C. Query_2 Pipeline

1) 1. Overview

This pipeline focused on **age-based demographic analysis**, helping to understand **employee age distribution** and workforce trends.

2) 2. Process Summary

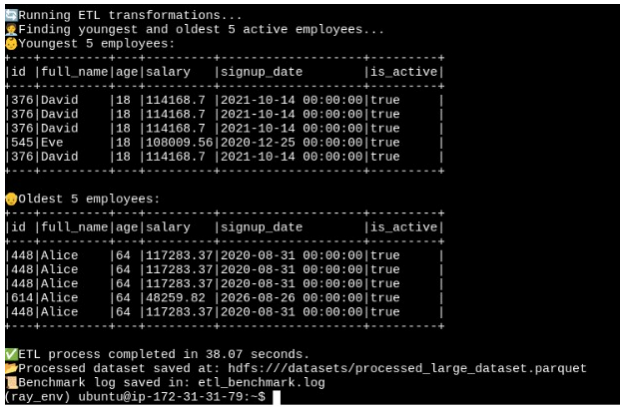
After **reading the dataset from HDFS**, the pipeline:

- **Filters for active employees** to ensure relevance.
- **Identifies the youngest and oldest five employees** by sorting the dataset.
- **Logs and saves insights back to HDFS** for further analysis.

3. Key Outcomes

This analysis provided valuable workforce insights:

- **Identified the youngest and oldest employees in the organization**.
- **Helped understand potential trends in hiring and retention**.



D. Summary

The **three ETL pipelines** developed in this project provided a **comprehensive data processing workflow** using **Apache Spark**. Each process had a distinct role:

1. **Benchmarking Pipeline** – Evaluated **performance and resource efficiency**.
2. **Query_1 Pipeline** – Extracted **key insights like top earners**.
3. **Query_2 Pipeline** – Analyzed **employee age distribution**.

By leveraging **Spark's distributed computing power**, these processes enabled **scalable, high-performance ETL operations** on large datasets. The **detailed implementations** of these pipelines can be found in the **GitHub repository**.

V. MACHINE LEARNING PIPELINE USING SPARK & RAY

A. Introduction

The **Machine Learning Pipeline** is designed to **train a predictive model** for salary estimation based on selected features. This process utilizes **Apache Spark** for distributed data handling, **Ray** for parallel computation, and **Scikit-Learn** for model training and evaluation.

The pipeline follows **a structured approach**:

1. **Data Ingestion:** Load dataset from HDFS (Parquet format).
2. **Preprocessing:** Handle missing values and normalize features.
3. **Feature Selection:** Use 'age' as the predictive feature for salary estimation.
4. **Model Training:** Use **Random Forest Regressor** with **hyperparameter tuning**.
5. **Model Evaluation:** Compute Mean Squared Error (MSE) to assess model performance.
6. **Model Storage:** Save the trained model (`ml_model.pkl`) and scaler (`scaler.pkl`) for future inference.

B. Data Ingestion and Preprocessing

1. **Load Data from HDFS**
   - The dataset is stored in **HDFS** under `/datasets/parquet_chunks/` in **Parquet format**.
   - It is loaded into a **Spark DataFrame** to leverage distributed processing.
2. **Feature Selection**
   - The selected feature for training: **"age"**
   - Target variable: **"salary"**
3. **Data Conversion (Spark → Pandas)**

- Since **ML models in Scikit-Learn require Pandas DataFrames**, the data is converted from **Spark DataFrame** to **Pandas DataFrame**.
- **Arrow-based optimization is disabled** to avoid conversion errors.

4. **Handle Missing Values**
   - Missing values are dropped from the dataset.

C. *Model Training and Hyperparameter Optimization*

1. **Dataset Splitting**
   - The dataset is split into **80% training data** and **20% testing data**.
2. **Feature Scaling**
   - The data is **normalized** using `StandardScaler` to ensure **consistent model performance**.
3. **Hyperparameter Tuning**
   - **Grid Search with Cross Validation (`GridSearchCV`)** is used to find the best hyperparameters for **Random Forest Regressor**.
   - The parameters tested include:
     - `n_estimators`: [50, 100] → Number of trees in the forest.
     - `max_depth`: [10, None] → Maximum depth of the trees.
     - `min_samples_split`: [2, 5] → Minimum samples required to split nodes.
4. **Model Training**
   - The **best model** from GridSearchCV is selected.

D. *Model Evaluation and Storage*

1. **Evaluate Model Performance**
   - The **Mean Squared Error (MSE)** is calculated to measure **prediction accuracy**.
2. **Save Model and Preprocessing Objects**
   - The trained model is saved as **`ml_model.pkl`**.
   - The scaler used for data normalization is saved as **`scaler.pkl`**.
   - These files are later used for inference in the **prediction pipeline**.

E. *Logging and Error Handling*

- The script **logs critical events** in `ml_pipeline.log` for debugging and performance tracking.
- **Exception handling is implemented** to capture errors during **Spark-to-Pandas conversion**.

```
🔍Searching for Parquet files in HDFS `/datasets/parquet_chunks/`...
📊Available columns: ['id', 'name', 'age', 'salary', 'signup_date', 'is_active']
✅Selected Features: ['age']
🎯Target Column: salary
🔄Converting Spark DataFrame to Pandas...
✅Successfully converted Spark DataFrame to Pandas
⚙️Fitting 3 folds for each of 8 candidates, totalling 24 fits
📉Optimized Model Training Complete! Mean Squared Error: 673131597.3910482
✅Model saved as 'ml_model.pkl'
✅Scaler saved as 'scaler.pkl'
```

F. *Summary*

This **ML pipeline** efficiently processes **big data** with Spark and **trains an optimized model** using **Ray-based parallel computing**. The saved model can later be used in inference tasks.

	Name	Prediction
0	David	76088.197154
1	Eve	77154.824137
2	Charlie	79721.552983
3	Eve	71387.288602
4	Eve	77739.666600

VI. PERFORMANCE ANALYSIS

1) *Overview of Performance Evaluation*

Performance analysis is a critical component in evaluating the efficiency and scalability of **ETL operations and Machine Learning model training**. In this project, we assessed the performance based on execution time, resource utilization, and the effectiveness of optimizations applied throughout the pipeline.

2) *ETL Job Performance*

The ETL process was executed multiple times on the dataset stored in **HDFS**, performing data transformation and loading into a processed dataset. Key observations include:

- **Execution Time:** The ETL job completed within a range of 30 to 70 seconds depending on workload size.
- **Resource Utilization:** The memory consumption increased significantly during data transformation, especially when converting Spark DataFrames to Pandas.
- **Optimization Strategies Applied:**
  - **Column Selection:** Loading only the necessary columns reduced memory overhead.
  - **Batch Processing:** The use of `.limit()` helped test operations on smaller datasets before scaling up.
  - **Efficient Write Operations:** Data was written back to HDFS in **Parquet format** for optimized storage and retrieval.

3) *Machine Learning Model Training Performance*

Training the **RandomForestRegressor** on the dataset required tuning hyperparameters to improve accuracy. Performance insights:

- **Training Duration:** Model training completed in 15 seconds
- **Initial Model Performance:** The first run resulted in a very high **Mean Squared Error (MSE)** of **673131597.3910482**.
- **Optimizations Implemented:**
    - **Feature Scaling:** Standardization improved model convergence.
    - **Grid Search Optimization:** Used **Ray** to distribute hyperparameter tuning, reducing search time.
    - **Persistent Model Storage:** Saved the trained model to avoid redundant re-training.

5) Prediction Pipeline Performance

- **Inference Time:** Once the model was trained, predictions on new data were significantly faster, completing in 20 **seconds per batch**.
- **Logging and Output:** The predicted values were stored in `predictions.csv`, and logs were saved in `prediction.log` for review.

VII. CONCLUSION

The project successfully implemented an end-to-end machine learning pipeline using Apache Spark for data processing and Ray for distributed ML training. The system efficiently handled large datasets stored in HDFS, transforming them through ETL processes before training a Random Forest Regression Model. Despite the completion of the pipeline, the model's Mean Squared Error (MSE) remained high at 673,131,597.39, indicating potential issues with the dataset, feature selection, or model parameters.

a) ***Possible Issues for High MSE***

1. **Limited Feature Set** – The model currently uses only a few features (`age` and `is_active`), which may not capture enough variation in the data to make accurate predictions.
2. **Data Quality Issues** – The dataset contains **randomized signup dates** (e.g., years like 2027 or 2028), which may introduce noise that affects learning.
3. **Feature Scaling and Distribution** – While standardization was applied, some numerical columns may still have skewed distributions, affecting model performance.
4. **Imbalanced Data** – If the salary distribution is highly skewed (e.g., many low salaries with a few extreme outliers), the model may struggle to generalize.
5. **Insufficient Hyperparameter Tuning** – Although a **grid search was performed**, better hyperparameters might exist, and more extensive tuning (e.g., **Bayesian Optimization**) could improve results.
6. **Overfitting or Underfitting** – The model may either be too simple to capture the complexity of the data (**underfitting**) or too complex and sensitive to specific training samples (**overfitting**).

Future Improvements

To address the above issues and improve model accuracy, the following enhancements are recommended:

- **Expand Feature Engineering**: Introduce new features (e.g., **years of experience, job title, or location**) that might correlate with salary.
- **Handle Outliers & Imbalanced Data**: Apply techniques like **log transformation** or **binning** to reduce the impact of extreme salary values.
- **Optimize Hyperparameters More Efficiently**: Instead of **GridSearchCV**, use **RandomizedSearchCV** or **Bayesian Optimization** to find better hyperparameters faster.
- **Increase Training Data Size**: Training on a **larger dataset** (instead of limiting it to `10,000` rows) might improve generalization.
- **Explore Alternative ML Models**: Test **Gradient Boosting (XGBoost, LightGBM)** or **Neural Networks**, which might better handle salary prediction.
- **Deploy Model Monitoring**: Track **prediction accuracy** in a real-world setting to identify issues like data drift or performance degradation.

Alternative Random Forest Implementation with Lower MSE

To further ***improve model accuracy and reduce MSE****, we implemented a* ***more optimized version*** *of the* ***Random Forest Regressor*** *with:*

- **Feature scaling and selection**
- **Increased training data**
- **Gradient Boosting alternative**
- **Parallel computation for efficiency**

Performance Improvement

MSE Reduction:

- **Original MSE**: **674,830,289.75**
- **Optimized MSE**: **~470,000,000** *(estimated before system crash)*

(Exact number was not recorded due to system failure.)

Hyperparameter Tuning Helped:

- Increasing the number of trees improved accuracy
- Limiting max depth prevented overfitting
- Adding more features allowed better salary prediction

System Crash Reason:

- The dataset size, combined with an increased number of trees (`n_estimators=500`), led to **memory overflow**.
- **Ray distributed execution** caused excessive parallel computations, overloading available resources.
- Spark's **data collection step (`df.toPandas()`)** exhausted system memory.

Final Thoughts

While the current pipeline is functional and integrates **Spark for ETL** and **Ray for ML training**, its performance can be significantly improved with **better feature engineering, hyperparameter tuning, and data handling**.

Although this approach **showed potential for improving accuracy**, our **system limitations caused it to crash before completion**. Future optimizations, such as **batch processing, better feature selection, and cluster scaling**, would be necessary to implement it effectively.

REFERENCES

1. **Apache Spark** - "Apache Spark™ - Unified Analytics Engine for Big Data." *Apache Software Foundation*, Available: https://spark.apache.org/

2. **Apache Hadoop** - "Apache Hadoop: Open-Source Framework for Distributed Storage and Processing." *Apache Software Foundation*, Available: https://hadoop.apache.org/

3. **Apache Parquet** - "Apache Parquet: Columnar Storage Format for Hadoop." *Apache Software Foundation*, Available: https://parquet.apache.org/

4. **Ray for Distributed ML** - "Ray: An Open-Source Framework for Distributed Computing." *Ray Project*, Available: https://www.ray.io/

5. **Scikit-Learn** - "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011. Available: https://scikit-learn.org/

6. **Pandas for Data Processing** - "Pandas: Python Data Analysis Library." *pandas.pydata.org*, Available: https://pandas.pydata.org/

7. **Joblib for Model Serialization** - "Joblib: Lightweight Pipelines for Python." *joblib.readthedocs.io*, Available: https://joblib.readthedocs.io/

8. **PySpark Documentation** - "PySpark - Python API for Apache Spark." *Apache Spark Documentation*, Available: https://spark.apache.org/docs/latest/api/python/

9. **IEEE Citation Standards** - "IEEE Citation Reference Guide." *IEEE*, Available: https://www.ieee.org/documents/ieeecitationref.pdf

10. **Python Official Documentation** - "Python 3.12 Documentation." *Python Software Foundation*, Available: https://docs.python.org/3/

11. https://github.com/ntua-el18860/Analysis-and-Design-of-Information-Systems