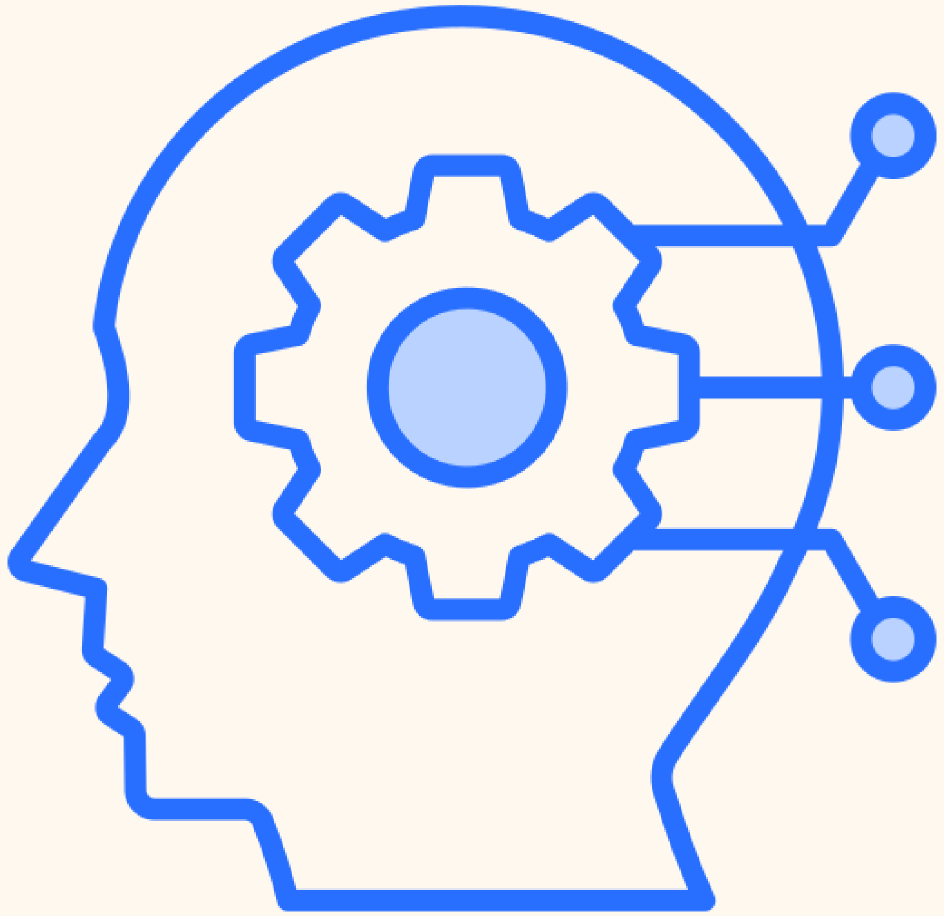
MACHINE LEARNING USING APACHE SPARK MLLIB



G Kalyani Krishna	22011102017
Harshita Anand	22011102023
MSR Varshadh	22011102050
M Suraj	22011102052

INTRODUCTION TO MACHINE LEARNING & BIG DATA

Machine Learning

- ML is a subset of artificial intelligence that enables systems to learn patterns from data and make predictions or decisions without being explicitly programmed.
- It includes techniques like supervised learning, unsupervised learning, and reinforcement learning.
- Common applications: recommendation systems, fraud detection, image recognition, and natural language processing.

Big Data

- Refers to extremely large datasets that cannot be processed using traditional methods.
- Characterized by the 4 Vs:
  - Volume – Huge amounts of data generated every second.
  - Velocity – Speed at which data is generated and processed.
  - Variety – Different formats (structured, semi-structured, unstructured).
  - Veracity – Data reliability and accuracy.

Intersection of Machine Learning & Big Data

- Machine learning models require vast amounts of data to improve accuracy.
- Big Data technologies (e.g., Hadoop, Spark) help in storing, processing, and analyzing large-scale data efficiently.
- ML on Big Data enables real-time analytics, predictions, and decision-making.

WHAT IS APACHE SPARK?

- Apache Spark is an open-source, distributed computing system designed for fast data processing.
- Developed at UC Berkeley and later donated to the Apache Software Foundation.
- It provides high-speed computing capabilities using in-memory processing.

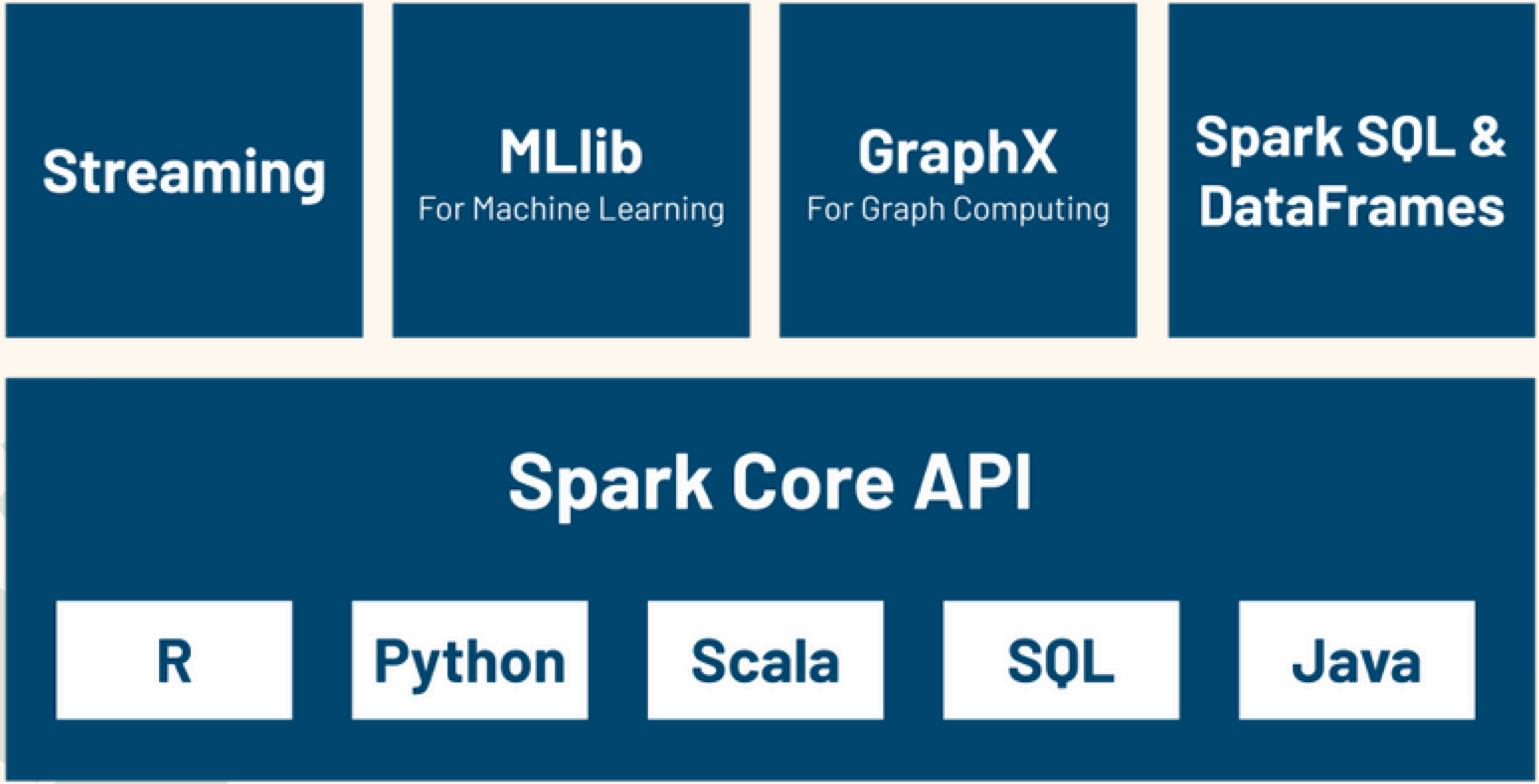
Key Features of Apache Spark

- Speed – Up to 100x faster than Hadoop MapReduce due to in-memory computing.
- Ease of Use – Supports Python (PySpark), Java, Scala, and R.
- Unified Analytics – Combines batch processing, real-time streaming, machine learning, and graph processing.
- Scalability – Works on a single machine or across thousands of nodes in a cluster.
- Fault Tolerance – Recovers from failures using RDD (Resilient Distributed Datasets).

Why Use Spark for Machine Learning?

- Handles Big Data efficiently with distributed processing.
- Fast iterative computations, ideal for ML algorithms.
- Built-in MLlib simplifies large-scale ML model development.

SPARK ECOSYSTEM



SPARK MLIB

What is Spark MLlib?

- MLlib (Machine Learning Library) is Spark's scalable machine learning framework.
- Designed to run on distributed clusters, making it ideal for big data ML tasks.
- Provides a set of high-performance ML algorithms and utilities for classification, regression, clustering, recommendation, etc.

Why Use Spark MLlib?

- Scalability – Works efficiently with large datasets.
- Speed – Uses in-memory computation for faster training.
- Ease of Integration – Supports Spark SQL, DataFrames, and Pipelines.
- Multi-language Support – Works with Scala, Java, Python (PySpark), and R.

MLLIB VS SPARK ML (COMPARISON OF APIS)

Features	MLLib	Spark ML
Data Structure	Uses RDDs (Resilient Distributed Datasets)	Uses DataFrames (More efficient)
Ease of Use	Requires manual feature engineering	Supports ML Pipelines for automatio
Performance	Slower (RDD transformations)	Faster (Optimized for Catalyst & Tungsten)
Feature Engineering	Manual feature extraction & transformation	Built-in transformers (VectorAssembler, OneHotEncoder, etc.
Support for Pipelines	No built-in ML Pipelines	Supports ML Pipelines (like Scikit-Learn)

FEATURES OF MLIB

1. **Scalability & Performance –** Optimized for big data with in-memory computation, making it 100x faster than Hadoop.

2. **Comprehensive ML Algorithms –** Supports classification, regression, clustering, and recommendation systems.

3. **ML Pipelines & Feature Engineering –** Provides built-in transformers (VectorAssembler, StandardScaler, OneHotEncoder).

4. **Multi-language Support –** Works with Scala, Java, Python (PySpark), and R.

5. **Seamless Spark Integration –** Works with Spark SQL, Streaming, and GraphX.

6. **Distributed & Fault-Tolerant –** Built on RDDs, ensuring parallel processing and fault tolerance.

7. **Open-source & Actively Maintained –** Part of the Apache Spark ecosystem, with continuous improvements in Spark ML.

MLLIB ARCHITECTURE

Data Layer → **Core MLLib Components** → **Higher Level APIs** → **Distributed Execution Engine**

Data Layer

- Supports RDDs (Resilient Distributed Datasets) and DataFrames for handling structured and unstructured data.
- Works with HDFS, Apache Hive, HBase, and other data sources.

Core MLLib Components

- Feature Extraction & Transformation
- Machine Learning Algorithms
- Optimization Algorithms

Higher Level APIs

- RDD-based MLlib – Legacy API using low-level RDDs (deprecated).
- DataFrame-based Spark ML – Modern API supporting ML Pipelines, providing better optimization and usability.

Distributed Execution Engine

- Uses Apache Spark Core for parallel computing across a cluster.
- Ensures fault tolerance and scalability.

DATA PREPROCESSING IN MLLIB

1. Feature Transformation

```
from pyspark.ml.feature import StringIndexer, VectorAssembler

# Convert categorical column to numeric
indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
data = indexer.fit(data).transform(data)

# Combine multiple features into a single vector
assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
data = assembler.transform(data)
```

2. Handling Missing Values

```
data = data.fillna({'age': 30, 'salary': 50000})  # Replace NaN values
```

3. Splitting Data for Training & Testing

```
train, test = data.randomSplit([0.8, 0.2], seed=42)
```

SUPERVISED LEARNING IN MLLIB

```python
from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, RandomForestClassifier
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator, RegressionEvaluator

spark = SparkSession.builder.appName("SupervisedLearning").getOrCreate()

models = {
    "Logistic Regression": LogisticRegression(featuresCol="features", labelCol="label"),
    "Decision Tree": DecisionTreeClassifier(featuresCol="features", labelCol="label"),
    "Random Forest": RandomForestClassifier(featuresCol="features", labelCol="label"),
    "Linear Regression": LinearRegression(featuresCol="features", labelCol="label")  # For regression
}
# Train and Evaluate Models
for name, model in models.items():
    trained_model = model.fit(train)
    predictions = trained_model.transform(test)
    if "Regression" in name:
        evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
    else:
        evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

    metric = evaluator.evaluate(predictions)
    print(f"{name} - Score: {metric:.4f}")
spark.stop()
```

UNSUPERVISED LEARNING IN MLLIB

```python
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans, GaussianMixture
from pyspark.ml.feature import VectorAssembler, PCA
from pyspark.ml.evaluation import ClusteringEvaluator

models = {
    "K-Means Clustering": KMeans(featuresCol="features", k=2, seed=42),
    "Gaussian Mixture Model": GaussianMixture(featuresCol="features", k=2),
    "Principal Component Analysis": PCA(k=1, inputCol="features", outputCol="pca_features")
}

for name, model in models.items():
    trained_model = model.fit(data)

    if "PCA" in name:
        transformed_data = trained_model.transform(data)
        transformed_data.select("pca_features").show()
        print(f"{name} - PCA Transformation Complete")
    else:
        predictions = trained_model.transform(data)
        evaluator = ClusteringEvaluator(featuresCol="features", predictionCol="prediction", metricName="silhouette")
        score = evaluator.evaluate(predictions)
        print(f"{name} - Silhouette Score: {score:.4f}")
spark.stop()
```

RECOMMENDATION SYSTEMS WITH MLLIB

Apache Spark MLlib provides a collaborative filtering approach for recommendation systems using Alternating Least Squares (ALS).

1. Collaborative Filtering (ALS)

- Implicit & Explicit Feedback – Works with both rating-based (explicit) and user behavior-based (implicit) data.
- Scalability – Optimized for large-scale datasets with distributed computing.
- Personalization – Generates user-item recommendations efficiently.

2. Example: Movie Recommendation using ALS

```
# Train ALS model
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", coldStartStrategy="drop")
model = als.fit(train)

# Generate recommendations
predictions = model.transform(test)
predictions.show()
```

MODEL EVALUATION & HYPERPARAMETER TUNING

1. Model Evaluation Metrics

- Classification → MulticlassClassificationEvaluator (accuracy, F1-score)
- Regression → RegressionEvaluator (RMSE, MAE, R^2)
- Clustering → ClusteringEvaluator (Silhouette score)

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Model Accuracy: {accuracy:.4f}")
```

2.Hyperparameter Tuning

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol="features", labelCol="label")
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.01, 0.1, 1.0]) \
    .addGrid(lr.maxIter, [10, 50, 100]) \
    .build()
crossval = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid,
                          evaluator=MulticlassClassificationEvaluator(metricName="accuracy"),
                          numFolds=3)
cvModel = crossval.fit(train)
best_model = cvModel.bestModel
print(f"Best Regularization Parameter: {best_model._java_obj.getRegParam()}")
```

MLLIB INTEGRATION WITH OTHER TOOLS

Apache Spark MLlib seamlessly integrates with various tools and frameworks to enhance machine learning workflows.

1. Integration with Apache Spark Ecosystem

- Spark SQL – For querying and manipulating structured data before ML processing.
- Spark Streaming – For real-time machine learning on streaming data.
- GraphX – Graph-based machine learning (e.g., community detection, recommendation systems).

2. Integration with External Libraries

- TensorFlow & PyTorch – Spark MLlib can interact with deep learning frameworks using Petastorm and TensorFlowOnSpark.
- H2O.ai – MLlib works with H2O Sparkling Water for advanced AutoML capabilities.
- Scikit-Learn – MLlib supports exporting models and working with sklearn pipelines via MLeap.

3. Integration with Cloud & Databases

- HDFS, Hive, HBase – Supports data loading from distributed storage systems.
- Amazon S3, Google Cloud Storage, Azure Data Lake – Handles large-scale cloud-based data processing.
- JDBC, MySQL, PostgreSQL – MLlib can fetch and store data from relational databases.

REAL-WORLD USE CASES OF MLLIB

Apache Spark MLlib is widely used in industry for big data machine learning across various domains.

1. Recommendation Systems

- Example: Netflix, Amazon, Spotify
  - Uses ALS (Alternating Least Squares) to recommend movies, products, and music.
  - Personalizes content based on user preferences.

2. Fraud Detection

- Example: Banking & Fintech (PayPal, JPMorgan, Stripe)
  - Uses Logistic Regression & Decision Trees for anomaly detection.
  - Processes large-scale transactions to detect fraud patterns in real time.

3. Customer Churn Prediction

- Example: Telecom & SaaS Companies (AT&T, Salesforce)
  - Uses Random Forest & Gradient Boosted Trees (GBT) to identify customers likely to leave.
  - Helps in targeted retention strategies.

4. Predictive Maintenance

- Example: Manufacturing & IoT (General Electric, Siemens)
  - Uses Time Series Analysis & Regression Models to predict equipment failures.
  - Reduces downtime and maintenance costs.

CHALLENGES IN USING MLLIB

1. Limited Deep Learning Support

- MLlib lacks built-in support for deep learning models like CNNs and RNNs.
- Workarounds involve integrating TensorFlowOnSpark or Petastorm.

2. Memory Management Issues

- Inefficient garbage collection in large clusters may cause OutOfMemory errors.
- Requires manual tuning of Spark configurations (executor memory, driver memory).

3. Limited Algorithm Support

- MLlib lacks XGBoost, CatBoost, and LightGBM (popular in tabular ML).
- Only basic Neural Networks are available, with limited customization.

4. Hyperparameter Tuning Can Be Slow

- Cross-validation and grid search are expensive for large datasets.
- Alternative: Use Bayesian Optimization or RandomizedSearch.

5. High Latency for Small Datasets

- Spark MLlib is not optimized for small datasets; libraries like Scikit-Learn perform better.
- Overhead from distributed computing can slow down model training.

FUTURE OF MLLIB

1. Better Deep Learning Integration

- Improved support for TensorFlow, PyTorch, and Deep Learning Pipelines.
- Enhanced GPU acceleration for large-scale ML workloads.

2. AutoML Capabilities

- Automated hyperparameter tuning using Bayesian Optimization.
- Feature selection and engineering improvements with AutoML frameworks.

3. Real-time Machine Learning

- Closer integration with Spark Streaming for real-time predictions.
- Support for online learning algorithms that adapt to new data.

4. Improved Scalability & Performance

- Optimization of distributed training to handle petabyte-scale datasets.
- More efficient memory management to reduce Garbage Collection (GC) issues.

5. Enhanced Model Deployment & Serving

- Better integration with MLflow, Kubernetes, and cloud platforms.
- Easier deployment of trained models as APIs for real-world applications.

THANK YOU