ChatGPT

Study Plan (4 Weeks)

Week 1: Spark Core and Setup

- **Environment Setup:** Install Java and Python, download Spark (local or standalone) [1]. Start with Spark's official Quick Start to launch `pyspark` or use `SparkSession` [2]. Confirm Spark version and Python compatibility (Spark 4.0.0 uses Python 3.9+ [3]).
- **Core Concepts:** Learn Spark architecture: driver, executors, and RDDs. Understand that a Spark application consists of a **driver program** and parallel **tasks** on executors [4]. Study how RDDs (Resilient Distributed Datasets) work – they are immutable, fault-tolerant, distributed collections [5] [6]. Practice creating simple RDDs with `sc.parallelize()` or by reading local files. Explore transformations (e.g. `map`, `filter`) and actions (e.g. `count`, `collect`) on RDDs [7]. Use Spark's shell or notebooks to try examples (e.g. word count; see Spark Quick Start's word count example [8]).
- **DataFrames & Spark SQL:** Introduce the **DataFrame API**. Understand that a DataFrame is a distributed table of named columns (conceptually like a relational table) [9]. Use `SparkSession.read` to load CSV/JSON/Parquet files and explore DataFrame operations (`select`, `filter`, `groupBy`, `join`). Learn to convert RDDs to DataFrames and vice versa. Explore basic Spark SQL queries using `spark.sql()` and temporary views. Official Spark SQL guide notes that Spark SQL provides richer optimizations than RDDs [10]. Practice by running simple SQL queries on DataFrames.
- **Practice Project:** Implement the classic Word Count (reading text into RDD/DataFrame, `flatMap`, `groupBy`, `count`) [2] [8]. Alternatively, try a basic ETL: e.g. load a CSV (e.g. orders or flights dataset), filter/clean rows, and compute summary statistics (using DataFrames or Spark SQL). DataCamp suggests starting with cleaning a dataset as a PySpark project [11].

Week 2: Advanced Data Processing and Spark SQL

- **Data Handling:** Master reading and writing data: use `spark.read.format(...)` for CSV, JSON, Parquet, Avro, etc., and write data with `df.write`. Learn to handle schemas and infer or specify column types. Practice converting and saving data between formats.
- **Data Manipulation:** Dive deeper into DataFrame operations: complex filters, aggregations, and joins. Use built-in SQL functions (from `pyspark.sql.functions`) for common tasks (date handling, string ops, etc.). Explore `cache()` / `persist()` to improve performance for iterative algorithms [12]. Understand partitioning: use `df.repartition()` or `df.coalesce()` to control parallelism (repartition shuffles data [13], coalesce avoids full shuffle). Learn the difference between narrow (e.g. `map`) and wide transformations (e.g. `groupBy`, which cause shuffle).
- **Spark SQL:** Study how Spark SQL works under the hood. Use Spark SQL's **DDL/DML** features: create Hive-aware tables, use temporary views (`createOrReplaceTempView`) and run SQL queries. The Spark SQL guide highlights that Spark SQL uses an optimized execution engine with Catalyst optimizations [10]. Experiment with SQL vs DataFrame API for the same tasks. Explore window functions and user-defined functions (UDFs) to extend capabilities.
- **Project:** Build a small data analysis pipeline: for example, analyze a sample sales or sensor dataset. Use DataFrames to clean data (handle nulls, drop/replace missing values), then perform aggregations (group by time period, compute metrics). Visualize results (e.g., with Matplotlib or

Seaborn). Optionally use Spark SQL for part of the analysis. Aim to include file I/O, caching, and partitioning strategies.

Week 3: Streaming and Machine Learning Basics

- **Spark Streaming:** Learn about real-time data processing. Start with **DStreams** (Spark's legacy streaming) to grasp the concept of micro-batches [14]. Then focus on **Structured Streaming** (Spark's modern API) – an incremental query engine on DataFrames [15]. Understand concepts: input sources (Kafka, sockets, files), output modes (append, update, complete), event-time vs processing-time. Read the Structured Streaming guide overview (it provides exactly-once fault tolerance via checkpointing [16]). Try a simple streaming example: e.g. use socket text stream to do word count on incoming data.
- **MLlib and Spark ML:** Begin machine learning with Spark. Understand MLlib's scope: it includes common algorithms (classification, regression, clustering) and tools for feature extraction and model tuning [17]. Learn the **Spark ML pipeline** API: Transformers, Estimators, and Pipelines [18]. Practice basic tasks like feature vectorization (`VectorAssembler`), label indexing, and training a simple model (e.g. logistic regression or decision tree) on a DataFrame. Explore model persistence (`.save()`) and evaluation metrics (like BinaryClassificationEvaluator). The Spark guide notes that the DataFrame-based ML API (`spark.ml`) is now primary [19], so focus on that.
- **Optimization Introduction:** Understand basic tuning: use the Spark UI to view stages/tasks, and learn about the **Catalyst optimizer** (Spark's query optimizer) and **Tungsten** execution engine (in-memory optimized processing). Although deep optimization details can be complex, recognize that DataFrames benefit from query planning optimizations [10]. Experiment with persist/storage levels (e.g., `df.persist(StorageLevel.MEMORY_AND_DISK)` [12]). Learn about **broadcast joins**: Spark can automatically broadcast small tables to speed up joins (threshold controlled by `spark.sql.autoBroadcastJoinThreshold` [20]).
- **Project:** Implement a mini streaming/ML project. For example, simulate a stream of sensor or log data (via socket or files) and count events in near real-time. Separately, build a small ML pipeline: pick a dataset (e.g. Titanic or customer churn), apply feature engineering in Spark, train a model, and evaluate. Optionally combine: e.g. read streaming data, apply a pre-trained model to make predictions on the fly (use Structured Streaming + `foreachBatch`).

Week 4: Advanced Topics, Tuning, and Deployment

- **Performance Tuning:** Study Spark's tuning guide. Learn about memory management and serialization [21]. By default Spark uses Java serialization; switching to Kryo (`.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")`) can boost speed [21]. Practice setting `spark.default.parallelism`, `spark.sql.shuffle.partitions`, and adjusting executor memory. Use the UI to identify skew or hotspots. Explore partitioning strategies (range vs hash partitioning) and how to reduce data movement (e.g. using `partitionBy` when writing data). Learn about caching DataFrames used in iterative algorithms [12].
- **Deployment & Integration:** Understand how to deploy Spark jobs. Practice using `spark-submit` for standalone, YARN, or Kubernetes clusters [22] [23]. Compare **cluster** vs **client** mode (in cluster mode, the driver runs on a worker; in client mode it runs locally) [23]. Learn to integrate Spark with other tools: read/write from HDFS/S3, connect to Hive Metastore, and use JDBC sinks/sources. Explore Structured Streaming sinks (e.g. write results back to Kafka or files). Also review how Spark connects to data sources like Hive and Kafka [24] [25].
- **Advanced Features:** Study advanced Spark SQL features like **window functions**, **broadcast hints**, and **UDF vs pandas UDF** for performance. Explore Spark's **Adaptive Query Execution** (if

using Spark 3.x) for dynamic optimizations. Familiarize with **Spark Connect** (if up-to-date), which decouples client and server for PySpark.
- **Capstone Project:** Build an end-to-end Spark application. For example, use Spark on a cluster or EMR: ingest raw data (batch or streaming), process it with transformations and SQL, apply an ML model, and write the results to a database or dashboard. Ensure to use caching and partitioning for efficiency. Document your work (data source, code, results) in a repo or blog to showcase your PySpark proficiency.

Study Material

- **Books:** *Learning Spark (2nd Edition)* by Matei Zaharia et al. (O'Reilly) – comprehensive guide to Spark internals and APIs [26]. *Spark: The Definitive Guide* by Bill Chambers & Matei Zaharia. *High Performance Spark* by Holden Karau and Rachel Warren (focus on tuning). *Learning PySpark* by Tomasz Drabas (practical PySpark applications).
- **Courses & Tutorials:** Official Spark documentation and Quick Start [2] [10]. Databricks Academy "Data Engineering Essentials" and "Applied Data Science with Spark" courses. Online tutorials like SparkByExamples PySpark tutorial [27], and structured PySpark courses on Coursera/Udemy (e.g. *Taming Big Data with Spark + Python*). Free tutorials: [Spark official programming guides (RDD, DataFrames, Streaming, MLlib)] [10] [15], [PySpark API docs] [28].
- **Online Articles & Videos:** Datacamp blog "How to Learn PySpark from Scratch" (includes a sample 4-week plan) [29]. Medium articles on Spark (e.g. Spark interview Q&A [7], join strategies [30]). YouTube talks from Spark Summit or Databricks (Spark architecture, tuning). StackOverflow and official JIRA are great for troubleshooting.
- **Hands-on Practice:** Interactive notebooks (e.g. Databricks Community Edition, Google Colab + PySpark, or DataLab by Datacamp [31]). Kaggle datasets for Spark experiments. GitHub repos with Spark projects. Code along with Kaggle/Big Data community notebooks.

Interview Question Bank

RDDs

- **What is an RDD (Resilient Distributed Dataset)?** – Spark's core abstraction: a fault-tolerant, immutable distributed collection of objects [5].
- **What makes RDDs fault-tolerant?** – RDDs track lineage (the sequence of transformations) so lost data can be recomputed [5] [32].
- **How do you create an RDD?** – By parallelizing a local collection (`sc.parallelize([...])`) or reading from storage (`sc.textFile(...)`) [5].
- **What are common RDD transformations and actions?** – *Transformations* (e.g. `map`, `filter`, `flatMap`) create new RDDs lazily [7]. *Actions* (e.g. `count`, `collect`, `saveAsTextFile`) trigger computation and return results [7].
- **What is lazy evaluation in Spark?** – Transformations on RDDs/DataFrames build up a DAG and are not executed until an action is called [33]. This allows optimizations before execution.
- **How does Spark perform a MapReduce operation?** – You can chain operations: e.g., `textFile.flatMap(...).groupBy(...).mapValues(sum)` is a map-shuffle-reduce flow [8].
- **What are narrow vs wide transformations?** – Narrow transformations (e.g. `map`, `filter`) don't require data shuffle; wide transformations (e.g. `groupByKey`, `reduceByKey`) do shuffle data across the network. Shuffles are expensive operations (see Spark's RDD guide).

- **How do you persist or cache an RDD?** – Use `rdd.cache()` or `rdd.persist(StorageLevel.MEMORY_ONLY)` to store an RDD in memory across the cluster for reuse [12].
- **What is the default storage level of cache()?** – MEMORY_ONLY. Using `persist()` lets you choose levels (e.g. `MEMORY_AND_DISK`) [12].
- **How are RDDs partitioned?** – Data in RDDs is split into partitions across worker nodes [34]. The number of partitions can be set at creation or via `repartition()` / `coalesce()`. Partitioning increases parallelism and fault tolerance [35] [34].

DataFrames

- **What is a DataFrame in Spark?** – A Dataset organized into named columns, equivalent to a relational table with schema [9]. DataFrames are built on RDDs but benefit from Spark SQL's optimizations (Catalyst).
- **How do DataFrames differ from RDDs?** – DataFrames have a schema and support SQL; they use Spark's Catalyst optimizer for better performance [9]. RDDs are lower-level and untyped. DataFrames offer higher-level operations (filters, aggregations) and can automatically optimize query plans.
- **How to create a DataFrame from data?** – Via `SparkSession.read` methods (e.g. `spark.read.csv("file.csv", header=True)`), or by converting an RDD of Rows with a schema. You can also create DataFrames by loading existing tables.
- **What is SparkSession?** – The entry point to Spark SQL and DataFrame API (since Spark 2.0). It internally manages a `SparkContext` and the catalog for tables. Use `SparkSession.builder.getOrCreate()` to start.
- **What is a Dataset?** – In Scala/Java, a typed collection of objects. In Python, all DataFrames are untyped (a Dataset of Rows). (Spark's Dataset API is primarily Scala/Java.)
- **How do you register and query a DataFrame as a SQL table?** – Use `df.createOrReplaceTempView("table")`, then `spark.sql("SELECT ...")`. Spark SQL returns results as a DataFrame.
- **What are common DataFrame operations?** – `select`, `filter` / `where`, `groupBy` / `agg`, `join`, `orderBy`, `withColumn`. Many Spark SQL functions are in `pyspark.sql.functions`.
- **How do you handle missing data in DataFrames?** – Methods like `df.dropna()`, `df.fillna(value)` or using `Imputer` in `pyspark.ml.feature` to fill mean/median [36].
- **What is lazy evaluation for DataFrames?** – Same as RDDs: DataFrame transformations are lazy (building a DAG) and only run upon an action like `collect()` [33].
- **How do you optimize a DataFrame's execution?** – Use `df.explain()` to view the execution plan; cache intermediate results if reused; reduce data (select only needed columns, filter early); use partitioning (e.g. bucketing, partitioning columns on write); avoid UDFs if possible (use built-in functions for Catalyst optimization).
- **What is Catalyst optimizer?** – Spark's rule-based optimizer for DataFrame/SQL queries. It uses schema info to optimize queries (predicate pushdown, column pruning) [37]. (DataFrames benefit from Catalyst optimizations to run faster than equivalent RDD code.)

Spark SQL

- **What is Spark SQL?** – A Spark module for structured data processing. It allows querying data via SQL language or DataFrame API. Spark SQL uses the same execution engine and optimizations for both interfaces [10].

- **How do you run SQL queries in Spark?** – Through `spark.sql("...")` on a SparkSession, which returns a DataFrame. You can also use the Spark Thrift server and JDBC/ODBC for external SQL clients [38].
- **What data sources does Spark SQL support?** – All common formats: text (CSV/JSON), Parquet, ORC, Avro, JDBC, Hive tables, etc. You can read from Hive tables if Spark is configured with a Hive metastore [38]. Spark SQL automatically recognizes DataFrames as tables (you can `createOrReplaceTempView`).
- **How does Spark integrate with Hive?** – Spark SQL can read Hive tables by pointing to a Hive metastore. Use `enableHiveSupport()` in SparkSession to access Hive QL. (In Spark SQL guide: "can also be used to read data from an existing Hive installation" [38].)
- **What are views in Spark SQL?** – *Temporary views* (session-scoped) and *global temporary views*. Created via `createTempView` or `createOrReplaceGlobalTempView`, they let you run SQL queries on DataFrames.
- **How does Spark handle schema inference?** – When reading files, Spark can infer schema (if enabled). DataFrame columns have types; use `df.printSchema()` to inspect. Spark SQL can also define schema via DDL (in SQL) or the DataFrame API.
- **What is the role of the Spark Catalog?** – The catalog keeps track of metadata for databases, tables, views, and functions. You can list tables (`spark.catalog.listTables()`). Spark SQL relies on the catalog (e.g. Hive metastore) for table information.

PySpark Architecture

- **What are the Spark driver and executor?** – The *driver* is the process running the main program (SparkSession/SparkContext). It splits jobs into tasks. *Executors* are JVM processes on worker nodes that run these tasks. Each executor has separate memory and runs multiple tasks in threads.
- **How does a Spark application run on a cluster?** – Spark applications run in either *cluster* or *client* deploy mode [23]. The cluster manager (Standalone/YARN/Mesos/Kubernetes) allocates resources. Spark's DAGScheduler divides jobs into stages of tasks, and the TaskScheduler runs them on executors.
- **What cluster managers does Spark support?** – Standalone (Spark's built-in scheduler), Hadoop YARN, Apache Mesos, and Kubernetes [39]. This allows running Spark on Hadoop clusters or container orchestration platforms.
- **What is `spark-submit`?** – The command-line tool to launch applications on a cluster. It packages the application (Python/Scala/Java), uploads it to the cluster, and starts the driver. You specify deploy mode, master URL, etc.
- **Explain the Spark execution flow (DAG and stages).** – Spark creates a **DAG (Directed Acyclic Graph)** of transformations for each action [40]. The DAGScheduler splits the DAG into **stages** (sets of tasks with only narrow dependencies). Tasks are then executed on executors. Spark's ability to rearrange and pipeline tasks across stages is due to this DAG model [40].
- **What is SparkContext vs SparkSession?** – In Spark 2.x+, SparkSession replaced SQLContext and HiveContext. `SparkContext` is the entry point for core (RDD) functionality; `SparkSession` (which includes a `SparkContext`) is the unified entry point for DataFrame and SQL APIs. Calling `SparkSession.builder.getOrCreate()` initializes both.
- **What languages does Spark support?** – High-level APIs in Scala, Java, Python (PySpark), and R (SparkR). PySpark is Spark's Python API, allowing Python programs to use Spark features [41].
- **How does Spark achieve fault tolerance on failures?** – If a task fails, Spark reassigns it. If an entire executor is lost, Spark can recompute lost partitions from lineage. This is enabled by RDD's immutability and persisted state.

Transformations & Actions

- **Define transformation and action in Spark.** – A *transformation* (e.g. `map`, `filter`, `flatMap`) creates a new RDD/DataFrame from an existing one; it is lazy [7]. An *action* (e.g. `count`, `collect`, `saveAsTextFile`) triggers computation and returns a result or writes output [7].
- **Give examples of transformations.** – Examples include `map(func)`, `filter(func)`, `flatMap(func)`, `groupByKey()`, `reduceByKey(func)`, `join()`, `sortBy()`. All these build a new dataset without performing computation until an action.
- **Give examples of actions.** – `count()`, `collect()`, `take(n)`, `first()`, `reduce(func)`, `saveAsTextFile(path)`, `foreach()`. These force Spark to compute all needed transformations.
- **What is the difference between `map` and `flatMap`?** – `map` applies a function to each element and returns one element per input. `flatMap` can return zero or multiple elements per input (useful for splitting a sentence into words).
- **What is `reduceByKey` vs `groupByKey`?** – Both are pair-RDD aggregations. `groupByKey` gathers all values for a key then applies a function, causing a full shuffle. `reduceByKey` applies a reduce function locally on each partition before the shuffle, which is more efficient (combines data along the way).
- **When are transformations evaluated?** – Only when an action is called. Spark builds an execution plan (DAG) through transformations and waits for an action to execute it.
- **How does Spark handle UDFs?** – User-Defined Functions (Python UDFs) can be applied on DataFrames, but they can be slower since they often break Catalyst optimizations. Since Spark 2.3, **Pandas UDFs** (vectorized) are faster for Python.

Optimization Techniques

- **What is data partitioning and why is it important?** – Partitioning splits data into chunks across the cluster [35]. Proper partitioning increases parallelism and can reduce shuffle (if keys are partitioned similarly). You can explicitly repartition or partitionBy a DataFrame when saving.
- **What is predicate pushdown?** – Spark can push down filters (e.g. WHERE clauses) to data source readers (like Parquet) to avoid reading unneeded data. This optimization happens automatically in DataFrame operations when supported.
- **How to broadcast a small lookup table?** – Use Spark's broadcast variables to send a small dataset to all workers [42]. In SQL/DataFrames, Spark can automatically broadcast small tables in joins (controlled by `spark.sql.autoBroadcastJoinThreshold`) [20]. You can also hint a join as broadcast with `.hint("broadcast")`.
- **What is spark.sql.autoBroadcastJoinThreshold?** – A configuration that sets the maximum size (default 10 MB) for a table to be automatically broadcast in a join [20]. Tables smaller than this threshold will be broadcast to all executors for a broadcast hash join (avoiding shuffle).
- **What is a broadcast join?** – A join strategy where Spark broadcasts the smaller table to all nodes, so the join can happen without shuffling the large table [30]. Very efficient when one dataset is small.
- **What is a shuffle sort-merge join?** – Spark's default join for large tables: it shuffles both datasets on the join key, sorts partitions, and then merges matching keys [43]. This incurs more network I/O but works for large data.
- **What is WholeStage code generation?** – An optimization where Spark compiles parts of the query plan into optimized bytecode for better performance (happens under the hood in Catalyst for DataFrame queries).
- **What are bucketing and partitioning on write?** – Data can be partitioned by a column (e.g. date) or bucketed into fixed buckets by key when writing to storage. Bucketing (sorting data into

fixed “buckets”) can speed up joins by reducing shuffle if both sides are bucketed with the same scheme. It is used in advanced joins like Sort-Merge Bucket join [44] .

Performance Tuning

- **How do you monitor a Spark application?** – Use the Spark Web UI (available on the driver’s port) to inspect DAGs, stages, tasks, storage memory, and SQL query plans.
- **How to tune memory usage?** – Adjust executor memory settings (`spark.executor.memory`, `spark.memory.fraction`, etc.). Persist (cache) only what’s needed. The Spark Tuning guide advises storing RDDs in serialized form or using Kryo to save space [45] . Use the Garbage Collection tuning to avoid long GC pauses.
- **What serialization formats does Spark use?** – By default Spark uses Java serialization (`ObjectOutputStream`), but you can switch to **Kryo** for better performance (faster and smaller) [21] . Kryo requires registering classes but yields ~10x faster serialization [21] .
- **How to enable Kryo serialization?** – Set `spark.serializer` to `org.apache.spark.serializer.KryoSerializer` in your SparkConf [46] and (optionally) register classes with `conf.registerKryoClasses(...)` [46] .
- **How does Spark handle shuffles?** – Shuffle operations (e.g. `groupByKey`, `reduceByKey`, joins) write data to disk and send it across the network to align keys. Spark writes shuffle files and manages partitioning for the next stage. Reducing shuffles (via partitioning or broadcast joins) improves performance.
- **What is level of parallelism?** – Number of partitions/tasks. Controlled by `spark.default.parallelism` (for RDD) or `spark.sql.shuffle.partitions` (for DataFrames). More partitions means more tasks – generally improves parallelism if tasks are balanced.

Serialization

- **What is Spark’s default serializer?** – Java serialization (via `java.io.Serializable`). It’s generic but often slow and bulky [21] .
- **What is Kryo serializer and why use it?** – Kryo is a more efficient binary serializer that can be ~10x faster than Java serialization [21] . Use it for network-intensive jobs (shuffle). Set `spark.serializer=KryoSerializer` in the config [46] .
- **When does Spark use Kryo by default?** – Since Spark 2.0, Spark uses Kryo internally for shuffling simple types like arrays of primitive types or strings [47] , but user-defined classes require manual registration.
- **What is `spark.kryoserializer.buffer`?** – Configures the initial buffer size for Kryo serialization. Increase it if you serialize large objects [48] .
- **How do broadcast variables get serialized?** – Broadcast variables are serialized (by default) and sent to each executor. If using Kryo serializer, they benefit from it. Large broadcast variables should also use efficient serialization and not be mutated (they are read-only) [42] .

Deployment

- **What are cluster deploy modes in Spark?** – *Client mode:* driver runs on the client machine; *Cluster mode:* driver runs inside the cluster (on a worker) [23] . Use cluster mode for production jobs to utilize cluster resources.
- **How do you run Spark on YARN?** – Deploy the application with `spark-submit --master yarn`. YARN acts as the cluster manager, allocating containers for the driver and executors [49] .
- **How do you run Spark on Kubernetes?** – Spark can run on Kubernetes by setting `--master k8s://...` and providing Docker images for Spark. This uses Kubernetes API for resource management [50] .

- **What is Spark Standalone mode?** – A simple cluster manager included with Spark. You manually start a master and workers (via `start-master.sh`, `start-worker.sh`), then submit jobs to the master URL.
- **How to submit a PySpark application?** – Package your `.py` script and any dependencies, then run `spark-submit --master [master-url] your_app.py`. This launches the driver and executors on the cluster.
- **What are Spark executors and how many should you use?** – Executors are JVM processes on worker nodes. The number depends on cluster size and available CPU cores (e.g. one executor per worker, with several cores each). Tune executors' count and cores via `--num-executors`, `--executor-cores`.
- **What is dynamic resource allocation?** – A Spark feature that can add/remove executors based on workload. (Enable with `spark.dynamicAllocation.enabled`.) It prevents idle executors from wasting resources.
- **What deployment tools support Spark?** – Spark can be deployed with scripts, or via cluster managers (YARN, Mesos, Kubernetes). Tools like Apache Livy or Databricks Runtime also manage Spark jobs.

Spark Streaming (DStreams)

- **What is Spark Streaming (DStreams)?** – A legacy Spark API for processing live data in micro-batches. It divides data into small batches (e.g. every 1 sec) and processes each with Spark core [14]. DStreams are sequences of RDDs representing the stream.
- **How do you create a DStream?** – Use a `StreamingContext` (built on `SparkContext`) and input sources: e.g. `ssc.socketTextStream(host, port)` or Kafka/Flume receivers [51].
- **What are receivers in Spark Streaming?** – Components that ingest data from sources (Kafka, Flume, TCP sockets). They run on executors, receive data, and store it in Spark for processing [52]. There are reliable (write-ahead logs) and unreliable receivers (no ack).
- **What is a window operation?** – Allows processing over sliding time windows (e.g. count events over last 1 minute, updated every 5 seconds). Use `window`, `reduceByKeyAndWindow`, etc. Windowed computations group multiple RDD batches.
- **What is checkpointing?** – Periodically saving streaming state (metadata, RDD lineage) to fault-tolerant storage. Required for stateful transformations and recovery. Ensures fault tolerance for state.
- **How do you handle stateful processing?** – Using functions like `updateStateByKey` (or the newer `mapWithState`), Spark Streaming can maintain and update state across batches (e.g. running counts). State data should be checkpointed.
- **How can you integrate Kafka with Spark Streaming?** – Use the Kafka integration libraries: either the older receiver-based approach (`KafkaUtils.createStream`) or the "Direct" approach (`createDirectStream`) which reads directly from Kafka partitions for exactly-once semantics.

Structured Streaming

- **What is Structured Streaming?** – Spark's newer stream processing API built on DataFrames/Datasets. Users write queries on streaming DataFrames the same way as batch. Spark runs them incrementally with exactly-once guarantees [15].
- **How does Structured Streaming differ from DStreams?** – Structured Streaming treats stream as an unbounded table. It supports event-time processing, SQL queries, and end-to-end fault tolerance by default [15]. It also introduces output modes (Append/Update/Complete) and watermarks for late data.

- **What are output modes?** – *Append:* only new rows are output. *Update:* updated results are output. *Complete:* all results are output each trigger. (E.g., aggregation queries often use Complete mode.)
- **What are watermarks?** – A mechanism to handle late data: it specifies how late data can arrive (e.g. "10 minutes") for event-time windows, allowing state to be dropped for older windows. Without watermarks, state could grow unbounded.
- **What are triggers in Structured Streaming?** – Control when queries process data: `Trigger.ProcessingTime("10 seconds")` for micro-batch intervals, `Trigger.Once()` for batch-at-a-time, and `Trigger.Continuous("1 second")` for low-latency continuous mode (available in Spark $\geq$2.3) [53].
- **What is exactly-once semantics?** – Structured Streaming guarantees that each record in the input contributes exactly once to the final output, even in face of failures [16]. It uses checkpointing and write-ahead logs internally to achieve this.
- **How do you read/write from Kafka in Structured Streaming?** – Use `spark.readStream.format("kafka")` with Kafka parameters, and `df.writeStream.format("kafka")` to send output. Structured Streaming handles offsets and retries automatically.

MLlib and Spark ML

- **What is MLlib?** – Apache Spark's machine learning library, providing scalable ML algorithms and utilities [17]. Includes classification, regression, clustering, and feature transformers.
- **What is Spark ML (in contrast to MLlib)?** – "Spark ML" refers to the DataFrame-based API in `spark.ml` (pipeline API). The original RDD-based MLlib (`spark.mllib`) is now in maintenance mode [54]. The DataFrame API (`spark.ml`) is the recommended interface.
- **What is a Transformer in Spark ML?** – An abstraction that converts one DataFrame into another, typically by adding or modifying columns [18]. E.g., a trained model (like a fitted `LogisticRegressionModel`) is a Transformer: it adds a column of predictions.
- **What is an Estimator?** – An algorithm that can be fit on a DataFrame to produce a Transformer. E.g., `LogisticRegression()` is an Estimator: calling `.fit(df)` produces a `LogisticRegressionModel` (a Transformer) [18].
- **What is a Pipeline?** – A sequence of Transformers and Estimators assembled into a workflow. You define a Pipeline with stages; calling `.fit()` runs all stages in order (fitting each Estimator) to produce a PipelineModel [55].
- **What is a Param?** – A parameter in a Transformer or Estimator (e.g. learning rate, number of trees). Spark ML provides a uniform API for setting parameters. This enables tools like `ParamGridBuilder` for hyperparameter tuning.
- **How do you do model selection (cross-validation) in Spark ML?** – Use `CrossValidator` or `TrainValidationSplit` with an `Estimator`, an `Evaluator`, and a `ParamMap` grid. `CrossValidator` performs k-fold CV; `TrainValidationSplit` does a single split. They use Pipeline or Estimator APIs.
- **What ML algorithms are available in Spark MLlib?** – Regression (Linear/Logistic Regression, Decision Tree, Random Forest, GBT), Classification (Naive Bayes, SVM), Clustering (K-Means, Gaussian Mixture), Collaborative Filtering (ALS), etc. Feature transformers include `StringIndexer`, `OneHotEncoder`, `VectorAssembler`, etc.
- **How does Spark ML handle feature vectors?** – Spark uses `Vector` types (`DenseVector`/`SparseVector`) to store feature arrays. Use `VectorAssembler` to combine raw features into a single vector column. Many ML algorithms expect a `features` column of type `Vector` and a label column.

Partitioning

- **Why is partitioning data important in Spark?** – Proper partitioning distributes data evenly and localizes computation. Good partitioning improves parallelism and can reduce shuffle.
- **What is the default number of partitions?** – For RDDs, default is `spark.default.parallelism` (often 2× number of cores). For DataFrames, default shuffle partitions is 200 (`spark.sql.shuffle.partitions`). Adjust as needed.
- **How do you repartition a DataFrame/RDD?** – Use `df.repartition(numPartitions)` or `df.repartition(col)` to shuffle and repartition by a column; use `df.coalesce(numPartitions)` to reduce partitions without full shuffle [13].
- **What is Hash vs Range partitioning?** – *HashPartitioner* (default for pair RDDs) assigns partitions by hashing the key. *RangePartitioner* (for sorted operations) splits data into ranges (requires sortable keys). You can use `rangeBy("key")` in DataFrame writes.
- **What is partitionBy when writing data?** – When writing to file formats, `df.write.partitionBy("col")` creates directory partitions by distinct values of that column. This can optimize queries that filter on that column by skipping unrelated partitions.
- **How to avoid data skew?** – If one partition is much larger (due to skewed key), it causes slowdown. Mitigate by increasing number of partitions, using salting (add random prefix to keys) or using custom partitioner.

Joins

- **What types of joins does Spark support?** – Spark DataFrames support inner, outer (full), left outer, right outer, left semi, left anti, and cross joins. Use `df.join(df2, on, how="inner")` etc.
- **What is a broadcast join?** – See **Join Strategies**. If one table is small (fits in memory), Spark can broadcast it to all nodes and join locally [30]. This avoids shuffle of the large table. Use hints or rely on `autoBroadcastJoinThreshold`.
- **What is sort-merge join?** – Default join for large tables. Spark shuffles both tables by key, sorts them, and merges matching rows [43]. Requires shuffle and sort, good for large data.
- **What is shuffle hash join?** – Alternative where Spark builds a hash table per partition instead of sorting [56]. (Spark may use this in some scenarios or on hint.) It still shuffles both sides.
- **What is a bucketed join (Sort-Merge Bucket join)?** – If both tables are bucketed/sorted by the join key with the same number of buckets, Spark can perform a bucket join without full shuffle [44]. Both tables must have identical bucketing.
- **When should you broadcast a join?** – When one table is much smaller (e.g. dimension table) and can fit in memory on each executor. This is more efficient than shuffling both. The threshold is set by `spark.sql.autoBroadcastJoinThreshold` (default 10MB) [20].
- **How to force a join strategy?** – Use DataFrame hints: e.g., `df.join(df2.hint("broadcast"), on, how="inner")` to force broadcast, or `.hint("shuffle_merge")`.
- **Why are join operations expensive?** – They often involve shuffling large amounts of data across the network and sorting. Reducing data size (filtering before join), increasing cluster resources, or using broadcast can help.
- **What is a sort-merge vs shuffle-hash cost trade-off?** – Sort-merge joins have high cost due to sorting both datasets [43], but are scalable for very large data. Shuffle-hash can be faster for moderate-sized joins but still needs enough memory per partition.

Broadcast Variables

- **What are broadcast variables in Spark?** – Read-only variables cached on all executors for efficiency [42]. Instead of shipping copies of a large value to each task, Spark sends it once to each node. Useful for small lookup data used by all tasks.
- **How to use a broadcast variable in PySpark?** – In Python: `b = sc.broadcast(myVar)`. Then use `b.value` in transformations. All tasks on each node share the single copy of `myVar` via `b`.
- **When should you use broadcast variables?** – When you have a relatively small dataset (like a lookup table or a pre-trained model) that is used by all tasks. Broadcasting avoids expensive shuffles of that data [30] [42].
- **How are broadcast variables serialized?** – They are serialized (using the configured serializer) and sent to each executor once. Large broadcast may exceed the default size limit, so check `spark.driver.maxResultSize` or use a distributed file for very large data.
- **What is the difference between a broadcast variable and an accumulator?** – Broadcast variables are read-only shared data. Accumulators are write-only from executors to the driver (e.g., counters or sums) [57].

Integration with Other Tools

- **How does Spark integrate with Hadoop?** – Spark can read/write from HDFS, and use YARN as a cluster manager [50]. It can be added to the Hadoop ecosystem by placing it on HDFS and using Yarn for resource scheduling [22] [49].
- **How does Spark integrate with Hive?** – By enabling Hive support in SparkSession, Spark can use the Hive metastore. Tables defined in Hive can be queried via Spark SQL.
- **How does Spark connect to Kafka?** – Structured Streaming provides a Kafka connector: `spark.readStream.format("kafka")`. DStreams (legacy) used separate Kafka receivers. Both allow Spark to consume Kafka topics.
- **Can Spark read from Cassandra/HBase/ElasticSearch?** – Yes, through connector libraries (e.g. Spark Cassandra Connector, HBase Spark Connector, Elasticsearch-Hadoop). These are not built-in but widely used. (No official ref cited here.)
- **How does Spark use JDBC sources?** – Use `spark.read.jdbc(url, table, properties)` to load a table from a database into a DataFrame. Similarly, `df.write.jdbc()` can save DataFrames back to a database.
- **What is Spark's support for cloud storage?** – Spark can access AWS S3, Google Cloud Storage, Azure Blob, etc., using Hadoop-compatible file system URIs (e.g. `s3://` paths), assuming proper credentials.
- **How does Spark integrate with Kubernetes?** – Spark can be deployed in Kubernetes: the cluster manager is Kubernetes (via the spark-kubernetes integration). Spark pods (driver + executors) run as containers managed by Kubernetes.
- **What about Spark on EMR or Databricks?** – AWS EMR and Databricks provide managed Spark clusters. Code runs the same; these platforms simplify setup and add features like optimized storage formats or notebooks.

Sources: Apache Spark official documentation [26] [10] [5] [15] [17], DataCamp and Medium tutorials [29] [7] [30], Spark community Q&A and blogs.

---

[1] [3] [22] [26] [39] [50] Overview - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/

2 8 Quick Start - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/quick-start.html

4 5 42 57 RDD Programming Guide - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/rdd-programming-guide.html

6 32 34 What Is a Resilient Distributed Dataset (RDD)? | IBM
https://www.ibm.com/think/topics/resilient-distributed-dataset

7 12 13 33 35 Apache Spark Interview Questions and Answers | by Sanjay Kumar PhD | Medium
https://skphd.medium.com/apache-spark-interview-questions-cba3629a5bbe

9 10 24 38 Spark SQL and DataFrames - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/sql-programming-guide.html

11 29 31 How to Learn PySpark From Scratch in 2025 | DataCamp
https://www.datacamp.com/blog/learn-pyspark

14 25 51 Spark Streaming - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/streaming-programming-guide.html

15 16 53 Structured Streaming Programming Guide - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/streaming/index.html

17 19 37 54 MLlib: Main Guide - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/ml-guide.html

18 55 ML Pipelines - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/ml-pipeline.html

20 30 43 44 56 Join Strategies in Apache Spark. In this blog, we'll break down the... | by Hrushikesh Kute | Medium
https://medium.com/@hrushikeshkute/join-strategies-in-apache-spark-1bbc0a698896

21 45 46 47 48 Tuning - Spark 4.0.0 Documentation
https://spark.apache.org/docs/latest/tuning.html

23 40 49 52 Top 24 Applications of AI: Transforming Industries Today
https://www.simplilearn.com/top-apache-spark-interview-questions-and-answers-article

27 41 PySpark 3.5 Tutorial For Beginners with Examples - Spark By {Examples}
https://sparkbyexamples.com/pyspark-tutorial/

28 Getting Started — PySpark 4.0.0 documentation
https://spark.apache.org/docs/latest/api/python/getting_started/index.html

36 Top 36 PySpark Interview Questions and Answers for 2025 | DataCamp
https://www.datacamp.com/blog/pyspark-interview-questions