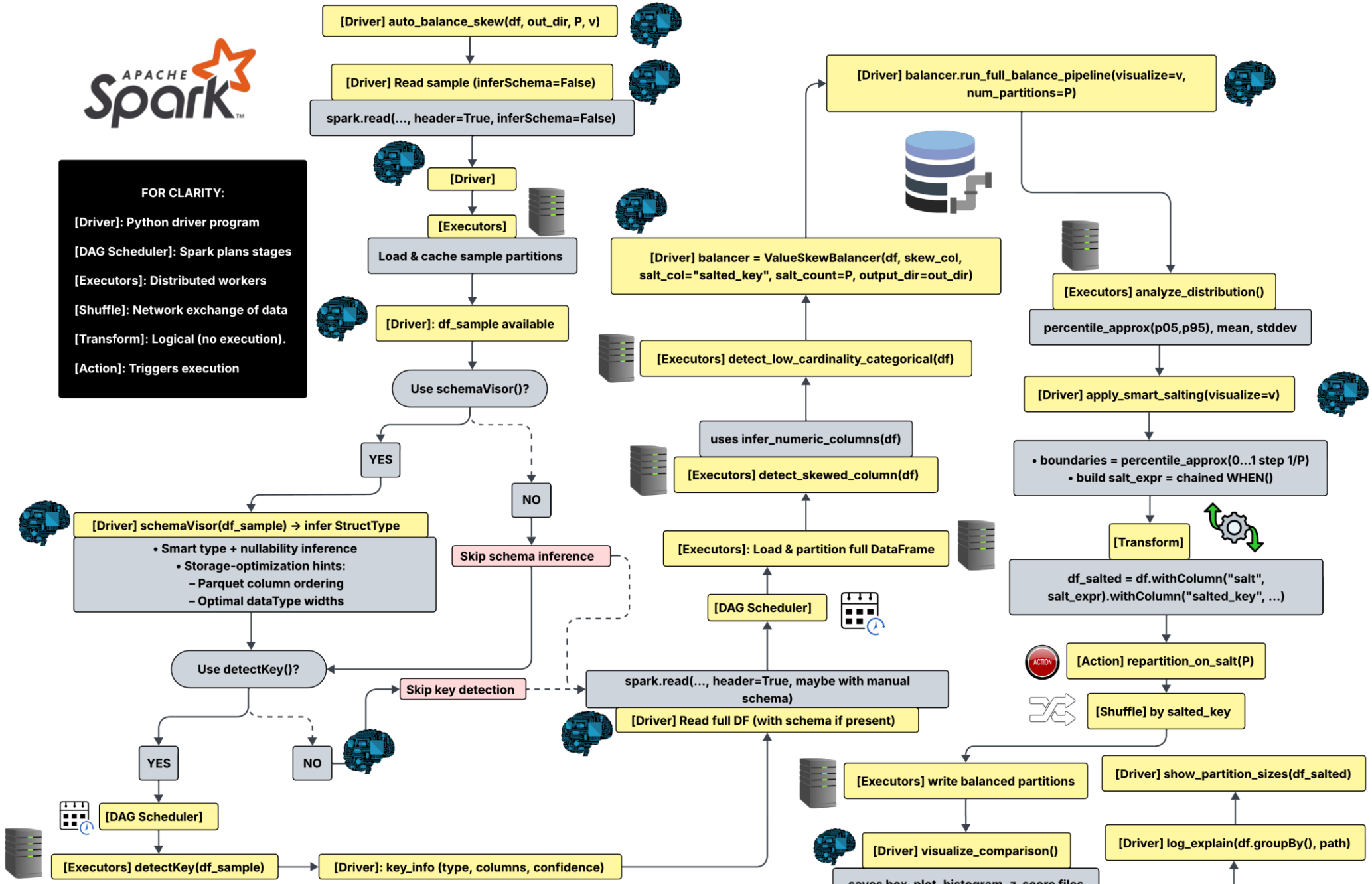
By Omar Attia // IBM Data Engineer

SkewBalancer: Auto-Optimized Spark Salting V2

- **Smart Schema Detection (schemaVisor)**
  - Emits Parquet column ordering hints and minimal-width types for maximum I/O throughput.
- **Key Auto-Detection (detectKey)**
  - Scores columns by distinct/N – nulls/N (≥0.99 → primary), tries combos up to size 3, then length heuristics for surrogates.
- **Hash-Key Mapping**
  - Guarantees reproducible, uniform binning by mapping each bucket ID through a consistent hash function.

Full Architecture



By Omar Attia // IBM Data Engineer

Full Architecture (Explanation) - 1

Method (1) auto_balance_skew(df, output_dir, partitions, verbose)

- Runs the entire skew-fix pipeline in one call: it picks your worst column, applies salting, rebalances partitions, then makes before/after charts and logs.

```
skew_col = detect_skewed_column(df)
group_key = detect_low_cardinality_categorical(df)
balancer = ValueSkewBalancer(df, skew_col, salt_count=partitions,
output_dir = output_dir)
df_salted = balancer.run_full_balance_pipeline(visualize=verbose,
num_partitions=partitions)
timeit(groupBy.count); log_explain(); show_partition_sizes()
```

Method (2) infer_numeric_columns(df)

- Automatically finds which DataFrame columns hold ints or floats so you know what to test for skew.

Filters the schema by:

f.dataType.simpleString() in {"int","double","float","bigint","long"}

in O(m) time for m columns.

By Omar Attia // IBM Data Engineer

Full Architecture (Explanation) - 2

Method (3) detect_skewed_column(df, verbose=False)

- Chooses the single numeric column whose distribution is most "lopsided" (skewed) via its quartile gaps.

For each col:

1. Compute Q1,Q2,Q3 via percentile_approx(...,[0.25,0.5,0.75])
2. IQR = Q3–Q1
3. Skewness = $|(Q3-Q2)-(Q2-Q1)|$IQR|(Q3−Q2)−(Q2−Q1)| Select max if >0.1. Complexity O(n·m).

Method (4) detect_low_cardinality_categorical(df)

- Picks a string column with 20 or fewer unique values so you can do a cheap groupBy.

✓ Scans each string field, runs countDistinct until it finds distinct_count ≤ 20. Runs in O(n·k) for k string cols.

By Omar Attia // IBM Data Engineer

Full Architecture (Explanation) - 3

Method (5) detectKey(df, max_composite=3, verbose=False)

- (Optionally) finds a primary—or composite—key by checking which columns (or small combos) uniquely identify rows.

✓ Score each col:

$$\text{score}(c) = \frac{|\text{distinct}(c)|}{N} - \frac{|\text{nulls}(c)|}{N}$$

✓ If max score ≥ 0.99 → Primary.

✓ Else try combos up to size 3 ($O(m^3 \cdot n)$), then surrogate if avg_len>8.

✓ Chebyshev bound ⇒ ≤1% collision risk at score 0.99.

By Omar Attia // IBM Data Engineer

Full Architecture (Explanation) - 4

Method (6) schemaVisor(df, n_chunks=8)

- (Optionally) samples up to 5 000 rows per text column to guess whether it's an integer, a float, or text—and locks key columns as non-nullable.

✓ Samples 20% (min 5 000 rows) → Pandas to compute
  - numeric_ratio = isdigit/S
  - float_ratio = isfloat/S

✓ Assigns **IntegerType** if ≥95% numeric, **DoubleType** if ≥95% float, else **StringType**.

✓ Hoeffding bound ⇒ ε≈1.1% at 95% CI for n=5 000.

Method (7) analyze_distribution() - CORE

- Quickly pulls the 5th and 95th percentiles, plus the mean and standard deviation of your skew column.

```
%%sql

SELECT
  percentile_approx(col, 0.05) AS p05,
  percentile_approx(col, 0.95) AS p95,
  mean(col)            AS mean,
  stddev(col)          AS std
```

By Omar Attia // IBM
Data Engineer

Full Architecture (Explanation) - 5

Method (8) apply_smart_salting(visualize=True)

- Automatically divides your skewed values into equal-count bins, tags each row with a “salt” index, and (optionally) shows before/after plots.

1. Executors compute

```
%%sql

percentile_approx(col, array(0,1/N, 2/N, …, 1)) → boundaries
```

2. Driver builds

```
%%python

salt_expr = when(cond0, 0).when(cond1, 1)…otherwise(0)
```

3. Adds .withColumn("salt", salt_expr) & .withColumn("salted_key", …)

- Local compile cost O(N)
- Shuffle cost O(n).

By Omar Attia // IBM Data Engineer

Full Architecture (Explanation) - 6

Method (8) repartition_on_salt(num_partitions)

- Tells Spark to re-shuffle your data by the new salted_key, spreading rows evenly.

Action Trigger

```
%%python

df_salted.repartition(num_partitions, col("salted_key"))
```

→ triggers a full exchange hashpartitioning on salted_key.

Method (9) visualize_comparison()

- Saves three visuals—boxplot, histogram, and Z-Score charts—so you can see how skew was reduced.

✓ Samples 5% of rows → Pandas.

✓ Z-scores = $(x-\mu)/\sigma$(x−μ)/σ; plots ±3 thresholds.

✓ Boxplot & KDE histogram via Matplotlib/Seaborn.

By Omar Attia // IBM Data Engineer

Full Architecture (Explanation) - 7

Method (10) timeit(func, *args, label="", **kwargs)

- Measures and prints how long any given Spark action takes.

Wraps Python's time.time() before/after:

```
%%python

start = time.time(); result = func(); end = time.time()
print(f"[{label}] Time: {end - start:.3f} sec")
```

Method (11) log_explain(df, filename)

- Writes Spark's physical execution plan to a text file for side-by-side review.

```
%%python

plan = df._jdf.queryExecution().toString()

with open(filename, "w") as f:
   f.write(plan)
```

By Omar Attia // IBM Data Engineer

Full Architecture (Explanation) - 8

Method (12) show_partition_sizes(df, label="")

- Prints how many rows ended up in each Spark partition, so you can verify balance.

```
%%python

sizes = df.rdd.glom().map(len).collect()
```

Gathers per-partition counts in O(n) after the shuffle.

By Omar Attia // IBM Data Engineer