Option 1 – One-Time Ingestion (Offline Simulation)

This means:

- We have an **open-source static dataset** (e.g., CSV with millions of rows of sensor readings).
- build a **Kafka producer script** that reads that dataset and sends the messages *sequentially* (as if they're arriving live) — but **it ends when the file ends**.
- can rerun it whenever we want for new experiments.

Pros:

- Much easier to control and reproduce experiments.
- We can run the whole training in **finite rounds** (like FedAvg round 1, 2, 3).
- Ideal for our academic / prototype project setup.

Cons:

- It's not *true streaming* — after the dataset ends, the system stops producing new data unless restarted.

So in this case:

- simulate "live" behavior by just **sending data row by row with small delay (e.g. 1 sec)**.
- But it's technically **a one-time ingestion** — not infinite stream.

Option 2 – Continuous Live Ingestion (True Streaming)

This means:

- Kafka producer keeps producing data **without stopping** (e.g., a synthetic data generator that never ends).
- Flink keeps training local models continuously (like edge devices operating in real time).
- Aggregator receives constant model updates and keeps producing new global versions.

Pros:

- Most realistic (simulates a real IoT ecosystem).
- Good for demonstrating “stream learning” and online adaptation.

Cons:

- Very heavy — models keep updating endlessly.
- Hard to evaluate convergence.
- Hard to run Spark batch analytics (because the model never stabilizes).

The recommended hybrid

Since we have open-source datasets and not real devices:

Use **Offline Simulation + Stream-like Behavior**

- Treat datasets as **finite but replayable streams**.

Producer will **loop through the dataset continuously**:

- So Kafka always has live-like flow,
  but the source data is still finite and controlled.

- This gives us **stream behavior**, but keeps system manageable.