🔄 Scalable Architecture

1. **Ingestion (Thousands of Documents)**
    - Source documents from **Cloud Storage** (e.g., AWS S3, GCP Storage, Azure Blob).
    - Use parallel processing (e.g., Spark, Ray, or multiprocessing in Python) to read and distribute workloads or batch processing.
    - In case parallel processing Each file is queued (e.g., with Kafka, SQS, or Pub/Sub) to ensure scalability.
2. **Processing (Normalization & Chunking)**
    - Workers pull jobs from the queue to **extract text, normalize, and chunk** documents in parallel.
    - Scales horizontally (add more workers for more throughput).
    - Orchestration tools like **Airflow, Dagster** schedule and monitor tasks.
3. **Storage**
    - Instead of SQLite, use scalable stores:
        - **Metadata & chunks** → Cloud Warehouse (AWS Redshift, BigQuery, Snowflake).
        - **Semantic search** → Vector DB (Pinecone, Weaviate, Milvus, FAISS).
    - Keeps queries efficient even with millions of chunks.
4. **Query Interface**
    - Expose queries via **API layer** (FastAPI or Flask).
    - API retrieves relevant chunks from the warehouse/vector DB.
    - Passes context into **LLM** (hosted via Ollama, OpenAI, Anthropic, or a fine-tuned local model).
    - API scales with load balancers (Kubernetes, etc.).

5. **Orchestration**
    - **Airflow** (or similar) manages:
        - Scheduled ingestion (daily/hourly crawls of new documents).
        - Monitoring and retries for failed tasks.
        - Dependency management (e.g., don't run chunking before ingestion completes).
6. **Cloud-Native Scaling**
    - **Storage layer**: resilient and scalable (S3 + Redshift/BigQuery).
    - **Compute**: containerized workers (Docker + Kubernetes).
    - **APIs**: autoscale with Kubernetes or serverless (AWS Lambda, Cloud Run).