RAG

From search to answers

Building an agentic, multi-modal RAG platform with OpenSearch

Naresh Waswani
Senior Architect, Simpplr Inc.

Jyoti Notani
Architect, Persistent Systems Ltd

WHERE THIS COMES FROM

Search has been evolving

Keyword

Lexical, BM25. Matches the exact words. Precise, literal, blind to meaning.

→

Semantic

Vector similarity. Matches meaning even when the words differ.

→

Answers

Generative AI composes a response instead of returning a list of links.

Each step adds intelligence. Each step also adds a way to get the answer wrong. This talk is about doing the last two reliably at enterprise scale.

THE GAP

A model only knows what it was trained on

What an LLM gives you

- You send it a question, it returns an answer.
- Fluent, fast, and confident.
- But grounded only in its training data, with a fixed cutoff.
- It has never seen your wiki, your tickets, or your HR policies.

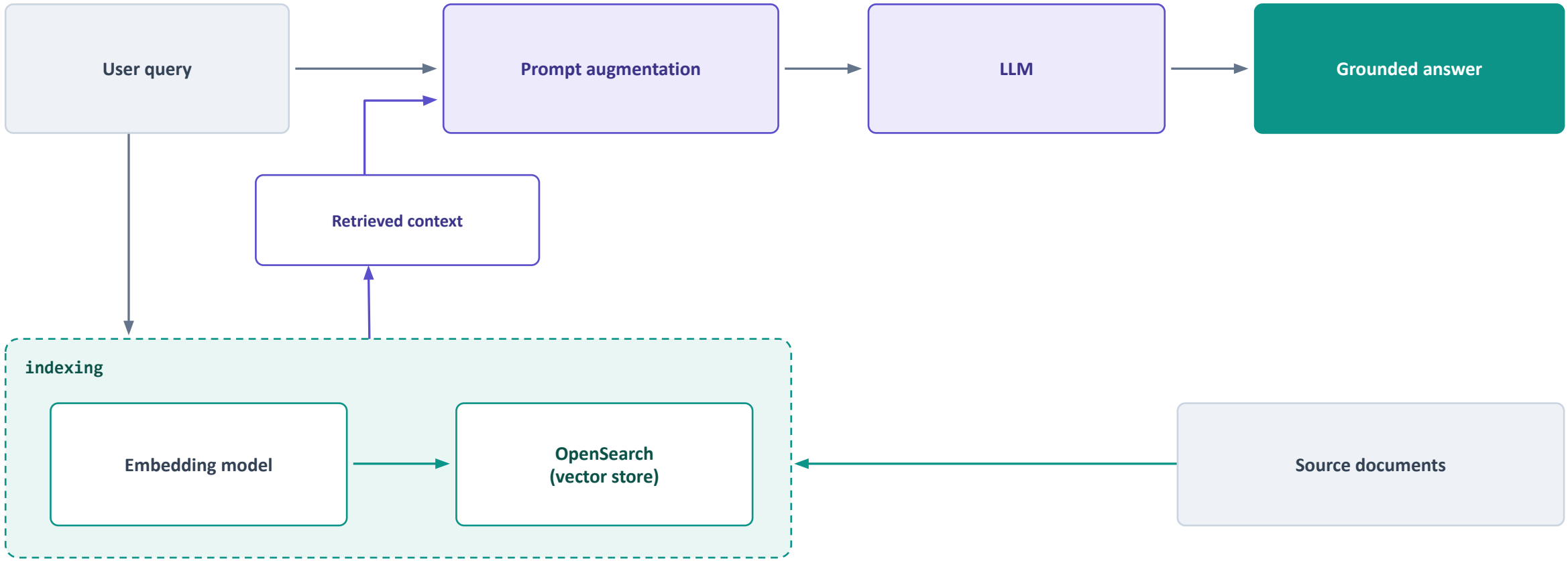
So we build RAG

- Retrieve your enterprise data first.
- Hand it to the model as context.
- Now the answer is grounded in your documents, with citations.
- OpenSearch is where that retrieval happens.

Retrieval-Augmented Generation: give the model your data at answer time, instead of hoping it memorized it.

RAG, HIGH LEVEL

Push data in, retrieve, then answer



THE ONE THING TO REMEMBER

Your answer is only as good as your input.

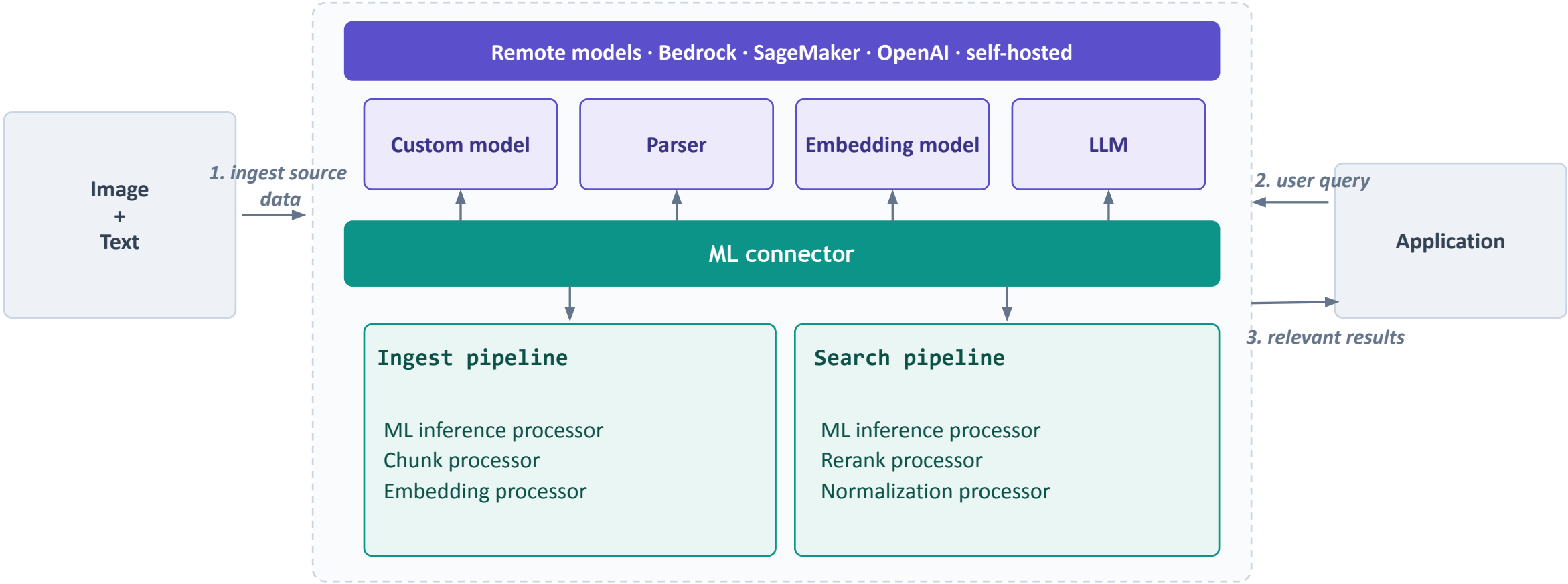
The model is the easy part. Everything interesting and everything that breaks is in getting the right data into OpenSearch and the right chunks back out. That is what the rest of this talk is about.

PART 1 · OPENSEARCH AS THE ENGINE

OpenSearch is not just the database

Ingest pipelines, search pipelines, and ML connectors.

The connector, the pipelines, the models



SUPPORTED CONNECTORS

Two execution modes under one framework

Local, in-cluster

- Lightweight models run on the cluster nodes.
- Good for BERT-class text embedders.
- No external hop, simplest to operate.

Remote, via connector

- Heavy or generative models run on a remote service.
- Embedding, captioning, parsing, full LLMs.
- OpenSearch streams the payload out and writes the result back.

Connector blueprints today: OpenAI · Amazon Bedrock · Amazon SageMaker · Cohere · Voyage · and any HTTP-compatible endpoint via a custom blueprint.

PART 2 · GETTING THE DATA IN

That one box hides a lot of work

Source connectors, a staging buffer, and change detection.

SO FAR SO GOOD

Everything lands in OpenSearch, and you search it

Text comes in, it is chunked and embedded, it lands in OpenSearch, and a user can search it with keyword, semantic, or hybrid retrieval. Clean. But two assumptions are hiding in plain sight.

Assumption 1

The documents come from one place and arrive cleanly. In reality they are scattered across many enterprise systems.

Assumption 2

The documents are plain text. In reality they are PDFs, slides, spreadsheets, audio, and video.

ASSUMPTION 1 · SOURCES

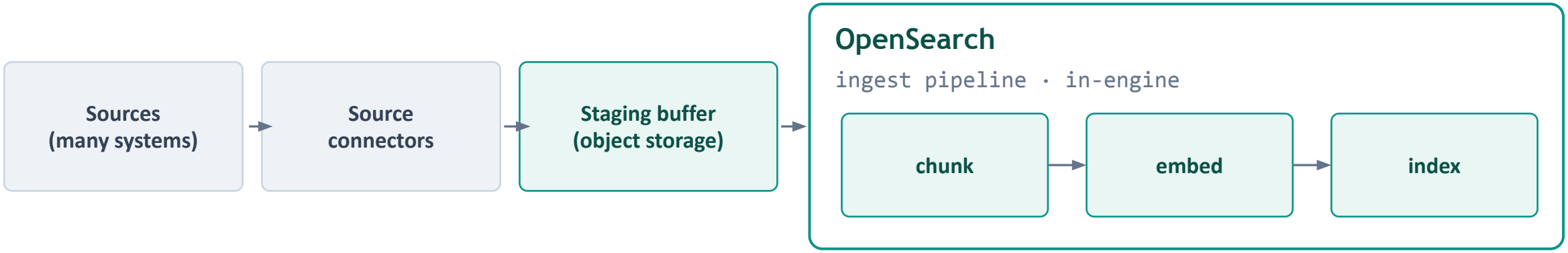
Enterprise data lives everywhere

- Confluence
- Jira
- Google Drive
- SharePoint
- Slack
- Wikis & file shares

Each source has its own API, its own auth, its own permissions model, and its own idea of what 'changed since last time' means. A connector per source pulls both the content and its access-control list.

THE INGESTION PIPELINE

Connectors and a buffer outside, embedding inside



Retry & backoff

Sources fail. Pull is durable and resumable.

Change detection

Delta where the source supports it, full-pull-and-diff where it doesn't. Plus deletes.

Freshness SLA

Measured source-change to searchable. Live updates beat backfill.

Decoupling

The buffer lets fetch and processing fail and scale independently.

PART 3 · THE MULTIMODAL PROBLEM

The documents are not plain text

PDFs, slides, spreadsheets, audio, video.

ASSUMPTION 2 · FORMATS

We just entered the multi-modal plane

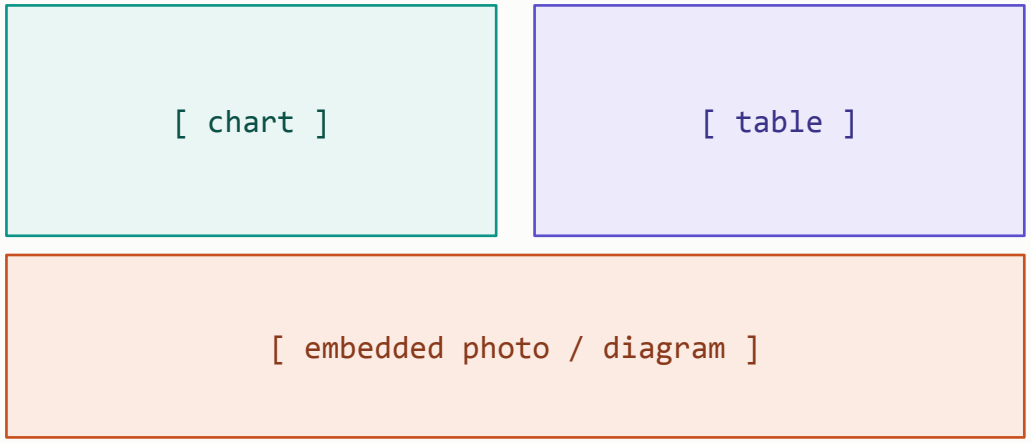
A PDF carries text, tables, charts, and images on one page. A town-hall recording carries words and tone. None of this is a clean string. So how do you get it into OpenSearch and let a user search across all of it?

WHAT WE ARE DEALING WITH

One page, four content types

Annual benefits summary

Paragraph of policy text describing eligibility and coverage in prose...



Each region needs different handling

- Prose text embeds directly.
- A chart's meaning is in its shape and numbers, not its alt text.
- A table is structured data, not a sentence.
- An embedded image may carry the only copy of a key fact.
- Treat the page as one blob and retrieval quietly misses most of it.

THE QUERY THAT BREAKS NAIVE RAG

"Who will be my next manager?"

The CEO announced a reorganization in a town-hall video. The answer exists, but it was spoken, not written. There is no document that says it.

What the words give you

- Transcribe the video and the spoken content becomes searchable text.
- Now 'reorganization' and names are findable.
- This covers most multimodal search needs.

What the words lose

- Tone, emphasis, who sounded uncertain, the mood of the room.
- A query about how something was said cannot be answered from a transcript.
- That gap is the bridge to native embedding later.

TWO WAYS TO HANDLE IT

Flatten to text, or embed natively

Approach 1 · Text-centric

- Convert every modality into text at ingestion.
- Caption images, turn tables into CSV, transcribe audio.
- Embed all of it in one text vector space.
- Cheap, precise on numbers and names, loses nuance.

Approach 2 · Native embedding

- Embed images as images and text as text into a shared space.
- Embed audio in its own acoustic space.
- Keeps layout, visual, and tonal nuance.
- Heavier to store and to query.

APPROACH 1 · FLATTEN TO TEXT

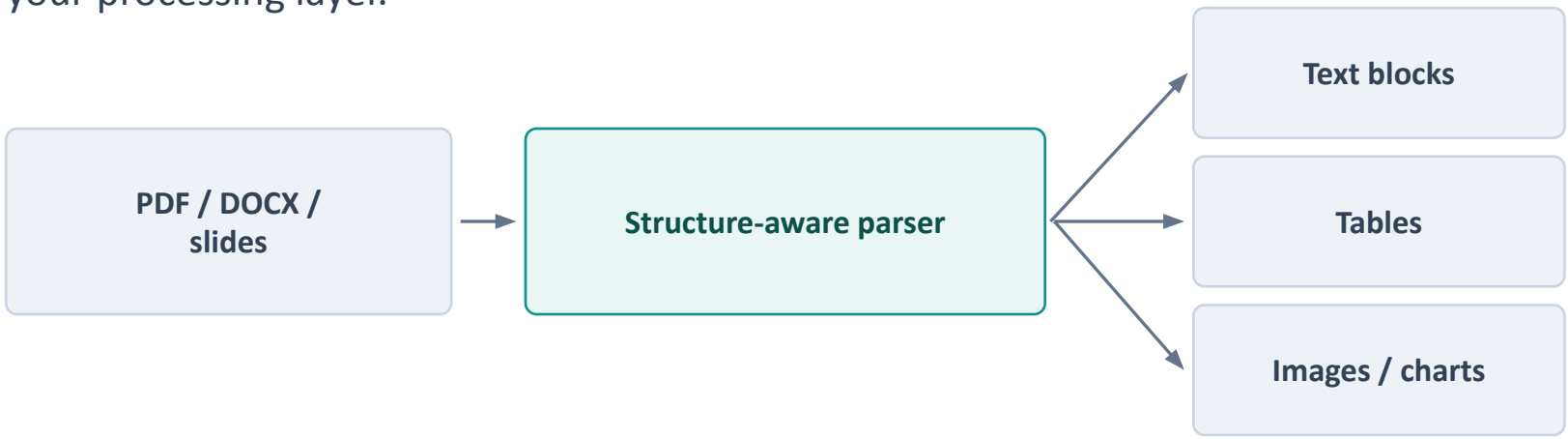
Convert everything into text

Extract, caption, transcribe, then embed one space.

STEP 1 · EXTRACTION

Parse the document into components

Before anything can be embedded, the document is broken into its parts. This parsing step runs outside OpenSearch, in your processing layer.



Tools that do this

- Amazon Textract
- Llama Parse
- Docling, Marker

Parsing fragility is real: a miscalculated column boundary can truncate text or drop an image. Validate it.

STEP 2 · COMPONENTS TO TEXT

Caption, tabulate, transcribe

Images → caption

A vision-language model writes a description of each image or chart.

Tables → CSV / markdown

Structured cells become text the embedder and the LLM can read.

Audio → transcript

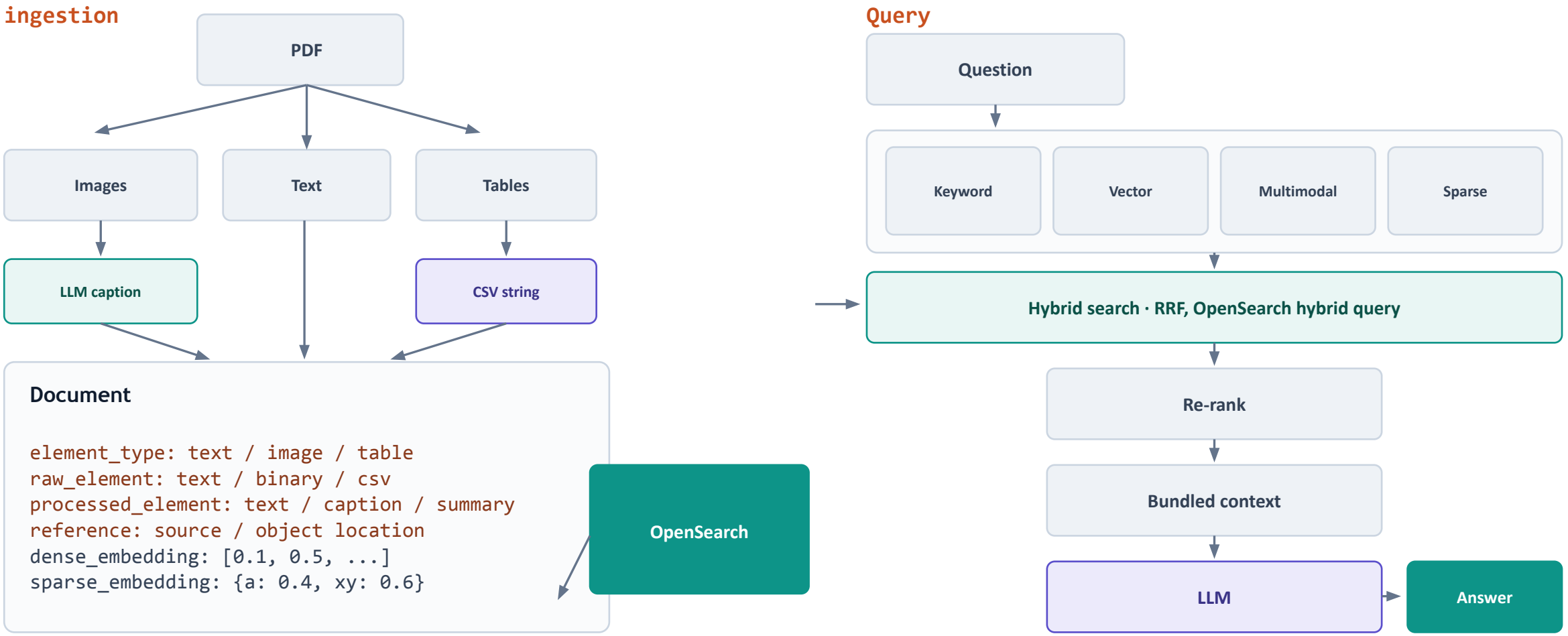
Speech-to-text turns the spoken content into searchable words.

Where OpenSearch fits

Captioning can be orchestrated by the OpenSearch ingest pipeline through the ML connector, but the generation runs on the remote vision model, not on the cluster. Light embedders can run in-cluster; heavy generative models stay remote.

APPROACH 1 · THE FULL FLOW

Multimodal RAG workflow



APPROACH 1 · TRADE-OFFS

Cheap and precise, but flattened

Strengths

- Pinpoint accuracy on numbers and names, they are literal text.
- One cheap text vector space, fast to query.
- Works with standard text embedders and any LLM.
- Simple to reason about and operate.

Costs

- Loses acoustic tone, visual style, and layout.
- Heavy upfront processing: VLMs captioning thousands of images is slow.
- A caption is only as good as the model that wrote it.
- Quality depends entirely on extraction and captioning.

APPROACH 2 · NATIVE EMBEDDING

Embed the modality itself

Shared vector space for text and images, a separate one for audio.

APPROACH 2 · HOW IT WORKS

A shared space for what text cannot say

Native multimodal embedding

- A multimodal model embeds an image as an image and text as text into one shared space.
- A text query can then match an image directly.
- Audio uses a separate acoustic-text space for tone and sound.

What OpenSearch supports natively

- A text_image_embedding processor and multimodal connectors (e.g. Titan, Cohere).
- These produce one combined vector per item.

APPROACH 1 VS APPROACH 2

Text-centric vs Native embedding

	Text-centric (caption to text)	Native embedding (rasterize)
Ingestion speed	Slow, VLMs caption page by page	Fast, encoders process pixels
Accuracy on numbers	Very high, numbers are literal text	Lower, encoders struggle to read tiny fonts
Style & tone matching	Poor, captions miss design and vocal tone	Excellent, finds similar layouts and sounds
Query-time cost	Cheap, tiny text fragments to the LLM	Expensive, large images

THE HONEST PART

When not to lean on OpenSearch's ML

And what you then have to own yourself.

THE DECISION

In-engine when simple, external when in control

Use the in-engine connector when

- You want one self-contained system to operate.
- Vectors are consumed only by OpenSearch.
- Ingestion is small to moderate.
- Query-time and index-time embedding should auto-match.

Step outside when

- Embedding must decouple from indexing for backpressure and SLA.
- Parsing or chunking must be sophisticated.
- Embeddings are shared with other services.
- You need model freedom, large-scale ingestion, or cost control.

OpenSearch's own guidance: ingest pipelines suit simple pre-processing and small datasets; large or complex ingestion belongs in an external pipeline.

THE PRICE OF CONTROL

Step outside, and here is what you own

Embedding service
Scaling, batching, availability.

Backpressure & retries
A slow model must not stall ingestion.

Idempotency & dedup
Replays must not double-write.

Dead-letter path
Poison documents quarantined, not blocking.

Model lockstep
Query-time and index-time models must match, or search silently breaks.

Vector versioning
Own the re-embed and reindex pipeline on a model change.

Freshness SLA
Orchestrate live updates ahead of backfill.

Cost & rate limits
Handle provider throttling and spend.

Security at query time
Tenant and ACL filtering is yours to enforce.

Every one of these was handled for you in-engine. Stepping out is choosing control at the price of all of it.

TAKEAWAYS

Six things to take home

1. Your answer is only as good as your retrieval.
2. OpenSearch is the engine: pipelines and connectors, not just storage.
3. Sources are scattered; a buffer and change detection are the real ingestion.
4. Enterprise data is multimodal; decide how each modality becomes searchable.
5. Text-centric is the precise, cheap default; native embedding for nuance.
6. In-engine ML buys simplicity; stepping out buys control, at a known price.

Generic at the edges. OpenSearch in the core.

Connect anything, buffer anywhere, but let OpenSearch own retrieval.

Flatten to text where you can, embed natively where you must, and know exactly when to step outside.

Thank you. Questions?