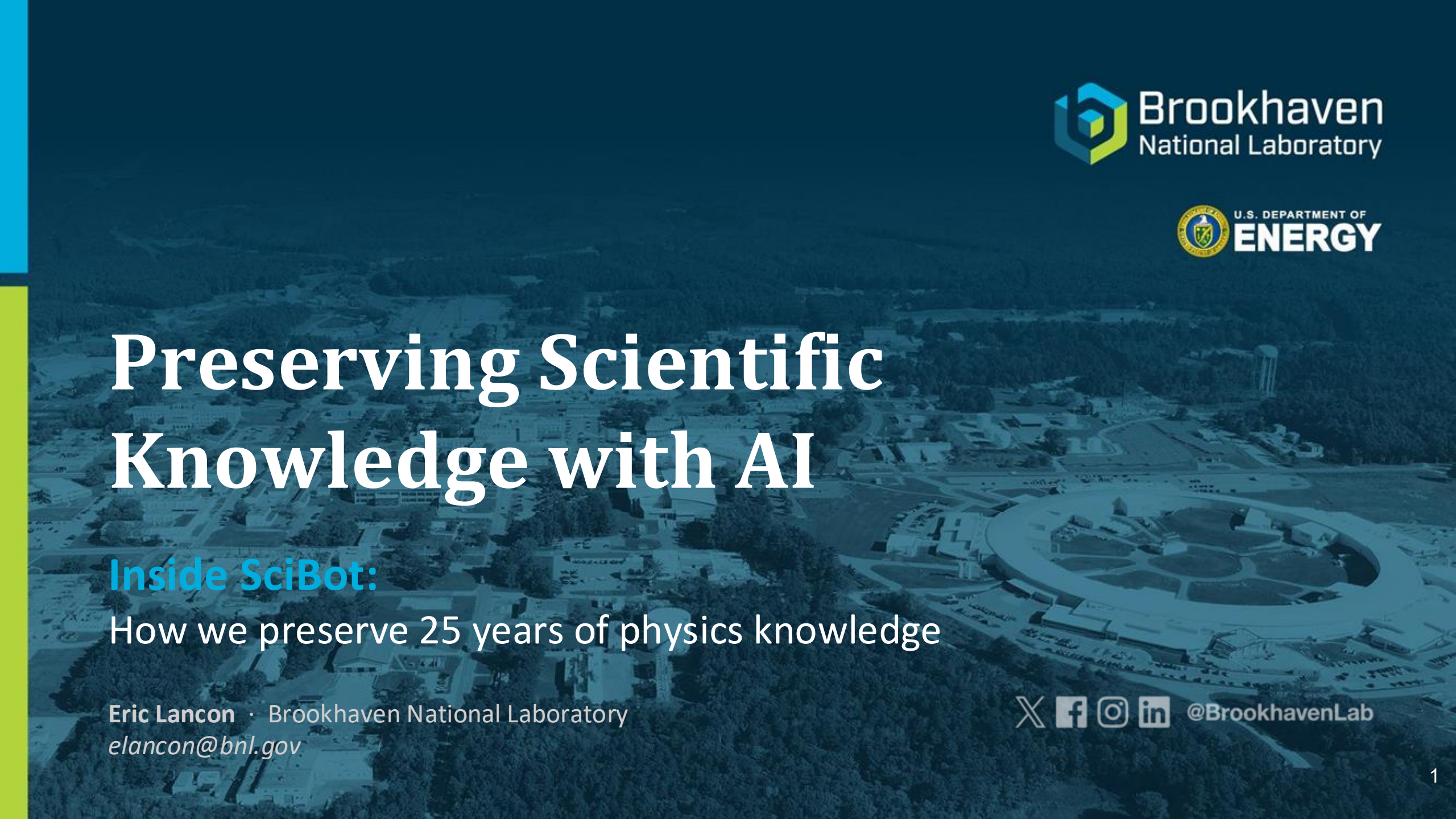


Preserving Scientific Knowledge with AI

Inside SciBot:

How we preserve 25 years of physics knowledge

Eric Lancon · Brookhaven National Laboratory
elancon@bnl.gov

@BrookhavenLab

RHIC

(Relativistic Heavy Ion Collider)

sPHENIX

STAR

LINAC

EBIS

NSRL

BOOSTER

AGS

RHIC stopped running after 25 years. The detectors are switched off, but the knowledge shouldn't disappear with the people.

RHIC's legacy

After 25 years, RHIC operations ended in February 2026. The data will remain, but the expertise won't unless we preserve it now.

~1 ExaByte

of experimental data

25 years of collisions, an irreplaceable legacy.

680+

publications

Plus tens of thousands of notes, slides, and memos.

2030

The knowledge cliff edge

A closing window to capture how it was done, before the experts move on.

The challenge: Data are easy to preserve. Understanding how to use them is much harder.
So a student in 2040 should be able to ask, "How did STAR measure this?" and get a real, sourced answer.

Preserving scientific knowledge

When people think about preserving science, they picture safeguarding the DATA. That matters, but data alone are inert.
To reuse an experiment's data years later, you need more than just the data.

PRESERVING DATA

Bits kept safe, redundant, verified, a locked vault of files.

Necessary, but not sufficient.

+

PRESERVING THE ABILITY TO DO SCIENCE WITH IT


Documentation


Software & workflows


Calibration procedures


Analysis notes & slides


Expert know-how


Years of discussion

SciBot preserves the know-how, the ability to do science

SciBot

SciBot is an **AI assistant** built at Brookhaven to help researchers search and navigate this scientific knowledge. Instead of manually searching through hundreds of documents, users ask a question in plain language.

1 You ask

"What trigger conditions did STAR use for the Run-14 jet analysis?"

2 SciBot searches

It searches thousands of notes, papers, slides and manuals and finds the passages that answer it

3 You get an answer

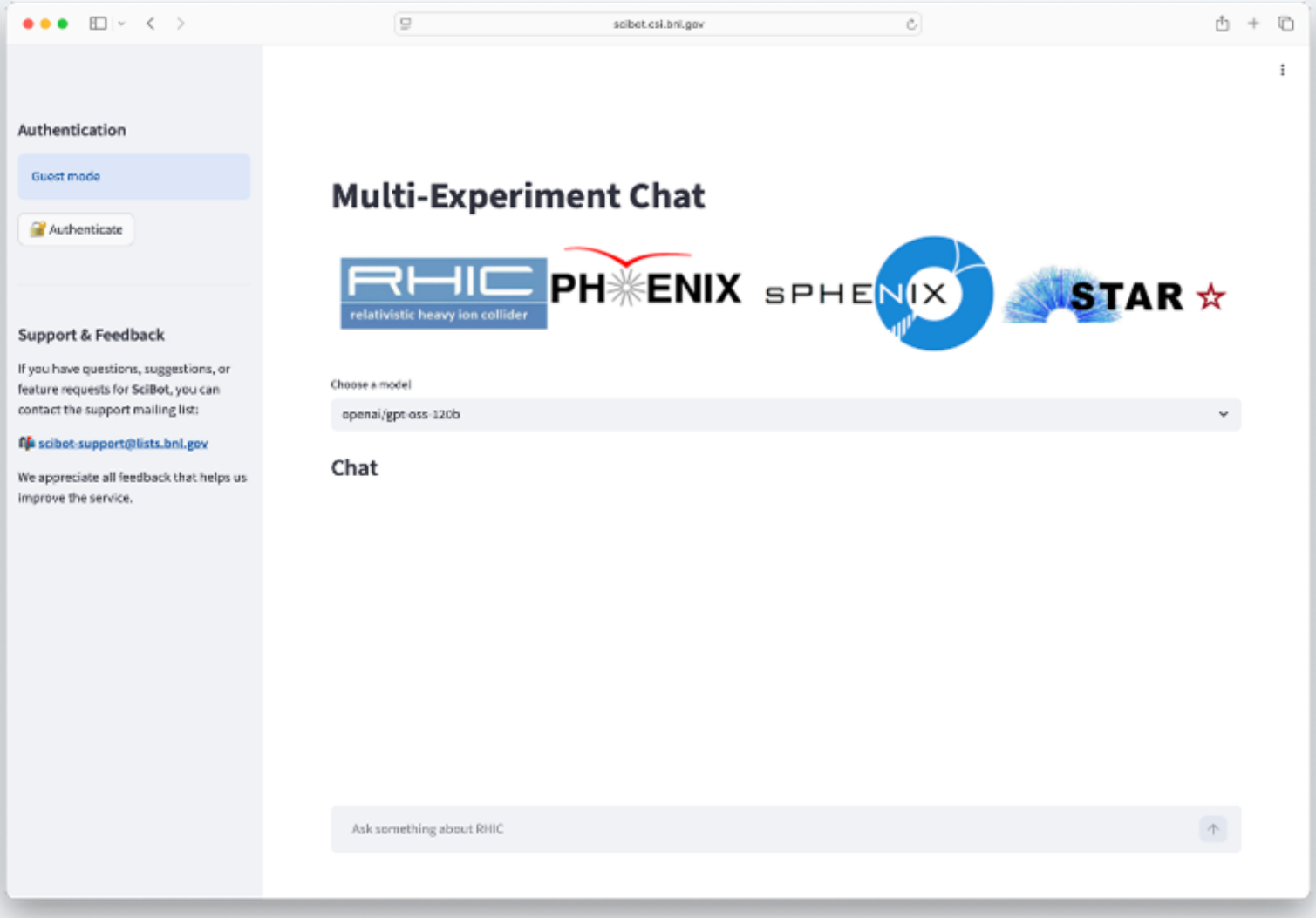
A clear response, with the source documents it drew from, so you can verify

SciBot doesn't analyze RHIC data. It helps people find the knowledge needed to use RHIC data.

Next, let's see how it works inside.


BEFORE
AFTER
AI
TAGG
IS
TAG

Here is SciBot entry



Looking inside the black box

You ask a question, and seconds later, an answer appears.
Inside SciBot there isn't any magic. It's just a sequence of steps.
The rest of this talk walks through them.

1. **Chunk & index** documents are split into pieces and stored
2. **Turn text into numbers** each piece becomes coordinates in a vector space
3. **Search two ways** by meaning AND by exact term, then merged
4. **Generate the answer** a language model writes it, grounded and cited

Everything we've discussed so far explains why we built SciBot. The rest of the talk explains how it works.

Why not just use ChatGPT?

Three issues.

Can't see our documents

Most RHIC documentation is internal or restricted, including analysis notes, chats, and theses. A public model was never trained on it and cannot be shown it.

No access.

Retrieved text would leak

Even for a public model, the passages we'd include in its prompt contain information it has no right to access. Sending those documents to a commercial API would expose internal information. Our model runs 100% locally.

Can't send the text out.

Hallucinates when it lacks reliable information

Asked a detailed physics question with no source, a general model invents confident, plausible-sounding answers, which is dangerous for science.

Not trustworthy.

Our solution: Keep a standard LLM for writing the answers, but feed it ONLY our own retrieved documents, locally, with access control.

The basic idea behind SciBot

Without retrieval: The LLM knows what it has learnt during training
With RAG: The LLM first reads YOUR documents

RAG: Retrieval Augmented Generation

1 Retrieve — Find

Find the few document passages most relevant to the question.

2 Augment — Read

Put those passages into the LLM prompt.

3 Generate — Answer

The LLM writes an answer grounded in those passages, and can cite them.

Example: "What trigger conditions did STAR use for the Run-14 jet analysis?" → RAG retrieves the analysis note, and the answer cites it.
A bare LLM would likely hallucinate plausible conditions or, for internal notes, have no access to the document.

Computer scientists call this Retrieval-Augmented Generation, or RAG.

What RAG does: one combined prompt

We don't retrain LLMs. We use a STANDARD LLM and give it a prompt that combines the retrieved chunks with the user's question, then ask it to answer using only that prompt.

THE QUESTION

"What trigger did STAR use for the Run-14 jet analysis?"

RETRIEVED CHUNKS

[1] …BEMC trigger, E_t > 5.4 GeV…

[2] …|z| < 30 cm vertex cut…

[3] …anti-k_t, R = 0.4 jets…

+

THE PROMPT WE SEND

"Using ONLY the context below, answer the question.

CONTEXT:

[1] BEMC trigger…
[2] vertex cut…
[3] anti-k_t jets…

QUESTION:

What trigger did STAR use…?

Cite your sources."

→

STANDARD LLM

(GPT, Claude, Llama, Gemini, …) off-the-shelf, nothing retrained

↓

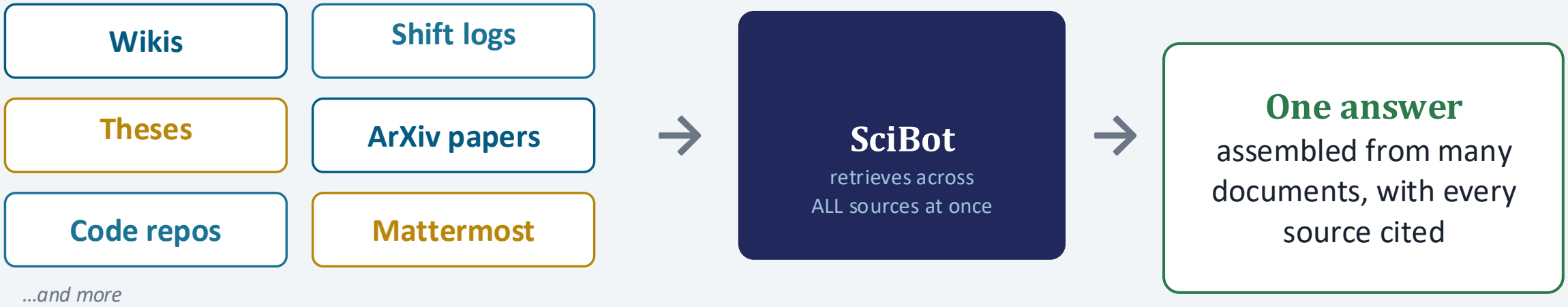
GROUNDED ANSWER

"STAR required a BEMC trigger tower with E_t > 5.4 GeV and |z| < 30 cm [1][2]."

"Retrieval-Augmented" simply means we AUGMENT the prompt with retrieved text before the LLM sees it. The model answers using the retrieved text.

One question, many scattered sources

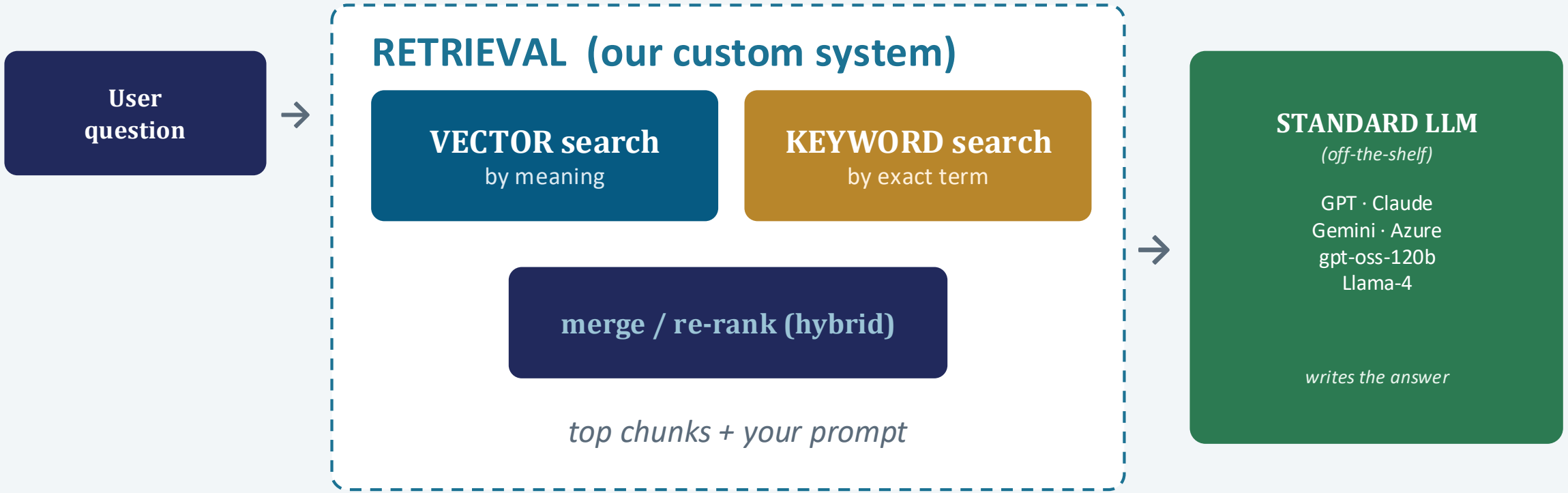
RHIC knowledge isn't in one place. It's spread across papers, notes, chats, webpages, and code repositories: ~22,000 webpages, ~900 arXiv papers, ~800 Zenodo PDFs, ~800 analysis notes, ~300 theses, and 120+ Mattermost channels. SciBot searches across all of them at once.



And it crosses experiments. The same question can draw on different RHIC experiments, e.g. "how has calorimeter calibration evolved across RHIC detectors?" pulls together information that no single collaboration archive contains.

How SciBot is organized

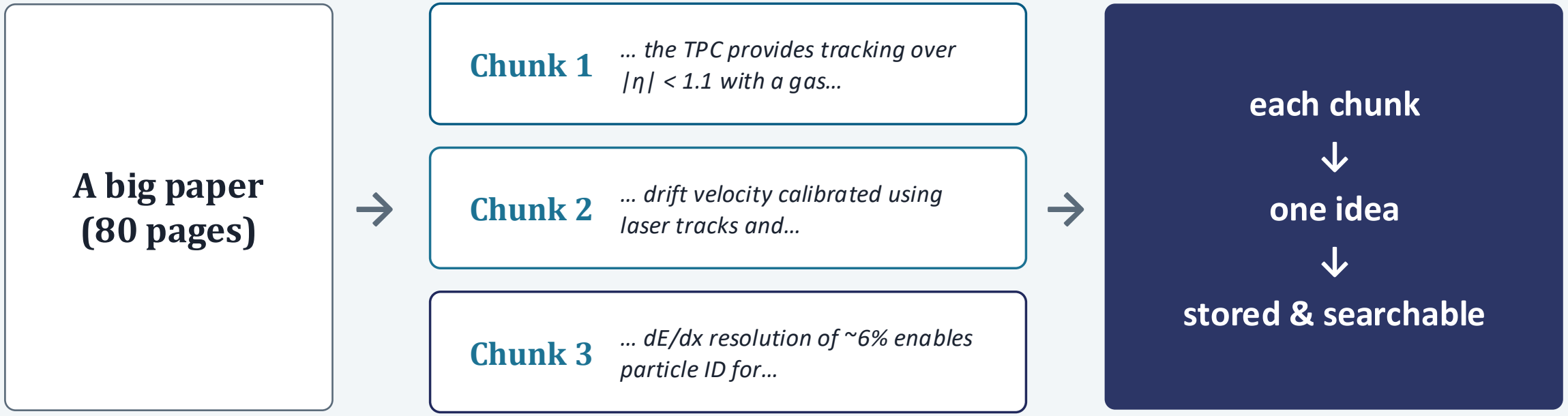
There are two parts: retrieval and the language model. Most of the engineering is in **retrieval**.



We can replace the LLM without rebuilding SciBot. We can swap GPT for Claude without changing the retrieval part.

Chunking: cutting documents into searchable pieces

Most questions are answered by one or two paragraphs, not an entire paper. Instead of treating an 80-page document as one object, we split it into overlapping chunks of about 1000 characters. Ideally each chunk contains one idea and becomes an independent searchable unit.



Why these sizes? Large enough to capture one idea, small enough for efficient retrieval, and overlapping so nothing important is split at a boundary. Next: the computer must turn each chunk into something it can compare mathematically.

From chunks to tokens

The tokenizer splits each chunk into tokens, the pieces that are **input to embedding model.**
Many common words stay whole; rare technical terms get split into several pieces.

One chunk

“The calorimeter measured vpdtiming for run 54280.”

SentencePiece tokenizer

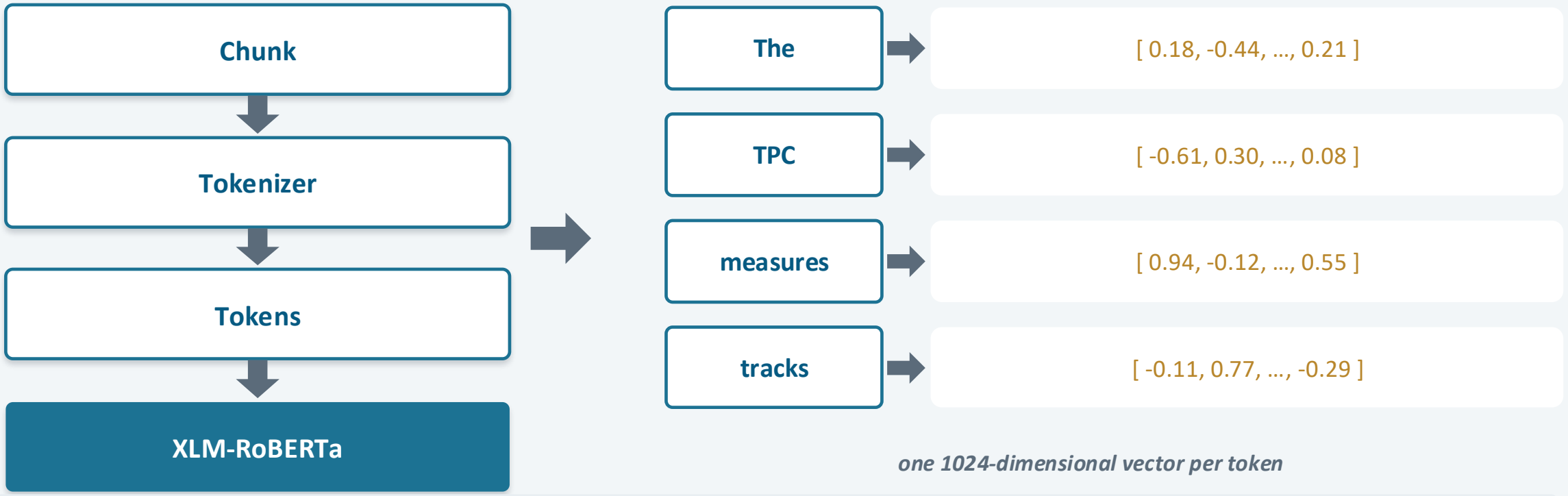
_The	_calor	i	meter	_measured	_v	p	d	tim	ing	_run	_54	280

Common words often stay whole · technical term split into pieces · number split too

A 1000-character chunk becomes about 170 to 420 tokens, comfortably below our model’s 512-token limit. Now the model can read the chunk, but it still doesn’t understand it.

The embedding model

The model processes all tokens and produces one 1024-dimensional vector per token.



These are contextual vectors: the same word in a different sentence produces a different vector.

https://huggingface.co/docs/transformers/model_doc/xlm-roberta?utm_source=chatgpt.com

How did it learn these vectors?

The model learns by repeatedly predicting missing words from context. Words appearing in similar contexts gradually develop similar internal representations.

Words used in similar contexts get similar representations.

1 Hide a word
"The ____ leaves a track in the TPC." Blank out one word.

2 Guess it
The model predicts the missing word from its neighbours.

3 Adjust & repeat
Wrong guess → adjust the numbers slightly. Repeat billions of times.

The ____ leaves a track in the TPC.

electron ✓*fits the context*

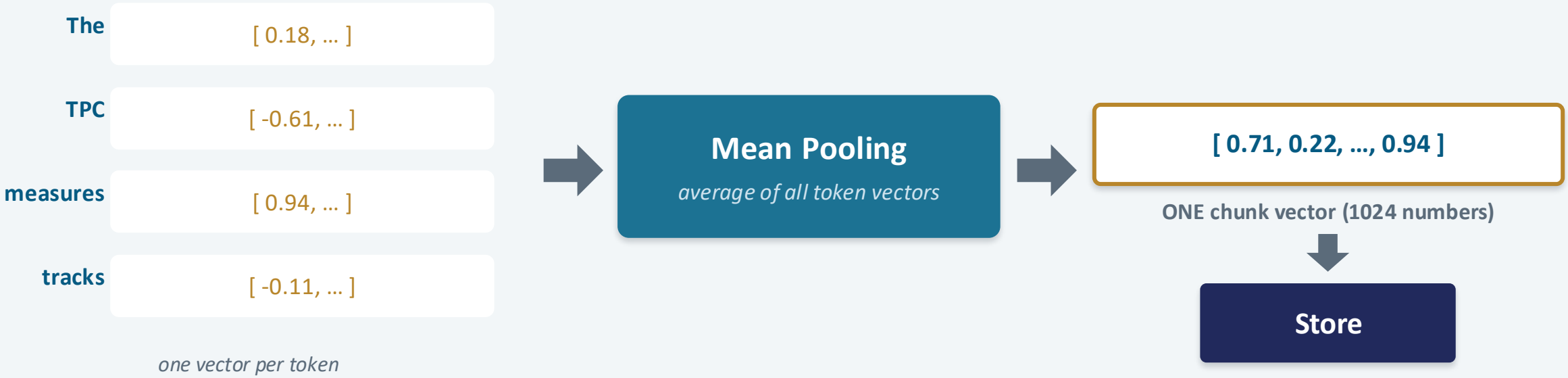
muon *also plausible*

banana ✗*never appears here*

Nobody programs the concept of "electron" into the model. **It learns context, not definitions**.

One chunk → one embedding

Each token has its own contextual vector. To search whole paragraphs we combine all token vectors into one vector for the chunk.



This single vector is what we store and search. Because every token vector already carries information about the whole chunk, averaging them gives a surprisingly good representation of the entire chunk.

Meaning becomes geometry

You can’t interpret any single number by itself. Meaning comes from the position of the entire vector: chunks about similar ideas end up close together.

Two chunks about the same idea:

“The TPC measures charged particle tracks”

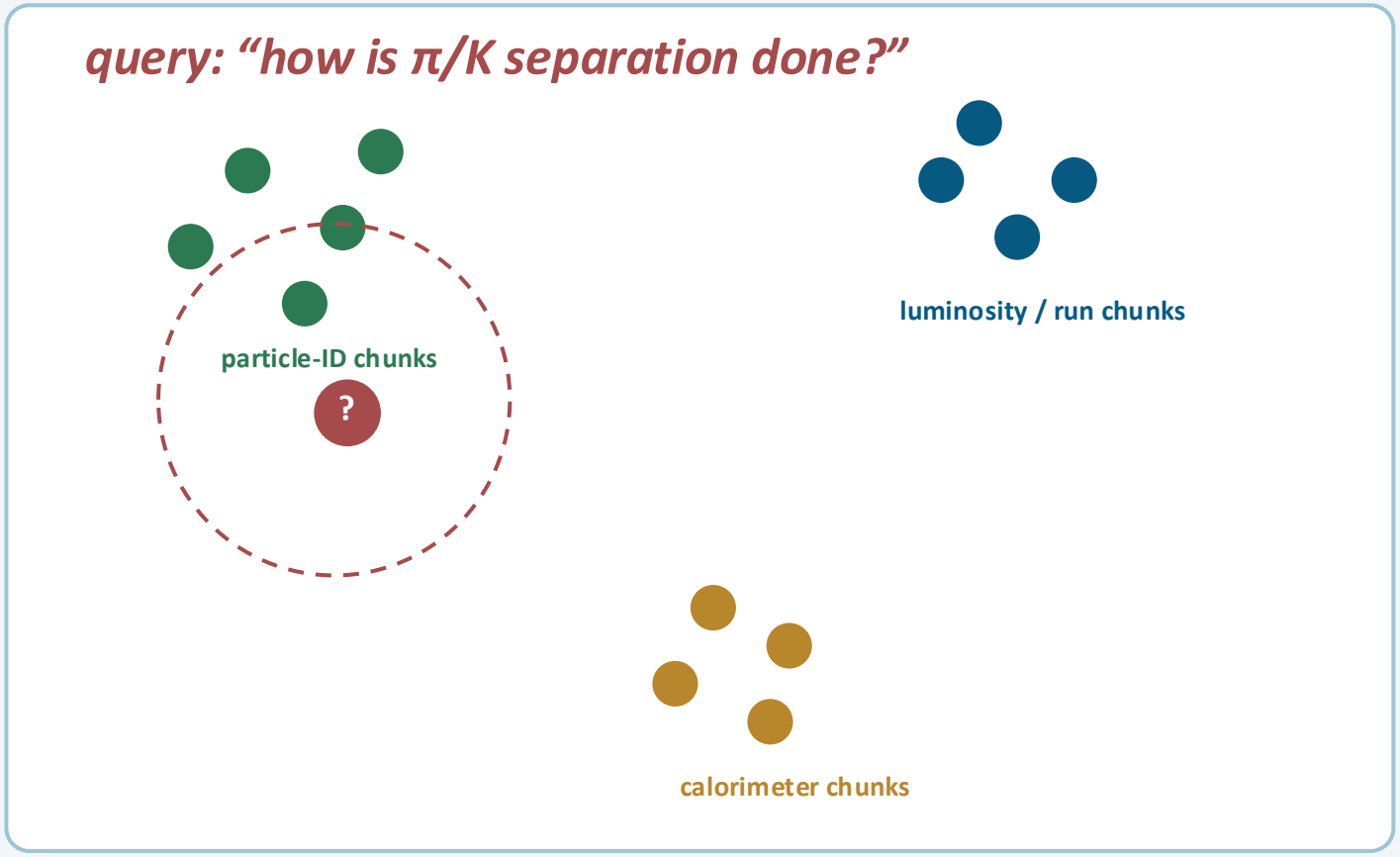
“Tracking detector records particle paths”

end up close together

So searching by meaning becomes searching for nearby points, even when the wording is completely different.

The vector database: a library organized by meaning

Every stored chunk has one embedding.
When you ask a question, the same embedding model turns it into another vector. The database finds the stored vectors nearest to it.



Why a vector database?

- Stores every chunk-vector + its source text
- Organizes similar vectors so nearby ones can be found quickly
- Given a query vector, returns the top-k closest chunks, no full scan of millions of points

This is semantic search: it finds related ideas even when the wording is completely different.

Semantic search: nearest neighbours win

The question is embedded and compared with all stored embeddings. Close vectors indicate similar ideas. The nearest chunks are passed to the language model.

Why it beats letter-matching

Question: *"How do you tell pions from kaons?"*

It retrieves chunks about **dE/dx, time-of-flight, Cherenkov ring radius**, even though NONE of those words appear in the question. That's matching **meaning, not spelling.**

It also generalizes

Because it matches meaning rather than spelling, a question phrased in everyday language can retrieve a chunk written in dense technical language, as long as both describe the same physics.

"how precisely can it measure energy?" sits right beside *"calorimeter energy resolution."*

Semantic search excels at finding ideas. But physicists also search for exact run numbers, detector names, and variable names, and that is where it struggles.

Why keyword search is still needed

Semantic search captures meaning, but not exact identifiers, and physics documents are full of them.

Exact codes & run numbers

Ask for run “54280,” and you may get runs that merely look similar. “Close in meaning” is useless when you need that exact one.

Unusual words

“vpdtiming” and “eic-shell” were chopped into sub-word fragments and appeared rarely during training, so their embeddings are less reliable.

Names & acronyms

A collaborator's name or a one-off label can sit near unrelated text just because neighboring words felt similar; a near-miss is still wrong.

The fix: keyword search (full-text search)

Some questions ask for an exact string, not a concept. Keyword search matches the literal term.

1	**Tokenize**	Split text into words.	`"The VPD timing for run 54280"`
2	**Drop stop-words**	Remove "the," "for" (no search value).	`VPD · timing · run · 54280`
3	**Normalize**	Lowercase, reduce to root forms.	`vpd · timing · run · 54280`
4	**Build the index**	Record which words each chunk contains, so a query jumps straight to matching chunks.	`run 54280 → chunk #257, #903 …`

Keyword search uses the same chunks but a different index. So each chunk has two independent indexes pointing to it: one by meaning (vector) and one by exact term (keyword).

Hybrid search: merge the two rankings

Semantic search is good with concepts. Keyword search is good with exact terms. We run both and merge the results: we keep the strongest results from each search and drop a chunk only if BOTH agree it's irrelevant. Each compensates for the other's weakness.

Why we merge: If either search finds a good chunk, we keep it, so we merge the two rankings (the recipe is called Reciprocal Rank Fusion). Chunks both searches liked rise to the top. So "how do you separate pions and kaons?" still works, ***and*** exact term "vpdtiming" is never lost.

Now we've found the right documents. The last step is to get the language model to use them correctly.

Giving instructions to the LLM

Retrieval finds the right chunks; the prompt determines what the model DOES with them. These are the instructions we give the model.

1	**Give a role**	*"You are an expert on the RHIC experiments, STAR, sPHENIX, PHENIX. Refuse other topics."*	Sets scope, tone, and audience.
2	**Ground in context**	*"Answer based ONLY on the provided search results and chat history."*	Forces answers to come from retrieved chunks.
3	**Say when unsure**	*"If there's no relevant info in the context, just say 'Hmm, I'm not sure.' Don't make up an answer."*	If the documents don't contain the answer, the model should say so. The key guard against hallucination.
4	**Demand citations**	*"Cite sources as [1], [2]… end with a numbered References list. Do not invent references."*	Makes answers checkable against real documents.
5	**Constrain format**	*"500 words max, bullet points, bold key terms."*	Keeps replies precise and readable.

The whole pipeline, end to end

OFFLINE (done once per document)

Chunk
split docs into 1000-char pieces

→

Embed
each chunk → 1024-d vector

→

Index
build TWO indexes: vector + keyword

LIVE (every question)

Embed Q
question → vector

→

Search
vector + keyword, merged

→

Build prompt
chunks + rules

→

Generate
a STANDARD LLM writes the answer

In one sentence: We chunk the documents, build two indexes over them (meaning and keyword), then for every question search both indexes, merge the results, and hand the best chunks to a standard LLM that writes the cited answer. That's RAG.

Most of the engineering is in preparing the right information before the model even sees the question.

What SciBot does — and doesn't do

✓ It DOES

- Finds the right documents
- Summarizes them in plain language
- Cites its sources so you can check
- Preserves expertise for the future

✗ It does NOT

- Analyze detector data
- Discover new physics
- Replace scientists
- Guarantee every answer is correct

"Hmm, I'm not sure." When the documents do not answer your question, SciBot says so.

For a scientific assistant, admitting uncertainty is a strength, not a weakness.

In summary

1

The problem
25 years of expertise are scattered across documents, and a knowledge cliff appears as the experts retire.

2

Retrieval
Two complementary searches, by meaning and by exact term, find the right passages and merge them.

3

A grounded answer
A standard LLM writes the answer from those passages, with citations, and says “I'm not sure” when nothing fits.

The goal isn't just to answer questions. *It's to keep the ability to do science with RHIC's data.*

RHIC has stopped collecting data.

Its scientific knowledge should remain accessible for **generations of future researchers**

AI isn't replacing scientists. Here, it helps preserve their knowledge so future scientists can keep building on it.

Brookhaven National Laboratory