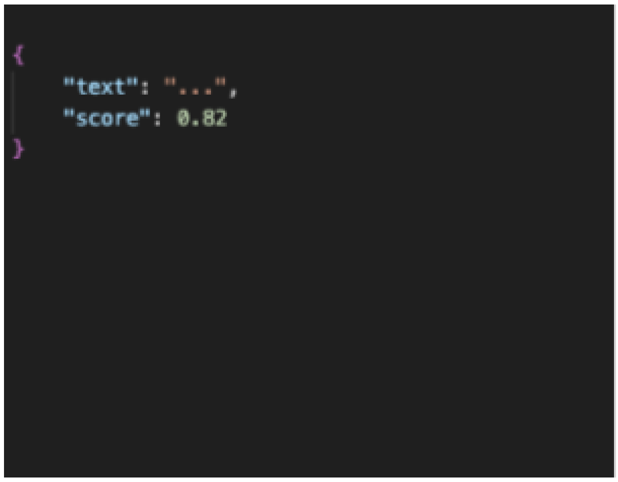
Caber DREAM

Caber DREAM[1] dynamically enriches every chunk in your Retrieval-Augmented Generation (RAG) datastore with **trusted source attributed metadata** so you always know **what** the data is, **where** it came from, **who** owns it, and **how** it may be used.

By eliminating blind spots in Data lineage, RBAC data authority, and chunk context, Caber DREAM dynamically optimizes the LLM's ability to interpret each RAG chunk intelligently, and improve Answer quality + precision. Enabling developers to build high accuracy context aware Agentic applications that are **explainable, policy-aware, and trustworthy.**.

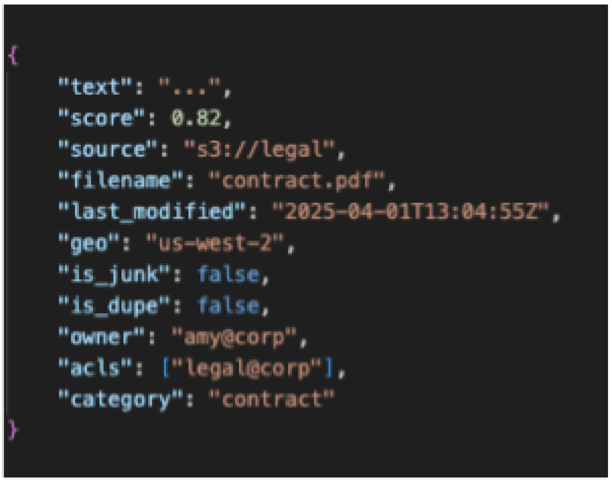
What Caber DREAM Delivers

Retrieved RAG chunk before



```
{
    "text": "...",
    "score": 0.82
}
```

Chunk augmented by Caber DREAM



```
{
    "text": "...",
    "score": 0.82,
    "source": "s3://legal",
    "filename": "contract.pdf",
    "last_modified": "2025-04-01T13:04:55Z",
    "geo": "us-west-2",
    "is_junk": false,
    "is_dupe": false,
    "owner": "amy@corp",
    "acls": ["legal@corp"],
    "category": "contract"
}
```

Transparent Data lineage, usage policy, risk flags, multiple scores (Rel, Faith, Acc), RBAC authority, and other metadata - added transparently in-flight with < 1 ms overhead.

The Problem With RAG Today

All RAG systems suffer from a core issue: RAG pipelines ignore & discard highly valuable metadata on ETL ingest, excluding its intelligence from the vectorized embeddings corpus. Retrieved datastore chunks lack temporal context precision, to determine their appropriateness for inclusion into the answer for each User or Application question. - Post-ingest Metadata retrofitting is a problematic kludgy

[1] Dynamic RAG Enrichment with Attributes and Metadata

temporary hack that fails to account for future needs. The Metadata is immediately stale upon creation with no ability for live dynamic on-demand updates.

Since ingest resource costs (i.e. compute, storage, vectorization, embedding generation etc) are expensive; this limits how often RAG refreshes/updates can be executed. Contributing to a constant "stale state" for the RAG vectorDB dataset.

The result is; 90% of all RAG queries suffer from low accuracy and relevance; with poor MRR, MAP and @K scores (recall/precision). - Stimulating indeterministic hallucinations and confusion. Without Metadata enforced "ground-truth" of what chunk data truly represents and with no viable hierarchy of chunk-to-document RBAC relevancy, tracing the root-cause of poor answer quality and performance is nearly impossible (i.e. mostly guess work).

Why RAG Answer quality, precision and fidelity are so low

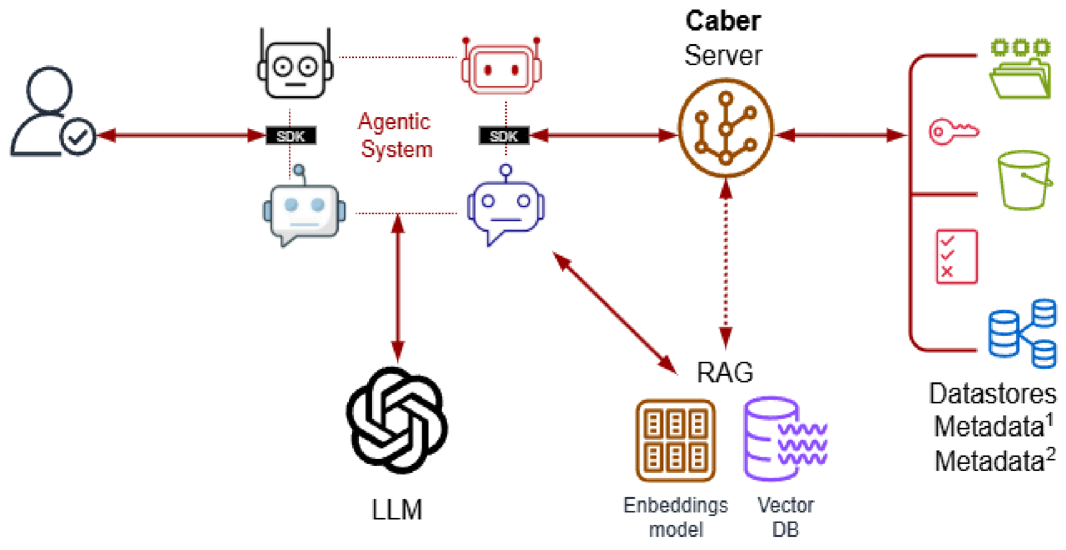
Stale metadata – updates to the original file do not propagate to datastore chunks.

Copy/paste across documents – a single chunk may be duplicated across many documents; with indeterministic complex. Authority becomes uncertain.

Missing metadata – key attributes often live in separate silos or never existed at all.

Junk & duplicate chunks – low-value generic duplicate data pollutes embeddings and increases hallucination and confusion rates.

Architecture



Caber DREAM consists of a client side SDK and server side container stack.

Server Side

Component	Responsibility
Indexer	Encodes content-defined Micro-fingerprints, deduplicates, computes Data lineage graph.
Metadata Enricher	Pulls attributes from source systems (SQL / REST) and resolves authoritative values via *precedence* or *prevalence*.
Graph API	Serves enriched chunks and lineage to SDK queries at high performance + low latency.
Dashboard Portal	One-click deployment, monitoring and policy configuration.

Client / SDK

The Python SDK instruments your retrieval calls, transparently enriches results and hands them back to LangChain, LlamaIndex, and other frameworks.

Deployment options

Automated AWS Launch – From the Caber portal pick a region, click *Deploy*, and spins up the container stack in your VPC with least-privilege IAM roles you choose. Removal is just as simple.

Docker Compose – Grab the docker-compose.yaml, adjust environment variables, and run 'docker compose up' to launch into your environment.

How It Works (End-to-End)

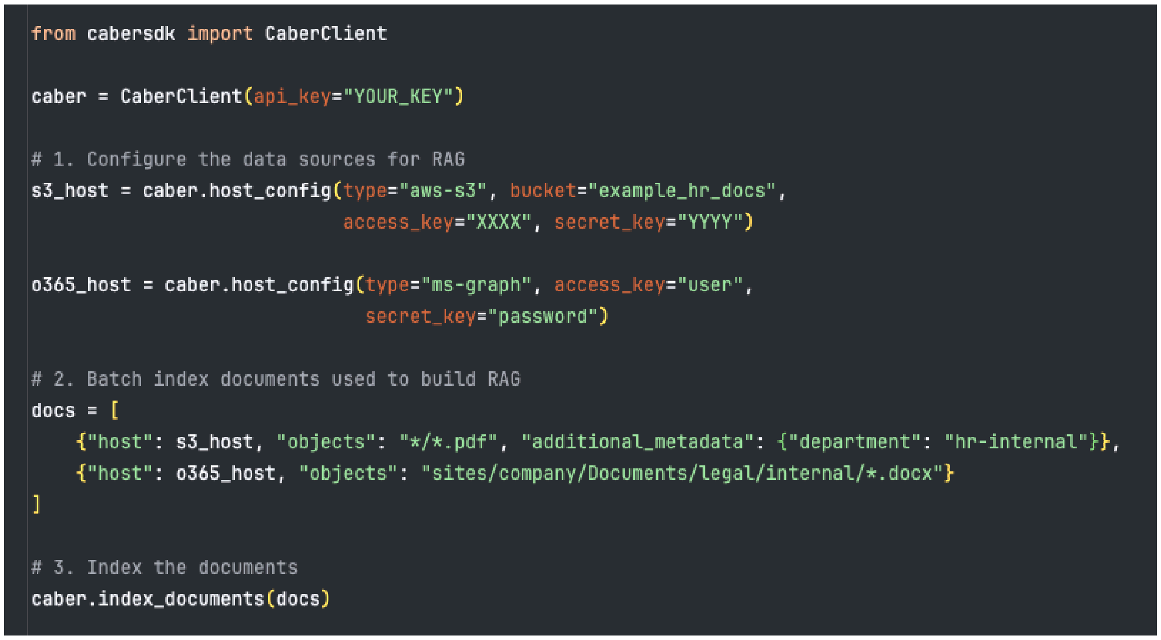
Dataset connectors	Tell Caber where your source Data set corpus and metadata lives	
Pipeline Ingest	2 pipeline modes (more coming…)	
	In-line ETL ingest	Add one line to call `caber.update_object(doc_name)` in the pipeline where you pre-process and ETL ingest

		data into your Embeddings model and vector database.
	Post ETL digest	Link Caber to your ETL pipeline datastore access logs (in any format), optionally provide filter criteria, and Caber will post-process batch index all data entities.
Fingerprinting	Caber's Micro-fingerprinting engine encodes each data payload and computes Graph relationships. Can happen anywhere in the ETL pipeline network. For high resolution Data Lineage, more fingerprinting localities = greater Data lineage resolution.	
Enrich	The Caber intelligence engine deterministically computes authoritative metadata structure & Tags from our Intelligence Graph. It identifies junk & duplicate chunk flags, and calculates scores.	
Query & Retrieval	Your LLM receives High fidelity metadata enriched chunks for CoT and reasoning phase computation to compute answers - explainability included.	

Feature Matrix

Feature	**Caber DREAM v1.0**	**Roadmap**
Source-level lineage graph	✅	shipping
Precedence & prevalence duplicate resolution	✅	Policy resolution
Chunklet viability scoring	✅	Adaptive thresholds
External metadata query (SQL / API)	✅	Auto-sync on change
Real-time datastore endpoint monitoring	in dev	—
Usage analytics dashboard	in dev	—
Multi-agent API data tracing	in dev	—
Extra SaaS Apps Data connectors	in dev	—
Extra Database Data connectors	in dev	—
Extra File store Data Connectors	in dev	—
Extra Metadata connectors	in dev	—
In-line Dynamic Data Classification	dev complete	Beta testing

QuickStart

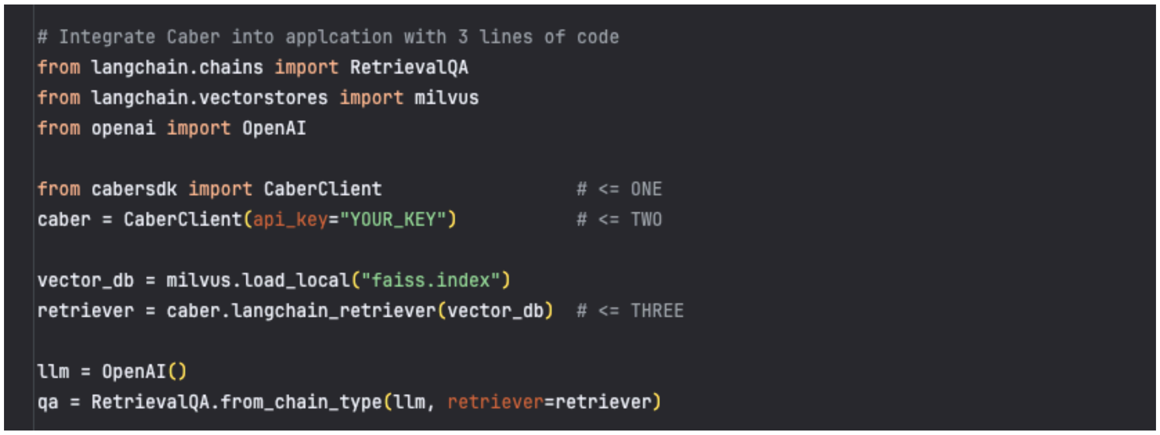


```python
from cabersdk import CaberClient

caber = CaberClient(api_key="YOUR_KEY")

# 1. Configure the data sources for RAG
s3_host = caber.host_config(type="aws-s3", bucket="example_hr_docs",
                            access_key="XXXX", secret_key="YYYY")

o365_host = caber.host_config(type="ms-graph", access_key="user",
                              secret_key="password")

# 2. Batch index documents used to build RAG
docs = [
    {"host": s3_host, "objects": "*/*.pdf", "additional_metadata": {"department": "hr-internal"}},
    {"host": o365_host, "objects": "sites/company/Documents/legal/internal/*.docx"}
]

# 3. Index the documents
caber.index_documents(docs)
```

Server-side ingest

Source Connectors		
AWS S3 (and compatible)	✅	—
Azure Blob	✅	—
Google Cloud Storage	✅	—
MS Sharepoint (CIFS/SQL Svr)	✅	—
Confluence	✅	—
Dropbox	✅	—

Client-side query retrieval enrichment



```
# Integrate Caber into applcation with 3 lines of code
from langchain.chains import RetrievalQA
from langchain.vectorstores import milvus
from openai import OpenAI

from cabersdk import CaberClient                  # <= ONE
caber = CaberClient(api_key="YOUR_KEY")           # <= TWO

vector_db = milvus.load_local("faiss.index")
retriever = caber.langchain_retriever(vector_db)  # <= THREE

llm = OpenAI()
qa = RetrievalQA.from_chain_type(llm, retriever=retriever)
```

v1.0 Specifications & guidelines

- Variable-length content-defined Micro-fingerprinting - independent and agnostic to all RAG chunking strategies.
- Dynamic Text and Intra document Content Classification extraction & hierarchy encoding into structured Metadata schema output.
- Text chunking respects multi-byte UTF-8 character boundaries
- Exact-match fingerprints; images, video are not recognized across resize and format conversions (e.g., PNG→JPG) in this release.
- Dataset viability scoring (Junk detection) requires time to build intelligence. It activates only after dataset size approximates a normal frequency distribution
- Selectable min/max Fingerprint chunk-size down to 64 bytes - Once set, the parameters are immutable until a full data corpus re-ingest overwrite occurs.
- Intelligent Micro-fingerprint Knowledge Graph (powered by neo4j) that traces deep relationships between all payload entities that Caber intercepts, collects and fingerprints as data moves through the network.
- Powerful in-line Data + Metadata Content Classification and Detection engine with 500+ detectors and analysis parsers.

Deployment Details

Automated AWS Launch

1. Sign into(https://caber.com)

2. Create Cross-Account IAM role for Caber via automated workflow

2. Click **Deploy to AWS**, choose region and IAM principal, and confirm.

3. CodeBuild/Terraform spins up the Caber stack (ECS/EC2 or ECS/Fargate).

4. Copy the generated `CABER_API_KEY` for input into Caber SDK.

Docker Compose

```bash
git clone < repo >
cd caber-rme-deploy
cp .env.sample .env                                        # edit config variables
docker compose up -d
```