Semantic RAG: How to train AI to *Understand* not just *Generate*

Where Classic RAG Fails
Limitation of Language Models: no world model
Our Hypothesis
Our Approach: Semantic RAG
Domain Example: Legal Compliance
- What We Use to Build the Knowledge Base
  - Data
  - Data Model
  - Tools: How the Assistant Retrieves Data
  - Playbook: how the assistant reasons
- How the assistant chooses tools
- How It Works in Practice
  - User question
  - Step 1 – Finding candidate CN codes
  - Step 2 – Structured query by CN code
  - Step 3 – Adding practical context from FAQ
  - Step 4 – Building the final answer
- What Matters Here

Technique: Architecture in Practice
- Graph vs. Relational Databases (Cypher vs. SQL)
  - Choosing Between Neo4j and Memgraph
  - Memory and Performance Considerations
  - Limitations and Trade-offs
- Structured Manual Data
- Vector Store
- Overall System Workflow

How to check assistant answer quality
Value of the approach
Conclusion

One of the most common tasks for an AI assistant is to answer user questions. People expect it to find information across internal wikis, support knowledge bases, Word documents, Excel sheets, and other corporate data sources.

Today, this capability is usually implemented through the **Retrieval-Augmented Generation (RAG)** approach. The idea is simple: first, the assistant retrieves document fragments relevant to the query; then, it generates a coherent answer based on them.

At first glance, the setup seems perfectly reasonable. But in practice, **classic RAG** reveals a number of limitations that become obvious in real-world use.
In this article, we'll go through the key issues and show how they can be solved.

Where classic RAG fails

Over the past few years, we've delivered AI projects for e-commerce, legal, and medical companies. Along the way, we kept running into the same weak spot of the "classic RAG" setup:

vector search → take top-N fragments → insert into the LLM context.

This pipeline relies solely on text similarity and ignores the actual data structure.

1. Queries like *"How many?"* and *"Show all"*

Typical e-commerce examples:

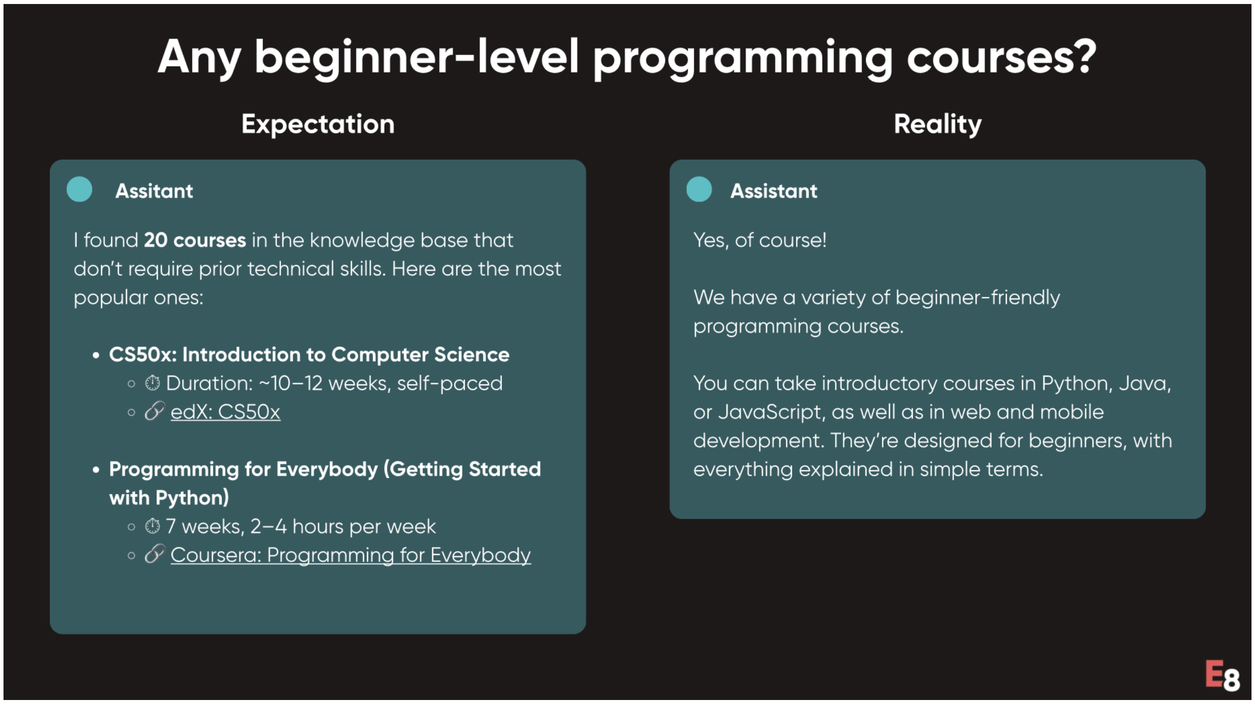
- "Show all bathroom lights"
- "How many creams do you have for sensitive skin?"
- "Are there any beginner programming courses?"

In a catalog, these are trivial – you'd just filter by fields such as light_type = bathroom, skin_type = sensitive, or course_level = beginner.

But a classic RAG assistant typically:

- grabs only the top N fragments (top-5 or top-10),
- returns a short snippet or part of a list,
- and almost never counts the total number of items correctly.

Result: the user expects a complete list or an exact number – but instead gets only a small sample.



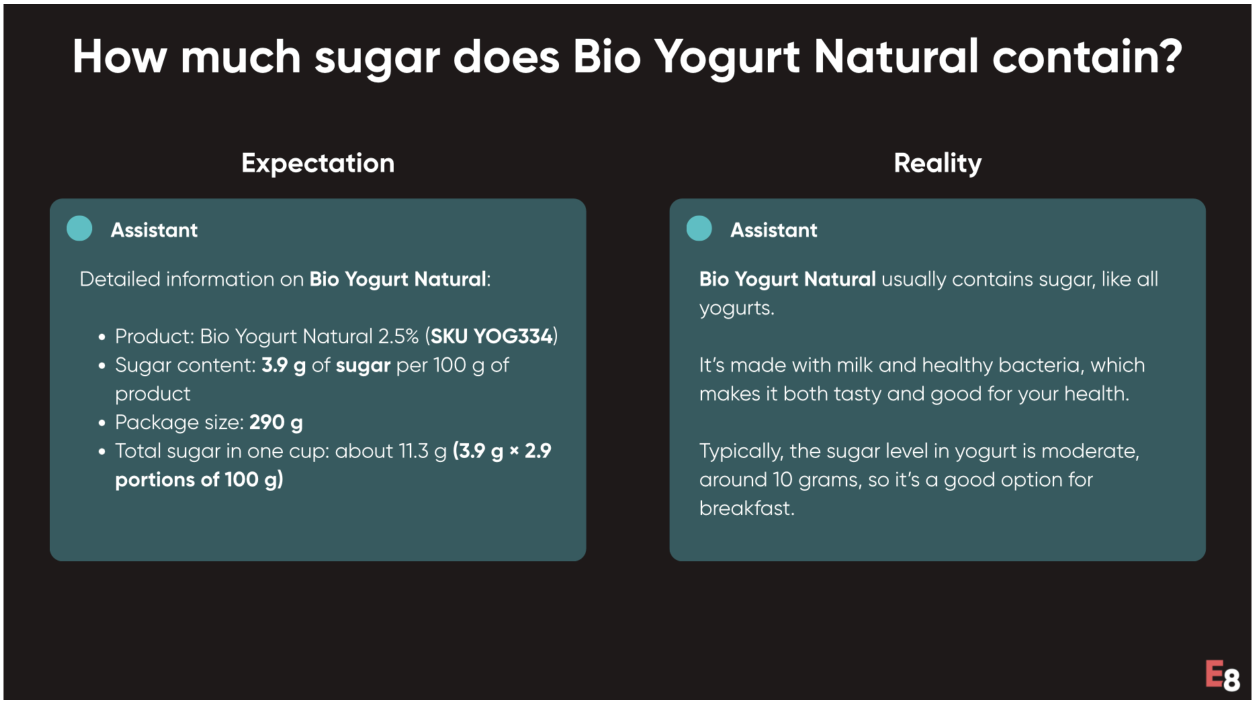
2. When you need a precise fact, not a guess

There's a whole class of queries where *"close enough"* is not acceptable. The user expects one exact value – not an approximation.

Examples:

- "What are the ingredients of product X?" → confusing it with product Y, even if the names are similar, is unacceptable.
- "What is the concentration of ingredient A in product X?" → the assistant must find the correct product and extract the exact number, not invent one.

Classic RAG, based purely on semantic similarity, often **guesses** in such cases.
The answer may look confident – but it isn't guaranteed to be correct.
For a business, even a single wrong number or ingredient can damage reputation or lead to legal consequences.



3. Domain-specific questions

A classic e-commerce example:

> "Which accessories are compatible with this model?"

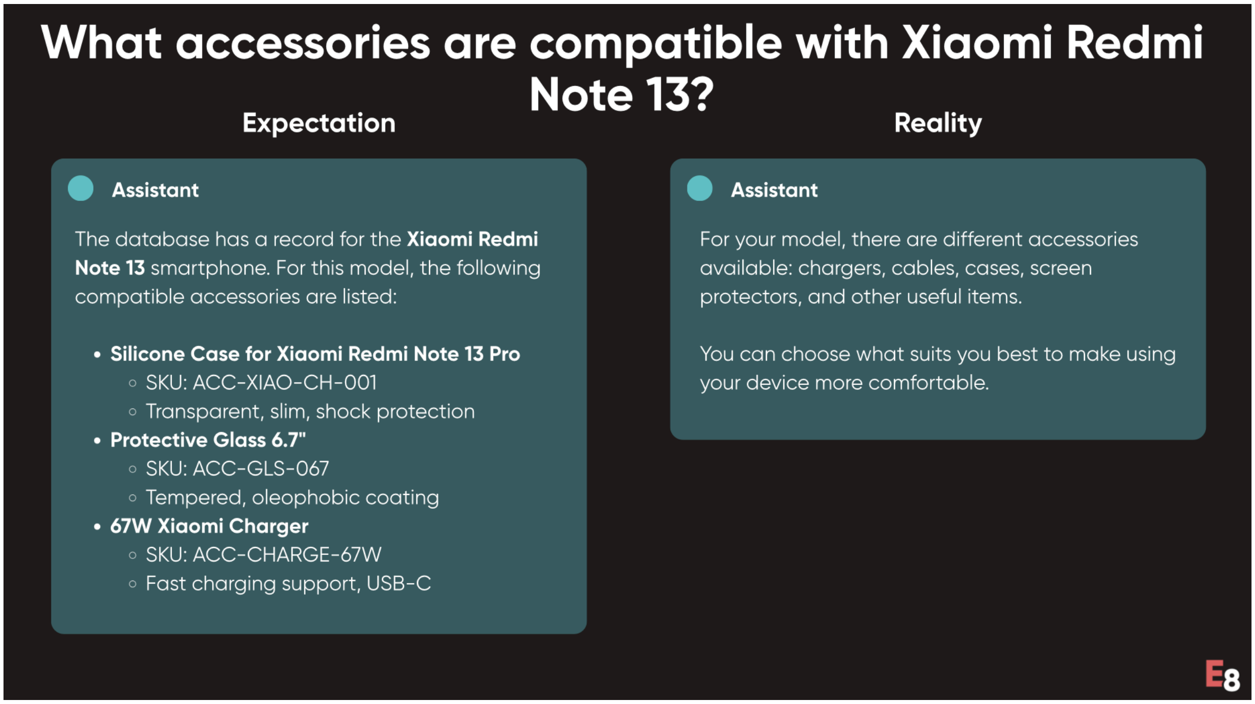
A classic RAG system looks for *similar* texts and may suggest items that **sound right but don't actually fit**.

For instance, it might return a case with the wrong form factor or a cable with the wrong connector – simply because the product descriptions shared similar words.

The root problem is that **naive RAG relies solely on semantic similarity**.
It ignores:

- actual compatibility logic (Bluetooth versions, dimensions, proprietary connectors);
- business rules ("this accessory fits models X, but not Y");
- relationships between entities in the data.

As a result, the user doesn't get an answer grounded in the logic of the domain – only a plausible *guess* stitched together from matching fragments.



Limitation of Language Models: no world model

All the issues with classic RAG trace back to one core fact: modern language models **lack a world model** to rely on.

LLMs don't build an internal *ontology* of a domain.
For them, facts and relationships are just patterns mixed into the text they were trained on.
When answering, the model doesn't truly *reason* — it simply stitches together the most probable sequence of words.

This absence of an explicit mental model shows up in predictable ways:

- weak generalization when faced with new or unseen cases,
- self-contradictions in longer reasoning chains,
- no ability to mentally simulate a system (e.g., an experiment or a business process),
- ignoring fixed rules and invariants unless they're explicitly encoded in the data.

For domains where strict logic and accuracy are critical – such as engineering, science, law, or finance – this becomes a **fundamental limitation**.

Our hypothesis

We start from a simple idea: **the domain model should be created by humans and stored outside the LLM.**

This model then serves as the assistant's *reasoning context*.
In this setup, the LLM is no longer a *knowledge holder* – it becomes an **interface**: understanding questions and translating them into precise actions on the data model.

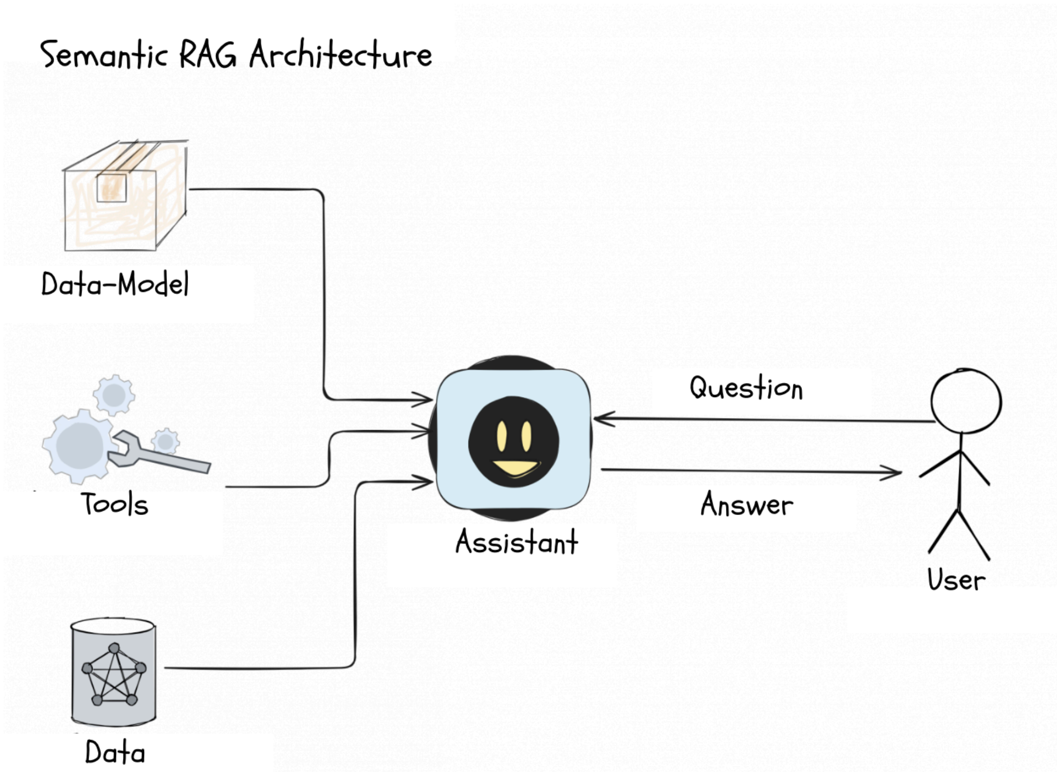
We propose rethinking the traditional RAG pipeline.
Instead of relying solely on semantic text similarity, we:

- **define the structure of the domain** — its entities, attributes, and relationships;
- **equip the assistant with tools** for data retrieval, not just vector search but also structured queries (SQL, Cypher) and API calls;
- **provide a playbook** — a set of instructions on how to choose tools and build reasoning logic for different query types.

This approach combines the **flexibility of language models** with the **precision of formal structures**.

Our approach: Semantic RAG

Building on this hypothesis, we developed the **Semantic RAG** architecture.



When the assistant receives a user question, it first examines the **domain model** to determine which **entities, attributes, and relationships** are required to answer it.

Next, the assistant constructs a reasoning plan and uses the appropriate tools:

- **semantic (vector) search** across documents,
- **structured queries** to databases (SQL, Cypher),
- **external API calls** when data must be retrieved from live systems.

This workflow produces not only *similar* text fragments but also **precise data operations** – filters, aggregations (COUNT, GROUP BY), and joins.

All retrieved facts are compiled into an **answer context** with explicit references to their sources:
doc_id:chunk_id, row_id in the database, or API response IDs.

Finally, based on these verified facts, the assistant generates a coherent answer – and **cites the sources**.

Now, let's look at how this works in detail.

Domain example: Legal compliance

To illustrate the foundation of **Semantic RAG**, let's look at a simplified example from the **legal domain**.

Context: we're working with manufacturers who must obtain official documents — such as quality or conformity certificates – to prove compliance with regulations before bringing products to market.

What kinds of questions should the assistant handle?

General legal and procedural questions:

- "What is a quality certificate?"
- "When is a conformity certificate required?"

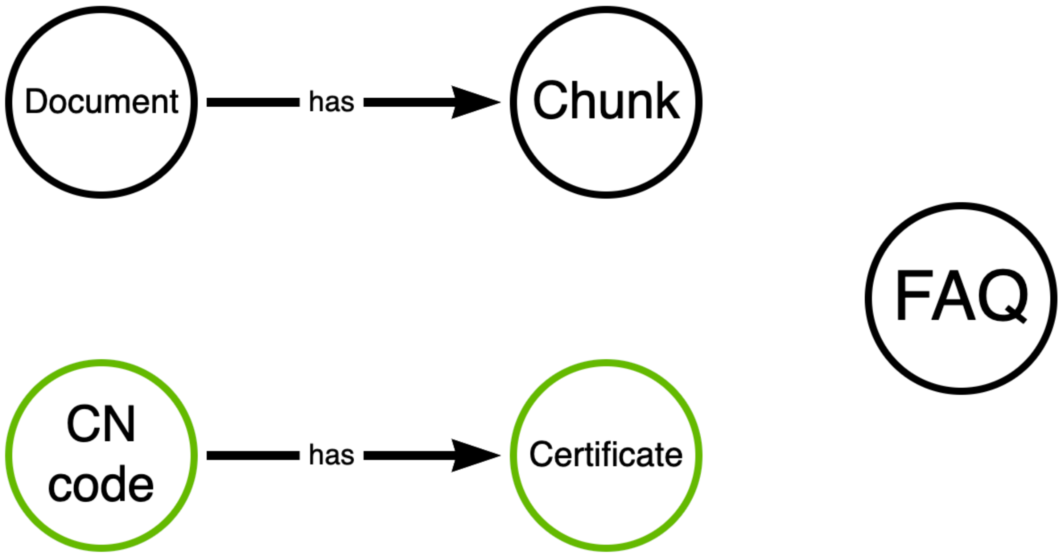
Precise, structured queries:

- "If I produce *[product]*, what CN (Combined Nomenclature) code applies to it?"
- "For CN code *X*, which documents must the manufacturer obtain?"

What we use to build the knowledge base

- **Regulations** – indexed *as is*, without modification.
- **FAQ (practical Q&A)** – expert answers illustrating how the rules are applied in practice.
- **CN code directory** – list of codes and corresponding product names.
- **Document directory** – catalog of all possible certificates and declarations.
- **Mapping table (CN → documents)** – distilled expert knowledge linking CN codes to required documents under specific conditions.
  This logic is extracted from multiple regulations – we don't ask the LLM to infer it; we prepare it in advance and load it as part of the structured data.

To make this clearer, let's visualize the full picture:



The main domain entities

Data

Below is the minimal set of tables that our example relies on.
(The screenshots illustrate their structure and sample data.)

- Document
  Regulatory acts and official letters.

ANCHOR_DOCUMENT

	document_id	document_title	document_content
1	document:001	Directive 2014/24/EU on public procurement	Establishes general rules for participation in public procurement across EU member states, including requirements on product origin, transparency of procedures, and equal access for EU manufacturers.
2	document:002	Regulation (EU) 2019/1020 on market surveillance	Defines the framework for checking compliance of industrial products in the EU, including certification mechanisms and verification of origin documents.
3	document:003	EU Rules of Origin Guidelines	Provides criteria for when a product is considered “Made in the EU” and sets out procedures for proving origin in trade agreements and support programs.
4	document:004	Critical Raw Materials Act (EU)	Regulates the use of critical raw materials and introduces localization requirements for production in strategic sectors.
5	document:005	Green Public Procurement Criteria (EU)	EU recommendations for sustainable public procurement, encouraging the use of products manufactured within the EU under environmentally friendly and localized production practices.
6	document:006	Industrial Strategy for Europe (2020)	Policy document of the European Commission setting measures to support EU industry, including localization, innovation, and participation in strategic value chains.

Table: **Regulatory Acts**

- Document Chunk
  Chunked fragments of documents used for semantic search.

ANCHOR_DOCUMENT_CHUNK

	document_chunk_id	document_chunk_text	link_document_has_chunk
1	chunk:000000003_document:001	The directive sets minimum transparency standards for all EU public procurement procedures.	Directive 2014/24/EU on public procurement
2	chunk:000000004_document:001	Contracting authorities must ensure equal treatment and non-discrimination of suppliers.	Directive 2014/24/EU on public procurement
3	chunk:000000005_document:002	Market surveillance authorities are responsible for verifying product compliance with EU legislation.	Regulation (EU) 2019/1020 on market surveillance
4	chunk:000000006_document:002	Products without proper conformity documents may be withdrawn from the EU market.	Regulation (EU) 2019/1020 on market surveillance
5	chunk:000000007_document:003	The Rules of Origin define when goods can be considered “EU originating”.	EU Rules of Origin Guidelines
6	chunk:000000008_document:003	Origin must be proven through certificates or approved self-declarations by manufacturers.	EU Rules of Origin Guidelines
7	chunk:000000009_document:004	The Critical Raw Materials Act introduces mandatory supply chain reporting for strategic sectors.	Critical Raw Materials Act (EU)
8	chunk:000000010_document:004	Companies must demonstrate reduced dependency on non-EU sources of critical materials.	Critical Raw Materials Act (EU)
9	chunk:000000011_document:005	Green Public Procurement Criteria encourage contracting authorities to prioritize eco-friendly products.	Green Public Procurement Criteria (EU)
10	chunk:000000012_document:005	The Industrial Strategy outlines localization targets for key industries such as energy, mobility, and digital technologies.	Green Public Procurement Criteria (EU)

Table: **Document Chunks**

- FAQ
  Questions and expert answers collected from real-world practice.

ANCHOR_FAQ

	faq_id	faq_question_text	faq_answer_text
1	faq:002	Why are supporting documents linked to product codes?	Because codes alone only identify a product; documents prove that the product meets regulatory, localization, or quality requirements.
2	faq:003	What is Directive 2014/24/EU about?	It sets the rules for public procurement across EU member states, including transparency, non-discrimination, and requirements on product origin.
3	faq:004	How is product origin confirmed in the EU?	Through Certificates of Origin, supplier declarations, or compliance with the EU's Rules of Origin Guidelines.
4	faq:005	What is the EU equivalent of Russia's PP 719?	There is no direct equivalent, but similar functions are covered by EU procurement directives, Rules of Origin, and industrial policy documents.
5	faq:006	What is the purpose of the Critical Raw Materials Act?	To ensure secure and sustainable supply chains for critical resources and to reduce dependency on imports from outside the EU.
6	faq:007	Why does the EU promote Green Public Procurement?	To encourage sustainable production and consumption by giving preference to eco-friendly and locally manufactured products in public tenders.
7	faq:008	What documents prove compliance with EU regulations?	Typically Declarations of Conformity, CE marking, Certificates of Origin, and supporting technical documentation.
8	faq:009	What is the Industrial Strategy for Europe?	It is an EU policy framework that defines measures to strengthen industry competitiveness, localization, and innovation in strategic sectors.
9	faq:010	Why do companies need to provide technical documentation?	Technical documentation demonstrates that the product complies with safety, environmental, and localization requirements set by EU regulations.

Table: **Domain Expertise Q&A**

- CN code
  Directory of product codes and names under the Combined Nomenclature system.

ANCHOR_CN_CODE

	code_id	cn_code_number	cn_code_name
1	cn:84795000	8479 50 00	Industrial robots, not elsewhere specified or included
2	cn:84433210	8443 32 10	Printers capable of connecting to an automatic data-processing machine or to a network
3	cn:84713000	8471 30 00	Portable automatic data-processing machines (e.g., laptops), ≤10 kg, with CPU, keyboard and display
4	cn:85176200	8517 62 00	Machines for reception, conversion and transmission or regeneration of data (incl. switching/routing apparatus)
5	cn:85076000	8507 60 00	Lithium-ion accumulators (batteries)
6	cn:85423111	8542 31 11	Electronic integrated circuits — processors and controllers
7	cn:85258091	8525 80 91	Video camera recorders (digital), able to record only images and sound taken by the television camera
8	cn:85044090	8504 40 90	Static converters (e.g., power supplies, inverters), other

Table: **Combined Nomenclature Codes**

- Compliance document
  Catalog of all types of supporting documents (certificates, declarations, etc.).

ANCHOR_COMPLIANCE_DOCUMENTS

	compliance_document_id	compliance_document_name
1	compdoc:001	Certificate of Origin
2	compdoc:002	Declaration of Conformity
3	compdoc:003	CE Marking
4	compdoc:004	Technical Documentation
5	compdoc:005	EU Ecolabel
6	compdoc:006	Supply Chain Due Diligence Report
7	compdoc:007	ISO 9001 Quality Management Certificate
8	compdoc:008	ISO 14001 Environmental Management Certificate

Table: **Compliance Documents**

- **CN ↔ Compliance Document Mapping**

A link table that defines which compliance documents are required for each CN code.
This mapping is stored separately for clarity and maintainability.

LINK_CNCODE_HAS_COMPDOC

	cn_code_id	compliance_document_id	requirement_type
1	cn:84795000	compdoc:002	mandatory
2	cn:84795000	compdoc:003	mandatory
3	cn:84795000	compdoc:004	mandatory
4	cn:84795000	compdoc:010	mandatory
5	cn:84795000	compdoc:009	mandatory
6	cn:84795000	compdoc:016	conditional
7	cn:84795000	compdoc:011	conditional
8	cn:84795000	compdoc:001	optional
9	cn:84795000	compdoc:007	optional
10	cn:84433210	compdoc:002	mandatory
11	cn:84433210	compdoc:003	mandatory
12	cn:84433210	compdoc:004	mandatory
13	cn:84433210	compdoc:010	mandatory
14	cn:84433210	compdoc:009	mandatory
15	cn:84433210	compdoc:016	conditional
16	cn:84433210	compdoc:001	optional

Link table between CN codes and Compliance Documents

Data Model

To describe the data, we use Minimal Modeling approach. We originally used it for our analytics projects a few years ago – and it turns out to work perfectly for LLM-based systems as well.

To model any domain, we decompose data into three core parts:

- **Anchors** – the main nouns (e.g., *user*, *order*, *product*, *warehouse*).
- **Attributes** – properties of anchors (e.g., *a user has an email*, *a product has a name*).
- **Links** – relationships between anchors (e.g., *a user placed an order*).

Anchor Table:

Anchor	Description	Example IDs / Aliases	Example Query
`cn\_code`	Combined Nomenclature code; needed for exact product matching and requirement lookup	`cn:84795000`, `cn:85437090`, `okpd:13.95.10`	`cypher MATCH (cn:cn_code {…}) RETURN cn`
`compliance\_document`	Supporting document (certificate/declaration) used to confirm compliance	`compdoc:002` (DoC), `compdoc:003` (CE Marking)	`cypher MATCH (d:compliance_document {…}) RETURN d`
`document`	Regulatory act or regulator's letter — source of truth for citations	`document:pp719_2023`, `document:eu_act_123`	`cypher MATCH (doc:document {…}) RETURN doc`
`document\_chunk`	Fragment of a document for semantic search and referencing the original source	`document_chunk:pp719_2023_001`	`cypher MATCH (c:document_chunk {…}) RETURN c`
`faq`	Q&A from practice/support; adds real-life cases and clarifications to the regulations	`faq:012`, `faq:016`	`cypher MATCH (f:faq {id:$faqId}) RETURN f`

Attribute Table:

Anchor	Attribute	Description	Embeddable
cn_code	cn_code_number	CN code for exact matching	no
cn_code	cn_code_name	CN item name	yes
compliance_document	compliance_document_name	Document name	yes
document	document_title	Document title/name	yes
document	document_content	Annotation / short summary	yes
document_chunk	document_chunk_text	Fragment text (for RAG)	yes
document_chunk	link_document_has_chunk	Link to source document	no
faq	faq_question_text	Question from practice/support	yes
faq	faq_answer_text	Expert/practice answer	yes
LINK_CNCODE_HAS_COMPDOC (link)	requirement_type	Type of requirement	no

Link Table:

Link (Sentence)	Anchor1	Anchor2	Description	Example Cypher
DOCUMENT_has_DOCUMENT_CHUNK	document	document_chunk	A document has chunks (text fragments).	cypher MATCH (d:document {document_id:$doc_id})-[:DOCUMENT_has_DOCUMENT_CHUNK]->(c:document_chunk) RETURN d.document_id, c.document_chunk_id, c.document_chunk_text
CN_CODE_requires_COMPDOC	cn_code	compliance_document	A CN code requires a supporting compliance document, with the type of requirement specified.	cypher MATCH (cn:cn_code {code_id:$cn_id})-[r:CN_CODE_requires_COMPDOC]->(d:compliance_document) RETURN cn.code_id, d.compliance_document_id, d.compliance_document_name, r.requirement_type

The descriptions of anchors, attributes, and links are stored in simple tables and passed as text into the assistant’s context together with the user’s query.

To help the assistant plan its reasoning steps and choose the right tools or queries, each anchor, attribute, and link includes several **service fields**:

1. **`description`**

A short, clear explanation of when and why to use this anchor, attribute, or link.
This is what the LLM relies on to decide whether to use the object and how to interpret the results.

2. **`query_example`**

A sample query that retrieves the relevant data:

- **SQL**, if the data is in a relational database;
- **Cypher**, if it's in a graph database such as Memgraph or Neo4j;
- or an **API call example**, if the data is fetched on demand.

These examples act as few-shot prompts for the assistant's tool use, reducing syntax and parameter errors.

3. **`embeddable` – semantic search flag** (`true/false`)

Indicates whether embeddings should be built and used for semantic search.

- Not used for numeric fields, IDs, or exact codes (e.g., CN code).
- Used for long texts (regulation fragments, FAQ answers) and for names or descriptions that benefit from semantic similarity.

Tools: How the assistant retrieves data

Once the data is prepared and the data model is defined, the assistant only needs **tools to extract facts**.
We use three main classes of tools.

1. Vector search

This is the **classic RAG** approach – but with one important constraint: we explicitly specify **which anchors and attributes** to search through.
This dramatically reduces noise and improves the relevance of retrieved chunks.

Signature (simplified):

```
vts(anchor, label, query)
```

Examples:

```
vts(anchor="faq", label="faq_question_text", query=$q)
vts(anchor="document_chunk", label="document_chunk_text",
query=$q)
```

2. Structured queries (Cypher / SQL)

For precise operations – "How many?", "Show all", compatibility checks, or filters by condition/date – the assistant executes **structured queries**.

In our setup, data is stored in **Memgraph**, so we use **Cypher**; the same logic applies to SQL.

Tool:
`cypher(query)`

Example (which documents are required for a given CN code, with requirement type):

```
MATCH (cn:cn_code {cn_code_id: "cn:84795000"})
-[rel:CN_CODE_requires_COMPDOC]->(d:compliance_document) RETURN
d.compliance_document_id AS compdoc_id,
d.compliance_document_name AS compdoc_name, rel.requirement_type
AS requirement_type
```

3. API and external tools

The toolset can be expanded depending on the use case.
The assistant can be given tools to:

- call **internal or external APIs**,
- perform **web searches**,
- or interact with other systems for data retrieval.

Playbook: how the assistant reasons

Once the data and data model are prepared, we **teach the assistant the reasoning sequence** for different types of queries.

For our legal domain, we define two key instructions:

1. Questions about legislation

- First, run a **vector search** over document_chunk_text — this is where legal norms and case practice are described.
- Then, run another **vector search** over faq — to include expert clarifications.
- **Answer format:**
  "By law, it's *X*; in practice, it's *Y*."

2. Questions about certification documents for a product

- First, use **vector search** to identify the product's **CN code**.
- Then, run a **structured query** against the database — using the product code to find all required **compliance documents**.

Here's what a corresponding instruction in the **Playbook** looks like:

query_name	query_example
If the user asks about legislation	1) Run `vts(anchor='document_chunk', label='document_chunk_text')`. 2) Run `vts(anchor='faq', label='faq_answer_text')`. 3) Compose the answer in the form: *Legally:* … // from `document_chunk_text` *Practically:* … // from `faq_answer_text`
If the user asks which documents are required for a product	1) Run `vts(anchor='cn_code', label='cn_code_name')` to identify the CN code. 2) Run the Cypher query: `cypher\nMATCH (cn:cn_code {cn_code_number:$code})-[:CN_CODE_requires_COMPDOC]->(d:compliance_document)\nRETURN d.compliance_document_name\n` 3) Return the list of documents in the answer.

Example: playbook for the assistant

How the assistant chooses tools

Before generating the final answer, the assistant performs several iterations of fact extraction, combining different tools depending on the query type and the data model.

For example:

1. On the **first iteration**, it may run **vector search** across the legislative database and the FAQ.
2. On the **second**, it identifies the **CN code** for the requested product.
3. On the **third**, it uses that CN code to retrieve related **compliance documents** and assemble the final answer.

This multi-step reasoning allows the assistant to ground its answers in verified facts rather than a single search result.

Example prompt (template for the tool planner):

```
You are a tool-using assistant for a knowledge base with a graph (Cypher) and vec
tor search.
Your job: answer the user's question by planning and executing calls to {tools},
using the provided data model.

Inputs you receive:
- graph_composition: textual description of entities/links + example queries
- user_query: the user's question

Policies:
- Prefer structured queries (Cypher/SQL) for lists, counts, compatibility and dat
e-bound rules.
- Use vector search only on fields marked embeddable=true.
- Always include citations: Document#section for norms, row_id/ids for structure
d results.

Planner (high level):
1) Identify intent: {legal_question | cn_requirements | other}.
2) Choose tools accordingly (you may combine multiple tools).
3) Compose answer in markdown with a "Sources" section.

Available tools:
- vts(anchor, label, query)
- cypher(query)

graph_composition:
{graph_composition}

Now, plan and run the minimal set of tool calls needed to answer:
user_query:
{user_query
```

How it works in practice

Let’s see how the assistant answers different types of questions using demo data *(codes and names are illustrative).*

User question

> “What documentation is needed to bring industrial robots to the EU market?”

This query clearly falls under the category **“certification documents for products.”**
Now, let’s look at how the assistant analyzes and resolves it step by step.

Step 1 – Finding candidate CN codes

The assistant begins with a **vector search** in the **CN Code directory** to determine which code corresponds to *industrial robots.*

```
2025-08-18 06:45:07,183 - rag_debug - INFO - Calling tool vector_text_search
2025-08-18 06:45:07,642 - rag_debug - INFO - vts_fn(label=cn_code, property=cn_co
de_name, th=0.5, n=5)
2025-08-18 06:45:10,019 - rag_debug - INFO - Tool vector_text_search (Vector sear
ch for similar text in node properties, use for semantic search)
Result:
Query: "industrial robots eu ce documentation"
vector_text_search
Rows:
{ "node": { "id": "cn:84795000", "cn_code_number": "8479 50 00", "cn_code_name":
"Industrial robots, not elsewhere specified or included", "type": "cn_code", "lab
els": ["cn_code"] }, "similarity": 0.935 }
{ "node": { "id": "cn:85437090", "cn_code_number": "8543 70 90", "cn_code_name":
"Electrical machines and apparatus with individual functions, not elsewhere speci
fied, other", "type": "cn_code", "labels": ["cn_code"] }, "similarity": 0.741 }
{ "node": { "id": "cn:90318080", "cn_code_number": "9031 80 80", "cn_code_name":
"Measuring or checking instruments, appliances and machines, not elsewhere specif
ied, other", "type": "cn_code", "labels": ["cn_code"] }, "similarity": 0.683 }
```

Logs, step 1 - understanding CN code

The search returns three candidates:

- 8479 50 00 – Industrial robots, not elsewhere specified or included (relevant).
- 8543 70 90 – Electrical machines and apparatus with individual functions, not elsewhere specified, other.
- 9031 80 80 – Measuring or checking instruments, appliances and machines, not elsewhere specified, other.

The most relevant match is **8479 50 00** – *Industrial robots.*
However, at this stage, the assistant doesn’t have to commit to a single code: it can keep several close candidates in context and refine them in later steps, or ask the user for clarification if confidence is low.

Step 2 – Structured query by CN code

Following the Playbook, once a CN code is identified, the assistant runs a **structured query** against the **graph database** to retrieve all related compliance documents.

Goal:
Fetch linked documents and their requirement types — *mandatory*, *conditional*, or *optional* — for the given product code.

```
2025-08-18 06:45:11,204 - rag_debug - INFO - Calling tool cypher
2025-08-18 06:45:11,677 - rag_debug - INFO - cypher_fn(query='
MATCH (cn:cn_code {cn_code_id: "cn:84795000"}) -[rel:CN_CODE_requires_COMPDOC]->
(d:compliance_document)
RETURN d.compliance_document_id AS compdoc_id, d.compliance_document_name AS comp
doc_name, rel.requirement_type AS requirement_type
ORDER BY CASE requirement_type WHEN "mandatory" THEN 0 WHEN "conditional" THEN 1
ELSE 2 END, compdoc_name
')
2025-08-18 06:45:15,982 - rag_debug - INFO - Tool cypher (Execute a Cypher query
against the graph database. Use for structured data retrieval and graph traversa
l.) result: Query: cypher
Rows:
{ "compdoc_id": "compdoc:002", "compdoc_name": "Declaration of Conformity", "requ
irement_type": "mandatory" }
{ "compdoc_id": "compdoc:003", "compdoc_name": "CE Marking", "requirement_type":
"mandatory" }
{ "compdoc_id": "compdoc:004", "compdoc_name": "Technical Documentation", "requir
ement_type": "mandatory" }
{ "compdoc_id": "compdoc:010", "compdoc_name": "RoHS Compliance Statement", "requ
irement_type": "mandatory" }
{ "compdoc_id": "compdoc:009", "compdoc_name": "REACH Compliance Statement", "req
uirement_type": "mandatory" }
{ "compdoc_id": "compdoc:016", "compdoc_name": "Notified Body Assessment Report
(RED/Machinery)", "requirement_type": "conditional" }
{ "compdoc_id": "compdoc:011", "compdoc_name": "EU Type-Examination Certificate
(Module B, Notified Body)", "requirement_type": "conditional" }
{ "compdoc_id": "compdoc:001", "compdoc_name": "Certificate of Origin", "requirem
ent_type": "optional" }
{ "compdoc_id": "compdoc:007", "compdoc_name": "ISO 9001 Quality Management Certi
ficate", "requirement_type": "optional" }
```

Logs, step 2 - understanding documents by CN-code

Step 3 – Adding practical context from FAQ

After fetching the structured data, the assistant enriches the answer with **practical insights** that go beyond formal regulations.
It does so by running a **vector search** against the **FAQ knowledge base** to find clarifications or examples from expert practice.

```
2025-08-18 06:45:16,743 - rag_debug - INFO - Calling tool vts
2025-08-18 06:45:17,019 - rag_debug - INFO - vts(anchor='faq', label='faq_answer_
text', th=0.30, n=5, query="industrial robots eu procurement practice")
2025-08-18 06:45:21,532 - rag_debug - INFO - Tool vts (Vector Text Search; anchor
='faq', label='faq_answer_text') result:
Rows:
{ "node": { "id": "faq:012", "faq_question_text": "...", "faq_answer_text":
"...", "type": "faq", "labels": ["faq"] }, "similarity": 0.873 }
{ "node": { "id": "faq:013", "faq_question_text": "...", "faq_answer_text":
"...", "type": "faq", "labels": ["faq"] }, "similarity": 0.847 }
{ "node": { "id": "faq:014", "faq_question_text": "...", "faq_answer_text":
"...", "type": "faq", "labels": ["faq"] }, "similarity": 0.822 }
{ "node": { "id": "faq:015", "faq_question_text": "...", "faq_answer_text":
"...", "type": "faq", "labels": ["faq"] }, "similarity": 0.804 }
{ "node": { "id": "faq:016", "faq_question_text": "...", "faq_answer_text":
"...", "type": "faq", "labels": ["faq"] }, "similarity": 0.774 }
```

Logs, step 3 (fragment) - finding related faq

By combining results from these three stages – **semantic search**, **structured reasoning**, and **practical context retrieval** – the assistant produces a grounded, verifiable answer that includes both legal requirements and real-world guidance.

Step 4 – Building the final answer

At this stage, the assistant already has all the necessary facts:

- excerpts from **regulatory documents**,
- the **list of required certificates** linked to the CN code,
- **practical clarifications** from the FAQ.

Now it combines all of this into a **human-readable yet verifiable** answer.

For placing the product **"industrial robots"** (CN 8479 50 00 — cn:84795000) on the EU market, the following set of documents is required:

✅ **Minimum by law**

- EU Declaration of Conformity (DoC) — [compdoc:002]
- CE Marking — [compdoc:003]
- Technical File (risk analysis, diagrams, list of standards) — [compdoc:004]
- Compliance with directives EMC (2014/30/EU), LVD (2014/35/EU — if it falls under voltage), RED (2014/53/EU — if radio modules are included). These requirements are reflected in the DoC and the Technical File → [compdoc:002, compdoc:004]
- Declaration of Conformity with RoHS (Restriction of Hazardous Substances, 2011/65/EU) — [compdoc:010]
- Notification under REACH / SVHC (Art. 33 REACH) — [compdoc:009]

🏛️ **When a third party is involved**

- Assessment/report from a Notified Body (RED/Machinery) — [compdoc:016]
- EU Type-Examination Certificate (Module B, Notified Body) — [compdoc:011]

💡 **Tender practice**

For "partly completed machinery" it is usually required to provide a Declaration of Incorporation + installation instructions,
while the final DoC/CE is prepared by the integrator — [faq:012].

Customers often request up-to-date test reports for current hardware/firmware revisions; otherwise, clarification requests may arise — [faq:014].
The DoC and instructions must be in the official language of the destination country; English is usually insufficient for final acceptance — [faq:016].

Assistant's answer

What matters here

- The assistant returned the full list of documents, not a random subset.
- The answer includes references to specific IDs (`compdoc:…`, `faq:…`), so the result can be verified and reproduced.
- The answer is built from structured data and real documents, not "made up" by the LLM. This greatly reduces the risk of hallucinations.

- Fixability is built in: if something is missing or wrong, it's clear what to change – add a document to the database or refine a link–instead of chasing the model's guesses.

Technique: Architecture in practice

Now let's look at how the architecture works from the **data and infrastructure** side.

Graph vs. relational databases (Cypher vs. SQL)

At the start, we had to choose the right database for the assistant:

- **Relational:** PostgreSQL + SQL
- **Graph:** Neo4j / Memgraph + Cypher

We chose a graph database, and here's why.

1. Flexible Schema

In real projects, the knowledge base constantly evolves.

Today we have *CN codes* and *certificates*.
Tomorrow an expert adds a *compatibility directory* or a *notes* field.

In a **graph database**, such changes are trivial: you simply add a new node type, property, or relationship – and the assistant can use it right away.

In **relational databases**, schema changes are much harder to implement and maintain.

2. More Expressive Query Language

Cypher is shorter and more intuitive for describing relationships — queries look like diagrams.
In SQL, the same logic turns into long chains of JOINs, which are harder to read and much more error-prone, especially when generated by an LLM.

3. Better Resilience to Model Drift

If the LLM misnames a property or relationship, **Cypher** simply returns an empty result.
In **SQL**, a wrong column name breaks the entire query.

For an assistant that generates queries dynamically, this resilience is critical —
a graph database is far more forgiving of generation errors.

Choosing Between Neo4j and Memgraph

Our main criterion was simple: **the ability to scale read operations** without immediately requiring a paid Enterprise license.

- **Memgraph Community** offers this out of the box with a *one master* + *multiple read replicas* topology.
- **Neo4j Community** does not: it runs as a single instance, and horizontal scaling is only available in the Enterprise edition.

Memory and performance considerations

Memgraph keeps the entire graph **in memory**, so server resources must be provisioned with headroom.
However, in our architecture, the **main memory consumer is not the graph itself** (its structure is lightweight) – it's the **vector embeddings** used for semantic search.

In practice, this means allocating **4–16 GB of RAM** and closely monitoring the growth of the embedding database.

If the graph no longer fits into memory, Memgraph offers a **Transactional mode**, where the graph is stored on disk. However, this mode is still experimental – it's slower, and all transactional data still needs to fit into RAM.

Limitations and trade-offs

Both Memgraph and Neo4j lack **multi-tenancy**, which means each project requires its own instance.

Despite the additional resource planning required, Memgraph's advantages – **flexible schema**, **lightweight queries**, and **horizontal scaling** – outweighed the drawbacks.
We therefore chose **Memgraph** as the foundation for our Semantic RAG architecture.

Structured manual data

For managing **directories and reference tables**, we use **Grist**.

Grist provides an interface that feels familiar to business users — similar to Excel — but with native support for **relational links**: a field in one table can directly reference a field in another.

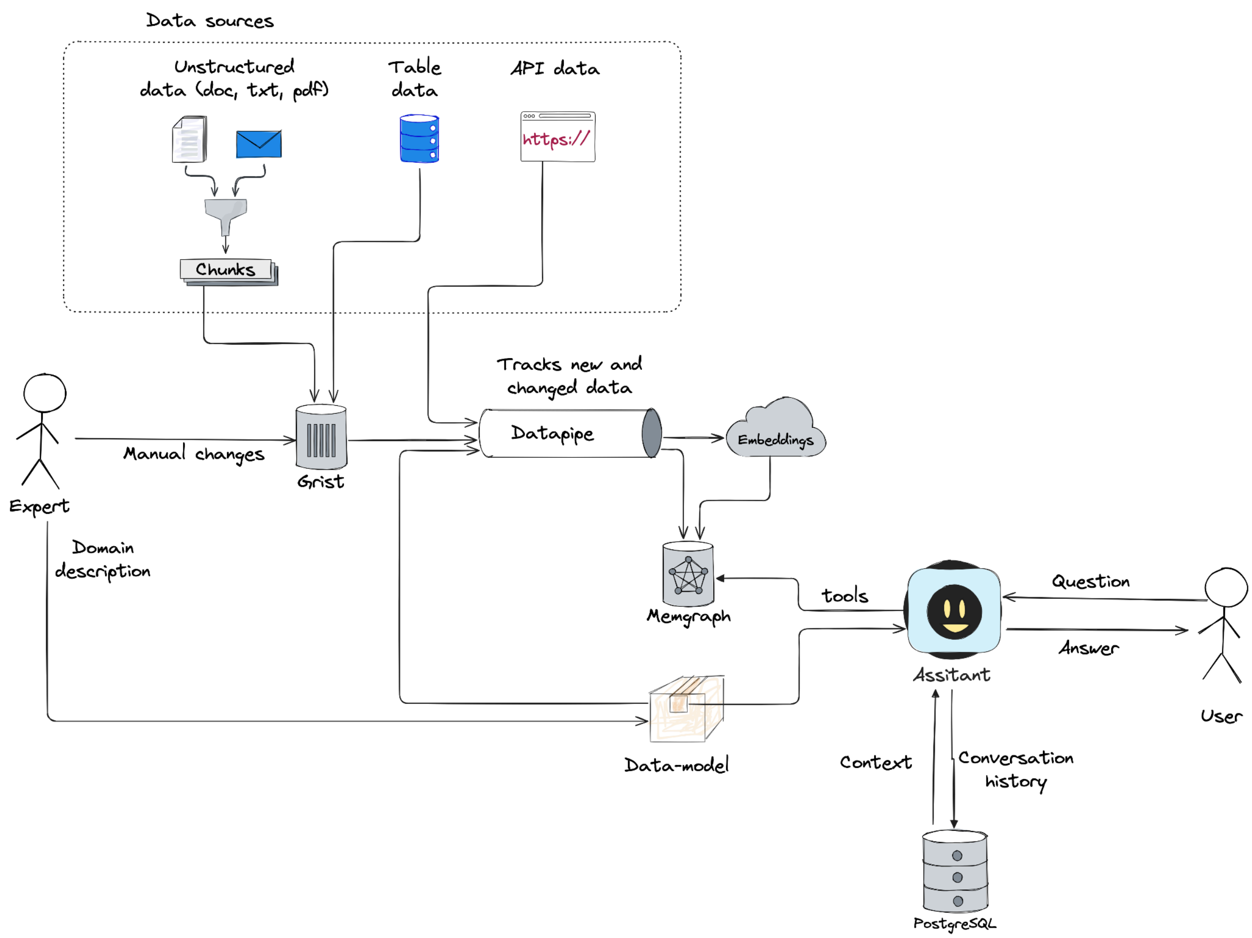
This makes data maintenance simple and transparent:
domain experts can add columns, update records, or create new fields **without developer involvement**.

Vector store

For searching through text fragments – such as regulatory documents or FAQ answers – we use a vector store embedded in Memgraph.

Overall system workflow

Here's how the **Semantic RAG** system operates – from data ingestion to the final user interaction.



1. Data sources

- **Documents** (`doc`, `pdf`, `txt`) → split into chunks and stored in **Grist** for review and editing.
- **Directories and reference tables** → also maintained in **Grist**, without any special preprocessing.
- **API data** (frequently updated or large-volume) → loaded directly into **Memgraph**.

2. Embeddings and data loading into Memgraph

To enable **vector search**, each data fragment must have an embedding.
The embedding is computed **only once** – when a record is added or updated.

3. Expert's role

The system remains accurate only with a **human expert in the loop**.
Their key responsibilities are:

- **Define the data model** – specify Anchors, Attributes, and Links.
- **Create the Playbook** – write step-by-step reasoning instructions describing how the assistant should answer typical queries (what data to check, what conclusions to draw).
- **Curate and correct data** – review the content loaded from automated sources and fix noise or missing entries.

4. Assistant's role

The final stage is the **assistant's interaction with the user**.

The assistant receives:

- the **user's question**,
- the **data model description**,
- the **Playbook** (reasoning recipes),
- and **access to data and tools**.

From there, it decides which tools to apply:

- **vector (embedding-based) search**,
- **structured queries** in Memgraph,
- **API calls** for dynamic or external context.

The assistant then generates a final answer with source references, ensuring that every fact is verifiable and the result reproducible.

All dialogue history and context metadata are stored in PostgreSQL.

How to check assistant answer quality

The quality of an assistant can't be judged by eye – it requires a systematic testing process. We use a golden dataset: a set of typical questions with reference answers.

1) Building the Golden Dataset

For each scenario in the Playbook, we prepare 20-30 sample questions that:

- cover all logic branches (different query formats, edge cases, exceptions);
- include a reference answer with explicit source IDs (chunks and structured rows) the answer must rely on.

> The LLM may rephrase text, but the sources must match – that's what we verify.

2) When to run tests

Run the test suite after every major change, including:

- data updates or expansion;
- data model changes (new anchors/links);
- LLM switch or model update.

3) Quality metric

Our primary metric is Accuracy – the share of answers that are fully correct in both content *and* sources.

Value of the approach

Clients often ask: “How good are the assistant’s answers?”

The honest answer is: it depends on the type of query and the data zone it belongs to.

1. Unstructured Texts (Chunks)

In this zone, answers are built purely through **semantic search** across text fragments.

Quality drivers:

- completeness and relevance of sources,
- chunking strategy,
- retriever performance,
- and how the user formulates the question.

Pros:

- quick to launch,
- covers the “gray zone” of knowledge without explicit modeling.

Cons:

- missed facts, duplicates, and ambiguities,
- difficult to test and verify rule correctness.

Typical accuracy: ~60–70%.

2. Structured Data (Data Model + Logic)

Foundation: a graph with defined entities, attributes, and relationships.
On top: a Playbook that encodes reasoning steps and validation checks.

Advantages:

- every fact's origin and calculation method are explicitly controlled,
- results are predictable, repeatable, and debuggable.

Outcome:
Accuracy reaches **90–97%**, depending on the domain's complexity and the completeness of the model.

Conclusion

Classic RAG, based solely on vector search, is a convenient way to *"connect an assistant to documents."*

But it inevitably runs into fundamental limitations:

- it can't return full or filtered lists of objects,
- it doesn't guarantee factual accuracy,
- and it struggles with domain-specific logic.

The root cause lies in the nature of LLMs: they have no stable *world model*, and without an external structure they drift toward inaccuracies and hallucinations.

Semantic RAG addresses these issues head-on.

We explicitly define the **domain model** – anchors, attributes, and links – connect structured data sources, and equip the assistant with a full **toolkit**: vector search, Cypher queries, and API calls.

In this setup, the LLM stops being a *knowledge store* and becomes an **interface** for reasoning over the knowledge base.

What this delivers:

- **Complete, verifiable answers** – with source references and record IDs.
- **Adherence to domain rules** – following real constraints, not imitating them.
- **Structured-answer accuracy of 90–97%** – a level unattainable with classic, vector-only RAG systems.