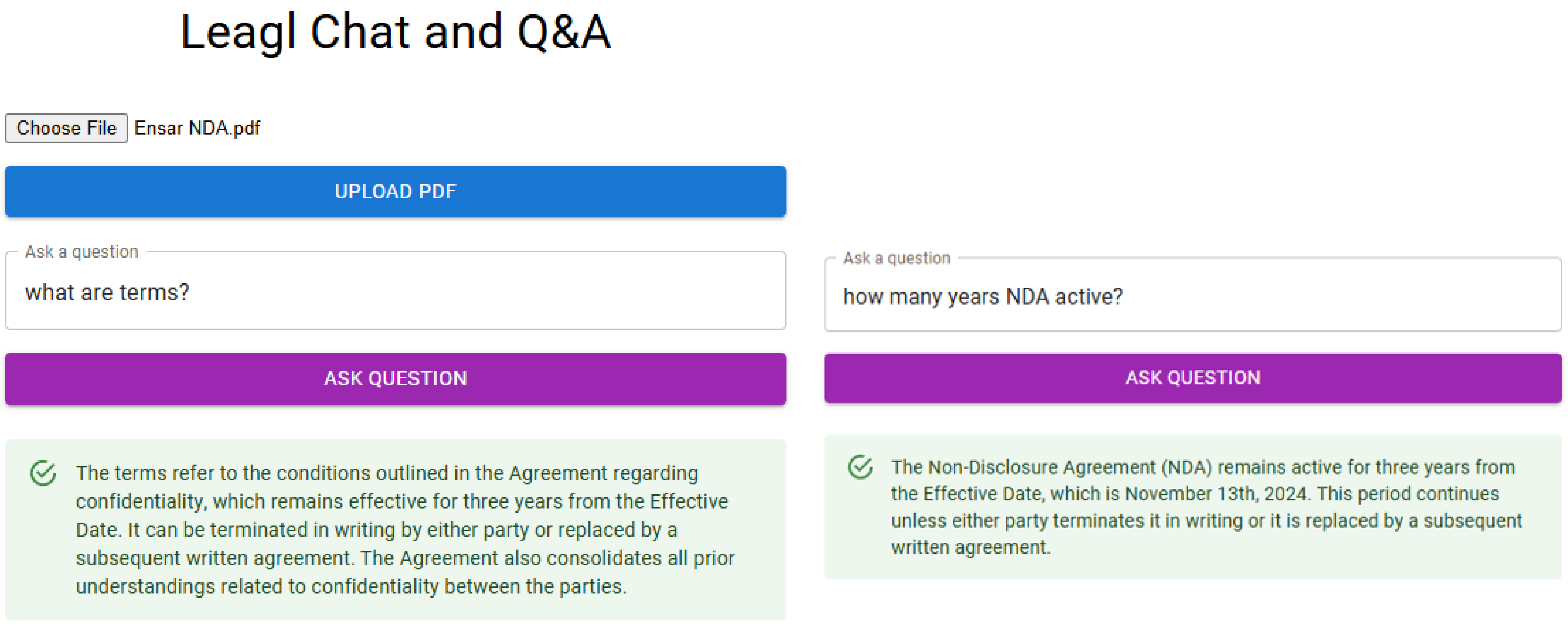
Ensar

Legal Chat – 6


Leagl Chat and Q&A

what are terms?

The terms refer to the conditions outlined in the Agreement regarding confidentiality, which remains effective for three years from the Effective Date. It can be terminated in writing by either party or replaced by a subsequent written agreement. The Agreement also consolidates all prior understandings related to confidentiality between the parties.

how many years NDA active?

The Non-Disclosure Agreement (NDA) remains active for three years from the Effective Date, which is November 13th, 2024. This period continues unless either party terminates it in writing or it is replaced by a subsequent written agreement.

https://github.com/ensardemos/leagalchat

React

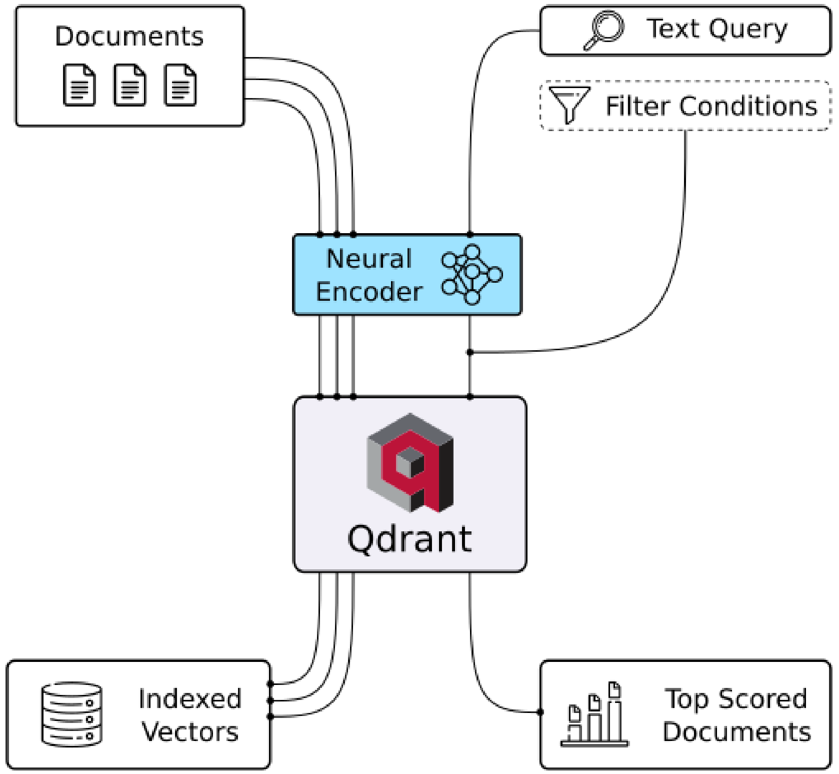
FastAPI

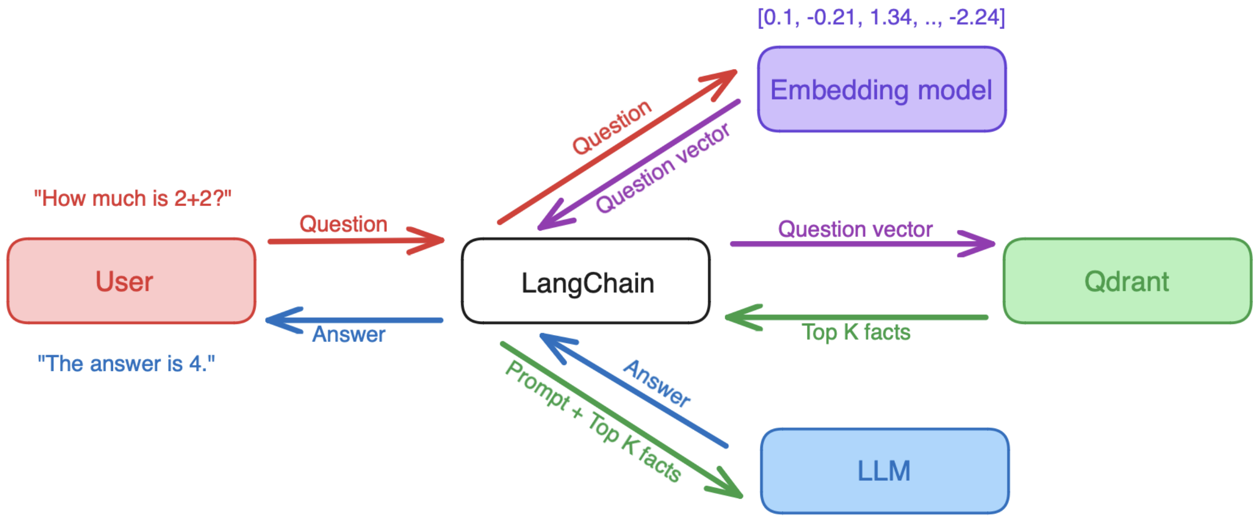
Anaconda

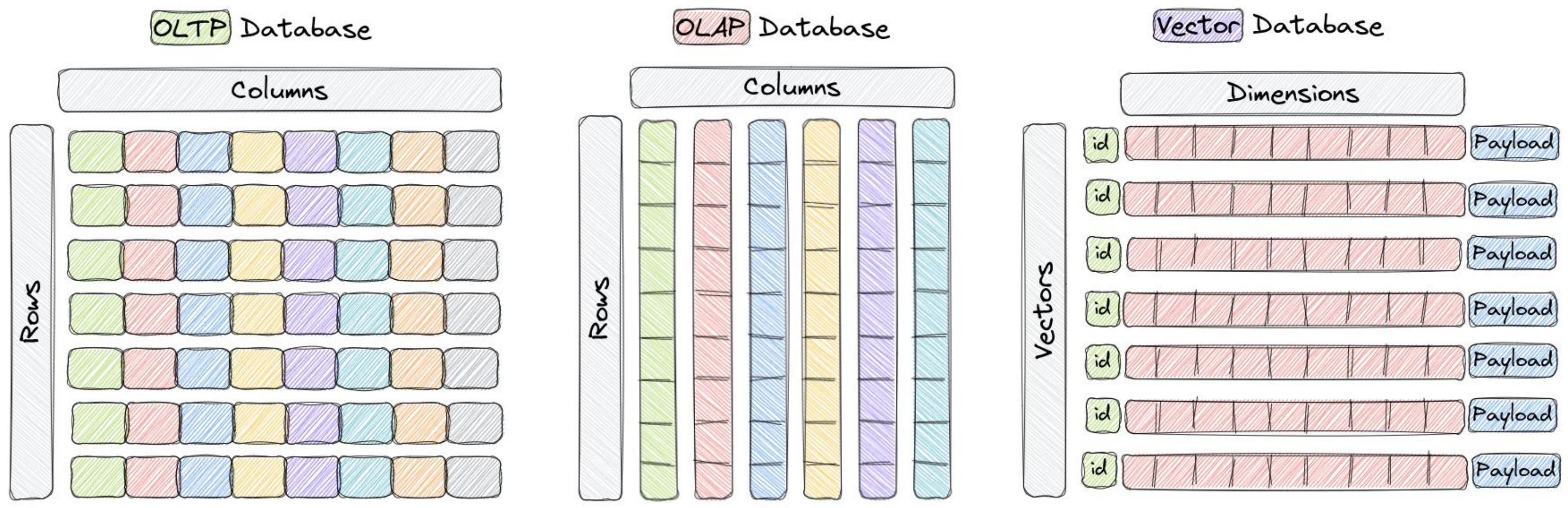
LangChain

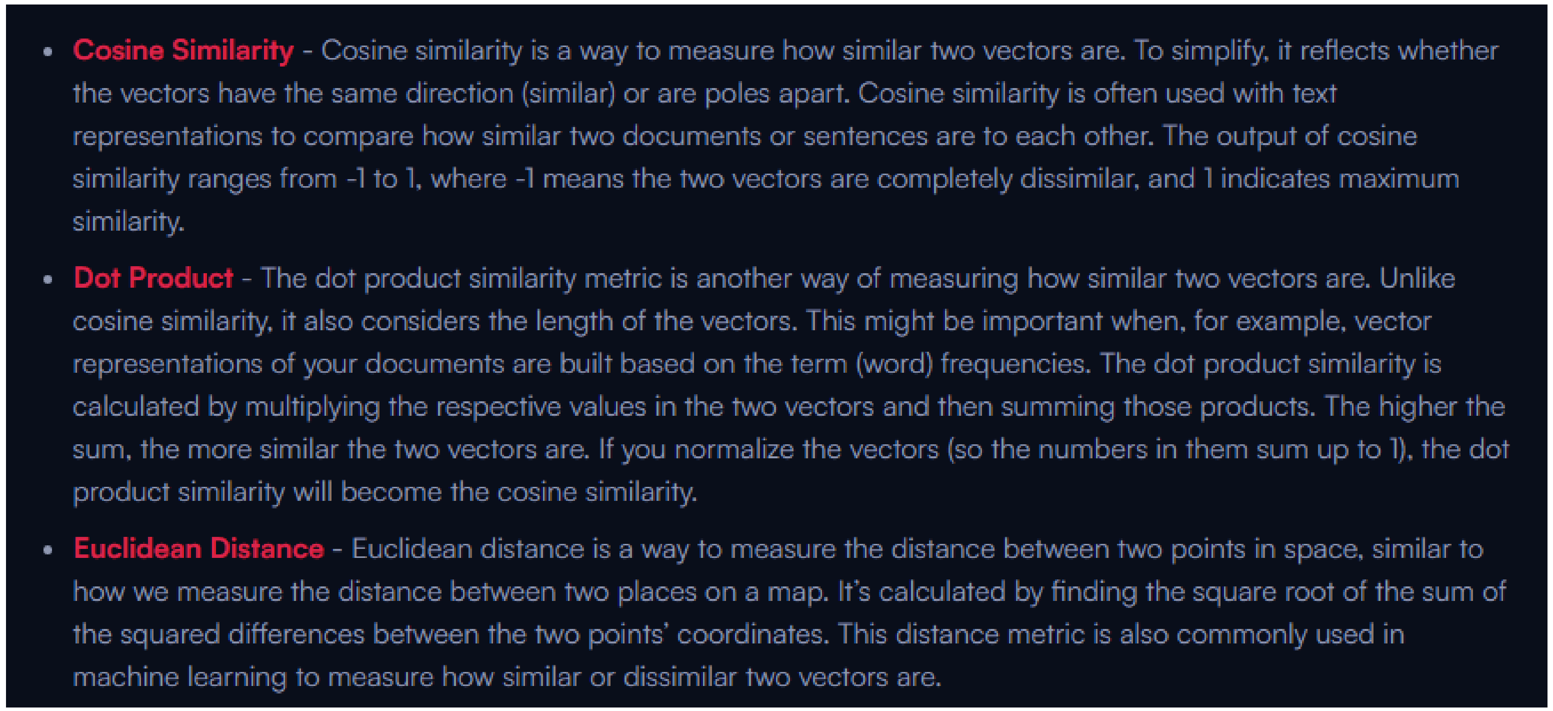
OpenAI

Qdrant

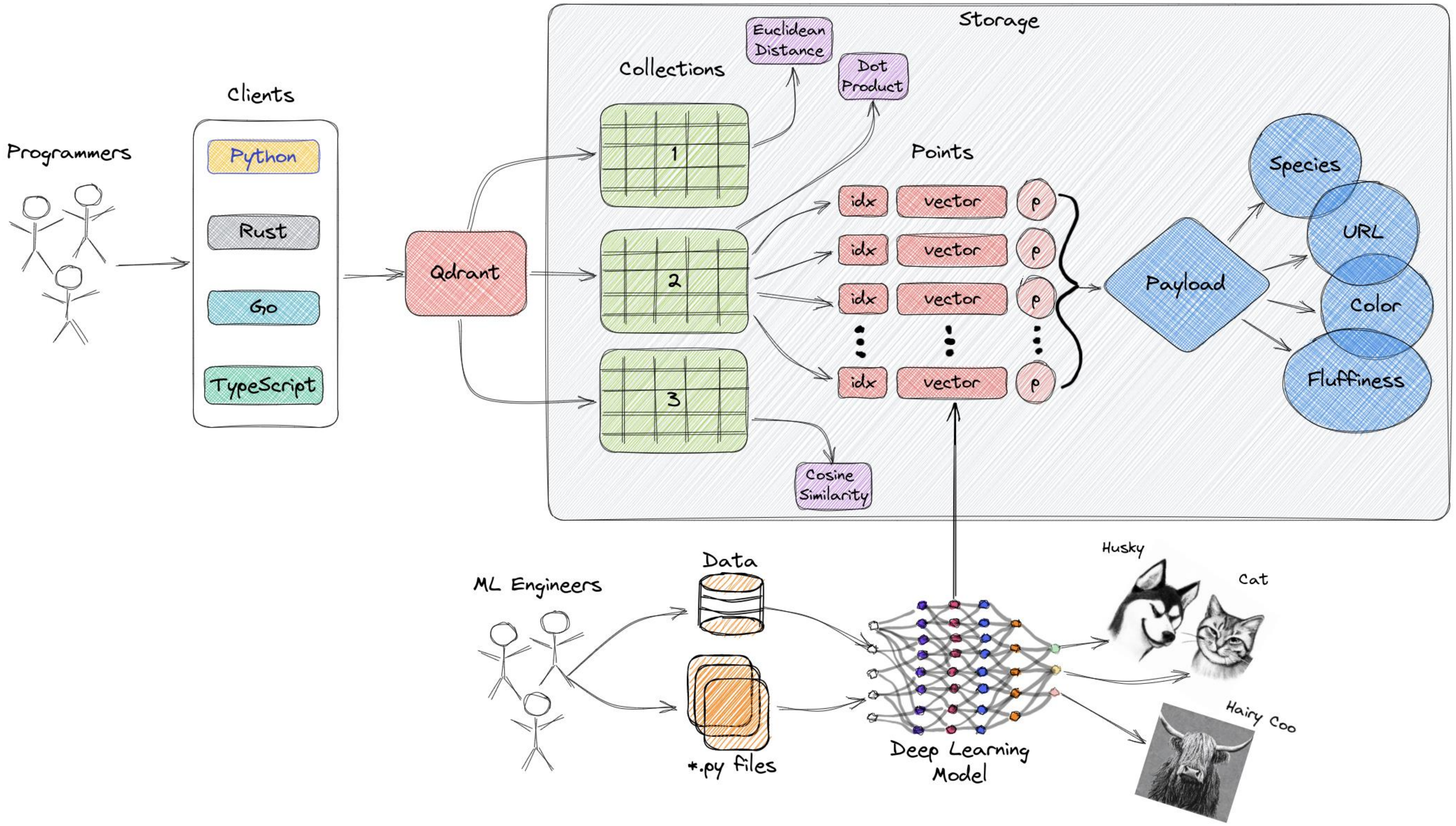

Documents
Text Query
Filter Conditions
Neural Encoder
Qdrant
Indexed Vectors
Top Scored Documents


[0.1, -0.21, 1.34, .., -2.24]
Embedding model
Question
Question vector
"How much is 2+2?"
Question
User
Answer
"The answer is 4."
LangChain
Question vector
Qdrant
Top K facts
Answer
Prompt + Top K facts
LLM





- **Cosine Similarity** - Cosine similarity is a way to measure how similar two vectors are. To simplify, it reflects whether the vectors have the same direction (similar) or are poles apart. Cosine similarity is often used with text representations to compare how similar two documents or sentences are to each other. The output of cosine similarity ranges from -1 to 1, where -1 means the two vectors are completely dissimilar, and 1 indicates maximum similarity.
- **Dot Product** - The dot product similarity metric is another way of measuring how similar two vectors are. Unlike cosine similarity, it also considers the length of the vectors. This might be important when, for example, vector representations of your documents are built based on the term (word) frequencies. The dot product similarity is calculated by multiplying the respective values in the two vectors and then summing those products. The higher the sum, the more similar the two vectors are. If you normalize the vectors (so the numbers in them sum up to 1), the dot product similarity will become the cosine similarity.
- **Euclidean Distance** - Euclidean distance is a way to measure the distance between two points in space, similar to how we measure the distance between two places on a map. It's calculated by finding the square root of the sum of the squared differences between the two points' coordinates. This distance metric is also commonly used in machine learning to measure how similar or dissimilar two vectors are.


Storage
Euclidean Distance
Dot Product
Collections
Clients
Programmers
Python
Rust
Go
TypeScript
Qdrant
1
2
3
Points
idx
vector
p
Payload
Species
URL
Color
Fluffiness
Cosine Similarity
ML Engineers
Data
*.py files
Deep Learning Model
Husky
Cat
Hairy Coo

1. FastAPI

- **Purpose**: A high-performance web framework for building APIs with Python 3.7+.
- **Key Features**:
  - Fast to develop: Allows for quick API development and clean code using async support and data validation.
  - Auto-generated docs: Automatically generates interactive Swagger and Redoc documentation.
  - Built-in request validation: Uses Python typing to validate request data automatically.
- **Common Requirements**:
  - High-performance API with async capabilities.
  - Automatic documentation generation for APIs.
  - JSON-based RESTful endpoints or WebSocket connections.

2. OpenAI

- **Purpose**: Interface to OpenAI's API for accessing language models like GPT-4.
- **Key Features**:
  - Access to various OpenAI models, including GPT-3.5, GPT-4, and embeddings models.
  - Chat and completion endpoints to generate text, answer questions, and more.
  - Model fine-tuning capabilities for custom applications.
- **Common Requirements**:
  - Text generation, language understanding, or conversation capabilities.
  - Embeddings generation for semantic search or retrieval tasks.
  - Integration with applications needing AI-driven natural language processing.

3. Uvicorn

- **Purpose:** ASGI server to run asynchronous Python applications (like FastAPI).
- **Key Features**:
  - Supports async and sync applications.
  - Lightweight and high-performance, ideal for production environments.
  - Hot-reload option for local development.
- **Common Requirements**:
  - Deployment of asynchronous web applications (e.g., FastAPI).
  - Efficient handling of WebSocket connections and async requests.
  - Production-ready server for Python web applications.

4. python-dotenv

- **Purpose**: Load environment variables from a `.env` file.
- **Key Features**:
  - Simplifies managing environment variables for local development.
  - Loads sensitive data (e.g., API keys) from a file without hardcoding them.
  - Avoids exposing secrets in code, enhancing security.
- **Common Requirements**:
  - Centralized configuration management with a `.env` file.
  - Securely storing sensitive data like database credentials, API keys, etc.
  - Environment-specific settings for staging, development, and production.

5. LangChain

- **Purpose:** Framework to build applications with language models.
- **Key Features:**
  - Provides chains to connect LLMs with other tools (search, databases, APIs).
  - Memory components for conversation context.
  - Tools for LLM-based decision making, question-answering, and more.
- **Common Requirements:**
  - Application workflows requiring orchestration with multiple language models.
  - Interactive and contextual applications where memory and tool use are needed.
  - Easy management of prompt engineering and chaining multiple models.

6. langchain_community

- **Purpose:** Contains community-contributed tools and integrations for LangChain.
- **Key Features:**
  - Extends LangChain with integrations not in the core framework.
  - Offers community-maintained utilities and modules.
- **Common Requirements:**
  - Leveraging experimental or niche tools for specific LangChain applications.
  - Access to community-supported integrations and functionalities.
  - Rapidly developing unique LLM-based applications.

7. langchain-core

- **Purpose:** Core package for LangChain's main functionality.
- **Key Features:**
  - Houses fundamental building blocks and modules for LangChain.
  - Tools for managing prompts, memory, and chains.
- **Common Requirements:**
  - Core infrastructure for LangChain applications.
  - Utilizes stable, non-community-dependent LangChain components.
  - Builds structured workflows for complex LLM applications.

8. langchain-openai

- **Purpose:** Provides OpenAI-specific modules and tools for LangChain.
- **Key Features:**
  - Streamlined OpenAI API integration with LangChain workflows.
  - Functions for accessing OpenAI's language models and embeddings.
- **Common Requirements:**
  - Direct integration with OpenAI models for LangChain workflows.
  - Using OpenAI's language models with LangChain memory and chaining.

9. qdrant-client

- **Purpose:** Client library for Qdrant, a vector database for high-performance search and retrieval
- **Key Features:**
    - Manages and searches large vector spaces.
    - Ideal for handling LLM embeddings and building semantic search systems.
- **Common Requirements:**
    - Efficient storage and retrieval of embeddings from language models.
    - Semantic search, similarity matching, or recommendation systems.
    - Vector-based storage for high-dimensional data in LLM applications.

10. python-multipart

- **Purpose:** Supports parsing `multipart/form-data` requests, commonly for file uploads.
- **Key Features:**
    - Parses files, images, and other data submitted via HTTP forms.
    - Useful for handling form-based file uploads in APIs.
- **Common Requirements:**
    - APIs handling file uploads, such as images or documents.
    - Parsing and validating incoming form-data requests.
    - File processing within web applications.

11. pypdf

- **Purpose:** PDF toolkit for reading, writing, and manipulating PDF files.
- **Key Features:**
    - Extracts text, merges, splits, or modifies PDF files.
    - Python-based PDF manipulation without requiring external dependencies.
- **Common Requirements:**
    - Reading or extracting text from PDFs for NLP or analysis tasks.
    - Basic manipulation (e.g., merging, splitting) of PDF documents.
    - Use in document processing workflows, such as parsing and summarizing.

1. Environment Setup and Configuration

- **Purpose:** Load environment variables required for the application, including API keys a
- **Components:**
  - `dotenv` to load API keys and URLs from environment variables (e.g., `OPENAI_API_I` `QDRANT_URL`, and `QDRANT_API_KEY`).
- **Workflow:** Load environment variables and store them in specific constants for use in downstream operations.

2. PDF Processing

- **Purpose:** Process a PDF document by loading its content and splitting it into manageable chunks.
- **Components:**
  - `PyPDFLoader` to load and read PDF documents.
  - `RecursiveCharacterTextSplitter` to split the document into chunks of text.
- **Workflow:**
  - Load the PDF and extract text.
  - Use a text splitter to divide the document into smaller chunks, which will later be vectorized for storage in the database.

3. Embedding and Vectorization

- **Purpose:** Generate embeddings for each document chunk to enable semantic searching and store them in Qdrant.
- **Components:**
  - `OpenAIEmbeddings` to create embeddings using OpenAI's model.
  - `Qdrant` and `QdrantClient` for storing these embeddings in a vector database.
- **Workflow:**
  - Initialize an embedding model with OpenAI API.
  - Generate embeddings for each chunked document.
  - Store these embeddings in the Qdrant vector database, enabling semantic search.

4. Qdrant Vector Store Client Initialization

- **Purpose:** Initialize and configure the Qdrant client to interact with the vector database.
- **Components:**
  - `QdrantClient` for handling connections to the Qdrant database.
  - `Qdrant` to create and manage collections for storing document embeddings.
- **Workflow:**
  - Set up a Qdrant client with the required URL and API key.
  - Configure the vector store by specifying the collection name and embedding model.

5. Question Answering (QA) Pipeline

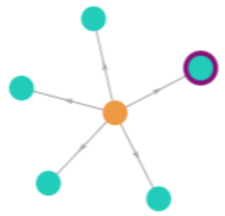
- **Purpose:** Provide answers to user queries based on semantically similar document chunks retrieved from Qdrant.
- **Components:**
  - `ChatPromptTemplate` for constructing structured prompts.
  - `RunnablePassthrough` and `RunnableParallel` for parallel data processing.
  - `ChatOpenAI` to generate responses using OpenAI's language model.
  - `StrOutputParser` to parse model responses.
- **Workflow:**
  - Configure a structured prompt template with response guidelines for the AI.
  - Retrieve relevant document chunks based on the similarity to the input query.
  - Pass the retrieved context and user query to the language model via the prompt template.
  - Generate a response using the model, parse the output, and return a concise answer.

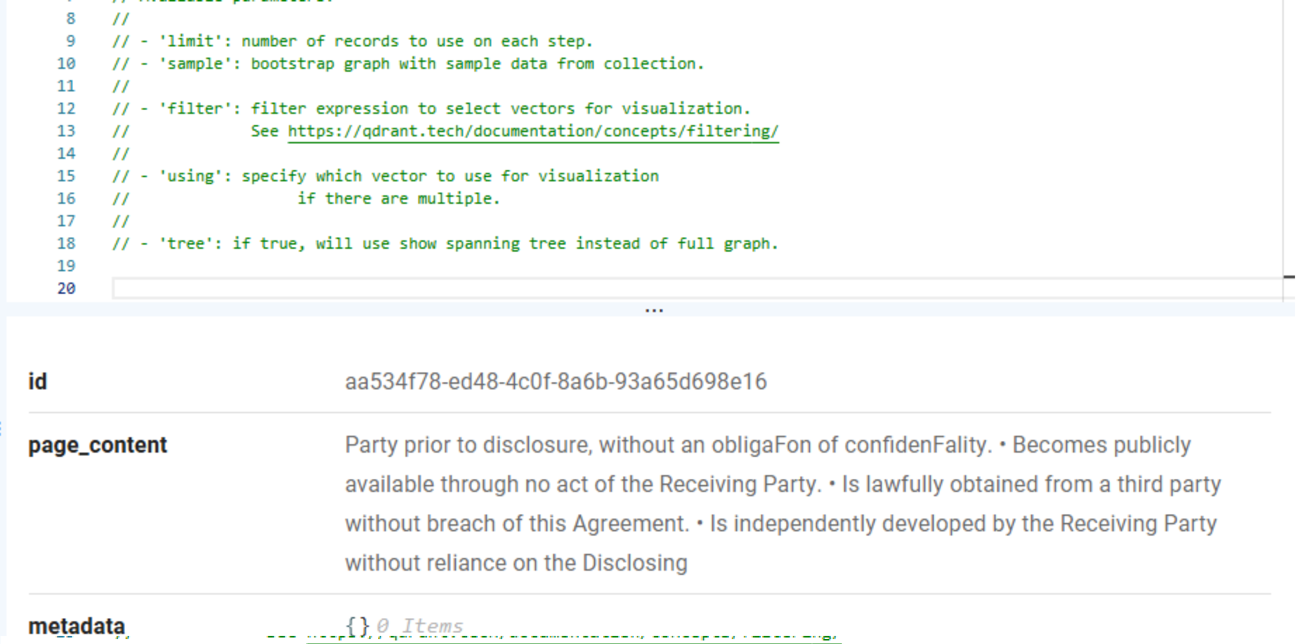
6. Error Handling and Logging

- **Purpose:** Handle errors gracefully across all functions and log issues if they arise.
- **Components:**
  - Exception handling blocks in each function.
- **Workflow:** When an error occurs, log the error details, and if applicable, return an informative error message to the user.

Potential Enhancements

- **Chunking Optimization:** Dynamically adjust chunk size and overlap based on document content characteristics.
- **Retrieval Improvement:** Experiment with different search parameters and configurations in Qdrant to improve retrieval accuracy.
- **Error Logging and Monitoring:** Implement structured logging for easier troubleshooting and performance monitoring.

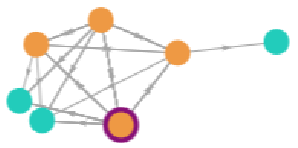


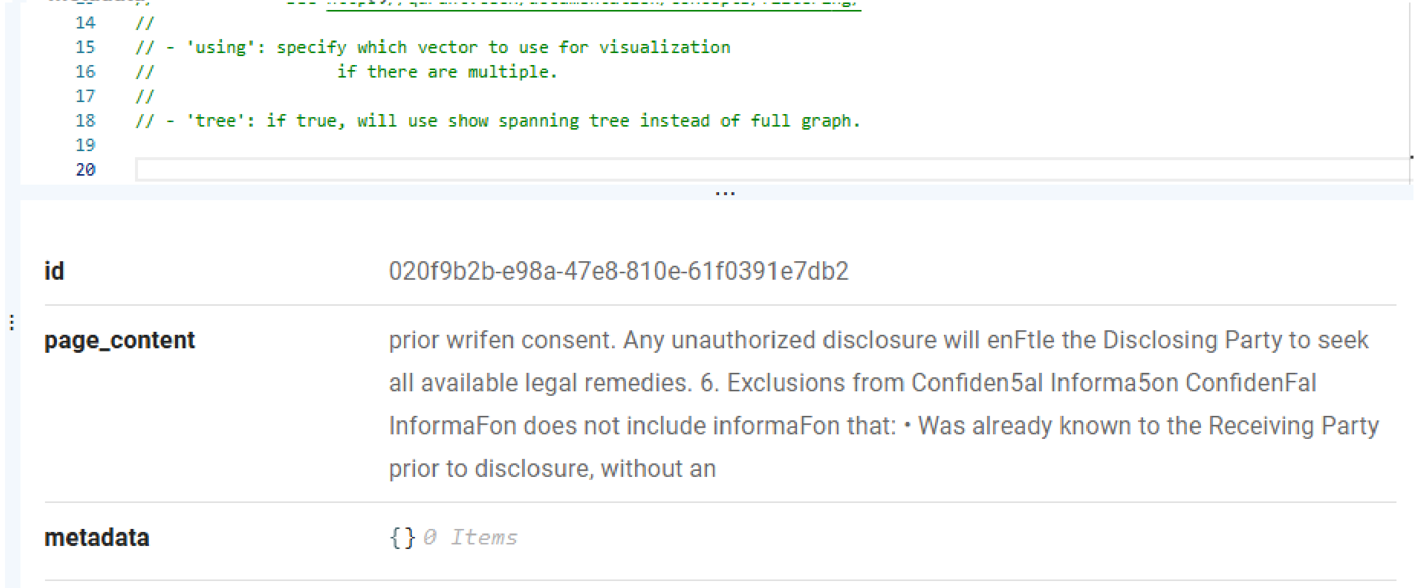


```
//
// - 'limit': number of records to use on each step.
// - 'sample': bootstrap graph with sample data from collection.
//
// - 'filter': filter expression to select vectors for visualization.
//             See https://qdrant.tech/documentation/concepts/filtering/
//
// - 'using': specify which vector to use for visualization
//                  if there are multiple.
//
// - 'tree': if true, will use show spanning tree instead of full graph.

```

id	aa534f78-ed48-4c0f-8a6b-93a65d698e16
page_content	Party prior to disclosure, without an obligaFon of confidenFality. • Becomes publicly available through no act of the Receiving Party. • Is lawfully obtained from a third party without breach of this Agreement. • Is independently developed by the Receiving Party without reliance on the Disclosing
metadata	{} *0 Items*





```
//
// - 'using': specify which vector to use for visualization
//                  if there are multiple.
//
// - 'tree': if true, will use show spanning tree instead of full graph.

```

id	020f9b2b-e98a-47e8-810e-61f0391e7db2
page_content	prior wrifen consent. Any unauthorized disclosure will enFtle the Disclosing Party to seek all available legal remedies. 6. Exclusions from Confiden5al Informa5on ConfidenFal InformaFon does not include informaFon that: • Was already known to the Receiving Party prior to disclosure, without an
metadata	{} *0 Items*