**MILVUS DB**

Minikube start

Kubectl get sc

kubectl patch storageclass local-path -p '{\"metadata\": {\"annotations\":{\"storageclass.kubernetes.io/is-default-class\":\"true\"}}}'

kubectl patch storageclass standard -p '{\"metadata\": {\"annotations\":{\"storageclass.kubernetes.io/is-default-class\":\"false\"}}}'

kubectl get-pods

kubectl port-forward pod/my-release-milvus-standalone-866576f7cc-jzt2m 19530:19530

**SUPERSTACK**

**The SuperStack**

**October 16, 2023**

**Dillon Martin**

Founder & co-CEO

**In this blog post:**

* Problems with Standard RAG Pipelines
* The SuperStack Features
* Using the SuperStack

A blue and white square with a diamond

Description automatically generated

In the world of LLM applications, accuracy and reliability are crucial to confidently deploying to production. Traditional Retrieval Augmented Generation (RAG) pipelines have served us well, but they come with their fair share of limitations.

From overlooking relevant information to misunderstanding the context of chunks and embeddings, there is space to improve both retrieval and text generation. Allow me to introduce the SuperStack, a collection of technologies designed to overcome the shortcomings of standard RAG pipelines.

In this blog post, we will cover the persistent problems with standard RAG pipelines, introduce the components of the SuperStack, and show where you can begin using these features.

**Problems with Standard RAG Pipelines**

**Misinterpreting Input:**

Let's start at the beginning of the pipeline - the user input. Relevance ranking models for information retrieval, like embeddings and cross-encoders, are trained on datasets of query-chunk pairs. Given a search query, they are trained to find the most relevant chunk out of a large set of possible chunks.

This means that to get optimal performance when using these models, you need your user inputs to be phrased as search queries.

Imagine a user input such as “Write a blog post about {new feature} and why it's better than the competition.”

If you embed this input and use it to search the knowledge base, you might get some good results, but you'll also return a lot of irrelevant results. For the best results, this user input would be converted into “Benefits and competitive advantage of {new feature}.”

**Lost Context in Chunking:**

Suppose you have a bunch of SEC filings in a knowledge base and you ask, “What were Apple’s key financial results in the most recent fiscal year?”

The information needed would be found in the Consolidated Statement of Operations section, which will be 5-10 chunks long. Without any connection between these chunks, the model can easily overlook relevant results, making generated output less accurate.

**Misunderstanding Text Chunks:**

When a text chunk is embedded in a standard RAG pipeline, it carries with it no context of the knowledge base or document from which it came. This can lead to an incomplete representation of the content and meaning of a given piece of text.

In the process of retrieval, this lack of context increases the rate at which irrelevant results show up in the search results.

**The SuperStack Features**

The SuperStack has three components that directly tackle the problems with standard RAG pipelines:

[**AutoQuery**](https://superpowered.ai/blog/introducing-auto-query) → Convert user inputs into well-formed search queries.

In cases where the input is long and confusing, AutoQuery can create up to three search queries to ensure the most relevant information is retrieved.

Beyond forming the query, AutoQuery also makes sure that search queries match the language of the knowledge base by translating queries into the appropriate language.

Lastly, AutoQuery handles references to previous messages by taking the conversation history as input, instead of just the most recent message. This makes implicit references to previous messages a breeze for AutoQuery to make sense of.

[**Relevant Segment Extraction (RSE)**](https://superpowered.ai/blog/introducing-relevant-segment-extraction) → Group clusters of relevant chunks into longer sections of text to provide better context to the LLM.

By allowing variable length context to be provided to the LLM, we dynamically retrieve relevant information spanning anywhere from a single paragraph to multiple pages.

The goal of RSE is to intelligently identify the section(s) of text that provide the most relevant information, without being constrained to fixed length chunks.

[**AutoContext**](https://superpowered.ai/blog/introducing-auto-context) → Automatically inject descriptive context into text chunks and embeddings, allowing the LLM to fully understand the meaning of every piece of text.

This injection of context uses both knowledge base and document-level context to give the embeddings a much more accurate and complete representation of the content and meaning of the text.

In addition to increasing the rate at which the correct information is retrieved, AutoContext also substantially reduces the rate at which irrelevant results show up in the search results. This reduces the rate at which the LLM misinterprets a piece of text in downstream chat and generation applications.

Auto context is particularly useful if you don’t already have a great idea of what is contained in the document and how it pertains to what kinds of questions may be asked of it. However, AutoContext is an additional charge that may be avoidable if the use of Chunk Headers can achieve the same result.

**Chunk Headers (AutoContext Alternative):**

Sometimes it makes sense to explicitly set the context yourself. For example, if I were to create a customer service chat bot for Superpowered AI, I would add Python SDK documentation to the knowledge base. I could certainly use auto context, but it might be more helpful to create a chunk header like:

Chunk header:

"These are the Superpowered Python SDK functions related to chat endpoints. To use any function in this document, you must first do pip install superpowered-sdk and use the functions like this:

**```python**

import superpowered

response = superpowered.<function\_name>```"

**Using the SuperStack**

This set of technologies is available to check out in both the REST API and user interface (UI), which reflects the functionality of our API. We will show the configuration for these features in both the API and UI.

**Getting Started With the API:**

pip install superpowered-sdk

**AutoQuery (API):**

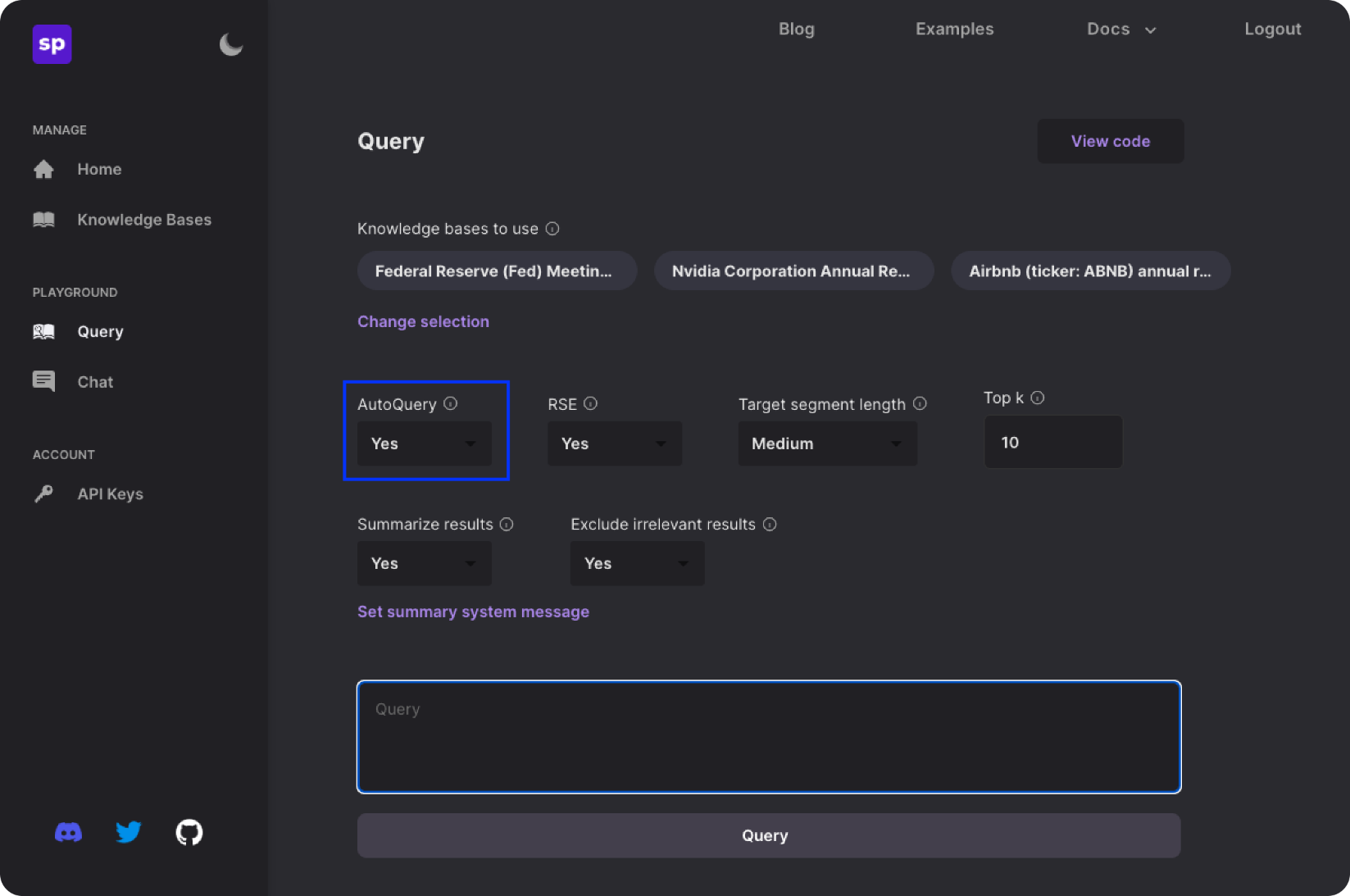
AutoQuery is used by default in the chat endpoint to synthesize recent chat history into well-structured search queries in the retrieval stage.

To use AutoQuery via the API, you can either use the /v1/knowledge\_bases/query endpoint in the REST API or the query\_knowledge\_bases() function in the Python SDK. In both cases, just set the use\_auto\_query parameter to true. To see the queries that were generated by AutoQuery, the response will be in the response JSON / dictionary in interaction.search\_queries.

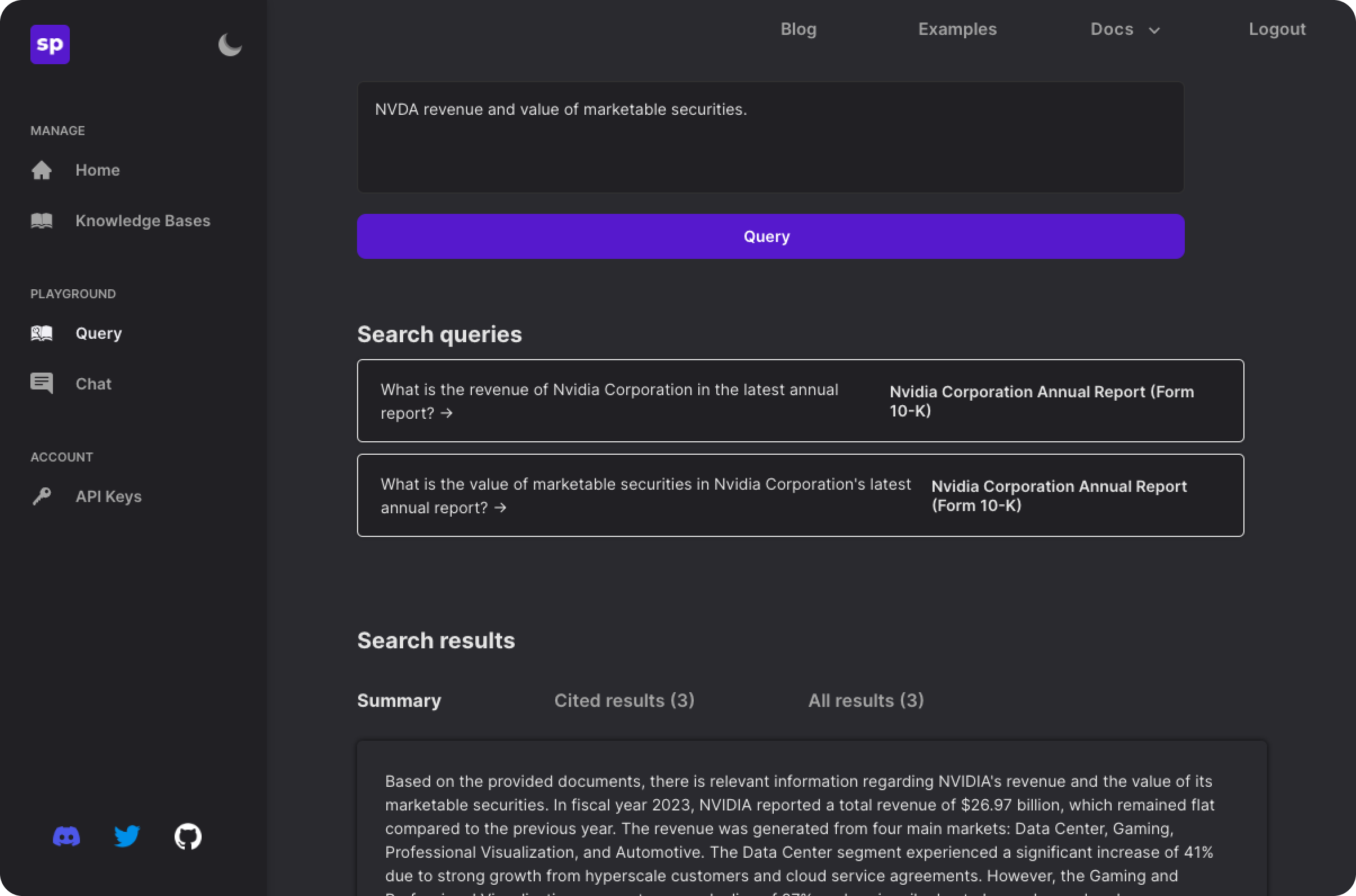
Please note that use\_auto\_query defaults to false in the query endpoint.

**AutoQuery (UI):**

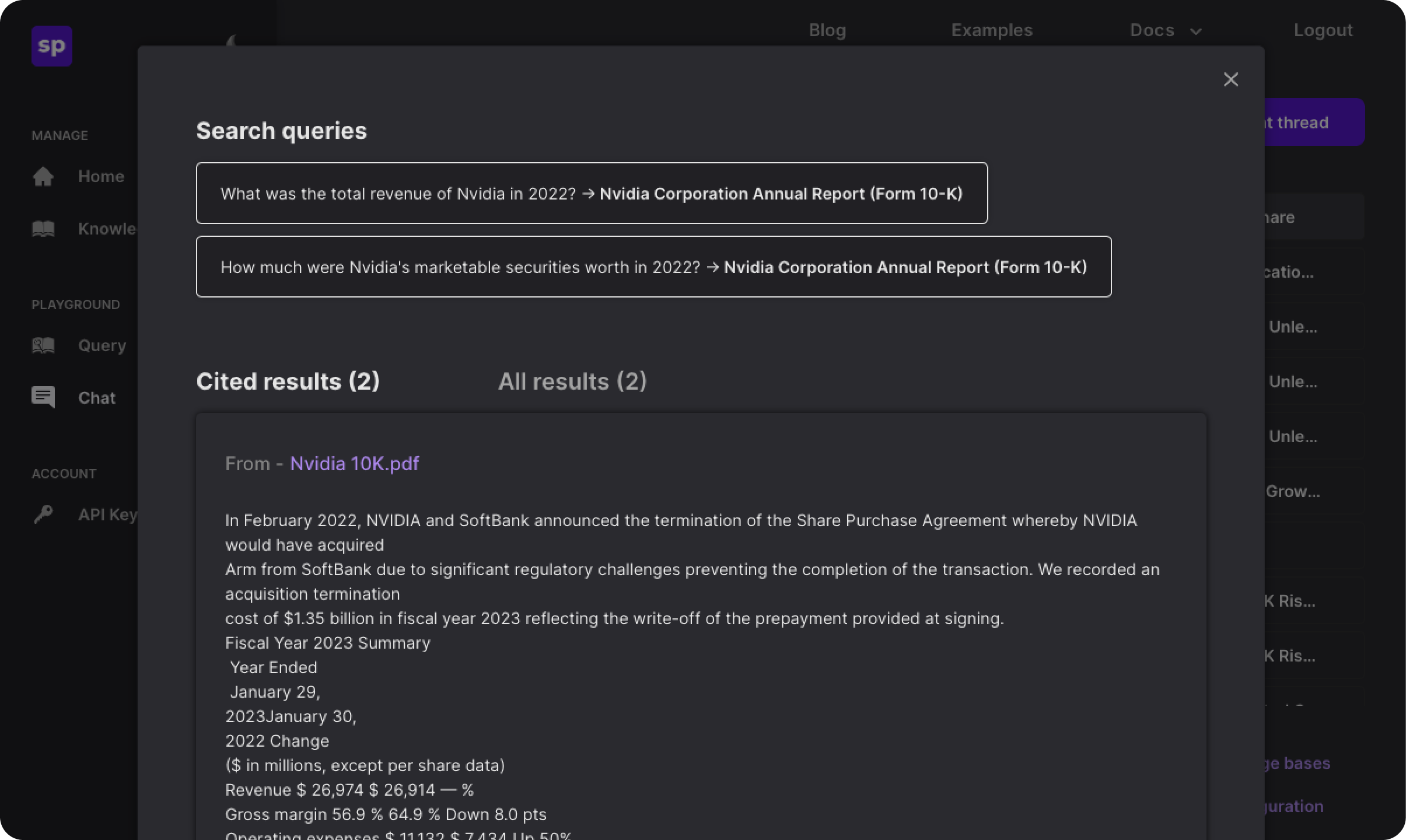
Query Endpoint: After creating a knowledge base, navigate to the Query page in the Playground and select “Yes” for AutoQuery.



When you query your knowledge base, you will see the search queries used at the top of your results.



Chat Endpoint: After creating a knowledge base, navigate to the Chat page and begin a conversation. You will see sources show up at the bottom of generated responses that used retrieval. Click “Sources” and you’ll be able to view the search queries used for your last input.



**Relevant Segment Extraction (API):**

RSE with a segment length of medium is the default behavior in both the query and chat endpoints if it is unspecified.

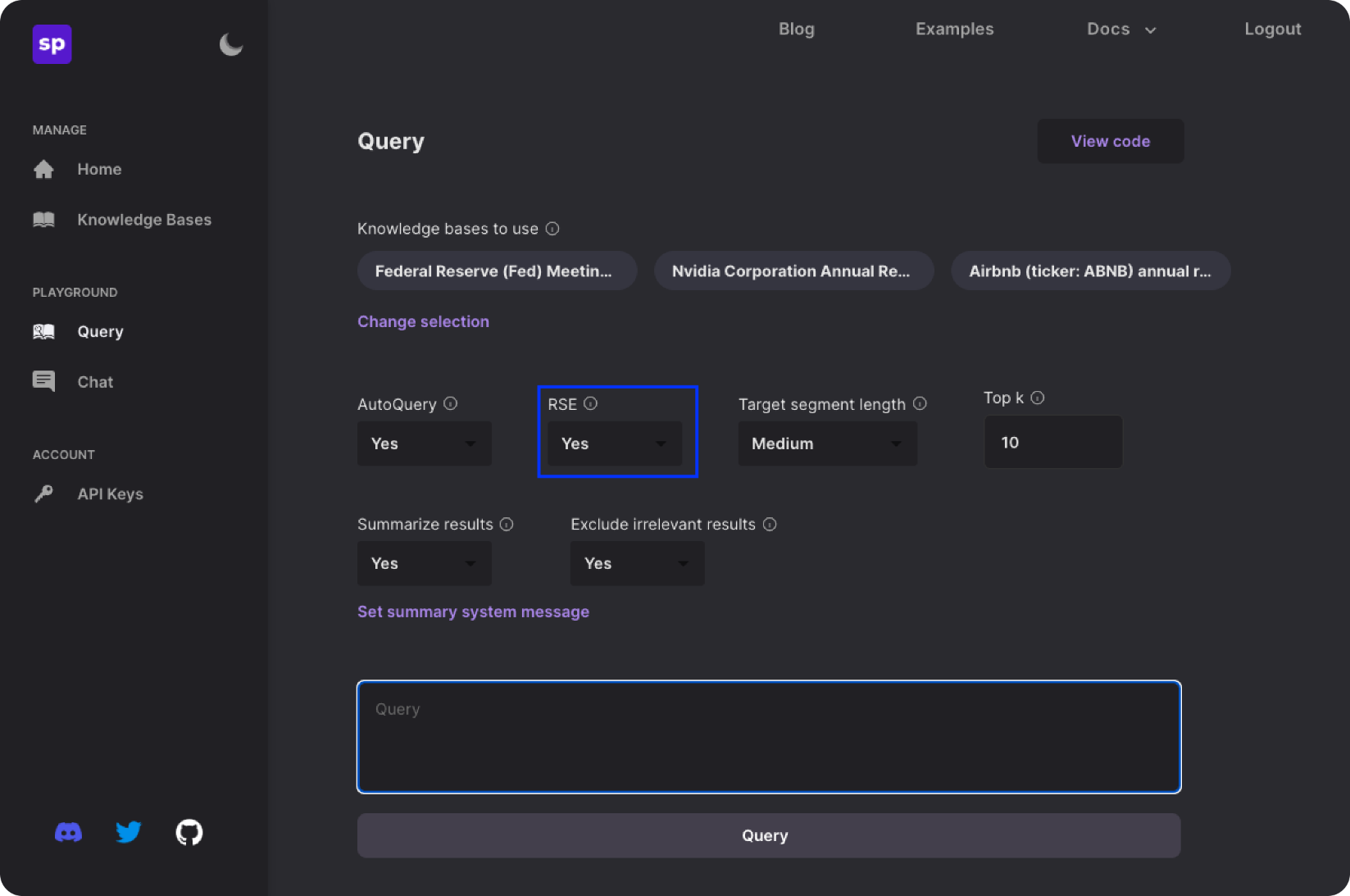
In the query endpoint, you can simply pass use\_rse as true or false.

The chat endpoint has a few more moving pieces. You must first create a “chat thread” that will contain the chat history for every conversation. Each chat thread has a series of default options. One of those options is use\_rse (which defaults to true). For all of the default options, please visit our [REST API docs](https://docs.superpowered.ai/api/rest/index.html)

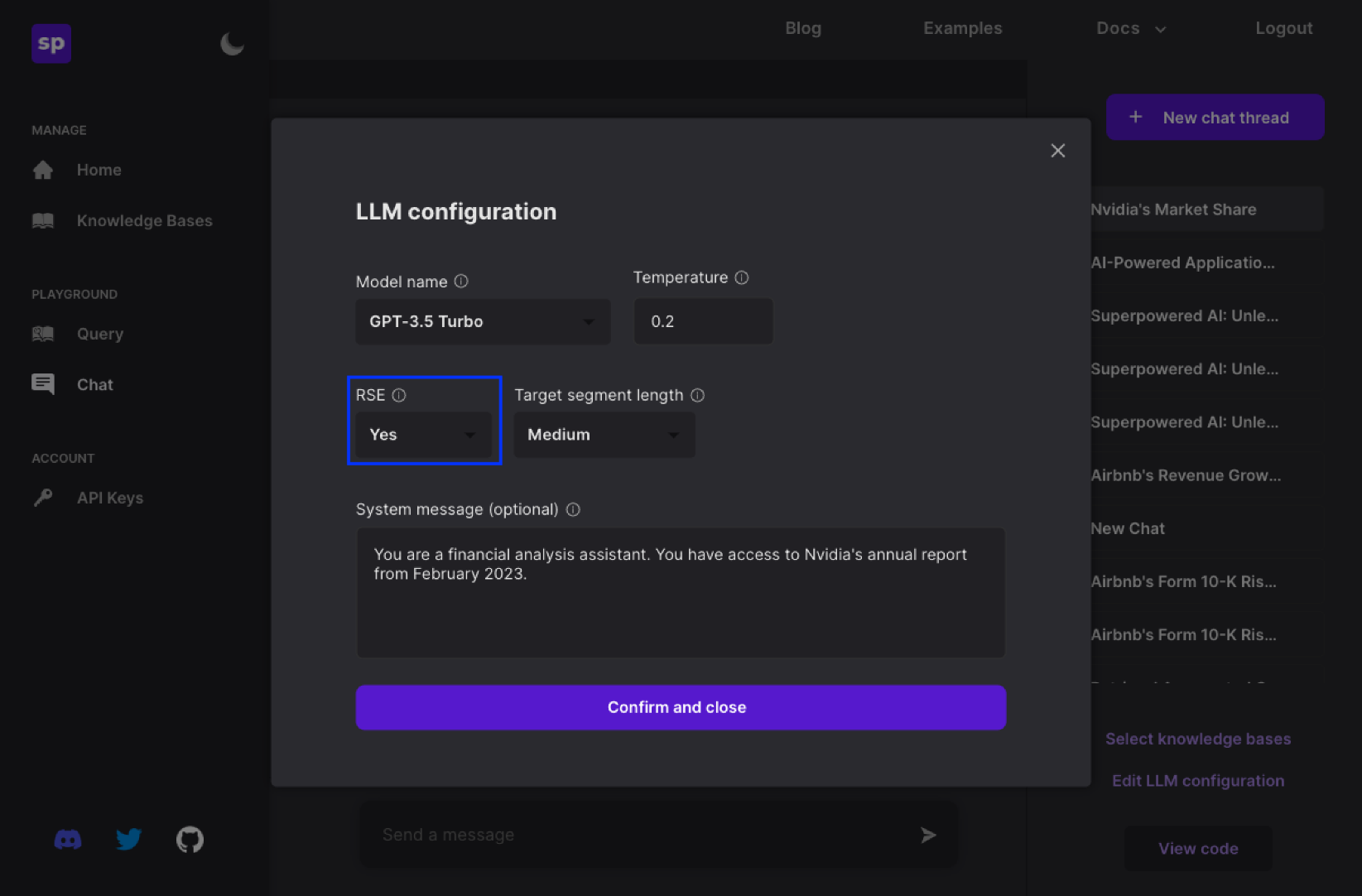
If you ever want to override the chat thread defaults, you can just pass use\_rse when calling the /v1/chat/threads/{thread\_id}/get\_response API endpoint or the get\_chat\_response() Python SDK function.

**Relevant Segment Extraction (UI):**

Query Endpoint: After creating a knowledge base, go to the Query page and set RSE to “Yes.” This will provide better context to the LLM than any individual chunk can.



Chat Endpoint: After creating a knowledge base, go to the Chat page in the Playground and select “Edit LLM Configuration.” Here you will be able to set RSE to “Yes.”



**AutoContext (API):**

Auto context is a parameter that will be passed for the various ways you can create documents in both the REST API and Python SDK.

In all of the following endpoints / functions, just pass auto\_context as true or false.

REST API:

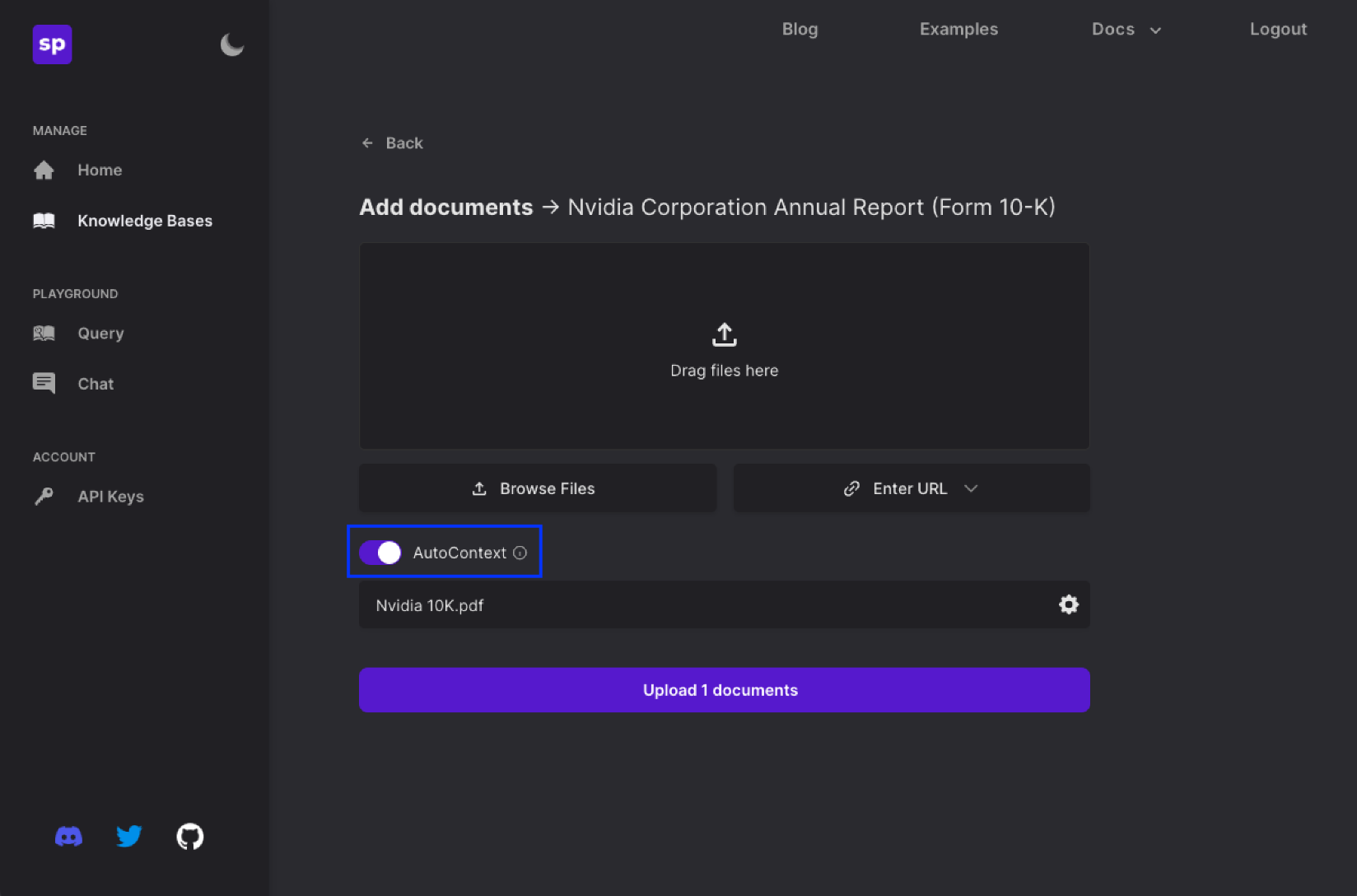
* /v1/knowledge\_bases/{knowledge\_base\_id}/documents/request\_signed\_file\_url
* /v1/knowledge\_bases/{knowledge\_base\_id}/documents/raw\_text
* /v1/knowledge\_bases/{knowledge\_base\_id}/documents/url

Python SDK:

* create\_document\_via\_file()
* create\_document\_via\_text()
* create\_document\_via\_url()

**AutoContext (UI):**

This feature is set when uploading documents to a knowledge base. Since AutoContext is a feature that gets set at the document level, it’s okay to have some documents in a knowledge base with AutoContext and some documents without it.



**THE SUPERSTACK 🦸**

The SuperStack has three components that directly tackle the problems with standard RAG pipelines:

[**AutoQuery**](https://superpowered.ai/blog/introducing-auto-query) → Convert user inputs into well-formed search queries for better retrieval results.

[**Relevant Segment Extraction (RSE)**](https://superpowered.ai/blog/introducing-relevant-segment-extraction) → Dynamically group clusters of relevant results into longer sections of contiguous text to provide better context to the LLM. This is especially useful for more complex questions, where the answer isn’t contained in a single sentence or paragraph.

[**AutoContext**](https://superpowered.ai/blog/introducing-auto-context) → Automatically inject descriptive context into text chunks and embeddings, to capture the full context of each chunk of text, reducing the likelihood of poor search results and hallucinations.

**Step 1: Text Preprocessing**

1. **Normalization**: Convert all text to a consistent format, e.g., lowercase, to avoid duplication due to case differences.
2. **Cleaning**: Remove any irrelevant content such as headers, footers, and special formatting that does not contribute to the factual content of the text.
3. **Tokenization**: Break the text into tokens or words. This can be useful for analysis and to ensure that embeddings capture word-level information.

**Step 2: Text Chunking**

1. **Section-wise Chunking**: Break the textbook into logical sections based on chapters or sub-chapters. If chapters are too long, consider further subdivision based on subheadings or a fixed number of paragraphs.
2. **Consider Interconnections**: For topics known to be interconnected, maintain pointers or references in your database that can link related sections even if they are not in the same chunk. This is crucial for complex subjects where understanding requires cross-referencing.
3. **Contextual Boundaries**: Ensure that each chunk ends at a natural stopping point to maintain context, such as the end of a paragraph or section, rather than cutting off mid-sentence.

**Step 3: Text Summarization (Optional)**

1. **Extractive Summarization**: Identify key sentences or paragraphs that represent the main points of each section. Tools like the **sentence-transformers** library can be used to rank sentences by importance based on their embeddings.
2. **Summarization Scale**: Decide on the extent of summarization based on the criticality and density of information. Highly technical sections may require finer details, while more general sections can be summarized more aggressively.

**Step 4: Generating Embeddings**

1. **Load Sentence Transformer Model**: Use models like `all-MiniLM-L6-v2