Introduction

One very common application of AI technology is Semantic Search. The goal of semantic search (which is also know as “Retrieval Augmented Generation” or RAG) is to query a body of data using not just simple matching heuristics (aka “lexical search), but by determining the intent and contextual meaning of the search query and finding matching concepts in the search target. Combining searches based on semantic similarity with Generative AI enables a very powerful “conversational” search style which is often referred to as “Chat Your Data”.

We see use cases involving users querying a knowledge base and combining their own knowledge with the real time state of the system. For example, a technician dispatched to repair a generator might use semantic search over the repair manual and technical specifications to determine how best to address an issue, given the generator’s current operating parameters. The recently added “AI Documentation Search” facility in Modelo is another example of Semantic Search.

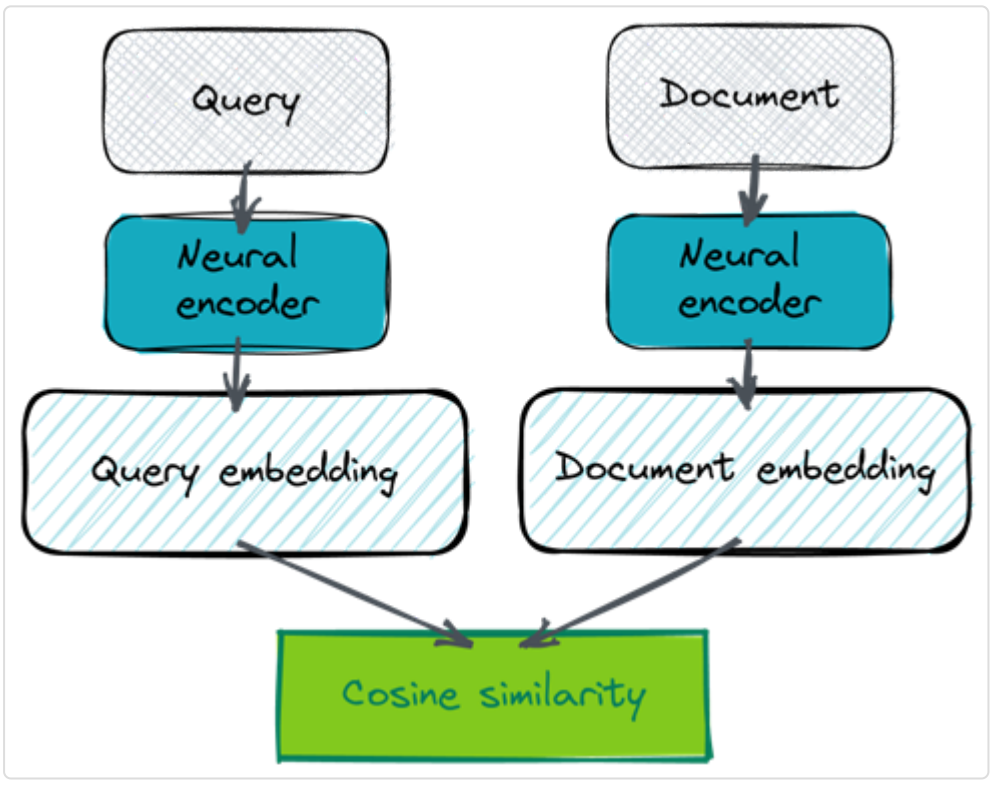
Semantic Search Architecture

Before we get into the specifics of how to enable semantic search within a Vantiq application, it will be useful to understand the general semantic search architecture. As part of this introduction it will be useful to understand a few commonly used terms/abbreviations:

- **Large Language Model** (LLM) – a type of artificial intelligence model designed to process input and perform some task. These models are trained on vast amounts of textual data to learn patterns, context, and relationships within language.
- **Embedding Model** – a type of machine learning model that transforms input data, typically text or categorical variables, into a numerical representation called embeddings. The purpose of embedding is to capture semantic relationships and similarities between the input elements.
- **Generative Model** – a type of artificial intelligence model designed to understand and generate human-like language. Generative LLMs have the ability to perform various natural language processing tasks, including text completion, translation, summarization, and question answering.

Vectors and Embeddings

The basis of semantic search is the fact that we can analyze a search target (such as a collection of documents) and use a neural network to “encode” it such that we can compare elements based on their semantic similarity. The encoded results are known as “embeddings” and they consist of a vector of floating-point numbers (the size of this vector varies depending on the neural network used to perform the encoding). The encoding works such that 2 elements whose vectors are “near” each other (based on a specific measurement such as the cosine of the angle between the vectors) are also close to each other in meaning. By applying this encoding to both the query and to the potential answers, we have a way to compare them semantically as shown below:

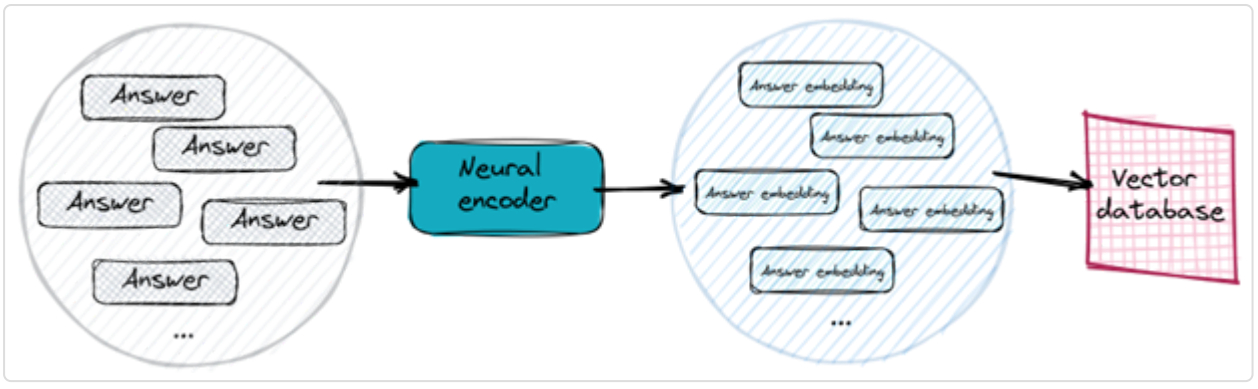


For our purposes, the “encoder” will always be an Embedding LLM. It is important to know that different models produce embeddings that must be compared in specific ways. For instance, the above example uses the cosine of the angle between the 2 vectors (aka cosine similarity). This is something you must just “know” based on the description of the model (there is nothing in the embedding vector which tells you this). Additionally, you must use the same embedding model to create both the query and document embedding. Otherwise, they likely won’t be comparable.

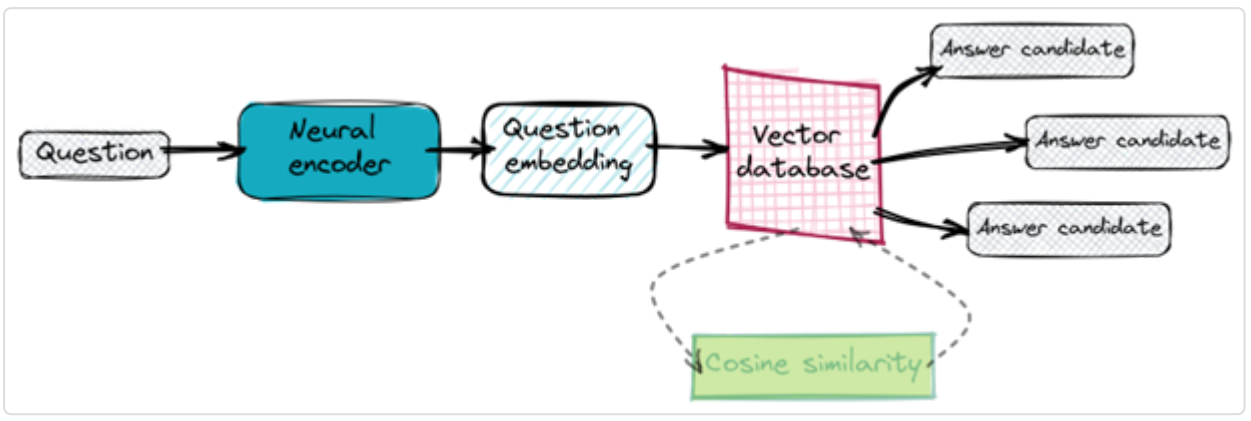
Efficiently storing and searching embeddings

Of course, we don’t want to have to continuously recreate embeddings from the documents we are searching, and we don’t want to have to perform similarity comparisons on each vector one at a time. We need a way to persistently store the embeddings once they have been created and to search those embeddings as shown above. This is the role of a Vector Database (Vector DB).

A Vector DB allows us to store the embeddings and any associated “metadata” (such as a reference to the document or the text that was encoded) and perform efficient searches using a target vector (shown as “query embedding” in the previous diagram) and optionally a filter based on the metadata values. The first step is to create embeddings for all the documents we wish to search and store those in the Vector DB as shown here:



Once that's done, we can then take any query, encode it to create an embedding and search the Vector DB to find the potential answers to our question:

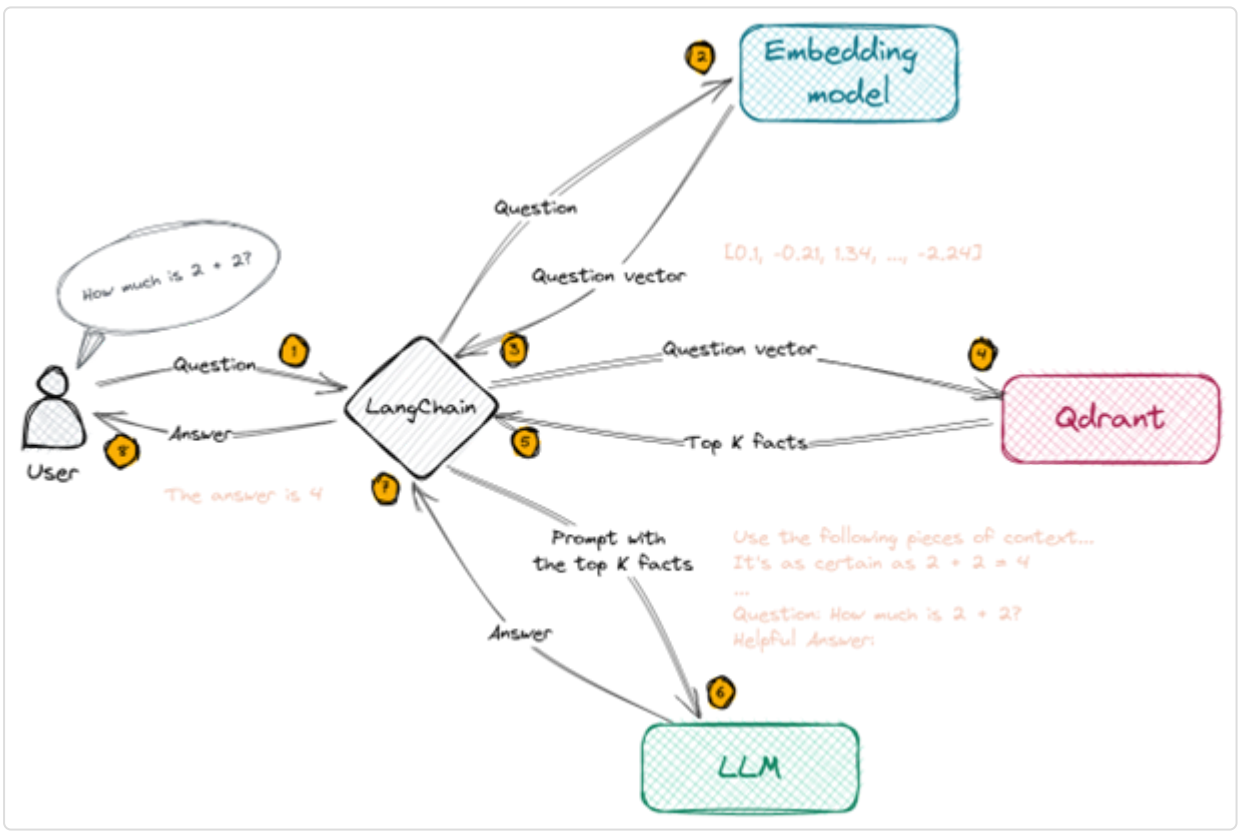


It is worth noting that when processing a document for storage in the Vector DB, we typically divide the document up into overlapping "chunks" and create embeddings for these. This improves the accuracy of the similarity queries and also reduces the amount of information retrieved (which we'll discuss in the next section).

Leveraging Generative AI models

Prior to the introduction of generative LLMs such as GPT, the process would have stopped there. We could display the search results to the user, and they could then access the original source material and read through it to (hopefully) answer their question. This is better than a purely lexical search, but still relies on the user's ability to read and understand the source material. Using a generative LLM enables us to go one step further and synthesize a more direct answer that incorporates the knowledge from candidate answers and leverages the general reasoning capabilities of the generative LLMs.

To accomplish this, we take the results of the vector similarity search (the candidate answers or facts) and format them, along with the original question, into a "prompt" which is sent to a generative LLM. The LLM is then able to formulate a response based on its general knowledge and the specific facts that it has been presented. The user is presented with this result (along with references to the source material used). It turns out that in most cases the response generated by the LLM is more useful and easier to understand than the underlying source material. This sequence is depicted here (Qdrant is a Vector DB vendor and Langchain is an LLM application framework, neither is specifically required, this is just for illustration purposes):



We can also allow the user to ask "follow up" questions to help clarify any aspects of the response that were unclear or to explore a related topic based on that initial response. This is accomplished by performing the same process using the new question (to obtain a new set of candidate answers) and presenting both the original interaction (or a summary of it) and the new question and facts to the LLM.

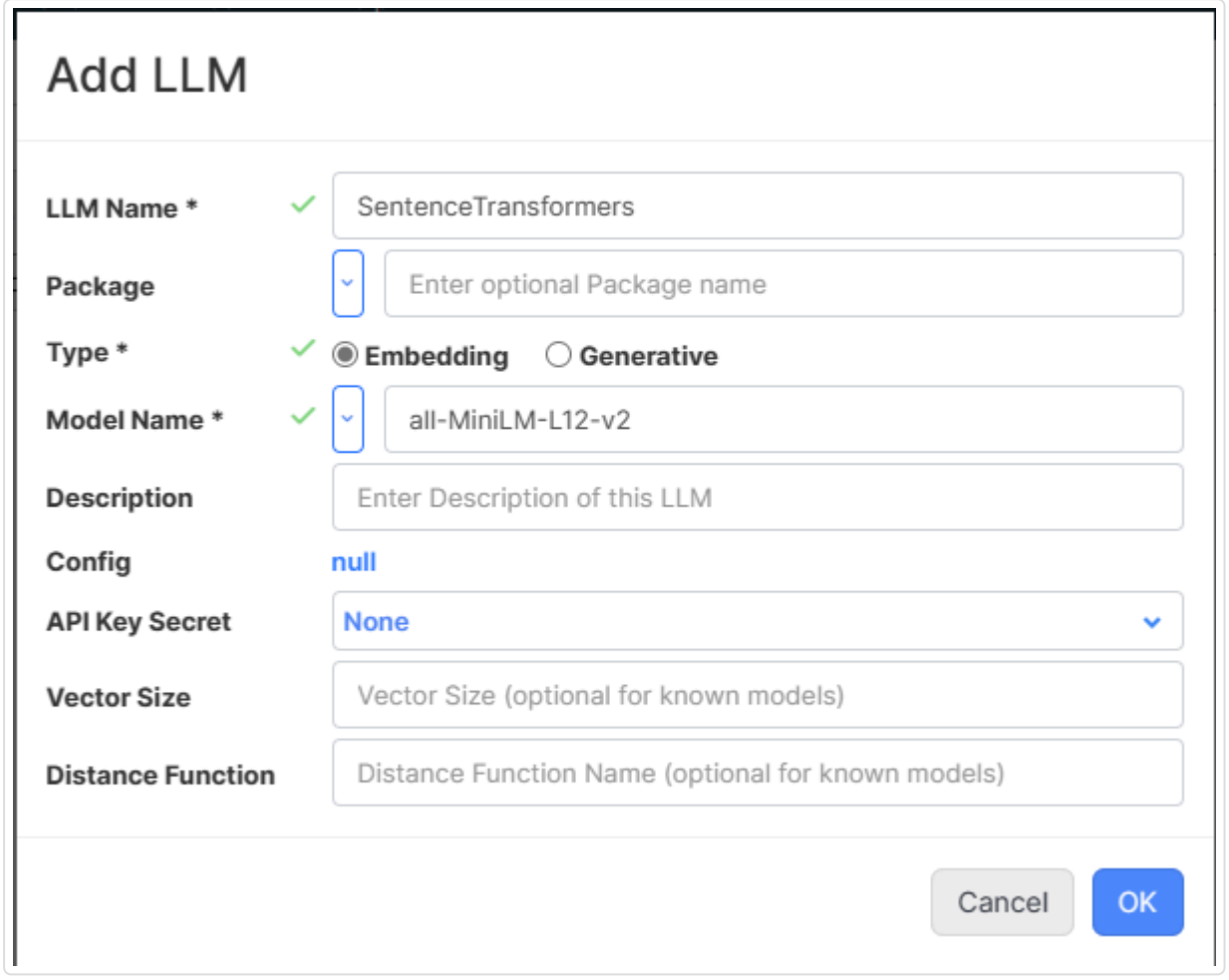
While this sounds great, it is worth noting that one limitation of the current generative LLMs is that too much information can be a bad thing. All the LLMs have hard limits on the amount of data they can accept measured in something called “tokens” (there are roughly 2-3 tokens per word). In addition, the more information the LLM receives the more likely it is to infer semantic connections that don’t exist. This can lead to it fabricating responses that simply are not true (a phenomenon known as “hallucinating ”). For example, when trying to answer a question based on the contents of a document, it is often better (or even necessary) to provide the LLM with only specific sections of the document, rather than the whole thing. This is why we divide the documents into “chunks” as they are loaded into the database (as mentioned earlier). Sending the chunks to the LLM results in both reduced token sizes and a more focused context (which typically yields better results).

Semantic Search in Vantiq

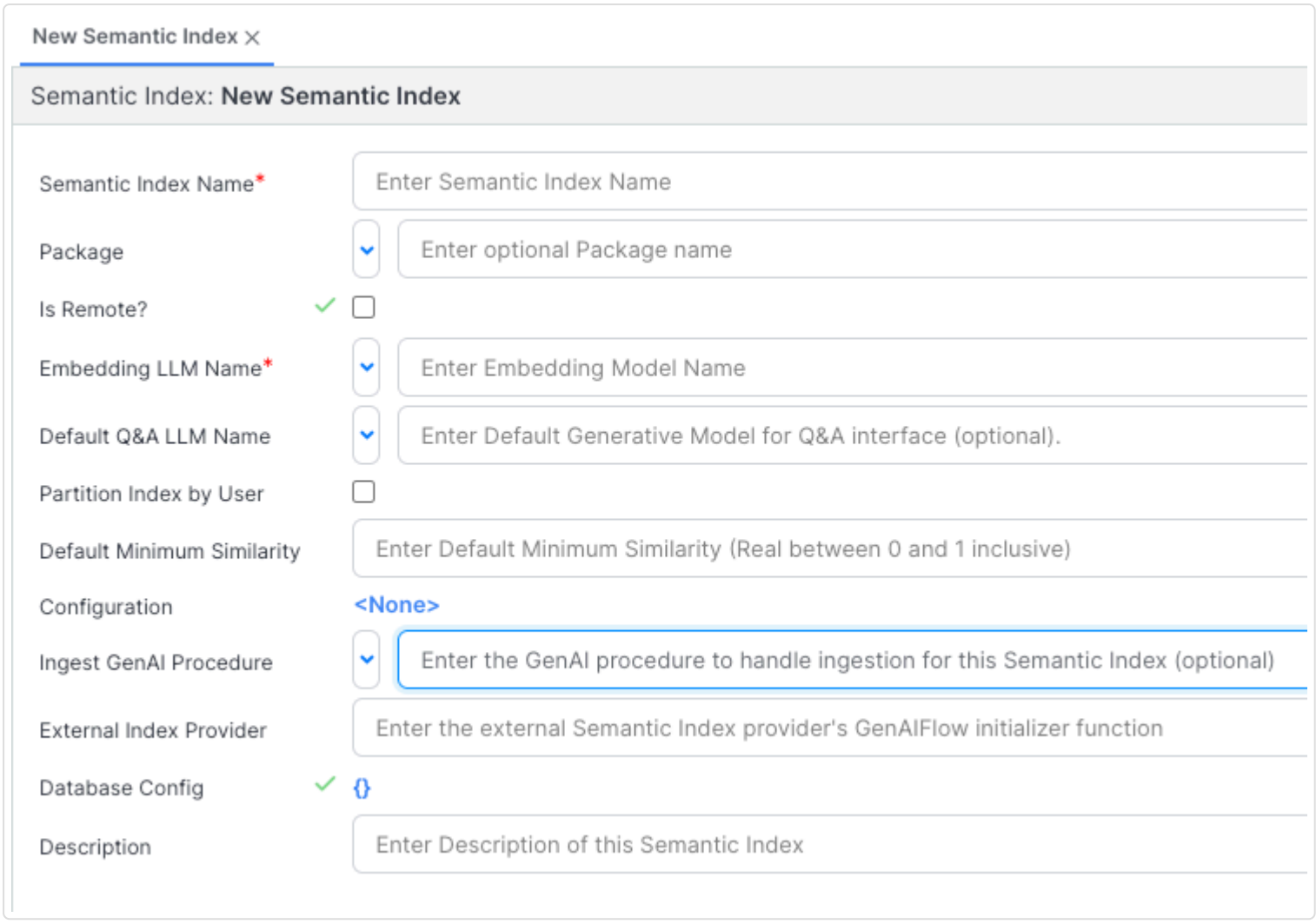
Now that we have a better understanding of how semantic search works in general, let’s take a look at how to use it in the Vantiq platform.

Creating a Semantic Index

The first step is to load the documents that you wish to search into a Semantic Index. To create a semantic index, you must first create an Embedding LLM which will be used to create the embeddings for the index’s content (once at least one such LLM has been created, it can be used by as many semantic indexes as needed). To create a new embedding model use the **Add…LLMs** menu item to bring up the list of available LLMs and then use the *+New* button to bring up the LLM creation pane:

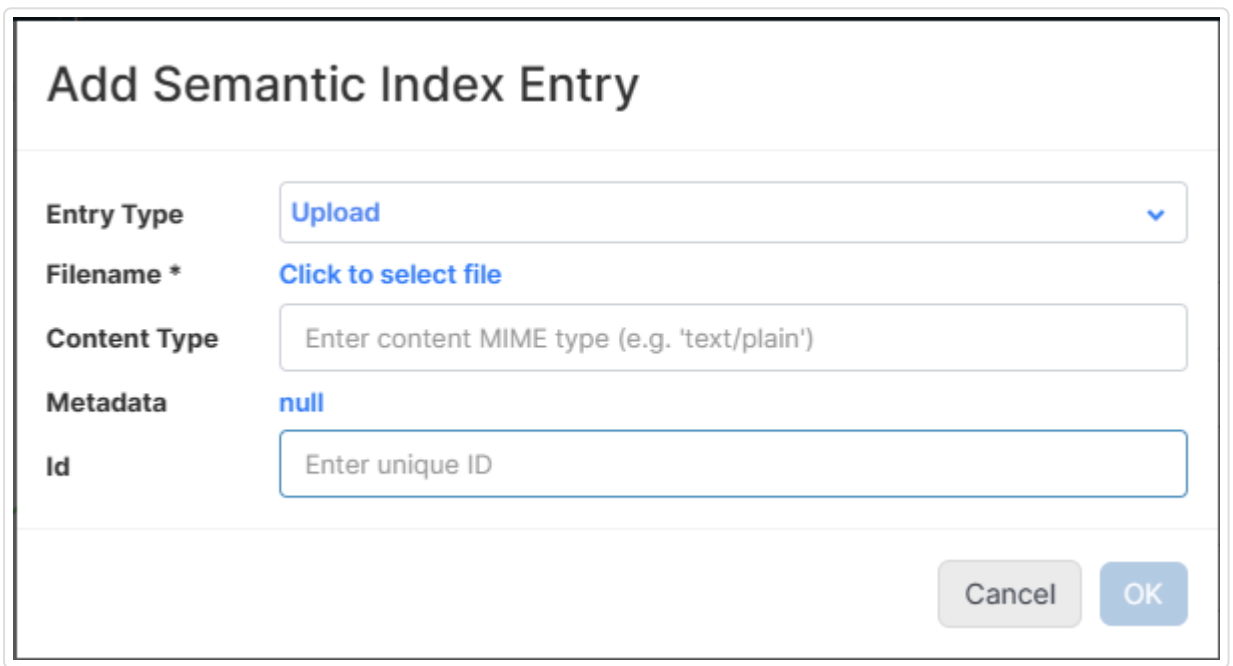


Once that’s done, use the **Add…Semantic Indexes** menu item to bring up the list of available Semantic Indexes and then use the *+New* button to bring up the Semantic Index creation pane:



Adding Semantic Index Entries

Now that you have a semantic index, the next step is to load it with your documents. To do this click on the name of the semantic index to bring up its pane and click on the *+New* button under “Semantic Index Entries” to launch the “Add Semantic Index Entry” dialog:



If the document you want to load is located on your computer, you can use the *Upload* option to load it into the index. The *Remote* option is used to load documents via a URL and *Resource* is used when the content to be loaded is in a Vantiq Document. The *Embedded* option is only really useful for quick testing since it requires you to type in the content you wish to load.

This process should be repeated for each of the documents you wish to load into the index. Semantic search works best when the entries of the index have a high degree of “relatedness”, so it is best to refrain from loading documents from different domains into the same index (having one index for each domain would be better).

Partitioning

When defining a semantic index there is an option to partition it by user. When this option is enabled, each user will have a separate view of the entries and underlying point data. As a user adds entries, they will be visible only to themselves and semantic searches will only consider entries that a given user can see. It is effectively providing each user a logically separate index, while keeping the management of the data confined to managing a single index. This can be useful when you want to isolate semantic index data to the user that added it without forcing each to add a separate index of their own.

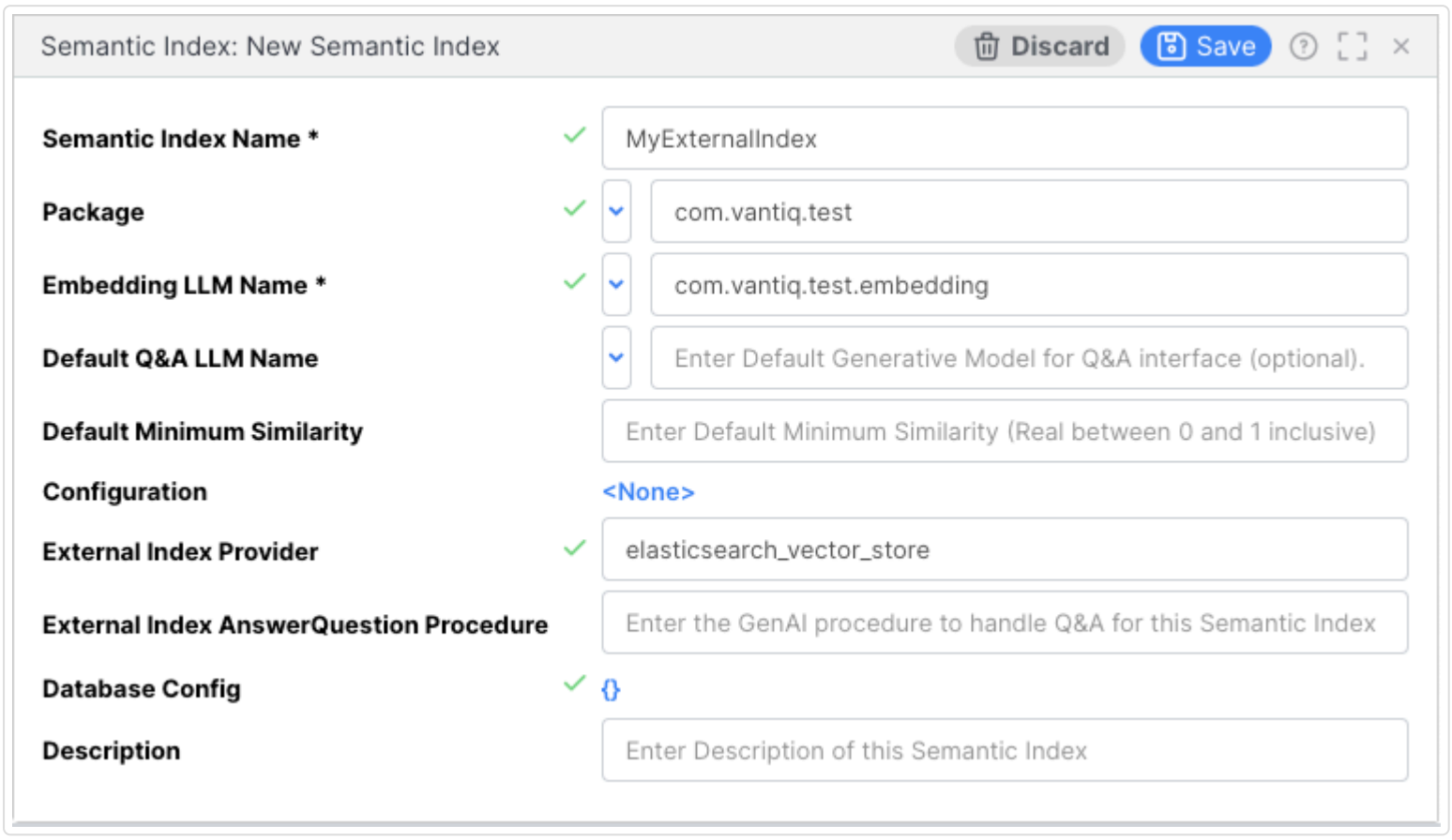
Performing Semantic Search

Now that you have a semantic index, you can make use of it to perform semantic search in your applications. The easiest way to do this is to use the Answer Question Activity Pattern. This allows you to pose a question to a generative LLM using the context found in a given index. This interaction can be a one-off or part of a larger LLM conversation (for more details on conversations see Conversation Memory).

Another way to perform a semantic search is by using the SemanticSearch system service. The `answerQuestion` procedure performs the end to end semantic search, returning the answer synthesized by a generative LLM. The `similaritySearch` procedure performs only the Vector DB query and returns the N most similar “chunks” from the database.

External Semantic Indexes

In some cases, you may already have a semantic index set up in an external system (such as Elasticsearch) containing your domain specific content. You can define a semantic index resource that that points to the external system and then use that index in the same way you would use a Vantiq managed index.



There are three pieces to specifying an external index:

- **External Provider** (required) - The Vantiq GenAI Flow service must know how to instantiate the client to talk to the external system. In the UI you provide the name of the function in the GenAI service connector docker image that will, when invoked, result in a VectorStore instance. In the case of Elasticsearch, the default service connector docker image contains the function: `elasticsearch_vector_store` which establishes a connection to an Elasticsearch instance and returns a VectorStore to handle semantic searches using the configured index.
- **Database Config** - Often the external system’s client will require configuration in order to establish a connection. This configuration is provided in the form of a JSON object. The exact contents will depend on the semantic index provider. For example, in the case of Elasticsearch, the configuration object contains, among other things, the host name for the Elasticsearch instance, the network port and credentials. The database config supports the `@secrets` notation allowing you to configure sensitive information such as passwords or API keys in a secure manner. An example database config for Elasticsearch might look like this:

```
{
  "host": "elastic.co.cloud.ep",
  "port": 443,
  "user": "elastic",
  "password": "@secrets(elasticsearch_password)"
}
```

In the above the client will connect to the cloud endpoint using the given username and password. Another option is to configure an API key: `"apiKey": "@secrets(elasticsearch_api_key)"`. In both bases the Vantiq server will replace the `@secrets` directive with the actual value of the secret at runtime. Additionally, Elasticsearch requires configuration in order to run semantic search queries. Specifically, it must know the name of the index as well as the vector and text fields defined within that index. The complete configuration would include these fields in addition to those needed for connection:

```
  "indexName": "PumpWorks610",
  "textField": "paragraph",
  "vectorField": "paragraph-vector",
```

The specific fields will depend on your index definition and content ingestion process. In general the database config is a grab-bag of configuration information needed by the semantic search engine client.

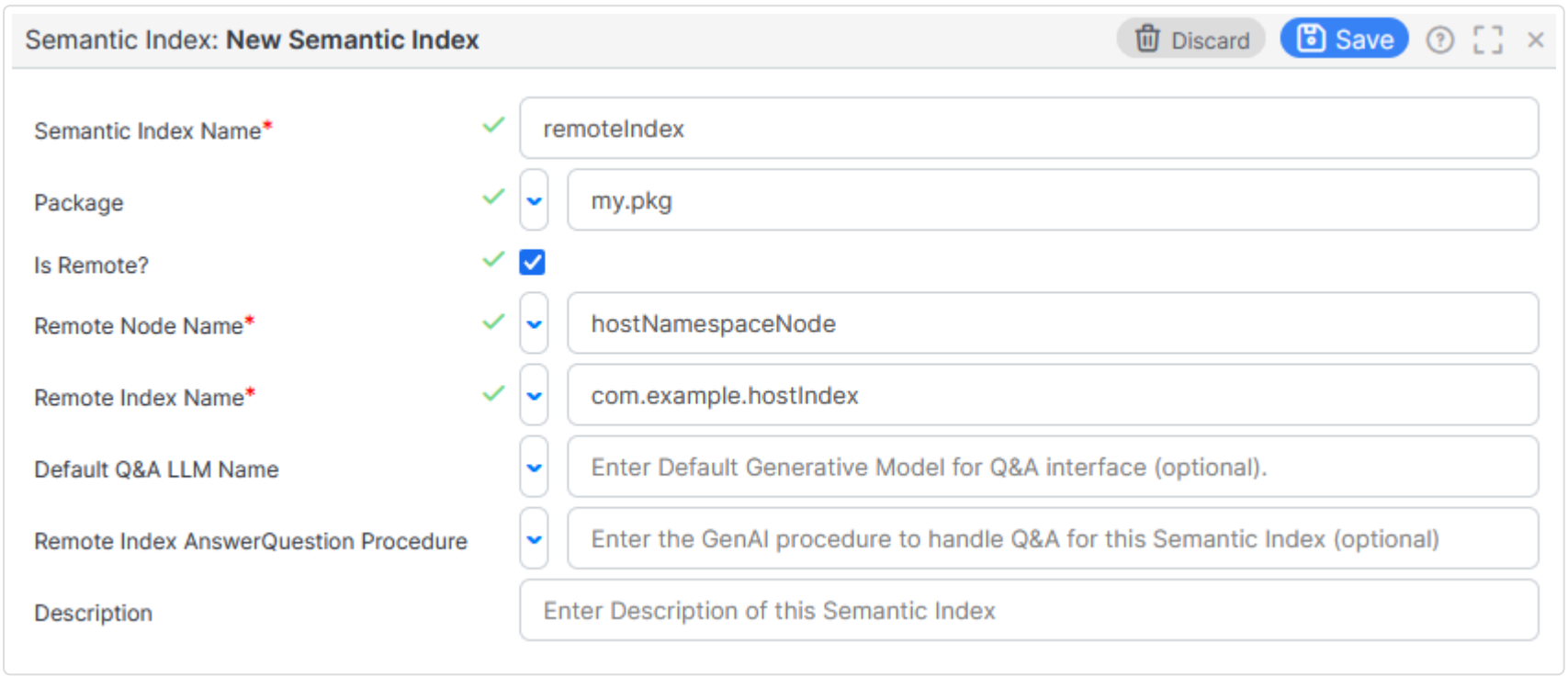
- **AnswerQuestion Procedure** - External indexes can only be instantiated in the context of a GenAI Flow. Therefore, if you want to use the `AnswerQuestion` Activity Pattern with the external index, you must provide the name of a GenAI procedure that provides the implementation. The input of the GenAI procedure will be an Object of the form `{"question": <provided question string>}` and the output an Object of the form `{"answer": <answer string>}`. The recommendation is to use the GenAI Flow Builder to construct the implementation using the external index as a resource, but the implementation can be any procedure definition. If no procedure is provided, the `AnswerQuestion` Activity Pattern will not be available for use with the external index and will return an error if attempted.

Content ingestion for external indexes is handled outside the purview of the Vantiq system. As a result there is no way within the Vantiq UI to add or remove entries for an external index. Further, semantic index dump and load are not supported.

Remote Semantic Indexes

It can be inconvenient to repeatedly upload semantic index contents for each new namespace. A remote semantic index allows you to use an index from another namespace while mostly treating it as if its contents are local.

To create a remote semantic index, you need a node to the host namespace. The token used for the node is expected to be user-level. Then, when creating the semantic index, check the box for “Is Remote?” and provide the name of the node and of the semantic index in the host namespace.



This index can be used in GenAI Flows in the same way as a local semantic index. However, you will need to create and set an AnswerQuestion Procedure to use the `AnswerQuestion` Activity Pattern or built-in procedure. The requirements for this are the same as for the AnswerQuestion Procedure of external semantic indexes.

You also cannot add, remove, or view entries for remote semantic indexes. The host namespace must be the one to manage entries. If entry management is a requirement for your design, you should instead create an Agent Service that manages the entries and publish the service to a catalog.