PineconeMemory Documentation

The `PineconeMemory` class provides a robust interface for integrating Pinecone-based Retrieval-Augmented Generation (RAG) systems. It allows for adding documents to a Pinecone index and querying the index for similar documents. The class supports custom embedding models, preprocessing functions, and other customizations to suit different use cases.

Parameters

Parameter	Type	Default	Description
`api_key`	`str`	-	Pinecone API key.
`environment`	`str`	-	Pinecone environment.
`index_name`	`str`	-	Name of the Pinecone index to use.
`dimension`	`int`	`768`	Dimension of the document embeddings.
`embedding_model`	`Optional[Any]`	`None`	Custom embedding model. Defaults to `SentenceTransformer('all-MiniLM-L6-v2')`.
`embedding_function`	`Optional[Callable[[str], List[float]]]`	`None`	Custom

embedding function. Defaults to `_default_embedding_function`. |

| `preprocess_function`| `Optional[Callable[[str], str]]` | `None` | Custom preprocessing function. Defaults to `_default_preprocess_function`. |

| `postprocess_function`| `Optional[Callable[[List[Dict[str, Any]]], List[Dict[str, Any]]]]`| `None` | Custom postprocessing function. Defaults to `_default_postprocess_function`. |

| `metric` | `str` | `'cosine'` | Distance metric for Pinecone index. |

| `pod_type` | `str` | `'p1'` | Pinecone pod type. |

| `namespace` | `str` | `''` | Pinecone namespace. |

| `logger_config` | `Optional[Dict[str, Any]]` | `None` | Configuration for the logger. Defaults to logging to `rag_wrapper.log` and console output. |

Methods

`_setup_logger`

```python
def _setup_logger(self, config: Optional[Dict[str, Any]] = None)
```

Sets up the logger with the given configuration.

`_default_embedding_function`

```python
def _default_embedding_function(self, text: str) -> List[float]
```

Generates embeddings using the default SentenceTransformer model.

`_default_preprocess_function`

```python
def _default_preprocess_function(self, text: str) -> str
```

Preprocesses the input text by stripping whitespace.

`_default_postprocess_function`

```python
def _default_postprocess_function(self, results: List[Dict[str, Any]]) -> List[Dict[str, Any]]
```

Postprocesses the query results.

`add`

Adds a document to the Pinecone index.

Parameter	Type	Default	Description
`doc`	`str`	-	The document to be added.
`metadata`	`Optional[Dict[str, Any]]`	`None`	Additional metadata for the document.

`query`

Queries the Pinecone index for similar documents.

Parameter	Type	Default	Description
`query`	`str`	-	The query string.
`top_k`	`int`	`5`	The number of top results to return.
`filter`	`Optional[Dict[str, Any]]`	`None`	Metadata filter for the query.

Usage

The `PineconeMemory` class is initialized with the necessary parameters to configure Pinecone and the embedding model. It supports a variety of custom configurations to suit different needs.

Example

```python
from swarms_memory import PineconeMemory
```

```
# Initialize PineconeMemory
memory = PineconeMemory(
    api_key="your-api-key",
    environment="us-west1-gcp",
    index_name="example-index",
    dimension=768
)
```

Adding Documents

Documents can be added to the Pinecone index using the `add` method. The method accepts a document string and optional metadata.

Example

```python
doc = "This is a sample document to be added to the Pinecone index."
metadata = {"author": "John Doe", "date": "2024-07-08"}

memory.add(doc, metadata)
```

Querying Documents

The `query` method allows for querying the Pinecone index for similar documents based on a query

string. It returns the top `k` most similar documents.

Example

```python
query = "Sample query to find similar documents."
results = memory.query(query, top_k=5)

for result in results:
    print(result)
```

Additional Information and Tips

Custom Embedding and Preprocessing Functions

Custom embedding and preprocessing functions can be provided during initialization to tailor the document processing to specific requirements.

Example

```python
def custom_embedding_function(text: str) -> List[float]:
    # Custom embedding logic
    return [0.1, 0.2, 0.3]
```

```
def custom_preprocess_function(text: str) -> str:
    # Custom preprocessing logic
    return text.lower()

memory = PineconeMemory(
    api_key="your-api-key",
    environment="us-west1-gcp",
    index_name="example-index",
    embedding_function=custom_embedding_function,
    preprocess_function=custom_preprocess_function
)
```

Logger Configuration

The logger can be configured to suit different logging needs. The default configuration logs to a file and the console.

Example

```python
logger_config = {
    "handlers": [
        {"sink": "custom_log.log", "rotation": "1 MB"},
        {"sink": lambda msg: print(msg, end="")},
    ]
```

}

memory = PineconeMemory(
    api_key="your-api-key",
    environment="us-west1-gcp",
    index_name="example-index",
    logger_config=logger_config
)
```

## References and Resources

- [Pinecone Documentation](https://docs.pinecone.io/)
- [SentenceTransformers Documentation](https://www.sbert.net/)
- [Loguru Documentation](https://loguru.readthedocs.io/en/stable/)

For further exploration and examples, refer to the official documentation and resources provided by Pinecone, SentenceTransformers, and Loguru.

This concludes the detailed documentation for the `PineconeMemory` class. The class offers a flexible and powerful interface for leveraging Pinecone's capabilities in retrieval-augmented generation systems. By supporting custom embeddings, preprocessing, and postprocessing functions, it can be tailored to a wide range of applications.
```