FAISSDB: Documentation

The `FAISSDB` class is a highly customizable wrapper for the FAISS (Facebook AI Similarity Search) library, designed for efficient similarity search and clustering of dense vectors. This class facilitates the creation of a Retrieval-Augmented Generation (RAG) system by providing methods to add documents to a FAISS index and query the index for similar documents. It supports custom embedding models, preprocessing functions, and other customizations to fit various use cases.

Parameters

Parameter	Type	Default	Description
`dimension`	`int`	`768`	Dimension of the document embeddings.
`index_type`	`str`	`'Flat'`	Type of FAISS index to use (`'Flat'` or `'IVF'`).
`embedding_model`	`Optional[Any]`	`None`	Custom embedding model.
`embedding_function`	`Optional[Callable[[str], List[float]]]`	`None`	Custom function to generate embeddings from text.
`preprocess_function`	`Optional[Callable[[str], str]]`	`None`	Custom function to preprocess text before embedding.
`postprocess_function`	`Optional[Callable[[List[Dict[str, Any]]], List[Dict[str, Any]]]]`	`None`

Custom function to postprocess the results. |

| `metric` | `str` | `'cosine'` | Distance metric for FAISS index (`'cosine'` or `'l2'`). |

| `logger_config` | `Optional[Dict[str, Any]]` | `None` | Configuration for the logger. |

Methods

`__init__`

Initializes the FAISSDB instance, setting up the logger, creating the FAISS index, and configuring custom functions if provided.

`add`

Adds a document to the FAISS index.

Parameters

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

Example Usage

```python
db = FAISSDB(dimension=768)
db.add("This is a sample document.", {"category": "sample"})
```

`query`

Queries the FAISS index for similar documents.

Parameters

Parameter	Type	Default	Description
`query`	`str`	None	The query string.
`top_k`	`int`	`5`	The number of top results to return.

Returns

Type	Description
`List[Dict[str, Any]]`	A list of dictionaries containing the top_k most similar documents.

Example Usage

```python
results = db.query("What is artificial intelligence?")
```

```
for result in results:
    print(f"Score: {result['score']}, Text: {result['metadata']['text']}")
```

Internal Methods

`_setup_logger`

Sets up the logger with the given configuration.

Parameters

Parameter	Type	Default	Description
`config`	`Optional[Dict[str, Any]]`	None	Configuration for the logger.

`_create_index`

Creates and returns a FAISS index based on the specified type and metric.

Parameters

Parameter	Type	Default	Description
`index_type`	`str`	'Flat'	Type of FAISS index to use.
`metric`	`str`	'cosine'	Distance metric for FAISS index.

Returns

Type	Description
`faiss.Index`	FAISS index instance.

`_default_embedding_function`

Default embedding function using the SentenceTransformer model.

Parameters

Parameter	Type	Default	Description
`text`	`str`	None	The input text to embed.

Returns

Type	Description
`List[float]`	Embedding vector for the input text.

`_default_preprocess_function`

Default preprocessing function.

Parameters

Parameter	Type	Default	Description
`text`	`str`	None	The input text to preprocess.

Returns

Type	Description
`str`	Preprocessed text.

`_default_postprocess_function`

Default postprocessing function.

Parameters

Parameter	Type	Default	Description
`results`	`List[Dict[str, Any]]`	None	The results to postprocess.

Returns

| Type | Description |

|------|-------------------------|

| `List[Dict[str, Any]]` | Postprocessed results. |

Usage Examples

Example 1: Basic Usage

```python
# Initialize the FAISSDB instance
db = FAISSDB(dimension=768, index_type="Flat")

# Add documents to the FAISS index
db.add("This is a document about AI.", {"category": "AI"})
db.add("Python is great for data science.", {"category": "Programming"})

# Query the FAISS index
results = db.query("Tell me about AI")
for result in results:
    print(f"Score: {result['score']}, Text: {result['metadata']['text']}")
```

Example 2: Custom Functions

```python
from transformers import AutoTokenizer, AutoModel
import torch
```

```
# Custom embedding function using a HuggingFace model
def custom_embedding_function(text: str) -> List[float]:
    tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
    model = AutoModel.from_pretrained("bert-base-uncased")
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().tolist()
    return embeddings

# Custom preprocessing function
def custom_preprocess(text: str) -> str:
    return text.lower().strip()

# Custom postprocessing function
def custom_postprocess(results: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
    for result in results:
        result["custom_score"] = result["score"] * 2  # Example modification
    return results

# Initialize the FAISSDB instance with custom functions
db = FAISSDB(
    dimension=768,
    index_type="Flat",
    embedding_function=custom_embedding_function,
```

```
    preprocess_function=custom_preprocess,
    postprocess_function=custom_postprocess,
    metric="cosine",
    logger_config={
        "handlers": [
            {"sink": "custom_faiss_rag_wrapper.log", "rotation": "1 GB"},
            {"sink": lambda msg: print(f"Custom log: {msg}", end="")}
        ],
    },
)

# Add documents to the FAISS index
db.add("This is a document about machine learning.", {"category": "ML"})
db.add("Python is a versatile programming language.", {"category": "Programming"})

# Query the FAISS index
results = db.query("Explain machine learning")
for result in results:
                print(f"Score: {result['score']}, Custom Score: {result['custom_score']}, Text: {result['metadata']['text']}")
```

Additional Information and Tips

- Ensure that the dimension of the document embeddings matches the dimension specified during the initialization of the FAISSDB instance.

- Use custom embedding functions to leverage domain-specific models for generating embeddings.
- Custom preprocessing and postprocessing functions can help tailor the text processing and

result formatting to specific needs.
- FAISS supports various types of indices; choose the one that best fits the application requirements (e.g., `Flat` for brute-force search, `IVF` for faster search with some accuracy trade-off).
- Properly configure the logger to monitor and debug the operations of the FAISSDB instance.

References and Resources

- [FAISS GitHub Repository](https://github.com/facebookresearch/faiss)
- [Sentence Transformers Documentation](https://www.sbert.net/)
- [Loguru Documentation](https://loguru.readthedocs.io/en/stable/)
- [HuggingFace Transformers](https://huggingface.co/transformers/)

By following this documentation, users can effectively utilize the `FAISSDB` class for various similarity search and document retrieval tasks, customizing it to their specific needs through the provided hooks and functions.