"""

ChromaDB vector store implementation.

This module provides functionality for storing and retrieving document

embeddings using ChromaDB.

"""

from typing import List, Dict, Any, Optional

from pathlib import Path

import logging

import shutil

from langchain\_community.vectorstores import Chroma

from langchain.schema import Document

from .base import BaseVectorStore

from ..embeddings.base import BaseEmbeddingModel

from ..utils.exceptions import VectorStoreError

from ..utils.logging\_config import get\_logger

logger = get\_logger(\_\_name\_\_)

class ChromaVectorStore(BaseVectorStore):

"""

Vector store implementation using ChromaDB.

This class handles document storage, retrieval, and persistence

using ChromaDB as the backend.

"""

def \_\_init\_\_(

self,

embedding\_model: BaseEmbeddingModel,

persist\_directory: Optional[str] = None,

collection\_name: str = "documents",

collection\_metadata: Optional[Dict[str, Any]] = None

):

"""

Initialize the ChromaDB vector store.

Args:

embedding\_model: Embedding model to use

persist\_directory: Directory to persist the database

collection\_name: Name of the collection

collection\_metadata: Additional metadata for the collection

"""

self.embedding\_model = embedding\_model

self.persist\_directory = persist\_directory

self.collection\_name = collection\_name

self.collection\_metadata = collection\_metadata or {}

self.\_vectorstore = None

logger.info(

"Initialized ChromaVectorStore",

extra={

"persist\_directory": persist\_directory,

"collection\_name": collection\_name

}

)

def add\_documents(

self,

documents: List[Document],

embeddings: Optional[Any] = None,

ids: Optional[List[str]] = None

) -> None:

"""

Add documents to the vector store.

Args:

documents: Documents to add

embeddings: Pre-computed embeddings (if None, will be computed)

ids: Optional IDs for the documents

Raises:

VectorStoreError: If adding documents fails

"""

logger.info(f"Adding {len(documents)} documents to vector store")

if not documents:

logger.warning("No documents to add")

return

try:

if self.\_vectorstore is None:

# Create new vector store

self.\_vectorstore = Chroma.from\_documents(

documents=documents,

embedding=self.embedding\_model,

persist\_directory=self.persist\_directory,

collection\_name=self.collection\_name,

collection\_metadata=self.collection\_metadata,

ids=ids

)

logger.info("Created new vector store with documents")

else:

# Add to existing vector store

self.\_vectorstore.add\_documents(documents, ids=ids)

logger.info("Added documents to existing vector store")

# Log statistics

self.\_log\_store\_stats()

except Exception as e:

logger.error(f"Error adding documents to vector store: {str(e)}", exc\_info=True)

raise VectorStoreError(f"Failed to add documents: {str(e)}")

def similarity\_search(

self,

query: str,

k: int = 4,

filter: Optional[Dict[str, Any]] = None,

\*\*kwargs

) -> List[Document]:

"""

Search for similar documents.

Args:

query: Search query

k: Number of results to return

filter: Optional metadata filter

\*\*kwargs: Additional search parameters

Returns:

List of similar documents

Raises:

VectorStoreError: If search fails

"""

logger.info(f"Performing similarity search: '{query[:50]}...'", extra={"k": k})

if self.\_vectorstore is None:

raise VectorStoreError("Vector store not initialized. Add documents first.")

try:

results = self.\_vectorstore.similarity\_search(

query=query,

k=k,

filter=filter,

\*\*kwargs

)

logger.info(f"Found {len(results)} similar documents")

return results

except Exception as e:

logger.error(f"Error during similarity search: {str(e)}", exc\_info=True)

raise VectorStoreError(f"Search failed: {str(e)}")

def similarity\_search\_with\_score(

self,

query: str,

k: int = 4,

filter: Optional[Dict[str, Any]] = None,

\*\*kwargs

) -> List[tuple[Document, float]]:

"""

Search for similar documents with relevance scores.

Args:

query: Search query

k: Number of results to return

filter: Optional metadata filter

\*\*kwargs: Additional search parameters

Returns:

List of (document, score) tuples

"""

logger.info(f"Performing similarity search with scores: '{query[:50]}...'")

if self.\_vectorstore is None:

raise VectorStoreError("Vector store not initialized. Add documents first.")

try:

results = self.\_vectorstore.similarity\_search\_with\_score(

query=query,

k=k,

filter=filter,

\*\*kwargs

)

# Log score distribution

if results:

scores = [score for \_, score in results]

logger.info(

"Search score statistics",

extra={

"min\_score": min(scores),

"max\_score": max(scores),

"avg\_score": sum(scores) / len(scores)

}

)

return results

except Exception as e:

logger.error(f"Error during similarity search: {str(e)}", exc\_info=True)

raise VectorStoreError(f"Search failed: {str(e)}")

def persist(self) -> None:

"""

Persist the vector store to disk.

Raises:

VectorStoreError: If persistence fails

"""

if not self.persist\_directory:

logger.warning("No persist directory configured, skipping persistence")

return

if self.\_vectorstore is None:

logger.warning("No vector store to persist")

return

try:

logger.info(f"Persisting vector store to {self.persist\_directory}")

self.\_vectorstore.persist()

logger.info("Vector store persisted successfully")

except Exception as e:

logger.error(f"Error persisting vector store: {str(e)}", exc\_info=True)

raise VectorStoreError(f"Failed to persist: {str(e)}")

def load(self) -> None:

"""

Load vector store from disk.

Raises:

VectorStoreError: If loading fails

"""

if not self.persist\_directory:

raise VectorStoreError("No persist directory configured")

persist\_path = Path(self.persist\_directory)

if not persist\_path.exists():

raise VectorStoreError(f"Persist directory does not exist: {persist\_path}")

try:

logger.info(f"Loading vector store from {self.persist\_directory}")

self.\_vectorstore = Chroma(

persist\_directory=self.persist\_directory,

embedding\_function=self.embedding\_model,

collection\_name=self.collection\_name

)

logger.info("Vector store loaded successfully")

self.\_log\_store\_stats()

except Exception as e:

logger.error(f"Error loading vector store: {str(e)}", exc\_info=True)

raise VectorStoreError(f"Failed to load: {str(e)}")

def delete\_collection(self) -> None:

"""Delete the entire collection."""

if self.\_vectorstore:

try:

self.\_vectorstore.delete\_collection()

self.\_vectorstore = None

logger.info(f"Deleted collection: {self.collection\_name}")

except Exception as e:

logger.error(f"Error deleting collection: {str(e)}", exc\_info=True)

raise VectorStoreError(f"Failed to delete collection: {str(e)}")

def clear\_persist\_directory(self) -> None:

"""Clear the persistence directory."""

if self.persist\_directory and Path(self.persist\_directory).exists():

try:

shutil.rmtree(self.persist\_directory)

logger.info(f"Cleared persist directory: {self.persist\_directory}")

except Exception as e:

logger.error(f"Error clearing persist directory: {str(e)}", exc\_info=True)

raise VectorStoreError(f"Failed to clear directory: {str(e)}")

def \_log\_store\_stats(self) -> None:

"""Log statistics about the vector store."""

if self.\_vectorstore and hasattr(self.\_vectorstore, '\_collection'):

try:

count = self.\_vectorstore.\_collection.count()

logger.info(f"Vector store contains {count} documents")

except Exception as e:

logger.debug(f"Could not get document count: {str(e)}")

def as\_retriever(self, \*\*kwargs) -> Any:

"""

Get the vector store as a retriever.

Args:

\*\*kwargs: Additional retriever parameters

Returns:

Retriever instance

"""

if self.\_vectorstore is None:

raise VectorStoreError("Vector store not initialized")

return self.\_vectorstore.as\_retriever(\*\*kwargs)