"""

Question-answering chain implementation for the RAG pipeline.

This module provides the retrieval and generation functionality

for answering questions based on the document store.

"""

from typing import Dict, Any, Optional, List

import logging

import time

from langchain.chains import RetrievalQA

from langchain.prompts import PromptTemplate

from langchain.schema import Document

from .base import BaseRetrievalChain

from ..vector\_store.base import BaseVectorStore

from ..llm.base import BaseLLM

from ..utils.exceptions import RetrievalError

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

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

# Default prompt template for QA

DEFAULT\_QA\_PROMPT = """You are a helpful AI assistant. Use the following pieces of context to answer the question at the end.

If you don't know the answer based on the context, just say that you don't know, don't try to make up an answer.

Always cite the specific parts of the context that support your answer.

Context:

{context}

Question: {question}

Answer:"""

class QAChain(BaseRetrievalChain):

"""

Question-answering chain using retrieval-augmented generation.

This chain retrieves relevant documents and uses them to generate

answers to user questions.

"""

def \_\_init\_\_(

self,

llm: BaseLLM,

vector\_store: BaseVectorStore,

chain\_type: str = "stuff",

return\_source\_documents: bool = True,

prompt\_template: Optional[str] = None,

retriever\_kwargs: Optional[Dict[str, Any]] = None,

verbose: bool = False

):

"""

Initialize the QA chain.

Args:

llm: Language model for generation

vector\_store: Vector store for retrieval

chain\_type: Type of chain to use ("stuff", "map\_reduce", "refine", "map\_rerank")

return\_source\_documents: Whether to return source documents

prompt\_template: Custom prompt template

retriever\_kwargs: Additional kwargs for the retriever

verbose: Whether to enable verbose output

"""

self.llm = llm

self.vector\_store = vector\_store

self.chain\_type = chain\_type

self.return\_source\_documents = return\_source\_documents

self.verbose = verbose

# Set up retriever

self.retriever\_kwargs = retriever\_kwargs or {"k": 4}

self.retriever = vector\_store.as\_retriever(\*\*self.retriever\_kwargs)

# Set up prompt

if prompt\_template:

self.prompt = PromptTemplate(

template=prompt\_template,

input\_variables=["context", "question"]

)

else:

self.prompt = PromptTemplate(

template=DEFAULT\_QA\_PROMPT,

input\_variables=["context", "question"]

)

# Create the chain

self.\_create\_chain()

logger.info(

"Initialized QAChain",

extra={

"chain\_type": chain\_type,

"return\_source\_documents": return\_source\_documents,

"retriever\_kwargs": self.retriever\_kwargs

}

)

def \_create\_chain(self) -> None:

"""Create the RetrievalQA chain."""

try:

self.chain = RetrievalQA.from\_chain\_type(

llm=self.llm,

chain\_type=self.chain\_type,

retriever=self.retriever,

return\_source\_documents=self.return\_source\_documents,

chain\_type\_kwargs={"prompt": self.prompt},

verbose=self.verbose

)

except Exception as e:

logger.error(f"Error creating QA chain: {str(e)}", exc\_info=True)

raise RetrievalError(f"Failed to create QA chain: {str(e)}")

def run(self, query: str) -> Dict[str, Any]:

"""

Run the QA chain on a query.

Args:

query: User question

Returns:

Dictionary containing the answer and optionally source documents

Raises:

RetrievalError: If the chain execution fails

"""

logger.info(f"Running QA chain for query: '{query[:100]}...'")

start\_time = time.time()

try:

# Run the chain

result = self.chain.invoke({"query": query})

# Process the result

response = {

"query": query,

"answer": result.get("result", ""),

"source\_documents": result.get("source\_documents", []),

"metadata": {

"processing\_time": time.time() - start\_time,

"num\_source\_documents": len(result.get("source\_documents", [])),

"chain\_type": self.chain\_type

}

}

# Log metrics

self.\_log\_retrieval\_metrics(response)

return response

except Exception as e:

logger.error(f"Error running QA chain: {str(e)}", exc\_info=True)

raise RetrievalError(f"QA chain execution failed: {str(e)}")

def run\_batch(self, queries: List[str]) -> List[Dict[str, Any]]:

"""

Run the QA chain on multiple queries.

Args:

queries: List of user questions

Returns:

List of results for each query

"""

logger.info(f"Running batch QA for {len(queries)} queries")

results = []

for i, query in enumerate(queries):

logger.info(f"Processing query {i+1}/{len(queries)}")

try:

result = self.run(query)

results.append(result)

except RetrievalError as e:

logger.warning(f"Failed to process query '{query[:50]}...': {str(e)}")

results.append({

"query": query,

"answer": None,

"error": str(e),

"source\_documents": []

})

return results

def \_log\_retrieval\_metrics(self, response: Dict[str, Any]) -> None:

"""Log metrics about the retrieval and generation."""

metrics = {

"query\_length": len(response["query"]),

"answer\_length": len(response["answer"]) if response["answer"] else 0,

"num\_sources": response["metadata"]["num\_source\_documents"],

"processing\_time\_seconds": response["metadata"]["processing\_time"]

}

# Calculate source document statistics

if response["source\_documents"]:

source\_lengths = [len(doc.page\_content) for doc in response["source\_documents"]]

metrics["avg\_source\_length"] = sum(source\_lengths) / len(source\_lengths)

metrics["total\_context\_length"] = sum(source\_lengths)

logger.info("QA chain metrics", extra=metrics)

def update\_retriever\_kwargs(self, \*\*kwargs) -> None:

"""

Update retriever parameters.

Args:

\*\*kwargs: New retriever parameters

"""

self.retriever\_kwargs.update(kwargs)

self.retriever = self.vector\_store.as\_retriever(\*\*self.retriever\_kwargs)

self.\_create\_chain()

logger.info(f"Updated retriever kwargs: {self.retriever\_kwargs}")

def get\_relevant\_documents(self, query: str) -> List[Document]:

"""

Get relevant documents without generating an answer.

Args:

query: Search query

Returns:

List of relevant documents

"""

try:

return self.retriever.get\_relevant\_documents(query)

except Exception as e:

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

raise RetrievalError(f"Document retrieval failed: {str(e)}")