"""

Text chunking implementation for document processing.

This module provides functionality to split documents into smaller chunks

suitable for embedding and retrieval.

"""

from typing import List, Optional, Dict, Any

import logging

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.schema import Document

from .base import BaseTextProcessor

from ..utils.exceptions import ChunkingError

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

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

class DocumentChunker(BaseTextProcessor):

"""

Document chunker using LangChain's RecursiveCharacterTextSplitter.

This chunker splits documents into smaller, overlapping chunks while

attempting to preserve semantic boundaries.

"""

def \_\_init\_\_(

self,

chunk\_size: int = 1000,

chunk\_overlap: int = 200,

separators: Optional[List[str]] = None,

length\_function: Optional[Any] = None

):

"""

Initialize the document chunker.

Args:

chunk\_size: Maximum size of each chunk

chunk\_overlap: Number of characters to overlap between chunks

separators: List of separators to use for splitting

length\_function: Function to calculate chunk length

"""

self.chunk\_size = chunk\_size

self.chunk\_overlap = chunk\_overlap

self.separators = separators or ["\n\n", "\n", " ", ""]

self.length\_function = length\_function or len

# Validate parameters

if chunk\_overlap >= chunk\_size:

raise ValueError("chunk\_overlap must be less than chunk\_size")

# Initialize the text splitter

self.text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=chunk\_size,

chunk\_overlap=chunk\_overlap,

separators=self.separators,

length\_function=self.length\_function

)

logger.info(

"Initialized DocumentChunker",

extra={

"chunk\_size": chunk\_size,

"chunk\_overlap": chunk\_overlap,

"separators": self.separators

}

)

def process(self, documents: List[Document]) -> List[Document]:

"""

Process documents by chunking them into smaller pieces.

Args:

documents: List of documents to chunk

Returns:

List of chunked documents

Raises:

ChunkingError: If chunking fails

"""

logger.info(f"Starting chunking process for {len(documents)} documents")

if not documents:

logger.warning("No documents provided for chunking")

return []

try:

# Split documents

chunks = self.text\_splitter.split\_documents(documents)

# Add chunk metadata

for i, chunk in enumerate(chunks):

chunk.metadata['chunk\_index'] = i

chunk.metadata['chunk\_size'] = len(chunk.page\_content)

chunk.metadata['chunking\_method'] = 'recursive\_character'

chunk.metadata['chunk\_params'] = {

'size': self.chunk\_size,

'overlap': self.chunk\_overlap

}

# Log statistics

self.\_log\_chunking\_stats(documents, chunks)

return chunks

except Exception as e:

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

raise ChunkingError(f"Failed to chunk documents: {str(e)}")

def \_log\_chunking\_stats(self, original\_docs: List[Document], chunks: List[Document]) -> None:

"""Log statistics about the chunking process."""

original\_chars = sum(len(doc.page\_content) for doc in original\_docs)

chunk\_chars = sum(len(chunk.page\_content) for chunk in chunks)

stats = {

"original\_documents": len(original\_docs),

"total\_chunks": len(chunks),

"original\_characters": original\_chars,

"chunk\_characters": chunk\_chars,

"average\_chunk\_size": chunk\_chars / len(chunks) if chunks else 0,

"chunks\_per\_document": len(chunks) / len(original\_docs) if original\_docs else 0

}

logger.info("Chunking statistics", extra=stats)

def chunk\_text(self, text: str, metadata: Optional[Dict[str, Any]] = None) -> List[Document]:

"""

Chunk a single text string.

Args:

text: Text to chunk

metadata: Optional metadata to attach to chunks

Returns:

List of chunked documents

"""

# Create a temporary document

temp\_doc = Document(page\_content=text, metadata=metadata or {})

return self.process([temp\_doc])

def update\_parameters(

self,

chunk\_size: Optional[int] = None,

chunk\_overlap: Optional[int] = None

) -> None:

"""

Update chunking parameters.

Args:

chunk\_size: New chunk size

chunk\_overlap: New chunk overlap

"""

if chunk\_size is not None:

self.chunk\_size = chunk\_size

if chunk\_overlap is not None:

self.chunk\_overlap = chunk\_overlap

# Recreate text splitter with new parameters

self.text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=self.chunk\_size,

chunk\_overlap=self.chunk\_overlap,

separators=self.separators,

length\_function=self.length\_function

)

logger.info(

"Updated chunking parameters",

extra={

"chunk\_size": self.chunk\_size,

"chunk\_overlap": self.chunk\_overlap

}

)