```
import os
from pathlib import Path
from typing import Optional

from dotenv import load_dotenv
from llama_index.core import SimpleDirectoryReader, VectorStoreIndex
from loguru import logger
from swarm_models import OpenAIChat

from swarms import Agent, AgentRearrange

load_dotenv()

# Get the OpenAI API key from the environment variable
api_key = os.getenv("GROQ_API_KEY")

# Model
model = OpenAIChat(
    openai_api_base="https://api.groq.com/openai/v1",
    openai_api_key=api_key,
    model_name="llama-3.1-70b-versatile",
    temperature=0.1,
)


class LlamaIndexDB:
```

```
"""A class to manage document indexing and querying using LlamaIndex.

    This class provides functionality to add documents from a directory and query the indexed documents.

    Args:
        data_dir (str): Directory containing documents to index. Defaults to "docs".
        **kwargs: Additional arguments passed to SimpleDirectoryReader and VectorStoreIndex.
            SimpleDirectoryReader kwargs:
                - filename_as_id (bool): Use filenames as document IDs
                - recursive (bool): Recursively read subdirectories
                - required_exts (List[str]): Only read files with these extensions
                - exclude_hidden (bool): Skip hidden files

            VectorStoreIndex kwargs:
                - service_context: Custom service context
                - embed_model: Custom embedding model
                - similarity_top_k (int): Number of similar docs to retrieve
                - store_nodes_override (bool): Override node storage
    """

    def __init__(self, data_dir: str = "docs", **kwargs) -> None:
        """Initialize the LlamaIndexDB with an empty index.

        Args:
            data_dir (str): Directory containing documents to index
```

```
    **kwargs: Additional arguments for SimpleDirectoryReader and VectorStoreIndex
"""
self.data_dir = data_dir
self.index: Optional[VectorStoreIndex] = None
self.reader_kwargs = {
    k: v
    for k, v in kwargs.items()
    if k
    in SimpleDirectoryReader.__init__.__code__.co_varnames
}
self.index_kwargs = {
    k: v
    for k, v in kwargs.items()
    if k not in self.reader_kwargs
}

logger.info("Initialized LlamaIndexDB")
data_path = Path(self.data_dir)
if not data_path.exists():
    logger.error(f"Directory not found: {self.data_dir}")
    raise FileNotFoundError(
        f"Directory {self.data_dir} does not exist"
    )

try:
    documents = SimpleDirectoryReader(
```

```
            self.data_dir, **self.reader_kwargs
        ).load_data()
        self.index = VectorStoreIndex.from_documents(
            documents, **self.index_kwargs
        )
        logger.success(
            f"Successfully indexed documents from {self.data_dir}"
        )
    except Exception as e:
        logger.error(f"Error indexing documents: {str(e)}")
        raise

def query(self, query: str, **kwargs) -> str:
    """Query the indexed documents.

    Args:
        query (str): The query string to search for
        **kwargs: Additional arguments passed to the query engine
            - similarity_top_k (int): Number of similar documents to retrieve
            - streaming (bool): Enable streaming response
            - response_mode (str): Response synthesis mode
            - max_tokens (int): Maximum tokens in response

    Returns:
        str: The response from the query engine
```

```
        Raises:
            ValueError: If no documents have been indexed yet
        """
        if self.index is None:
            logger.error("No documents have been indexed yet")
            raise ValueError("Must add documents before querying")

        try:
            query_engine = self.index.as_query_engine(**kwargs)
            response = query_engine.query(query)
            print(response)
            logger.info(f"Successfully queried: {query}")
            return str(response)
        except Exception as e:
            logger.error(f"Error during query: {str(e)}")
            raise



# Initialize specialized medical agents
medical_data_extractor = Agent(
    agent_name="Medical-Data-Extractor",
     system_prompt="You are a specialized medical data extraction expert, trained in processing and analyzing clinical data, lab results, medical imaging reports, and patient records. Your role is to carefully extract relevant medical information while maintaining strict HIPAA compliance and patient confidentiality. Focus on identifying key clinical indicators, test results, vital signs, medication histories, and relevant patient history. Pay special attention to temporal relationships between
```

```
symptoms, treatments, and outcomes. Ensure all extracted data maintains proper medical context and terminology.",
    llm=model,
    max_loops=1,
    autosave=True,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="medical_data_extractor.json",
    user_name="medical_team",
    retry_attempts=1,
    context_length=200000,
    output_type="string",
)

diagnostic_specialist = Agent(
    agent_name="Diagnostic-Specialist",
     system_prompt="You are a senior diagnostic physician with extensive experience in differential diagnosis. Your role is to analyze patient symptoms, lab results, and clinical findings to develop comprehensive diagnostic assessments. Consider all presenting symptoms, patient history, risk factors, and test results to formulate possible diagnoses. Prioritize diagnoses based on clinical probability and severity. Always consider both common and rare conditions that match the symptom pattern. Recommend additional tests or imaging when needed for diagnostic clarity. Follow evidence-based diagnostic criteria and current medical guidelines.",
    llm=model,
    max_loops=1,
    autosave=True,
```

```
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="diagnostic_specialist.json",
    user_name="medical_team",
    retry_attempts=1,
    context_length=200000,
    output_type="string",
)

treatment_planner = Agent(
    agent_name="Treatment-Planner",
      system_prompt="You are an experienced clinical treatment specialist focused on developing comprehensive treatment plans. Your expertise covers both acute and chronic condition management, medication selection, and therapeutic interventions. Consider patient-specific factors including age, comorbidities, allergies, and contraindications when recommending treatments. Incorporate both pharmacological and non-pharmacological interventions. Emphasize evidence-based treatment protocols while considering patient preferences and quality of life. Address potential drug interactions and side effects. Include monitoring parameters and treatment milestones.",
    llm=model,
    max_loops=1,
    autosave=True,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="treatment_planner.json",
    user_name="medical_team",
```

```
    retry_attempts=1,
    context_length=200000,
    output_type="string",
)

specialist_consultant = Agent(
    agent_name="Specialist-Consultant",
    system_prompt="You are a medical specialist consultant with expertise across multiple disciplines including cardiology, neurology, endocrinology, and internal medicine. Your role is to provide specialized insight for complex cases requiring deep domain knowledge. Analyze cases from your specialist perspective, considering rare conditions and complex interactions between multiple systems. Provide detailed recommendations for specialized testing, imaging, or interventions within your domain. Highlight potential complications or considerations that may not be immediately apparent to general practitioners.",
    llm=model,
    max_loops=1,
    autosave=True,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="specialist_consultant.json",
    user_name="medical_team",
    retry_attempts=1,
    context_length=200000,
    output_type="string",
)
```

```
patient_care_coordinator = Agent(
    agent_name="Patient-Care-Coordinator",
     system_prompt="You are a patient care coordinator specializing in comprehensive healthcare management. Your role is to ensure holistic patient care by coordinating between different medical specialists, considering patient needs, and managing care transitions. Focus on patient education, medication adherence, lifestyle modifications, and follow-up care planning. Consider social determinants of health, patient resources, and access to care. Develop actionable care plans that patients can realistically follow. Coordinate with other healthcare providers to ensure continuity of care and proper implementation of treatment plans.",
    llm=model,
    max_loops=1,
    autosave=True,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="patient_care_coordinator.json",
    user_name="medical_team",
    retry_attempts=1,
    context_length=200000,
    output_type="string",
)



# Initialize the SwarmRouter to coordinate the medical agents
router = AgentRearrange(
    name="medical-diagnosis-treatment-swarm",
      description="Collaborative medical team for comprehensive patient diagnosis and treatment
```

```
planning",
    max_loops=1,  # Limit to one iteration through the agent flow
    agents=[
        medical_data_extractor,  # First agent to extract medical data
        diagnostic_specialist,  # Second agent to analyze and diagnose
        treatment_planner,  # Third agent to plan treatment
        specialist_consultant,  # Fourth agent to provide specialist input
        patient_care_coordinator,  # Final agent to coordinate care plan
    ],
    # Configure the document storage and retrieval system
    memory_system=LlamaIndexDB(
        data_dir="docs",  # Directory containing medical documents
        filename_as_id=True,  # Use filenames as document identifiers
        recursive=True,  # Search subdirectories
        # required_exts=[".txt", ".pdf", ".docx"],  # Supported file types
        similarity_top_k=10,  # Return top 10 most relevant documents
    ),
    # Define the sequential flow of information between agents
            flow=f"{medical_data_extractor.agent_name}  ->  {diagnostic_specialist.agent_name}  ->
{treatment_planner.agent_name}  ->  {specialist_consultant.agent_name}  ->
{patient_care_coordinator.agent_name}",
)

# Example usage
if __name__ == "__main__":
    # Run a comprehensive medical analysis task for patient Lucas Brown
```

```
router.run(
    "Analyze this Lucas Brown's medical data to provide a diagnosis and treatment plan"
)
```