```
!pip install pydantic
!pip install netCDF4
!pip install xarray

from netCDF4 import Dataset
from pydantic import BaseModel, Field
from typing import List, Optional
import os
import xarray as xr

# Display when done
print('Libraries imported')
```

```
Requirement already satisfied: pydantic in /usr/local/lib/python3.10/dist-packages (2.9.1)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pydantic-core==2.23.3 in /usr/local/lib/python3.10/dist-packages (from p
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (fro
Collecting netCDF4
  Downloading netCDF4-1.7.1.post2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (
Collecting cftime (from netCDF4)
  Downloading cftime-1.6.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.7 kB)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from netCDF4) (2024.
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from netCDF4) (1.26.4)
Downloading netCDF4-1.7.1.post2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.0/9.0 MB 51.0 MB/s eta 0:00:00
Downloading cftime-1.6.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 44.4 MB/s eta 0:00:00
Installing collected packages: cftime, netCDF4
Successfully installed cftime-1.6.4 netCDF4-1.7.1.post2
Requirement already satisfied: xarray in /usr/local/lib/python3.10/dist-packages (2024.6.0)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.10/dist-packages (from xarray) (1.
Requirement already satisfied: packaging>=23.1 in /usr/local/lib/python3.10/dist-packages (from xarray)
Requirement already satisfied: pandas>=2.0 in /usr/local/lib/python3.10/dist-packages (from xarray) (2.
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateuti
Libraries imported
```

```
class NetCDFMetadata(BaseModel):
    dimensions: dict = Field(..., description="Dimensions of the NetCDF file.")
    variables: List[str] = Field(..., description="Variables available in the NetCDF file.")
    attributes: dict = Field(..., description="Global attributes of the NetCDF file.")
    file_name: str = Field(..., description="The name of the NetCDF file.")

# Display when done
print('NetCDFMetadata model created')
```

```
NetCDFMetadata model created
```

```
def extract_netcdf_metadata(file_path: str) -> NetCDFMetadata:
    with Dataset(file_path, 'r') as nc:
        dimensions = {dim: len(nc.dimensions[dim]) for dim in nc.dimensions}
        variables = list(nc.variables.keys())
        attributes = {attr: nc.getncattr(attr) for attr in nc.ncattrs()}
        file_name = os.path.basename(file_path)
        return NetCDFMetadata(
            dimensions=dimensions,
```

```
            variables=variables,
            attributes=attributes,
            file_name=file_name
        )

# Display when done
print('Metadata extraction function created')
```

```
Metadata extraction function created
```

```
# List your NetCDF files
netcdf_files = ['/content/gom_t008.nc']

# Extract metadata for each file
all_metadata = [extract_netcdf_metadata(f) for f in netcdf_files]

# Display the extracted metadata
for metadata in all_metadata:
    print(metadata)

# Display when done
print('Metadata extraction completed')
```

```
dimensions={'lat': 346, 'lon': 541, 'depth': 40, 'time': 1} variables=['time', 'tau', 'depth', 'lat', '
Metadata extraction completed
```

```
#!pip install llama-index
#!pip install chromadb
#!pip install openai
#!pip install llama-index-vector-stores-chroma

from llama_index.core import StorageContext, VectorStoreIndex, Settings
from llama_index.llms.openai import OpenAI
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.vector_stores.chroma import ChromaVectorStore
import chromadb
import os
from llama_index.llms.openai import OpenAI

os.environ['OPENAI_API_KEY'] = 'sk-RlyK9UahfWCbcQJcz7iEHqkxvCNj9vVHREOhSrRuS0T3BlbkFJIMns3sTSMPRlgx4f1x-k-X

# Set up LlamaIndex Settings
Settings.llm = OpenAI(model='gpt-4o-mini', temperature=0.1)
Settings.embed_model = OpenAIEmbedding()

# Chroma settings
chroma_path = './chroma_db'
chroma_collection_name = 'chrm'

# Display when done
print('LlamaIndex components loaded')
```

```
LlamaIndex components loaded
```

```
# Load or create vector store
if os.path.exists(chroma_path):
    chroma_client = chromadb.PersistentClient(path=chroma_path)
    chroma_collection = chroma_client.get_or_create_collection(chroma_collection_name)
    vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
    index = VectorStoreIndex.from_vector_store(vector_store)
    print('Vector store loaded')
else:
    chroma_client = chromadb.PersistentClient(path=chroma_path)
    chroma_collection = chroma_client.get_or_create_collection(chroma_collection_name)
    vector_store = ChromaVectorStore(chroma_collection=chroma_collection)

    # Ensure documents are properly formatted
    documents = [{'text': doc.text, 'metadata': doc.metadata} for doc in all_metadata]

    # Create the index
    storage_context = StorageContext.from_defaults(vector_store=vector_store)
    index = VectorStoreIndex.from_documents(documents, storage_context=storage_context)
    print('Vector store created')
```

```
Vector store loaded
```

```
from llama_index.core.retrievers import VectorIndexAutoRetriever
from llama_index.core.vector_stores.types import MetadataInfo, VectorStoreInfo

# Prepare metadata schema
all_metadata_info = []
for field_name, field_info in NetCDFMetadata.__fields__.items():
    all_metadata_info.append(
        MetadataInfo(
            name=field_name,
            type=str(field_info.annotation),
            description=field_info.description,
        )
    )

vector_store_info = VectorStoreInfo(
    content_info="list of NetCDF files metadata",
    metadata_info=all_metadata_info,
)

retriever = VectorIndexAutoRetriever(index, vector_store_info, verbose=True)
print('Metadata schema prepared')
```

```
Metadata schema prepared
```

```
import time
from llama_index.core.query_engine import RetrieverQueryEngine

def retry_request(func, retries=3, delay=5):
    for attempt in range(retries):
        try:
            return func()
        except Exception as e:
            print(f"Attempt {attempt + 1} failed: {e}")
            time.sleep(delay)
    raise RuntimeError("All retry attempts failed.")
```

```
# Set up the query engine
query_engine = RetrieverQueryEngine.from_args(retriever=retriever, streaming=True)

def make_query():
    return query_engine.query('What variables are in the first NetCDF file?')

# Retry the query request
try:
    resp = retry_request(make_query)
    for token in resp.response_gen:
        print(token, end="")
except Exception as e:
    print(f"Query failed: {e}")

print('Query executed')
```

```
Using query str: variables in the first NetCDF file
Using filters: []
Empty ResponseQuery executed
```

```
import netCDF4

# Check variables in the first NetCDF file
file_path = '/content/gom_t008.nc'
dataset = netCDF4.Dataset(file_path, 'r')

print("Variables in the NetCDF file:")
for var in dataset.variables:
    print(var)
```

```
Variables in the NetCDF file:
time
tau
depth
lat
lon
water_u
water_v
water_temp
salinity
surf_el
```

Start coding or generate with AI.

Start coding or generate with AI.

```
import netCDF4 as nc

def extract_metadata(netcdf_file):
    dataset = nc.Dataset(netcdf_file)
    metadata = {
        'variables': list(dataset.variables.keys()),
        'dimensions': list(dataset.dimensions.keys()),
        'attributes': {attr: getattr(dataset, attr) for attr in dataset.ncattrs()}
    }
    dataset.close()
    return metadata
```

```
# Example of extracting metadata
metadata = extract_metadata('/content/gom_t008.nc')
print(metadata)
```

```
{'variables': ['time', 'tau', 'depth', 'lat', 'lon', 'water_u', 'water_v', 'water_temp', 'salinity', 's
```

```
import json

# Store metadata as JSON
metadata_json = json.dumps(metadata, indent=4)
with open('metadata.json', 'w') as f:
    f.write(metadata_json)
```

```
!pip install langchain openai langchain_community
```

```
Requirement already satisfied: langchain in /usr/local/lib/python3.10/dist-packages (0.2.16)
Requirement already satisfied: openai in /usr/local/lib/python3.10/dist-packages (1.44.1)
Collecting langchain_community
  Downloading langchain_community-0.2.16-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages (from langchain)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-packages (from lang
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dist-packages (from l
Requirement already satisfied: async-timeout<5.0.0,>=4.0.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: langchain-core<0.3.0,>=0.2.38 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: langchain-text-splitters<0.3.0,>=0.2.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: langsmith<0.2.0,>=0.1.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain)
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packages (from langchai
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packages (from langchai
Requirement already satisfied: tenacity!=8.4.0,<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from openai)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from openai) (1.7.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from openai
Requirement already satisfied: jiter<1,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from openai)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages (from openai) (4.66.5)
Requirement already satisfied: typing-extensions<5,>=4.11 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohtt
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aioht
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aio
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.10/dist-packages (from httpx<1,>
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.10/dist-packages (from httpcor
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10/dist-packages (from la
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.10/dist-packages (from lan
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.10/dist-packages (from l
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pydantic-core==2.23.3 in /usr/local/lib/python3.10/dist-packages (from p
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requ
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlc
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dist-packages (from jsonpa
```

```
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from
Downloading langchain_community-0.2.16-py3-none-any.whl (2.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 19.0 MB/s eta 0:00:00
Installing collected packages: langchain_community
Successfully installed langchain_community-0.2.16
```

```
from langchain.chat_models import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

# Initialize OpenAI chat-based LLM (using gpt-3.5-turbo)
chat_llm = ChatOpenAI(temperature=0.1, model="gpt-3.5-turbo")

# Create a prompt template to query the metadata
template = """
You are an assistant that helps users explore metadata of NetCDF files. Below is the metadata:

{metadata}

Answer the following question: {question}
"""

prompt = PromptTemplate(
    input_variables=["metadata", "question"],
    template=template
)

# Create a chain to query the metadata
metadata_chain = LLMChain(llm=chat_llm, prompt=prompt)

# Example of querying metadata
query = "What are the variables in the NetCDF file?"
response = metadata_chain.run({
    "metadata": metadata_json,
    "question": query
})

print(response)
```

```
<ipython-input-27-c4c0c846fa24>:6: LangChainDeprecationWarning: The class `ChatOpenAI` was deprecated i
  chat_llm = ChatOpenAI(temperature=0.1, model="gpt-3.5-turbo")
The variables in the NetCDF file are:
1. time
2. tau
3. depth
4. lat
5. lon
6. water_u
7. water_v
8. water_temp
9. salinity
10. surf_el
```

```
# List of queries you want to test
queries = [
    "What are the variables in the NetCDF file?",
    "What is the time range of the data in the NetCDF file?",
```

```
    "What is the spatial resolution of the data in the file?",
    "Are there any missing values in the NetCDF file?",
    "What is the depth range covered in this dataset?"
]

# Loop through each question and query the metadata
for query in queries:
    response = metadata_chain.run({
        "metadata": metadata_json,
        "question": query
    })

    print(f"Question: {query}")
    print(f"Response: {response}\n")
```

```
:CDF file?
ıts the time dimension. Without specific values provided, we cannot determine the exact time range of th

ı the file?
.e can be determined by looking at the dimensions "lat" and "lon". In this case, the dimensions are lat

.le?
:ermine if there are any missing values in the NetCDF file. The metadata only includes information about

;et?
:termined by the "depth" variable, which is one of the dimensions in the NetCDF file. The depth dimensio
```

Start coding or generate with AI.

```
import netCDF4 as nc

# Function to extract metadata from the NetCDF file
def extract_metadata(nc_file):
    # Open the NetCDF file
    dataset = nc.Dataset(nc_file)

    # Extract basic information
    variables = list(dataset.variables.keys())
    time_range = None
    if 'time' in dataset.variables:
        time_range = (dataset.variables['time'][:].min(), dataset.variables['time'][:].max())

    latitudes = dataset.variables['lat'][:] if 'lat' in dataset.variables else None
    longitudes = dataset.variables['lon'][:] if 'lon' in dataset.variables else None
    depth_range = None
```

```
    if 'depth' in dataset.variables:
        depth_range = (dataset.variables['depth'][:].min(), dataset.variables['depth'][:].max())

    # Spatial resolution (assuming lat and lon are 1D arrays)
    lat_resolution = latitudes[1] - latitudes[0] if latitudes is not None and len(latitudes) > 1 else None
    lon_resolution = longitudes[1] - longitudes[0] if longitudes is not None and len(longitudes) > 1 else No

    # Check for missing values in each variable
    missing_value_info = {}
    for var in variables:
        if hasattr(dataset.variables[var], '_FillValue'):
            missing_value_info[var] = dataset.variables[var].__dict__.get('_FillValue', None)

    # Create a metadata dictionary
    metadata = {
        "variables": variables,
        "time_range": time_range,
        "lat_resolution": lat_resolution,
        "lon_resolution": lon_resolution,
        "depth_range": depth_range,
        "missing_value_info": missing_value_info
    }

    return metadata

# Function to handle specific queries based on extracted metadata
def handle_query(query, metadata):
    if "variables" in query.lower():
        return f"The variables in the NetCDF file are: {', '.join(metadata['variables'])}"

    elif "time range" in query.lower():
        if metadata['time_range']:
            return f"The time range of the data is from {metadata['time_range'][0]} to {metadata['time_range
        else:
            return "No time data available."

    elif "spatial resolution" in query.lower():
        if metadata['lat_resolution'] and metadata['lon_resolution']:
            return f"The spatial resolution is approximately {metadata['lat_resolution']} degrees in latitud
        else:
            return "No spatial resolution data available."

    elif "missing values" in query.lower():
        if metadata['missing_value_info']:
            missing_info = ', '.join([f"{var}: {val}" for var, val in metadata['missing_value_info'].items()
            return f"The following variables have missing values: {missing_info}."
        else:
            return "There are no missing values in the variables."

    elif "depth range" in query.lower():
        if metadata['depth_range']:
            return f"The depth range in this dataset is from {metadata['depth_range'][0]} to {metadata['dept
        else:
            return "No depth data available."

    else:
        return "Query not recognized or supported."

# Example of running multiple queries on the metadata
def run_queries(nc_file, queries):
```

```
    # Extract metadata from the NetCDF file
    metadata = extract_metadata(nc_file)

    # Handle each query and print the response
    for query in queries:
        print(f"Question: {query}")
        response = handle_query(query, metadata)
        print(f"Response: {response}\n")

# Define your queries
queries = [
    "What are the variables in the NetCDF file?",
    "What is the time range of the data in the NetCDF file?",
    "What is the spatial resolution of the data in the file?",
    "Are there any missing values in the NetCDF file?",
    "What is the depth range covered in this dataset?"
]

# Path to your NetCDF file
nc_file = '/content/gom_t008.nc'

# Run the queries
run_queries(nc_file, queries)
```

```
Question: What are the variables in the NetCDF file?
Response: The variables in the NetCDF file are: time, tau, depth, lat, lon, water_u, water_v, water_tem

Question: What is the time range of the data in the NetCDF file?
Response: The time range of the data is from 192884.00000000006 to 192884.00000000006.

Question: What is the spatial resolution of the data in the file?
Response: The spatial resolution is approximately 0.03999900817871094 degrees in latitude and 0.0399780

Question: Are there any missing values in the NetCDF file?
Response: The following variables have missing values: water_u: -30000, water_v: -30000, water_temp: -3

Question: What is the depth range covered in this dataset?
Response: The depth range in this dataset is from 0.0 to 5000.0 meters.
```

Start coding or generate with AI.