Langri-La

Langrila is an open-source third-party python package that is useful to use API-based LLM in the same interface. This package puts emphasis on simple architecture for readability. This package is just personal project.

Contribution

Coding style

1. Sticking to simplicity : This library is motivated by simplifying architecture for readability. Thus too much abstraction should be avoided.
2. Implementing minimum modules : The more functions each module has, the more complex the source code becomes. Langrila focuses on implementing minimum necessary functions in each module Basically module has only a responsibility expressed by thier module name and main function is implemented in a method easy to understand like `run()` or `arun()` methods except for some.

Branch management rule

- Topic branch are checkout from main branch.
- Topic branch should be small.

Pre-requirement

If necessary, set environment variables to use OpenAI API, Azure OpenAI Service, Gemini API, and Claude API; if using VertexAI or Amazon Bedrock, check each platform's user guide and authenticate in advance VertexAI and Amazon Bedrock.

Supported models for OpenAI

Chat models

- gpt-3.5-turbo-1106
- gpt-3.5-turbo-0125
- gpt-4-1106-preview
- gpt-4-vision-preview
- gpt-4-0125-preview
- gpt-4-turbo-2024-04-09
- gpt-4o-2024-05-13
- gpt-4o-mini-2024-07-18

Embedding models

- text-embedding-ada-002
- text-embedding-3-small
- text-embedding-3-large

Aliases

```
{'gpt-4o-mini': 'gpt-4o-mini-2024-07-18',
 'gpt-4o': 'gpt-4o-2024-05-13',
 'gpt-4-turbo': 'gpt-4-turbo-2024-04-09',
 'gpt-3.5-turbo': 'gpt-3.5-turbo-0125'}
```

Platform

- OpenAI
- Azure OpenAI

Supported models for Gemini

Chat models

- gemini-1.5-pro
- gemini-1.5-flash

Platform

- Google AI
- VertexAI

Supported models for Claude

Chat models

- claude-3.5-sonnet
- claude-3-opus
- claude-3-sonnet
- claude-3-haiku

Platform

- Anthropic
- Amazon Bedrock
- VertexAI (not tested)

Breaking changes

▶ v0.0.20 -> v0.1.0

▶ v0.0.7 -> v0.0.8

▶ v0.0.2 -> v0.0.3

Basic usage

Sample notebook 01.introduction.ipynb includes following contents:

- Basic usage with simple text prompt
  - ChatGPT of OpenAI
  - ChatGPT on Azure OpenAI
  - Gemini of Google AI
  - Gemini on VertexAI
  - Claude of Anthropic
  - Claude on Amazon Bedrock
- Image input
- Message system in langrila
- Multi-turn conversation with multiple client
- How to specify system instruction
- JSON mode completion
- Token management
- Usage gathering across multiple models
- Prompt template

02.function_calling.ipynb instruct function calling in langrila.

- Basic usage for ChatGPT, Gemini and Claude
- Multi-turn conversation using tools
- Multi-turn conversation using tools with multiple client

Dependencies

must

- Python >=3.10,<3.13

as needed

Langrila has various extra installation options. See the following installation section and pyproject.toml.

Installation

clone

```
git clone git@github.com:taikinman/langrila.git
```

pip

See pyproject.toml for more detailed installation options.

```
cd langrila

# For OpenAI
pip install -e .[openai]

# For Gemini
pip install -e .[gemini]

# For Claude
pip install -e .[claude]

# For both
pip install -e .[openai,gemini]

# For OpenAI and Qdrant
pip install -e .[openai,qdrant]

# For OpenAI and Chroma
pip install -e .[openai,chroma]

# For OpenAI and Usearch
pip install -e .[openai,usearch]

# For All
pip install -e .[all]
```

poetry

See pyproject.toml for more detailed installation options.

```
# For OpenAI
poetry add --editable /path/to/langrila/ --extras openai

# For Gemini
poetry add --editable /path/to/langrila/ --extras gemini

# For Claude
poetry add --editable /path/to/langrila/ --extras claude

# For both OpenAI and Gemini (can choose Claude as well)
poetry add --editable /path/to/langrila/ --extras "openai gemini"

# For OpenAI and Qdrant
poetry add --editable /path/to/langrila/ --extras "openai qdrant"

# For OpenAI and Chroma
poetry add --editable /path/to/langrila/ --extras "openai chroma"

# For OpenAI and Usearch
poetry add --editable /path/to/langrila/ --extras "openai usearch"

# For all extra dependencies
poetry add --editable /path/to/langrila/ --extras all
```

Optional

Retrieval

Now langrila supports qdrant, chroma and usearch for retrieval.

For Qdrant

```python
from qdrant_client import models

from langrila.database.qdrant import QdrantLocalCollectionModule, QdrantLocalRetrievalModule
from langrila.openai import OpenAIEmbeddingModule
```

```
#######################
# create collection
#######################

embedder = OpenAIEmbeddingModule(
    api_key_env_name="API_KEY",
    model_name="text-embedding-3-small",
    dimensions=1536,
)

collection = QdrantLocalCollectionModule(
    persistence_directory="./qdrant_test",
    collection_name="sample",
    embedder=embedder,
    vectors_config=models.VectorParams(
        size=1536,
        distance=models.Distance.COSINE,
    ),
)

documents = [
    "Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.",
    "LangChain is a framework for developing applications powered by language models.",
    "LlamaIndex (GPT Index) is a data framework for your LLM application.",
]

collection.run(documents=documents) # metadatas could also be used

# #######################
# # retrieval
# #######################

# In the case collection was already instantiated
# retriever = collection.as_retriever(n_results=2, threshold_similarity=0.5)

retriever = QdrantLocalRetrievalModule(
    embedder=embedder,
    persistence_directory="./qdrant_test",
    collection_name="sample",
    n_results=2,
    score_threshold=0.5,
)

query = "What is Langrila?"
retrieval_reuslt = retriever.run(query, filter=None)

# show result
retrieval_result.model_dump()

>>> {'ids': [0],
 'documents': ['Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.'],
 'metadatas': [{'document': 'Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.'}],
 'scores': [0.5303465176248179],
 'collections': ['sample'],
 'usage': {'prompt_tokens': 6, 'completion_tokens': 0}}
```

Qdrant server is also supported by `QdrantRemoteCollectionModule` and `QdrantRemoteRetrievalModule`. Here is a basic example using docker which app container and qdrant container are bridged by same network.

```
from qdrant_client import models

from langrila.database.qdrant import QdrantRemoteCollectionModule, QdrantRemoteRetrievalModule
from langrila.openai import OpenAIEmbeddingModule

#######################
# create collection
#######################

embedder = OpenAIEmbeddingModule(
    api_key_env_name="API_KEY",
    model_name="text-embedding-3-small",
    dimensions=1536,
)

collection = QdrantRemoteCollectionModule(
    url="http://qdrant",
    port="6333",
    collection_name="sample",
    embedder=embedder,
    vectors_config=models.VectorParams(
        size=1536,
        distance=models.Distance.COSINE,
    ),
)
```

For more details, see qdrant.py.

For Chroma

```python
from langrila.database.chroma import ChromaLocalCollectionModule, ChromaLocalRetrievalModule
from langrila.openai import OpenAIEmbeddingModule

#######################
# create collection
#######################

embedder = OpenAIEmbeddingModule(
    api_key_env_name="API_KEY",
    model_name="text-embedding-3-small",
    dimensions=1536,
)

collection = ChromaLocalCollectionModule(
    persistence_directory="./chroma_test",
    collection_name="sample",
    embedder=embedder,
)

documents = [
    "Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.",
    "LangChain is a framework for developing applications powered by language models.",
    "LlamaIndex (GPT Index) is a data framework for your LLM application.",
]

collection.run(documents=documents) # metadatas could also be used

# #######################
# # retrieval
# #######################

# In the case collection was already instantiated
# retriever = collection.as_retriever(n_results=2, threshold_similarity=0.5)

retriever = ChromaLocalRetrievalModule(
    embedder=embedder,
    persistence_directory="./chroma_test",
    collection_name="sample",
    n_results=2,
    score_threshold=0.5,
)

query = "What is Langrila?"
retrieval_result = retriever.run(query, filter=None)

# show result
retrieval_result.model_dump()

>>> {'ids': [0],
 'documents': ['Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.'],
 'metadatas': [{'document': 'Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.'}],
 'scores': [0.46960276455443584],
 'collections': ['sample'],
 'usage': {'prompt_tokens': 6, 'completion_tokens': 0}}
```

HttpClient is also supported by `ChromaRemoteCollectionModule` and `ChromaRemoteRetrievalModule`. Here is a basic example using docker which app container and chroma container are bridged by same network.

```python
from langrila.database.chroma import ChromaRemoteCollectionModule
from langrila.openai import OpenAIEmbeddingModule

#######################
# create collection
#######################

embedder = OpenAIEmbeddingModule(
    api_key_env_name="API_KEY",
    model_name="text-embedding-3-small",
    dimensions=1536,
)

collection = ChromaRemoteCollectionModule(
    host="chroma",
    port="8000",
    collection_name="sample",
    embedder=embedder,
)
```

For more details, see chroma.py.

For Usearch

Usearch originally doesn't support metadata storing and filtering, so in langrila, those functions are realized by SQLite3 and postprocessing.

```python
from langrila.database.usearch import UsearchLocalCollectionModule, UsearchLocalRetrievalModule
from langrila.openai import OpenAIEmbeddingModule

#######################
# create collection
#######################
```

```
embedder = OpenAIEmbeddingModule(
    api_key_env_name="API_KEY",
    model_name="text-embedding-3-small",
    dimensions=1536,
)

collection = UsearchLocalCollectionModule(
    persistence_directory="./usearch_test",
    collection_name="sample",
    embedder=embedder,
    dtype = "f16",
    ndim = 1536,
    connectivity = 16,
    expansion_add = 128,
    expansion_search = 64,
)

documents = [
    "Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.",
    "LangChain is a framework for developing applications powered by language models.",
    "LlamaIndex (GPT Index) is a data framework for your LLM application.",
]

# Strongly recommended because search result may be different when new vectors are inserted after existing vectors are removed.
# Instead, rebuilding the index is recommended using `delete_collection` before upserting.
# Or use exact search to avoid this issue when search time.
collection.delete_collection()

collection.run(documents=documents) # metadatas could also be used.

# #######################
# # retrieval
# #######################

# In the case collection was already instantiated
# retriever = collection.as_retriever(n_results=2, threshold_similarity=0.5)

retriever = UsearchLocalRetrievalModule(
    embedder=embedder,
    persistence_directory="./usearch_test",
    collection_name="sample",
    dtype = "f16",
    ndim=1536,
    connectivity = 16,
    expansion_add = 128,
    expansion_search = 64,
    n_results=2,
    score_threshold=0.5,
)

query = "What is Langrila?"
retrieval_result = retriever.run(query, filter=None, exact=False)

# show result
retrieval_result.model_dump()

>>> {'ids': [0],
 'documents': ['Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.'],
 'metadatas': [{'document': 'Langrila is a useful tool to use ChatGPT with OpenAI API or Azure in an easy way.'}],
 'scores': [0.46986961364746094],
 'collections': ['sample'],
 'usage': {'prompt_tokens': 6, 'completion_tokens': 0}}
```

When you need to filter retrieval results by metadata in search time, you can implement your custom metadata filter. Base class of metadata filter is in base.py. For more details, see : usearch.py.

Specific use case

The library supports a variety of use cases by combining modules such as these and defining new modules. For example, the following is an example of a module that combines basic Retrieval and prompt templates.