GraphRAG & Ollama Quick Start

背景知识文档

> Microsoft GraphRAG

将需要构建知识库的源文本按照相关性规则，构建摘要式的知识图谱，协助大模型更准确地执行任务。

前期环境准备

1. 本地安装 conda ，用于切换隔离的 python 环境
2. 切换使用新的 python 环境

`conda create --name graphrag python=3.11`

`conda activate graphrag`

本地部署Ollama

> 本地部署 Ollama

1. `curl -fsSL https://ollama.com/install.sh | sh`
2. 拉取大模型

`ollama pull qwen2`

`ollama pull nomic-embed-text`

3. 测试本地 ollama 服务

`curl http://localhost:11434/api/generate -d '{"model": "mistral","prompt":"Why is the sky blue?"}'`

修改 GraphRAG 源代码以支持 Ollama

> Microsoft GraphRAG Get Started

1. 安装 graphrag，ollama 依赖包 `pip install graphrag ollama`
2. 准备源知识库文本

`mkdir -p ./ragtest/input`

将下面文件复制到 `./ragtest/input/book.txt`

The Project Gutenberg eBook of A Christmas Carol

This ebook is for the use of anyone anywhere in the United States and most other parts of the world at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this ebook or online at www.gutenberg.org. If you are not located in the United States, you will have to check the laws of the country where you are located before using this eBook.

Title: A Christmas Carol

Author: Charles Dickens

Illustrator: Arthur Rackham

Release date: December 24, 2007 [eBook #24022]

Language: English

Original publication: Philadelphia and New York: J. B. Lippincott Company,, 1915

Credits: Produced by Suzanne Shell, Janet Blenkinship and the Online Distributed Proofreading Team at http://www.pgdp.net

book.txt

3. 初始化配置文件 `python -m graphrag.index --init --root ./ragtest`
4. 修改环境变量 `vi ./ragtest/.env`，将 GRAPHRAG_API_KEY 设置为 ollama

```
GRAPHRAG_API_KEY=ollama
```

5. 修改配置文件 vi ./ragtest/setting.yaml，修改成如下，绿色为修改内容。其中 concurrent_requests 根据自身机器的性能修改即可，性能不足时大模型可能会超时导致失败。

```
encoding_model: cl100k_base
skip_workflows: []
llm:
  api_key: ${GRAPHRAG_API_KEY}
  type: openai_chat # or azure_openai_chat
  model: qwen2
  model_supports_json: true # recommended if this is available for your model.
  # max_tokens: 4000
  # request_timeout: 180.0
  api_base: http://127.0.0.1:11434/v1
  # api_version: 2024-02-15-preview
  # organization: <organization_id>
  # deployment_name: <azure_model_deployment_name>
  # tokens_per_minute: 150_000 # set a leaky bucket throttle
```

```
  # requests_per_minute: 10_000 # set a leaky bucket throttle
  max_retries: 3
  # max_retry_wait: 10.0
  # sleep_on_rate_limit_recommendation: true # whether to sleep when azure suggests wait-times
  concurrent_requests: 2 # the number of parallel inflight requests that may be made

parallelization:
  stagger: 0.3
  # num_threads: 50 # the number of threads to use for parallel processing

async_mode: threaded # or asyncio

embeddings:
  ## parallelization: override the global parallelization settings for embeddings
  async_mode: threaded # or asyncio
  llm:
    api_key: ${GRAPHRAG_API_KEY}
    type: openai_embedding # or azure_openai_embedding
    model: nomic-embed-text
    api_base: http://127.0.0.1:11434/api
    # api_version: 2024-02-15-preview
    # organization: <organization_id>
    # deployment_name: <azure_model_deployment_name>
    # tokens_per_minute: 150_000 # set a leaky bucket throttle
    # requests_per_minute: 10_000 # set a leaky bucket throttle
    # max_retries: 10
    # max_retry_wait: 10.0
    # sleep_on_rate_limit_recommendation: true # whether to sleep when azure suggests wait-times
    # concurrent_requests: 25 # the number of parallel inflight requests that may be made
    # batch_size: 16 # the number of documents to send in a single request
    # batch_max_tokens: 8191 # the maximum number of tokens to send in a single request
    # target: required # or optional



chunks:
  size: 300
  overlap: 100
  group_by_columns: [id] # by default, we don't allow chunks to cross documents

input:
```

```
  type: file # or blob
  file_type: text # or csv
  base_dir: "input"
  file_encoding: utf-8
  file_pattern: ".*\\.txt$"

cache:
  type: file # or blob
  base_dir: "cache"
  # connection_string: <azure_blob_storage_connection_string>
  # container_name: <azure_blob_storage_container_name>

storage:
  type: file # or blob
  base_dir: "output/${timestamp}/artifacts"
  # connection_string: <azure_blob_storage_connection_string>
  # container_name: <azure_blob_storage_container_name>

reporting:
  type: file # or console, blob
  base_dir: "output/${timestamp}/reports"
  # connection_string: <azure_blob_storage_connection_string>
  # container_name: <azure_blob_storage_container_name>

entity_extraction:
  ## llm: override the global llm settings for this task
  ## parallelization: override the global parallelization settings for this task
  ## async_mode: override the global async_mode settings for this task
  prompt: "prompts/entity_extraction.txt"
  entity_types: [organization,person,geo,event]
  max_gleanings: 0

summarize_descriptions:
  ## llm: override the global llm settings for this task
  ## parallelization: override the global parallelization settings for this task
  ## async_mode: override the global async_mode settings for this task
  prompt: "prompts/summarize_descriptions.txt"
  max_length: 500

claim_extraction:
  ## llm: override the global llm settings for this task
  ## parallelization: override the global parallelization settings for this task
  ## async_mode: override the global async_mode settings for this task
  # enabled: true
```

```
  prompt: "prompts/claim_extraction.txt"
  description: "Any claims or facts that could be relevant to information discovery."
  max_gleanings: 0

community_report:
  ## llm: override the global llm settings for this task
  ## parallelization: override the global parallelization settings for this task
  ## async_mode: override the global async_mode settings for this task
  prompt: "prompts/community_report.txt"
  max_length: 2000
  max_input_length: 8000

cluster_graph:
  max_cluster_size: 10

embed_graph:
  enabled: false # if true, will generate node2vec embeddings for nodes
  # num_walks: 10
  # walk_length: 40
  # window_size: 2
  # iterations: 3
  # random_seed: 597832

umap:
  enabled: false # if true, will generate UMAP embeddings for nodes

snapshots:
  graphml: false
  raw_entities: false
  top_level_nodes: false

local_search:
  # text_unit_prop: 0.5
  # community_prop: 0.1
  # conversation_history_max_turns: 5
  # top_k_mapped_entities: 10
  # top_k_relationships: 10
  # max_tokens: 12000

global_search:
  # max_tokens: 12000
  # data_max_tokens: 12000
  # map_max_tokens: 1000
  # reduce_max_tokens: 2000
  # concurrency: 32
```

6. 修改 graphrag 包的代码文件，使其支持本地 ollama 的 embedding 模型来生成 GraphRAG，并且为 txt 的 embedding 模型。`vi /home/jonny/miniconda3/envs/graphrag/lib/python3.11/site-packages/graphrag/llm/openai/openai_embeddings_llm.py`

```
# Copyright (c) 2024 Microsoft Corporation.
# Licensed under the MIT License

"""The EmbeddingsLLM class."""

from typing_extensions import Unpack

from graphrag.llm.base import BaseLLM
from graphrag.llm.types import (
    EmbeddingInput,
    EmbeddingOutput,
    LLMInput,
)

from .openai_configuration import OpenAIConfiguration
from .types import OpenAIClientTypes
import ollama


class OpenAIEmbeddingsLLM(BaseLLM[EmbeddingInput, EmbeddingOutput]):
    """A text-embedding generator LLM."""

    _client: OpenAIClientTypes
    _configuration: OpenAIConfiguration

    def __init__(self, client: OpenAIClientTypes, configuration: OpenAIConfiguration):
        self.client = client
        self.configuration = configuration

    async def _execute_llm(
        self, input: EmbeddingInput, **kwargs: Unpack[LLMInput]
    ) -> EmbeddingOutput | None:
        args = {
            "model": self.configuration.model,
            **(kwargs.get("model_parameters") or {}),
        }
        #embedding = await self.client.embeddings.create(
        #    input=input,
```

```
        #    **args,
        #)
        #return [d.embedding for d in embedding.data]
        embedding_list = []
        for inp in input:
            embedding = ollama.embeddings(model="nomic-embed-text", prompt=inp)
            embedding_list.append(embedding["embedding"])
        return embedding_list

```

7. 安装必要的 langchain_community，langchain_core 包，修改 graphrag 包的代码文件，使其支持使用 ollama 部署的 embedding 模型进行知识库召回，并且为 txt 的 embedding 模型。

`pip install langchain_community langchain_core`

`vi /home/jonny/miniconda3/envs/graphrag/lib/python3.11/site-packages/graphrag/query/llm/oai/embedding.py`

```
# Copyright (c) 2024 Microsoft Corporation.
# Licensed under the MIT License

"""OpenAI Embedding model implementation."""

import asyncio
from collections.abc import Callable
from typing import Any

import numpy as np
import tiktoken
from tenacity import (
    AsyncRetrying,
    RetryError,
    Retrying,
    retry_if_exception_type,
    stop_after_attempt,
    wait_exponential_jitter,
)

from graphrag.query.llm.base import BaseTextEmbedding
from graphrag.query.llm.oai.base import OpenAILLMImpl
from graphrag.query.llm.oai.typing import (
    OPENAI_RETRY_ERROR_TYPES,
    OpenaiApiType,
)
from graphrag.query.llm.text_utils import chunk_text
from graphrag.query.progress import StatusReporter
```

```python

from langchain_community.embeddings import OllamaEmbeddings

class OpenAIEmbedding(BaseTextEmbedding, OpenAILLMImpl):
    """Wrapper for OpenAI Embedding models."""

    def __init__(
        self,
        api_key: str | None = None,
        azure_ad_token_provider: Callable | None = None,
        model: str = "text-embedding-3-small",
        deployment_name: str | None = None,
        api_base: str | None = None,
        api_version: str | None = None,
        api_type: OpenaiApiType = OpenaiApiType.OpenAI,
        organization: str | None = None,
        encoding_name: str = "cl100k_base",
        max_tokens: int = 8191,
        max_retries: int = 10,
        request_timeout: float = 180.0,
        retry_error_types: tuple[type[BaseException]] = OPENAI_RETRY_ERROR_TYPES,  # type: ignore
        reporter: StatusReporter | None = None,
    ):
        OpenAILLMImpl.__init__(
            self=self,
            api_key=api_key,
            azure_ad_token_provider=azure_ad_token_provider,
            deployment_name=deployment_name,
            api_base=api_base,
            api_version=api_version,
            api_type=api_type,  # type: ignore
            organization=organization,
            max_retries=max_retries,
            request_timeout=request_timeout,
            reporter=reporter,
        )

        self.model = model
        self.encoding_name = encoding_name
        self.max_tokens = max_tokens
        self.token_encoder = tiktoken.get_encoding(self.encoding_name)
        self.retry_error_types = retry_error_types

    def embed(self, text: str, **kwargs: Any) -> list[float]:
        """
        Embed text using OpenAI Embedding's sync function.
```

```

        For text longer than max_tokens, chunk texts into max_tokens, embed each chunk, then combine using weighted average.
        Please refer to: https://github.com/openai/openai-cookbook/blob/main/examples/Embedding_long_inputs.ipynb
        """
        token_chunks = chunk_text(
            text=text, token_encoder=self.token_encoder, max_tokens=self.max_tokens
        )
        chunk_embeddings = []
        chunk_lens = []
        for chunk in token_chunks:
            try:
                embedding, chunk_len = self._embed_with_retry(chunk, **kwargs)
                chunk_embeddings.append(embedding)
                chunk_lens.append(chunk_len)
            # TODO: catch a more specific exception
            except Exception as e:  # noqa BLE001
                self._reporter.error(
                    message="Error embedding chunk",
                    details={self.__class__.__name__: str(e)},
                )

                continue
        chunk_embeddings = np.average(chunk_embeddings, axis=0, weights=chunk_lens)
        chunk_embeddings = chunk_embeddings / np.linalg.norm(chunk_embeddings)
        return chunk_embeddings.tolist()

    async def aembed(self, text: str, **kwargs: Any) -> list[float]:
        """
        Embed text using OpenAI Embedding's async function.

        For text longer than max_tokens, chunk texts into max_tokens, embed each chunk, then combine using weighted average.
        """
        token_chunks = chunk_text(
            text=text, token_encoder=self.token_encoder, max_tokens=self.max_tokens
        )
        chunk_embeddings = []
        chunk_lens = []
        embedding_results = await asyncio.gather(*[
            self._aembed_with_retry(chunk, **kwargs) for chunk in token_chunks
        ])
```

```
        embedding_results = [result for result in embedding_results if result[0]]
        chunk_embeddings = [result[0] for result in embedding_results]
        chunk_lens = [result[1] for result in embedding_results]
        chunk_embeddings = np.average(chunk_embeddings, axis=0, weights=chunk_lens)  # type: ignore
        chunk_embeddings = chunk_embeddings / np.linalg.norm(chunk_embeddings)
        return chunk_embeddings.tolist()

    def _embed_with_retry(
        self, text: str | tuple, **kwargs: Any
    ) -> tuple[list[float], int]:
        try:
            retryer = Retrying(
                stop=stop_after_attempt(self.max_retries),
                wait=wait_exponential_jitter(max=10),
                reraise=True,
                retry=retry_if_exception_type(self.retry_error_types),
            )
            for attempt in retryer:
                with attempt:
                    embedding = (
                        #self.sync_client.embeddings.create(  # type: ignore
                        #    input=text,
                        #    model=self.model,
                        #    **kwargs,  # type: ignore
                        #)
                        #.data[0]
                        #.embedding
                        OllamaEmbeddings(
                            model=self.model,
                        ).embed_query(text)
                        or []
                    )
                    return (embedding, len(text))
        except RetryError as e:
            self._reporter.error(
                message="Error at embed_with_retry()",
                details={self.__class__.__name__: str(e)},
            )
            return ([], 0)
        else:
            # TODO: why not just throw in this case?
            return ([], 0)

    async def _aembed_with_retry(
        self, text: str | tuple, **kwargs: Any
```

```
    ) -> tuple[list[float], int]:
        try:
            retryer = AsyncRetrying(
                stop=stop_after_attempt(self.max_retries),
                wait=wait_exponential_jitter(max=10),
                reraise=True,
                retry=retry_if_exception_type(self.retry_error_types),
            )
            async for attempt in retryer:
                with attempt:
                    #embedding = (
                    #    await self.async_client.embeddings.create(  # type: ignore
                    #        input=text,
                    #        model=self.model,
                    #        **kwargs,  # type: ignore
                    #    )
                    #).data[0].embedding or []
                    embedding = (
                        await OllamaEmbeddings(
                            model=self.model,
                        ).embed_query(text) or [] )
                    return (embedding, len(text))
        except RetryError as e:
            self._reporter.error(
                message="Error at embed_with_retry()",
                details={self.__class__.__name__: str(e)},
            )
            return ([], 0)
        else:
            # TODO: why not just throw in this case?
            return ([], 0)

```

构建图知识库

1. 执行 `python -m graphrag.index --root ./ragtest`

执行完毕可以看到成功：

```
⠿ GraphRAG Indexer
├── Loading Input (InputFileType.text) - 1 files loaded (0 filtered) ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
├── create_base_text_units
├── create_base_extracted_entities
├── create_summarized_entities
├── create_base_entity_graph
├── create_final_entities
├── create_final_nodes
├── create_final_communities
├── join_text_units_to_entity_ids
├── create_final_relationships
├── join_text_units_to_relationship_ids
├── create_final_community_reports
├── create_final_text_units
├── create_base_documents
└── create_final_documents
🚀 All workflows completed successfully.
```

结果验证

1. 全局查询 `python -m graphrag.query  --root ./ragtest  --method global "What are the top themes in this story?"`

```
(graphrag) tts-trainer% python -m graphrag.query  --root ./ragtest  --method global  "What are the top themes in this story?"


INFO: Reading settings from ragtest/settings.yaml
creating llm client with {'api_key': 'REDACTED,len=6', 'type': "openai_chat", 'model': 'qwen2', 'max_tokens': 4000, 'request_timeout': 180.0, 'api_base': 'http://127.0.0.1:1
1434/v1', 'api_version': None, 'organization': None, 'proxy': None, 'cognitive_services_endpoint': None, 'deployment_name': None, 'model_supports_json': True, 'tokens_per_mi
nute': 0, 'requests_per_minute': 0, 'max_retries': 3, 'max_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True, 'concurrent_requests': 2}

SUCCESS: Global Search Response: The top themes in this story encompass several key aspects that enrich its narrative structure and deeper meanings:

1. **Personal Growth and Moral Development** (Importance Score: 90): This theme focuses on Ebenezer Scrooge's transformation from a selfish, greedy character to one who valu
es empathy, generosity, and kindness. It highlights the significance of moral development through personal experiences.

2. **Impact of Supernatural Encounters** (Importance Score: 85): The story explores how encounters with ghosts influence human behavior and societal relationships. These sup
ernatural elements add depth to Scrooge's character arc and provide a unique perspective on redemption.

3. **Significance of Family, Friendship, and Community** (Importance Score: 70): This theme underscores the importance of these social bonds in shaping individual values and
 actions. It illustrates how connections with others can lead to personal growth and moral awakening.

4. **Exploration of Time Concepts** (Importance Score: 65): Through Scrooge's encounters with ghosts representing past, present, and future, the story delves into the concep
t of time. This exploration adds layers to the narrative, emphasizing the importance of living in the moment and considering one's legacy.

5. **Critique of Capitalist Values** (Importance Score: 60): The story critiques capitalist values by contrasting Scrooge's materialistic pursuits with the virtues of empath
y and kindness. It suggests that societal values should prioritize human connection over profit.

These themes collectively contribute to a rich narrative that not only entertains but also encourages reflection on personal ethics, social relationships, and the value of t
ime in one's life.
```

2. 本地查询 `python -m graphrag.query  --root ./ragtest  --method local "Who is Scrooge, and what are his main relationships?"`

```
(graphrag) tts-trainer% python -m graphrag.query  --root ./ragtest  --method local  "Who is Scrooge, and what are his main relationships?"


INFO: Reading settings from ragtest/settings.yaml
creating llm client with {'api_key': 'REDACTED,len=6', 'type': "openai_chat", 'model': 'qwen2', 'max_tokens': 4000, 'request_timeout': 180.0, 'api_base': 'http://127.0.0.1:114
34/v1', 'api_version': None, 'organization': None, 'proxy': None, 'cognitive_services_endpoint': None, 'deployment_name': None, 'model_supports_json': True, 'tokens_per_minute
': 0, 'requests_per_minute': 0, 'max_retries': 3, 'max_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True, 'concurrent_requests': 2}
creating embedding llm client with {'api_key': 'REDACTED,len=6', 'type': "openai_embedding", 'model': 'nomic-embed-text', 'max_tokens': 4000, 'request_timeout': 180.0, 'api_ba
se': 'http://127.0.0.1:11434/api', 'api_version': None, 'organization': None, 'proxy': None, 'cognitive_services_endpoint': None, 'deployment_name': None, 'model_supports_json
': None, 'tokens_per_minute': 0, 'requests_per_minute': 0, 'max_retries': 10, 'max_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True, 'concurrent_requests': 25}

SUCCESS: Local Search Response: Scrooge is a central character from Charles Dickens' classic novel "A Christmas Carol." He is portrayed as a wealthy but miserly businessman wh
o prioritizes money over human connections. His main relationships include:

1. **Bob Cratchit**: Scrooge employs Bob as his clerk, and despite being mistreated by Scrooge, he remains loyal to him. Bob's family, especially Tiny Tim, plays a significant
 role in Scrooge's transformation.

2. **Marley**: Scrooge's former business partner who died under mysterious circumstances, leaving behind a warning about the consequences of Scrooge's actions and a ghostly ap
parition that haunts Scrooge.

3. **The Ghosts of Christmas Past, Present, and Future**: These supernatural entities are visited by Scrooge throughout the night on Christmas Eve. Each ghost serves to reveal
 different aspects of his past, present, and future life, influencing his decision to change his ways.

4. **Fred and Miss Fezziwig**: Fred is Scrooge's nephew who invites him to a festive party given by his wife, Miss Fezziwig. This event helps Scrooge see the joy in life that
he has been missing.

5. **Tiny Tim**: The disabled son of Bob Cratchit, whose suffering and eventual death deeply affect Scrooge, leading him to reconsider his cold-hearted nature.

These relationships are pivotal in Scrooge's journey from a selfish individual to a compassionate human being who learns the true meaning of Christmas through these interactio
ns.
```