AI Project

Configuration:

```yaml
services:
  mongo:
    image: mongo:latest
    container_name: "llm_engineering_mongo"
    logging:
      options:
        max-size: 1g
    environment:
      MONGO_INITDB_ROOT_USERNAME: "llm_engineering"
      MONGO_INITDB_ROOT_PASSWORD: "llm_engineering"
    ports:
      - 27017:27017
    volumes:
      - mongo_data:/data/db
    networks:
      - local
    restart: always

  qdrant:
    image: qdrant/qdrant:latest
    container_name: "llm_engineering_qdrant"
    ports:
      - 6333:6333
```

```
rag-py3.12(base) komalbagwe@10-17-99-138 RAG % /Users/komalbagwe/Library/Caches/pypoetry/virtualenvs/rag-KP5L7G78-py3.12/bin/python /Users/komalbagwe/Documents/RAG/run_tasks.py
Running command: docker-compose up -d

 Container llm_engineering_mongo  Running
 Container llm_engineering_qdrant  Running

Running command: clearml-init
ClearML SDK setup process
Configuration file already exists: /Users/komalbagwe/clearml.conf
Leaving setup, feel free to edit the configuration file.


Running command: python gradio_interface.py
```

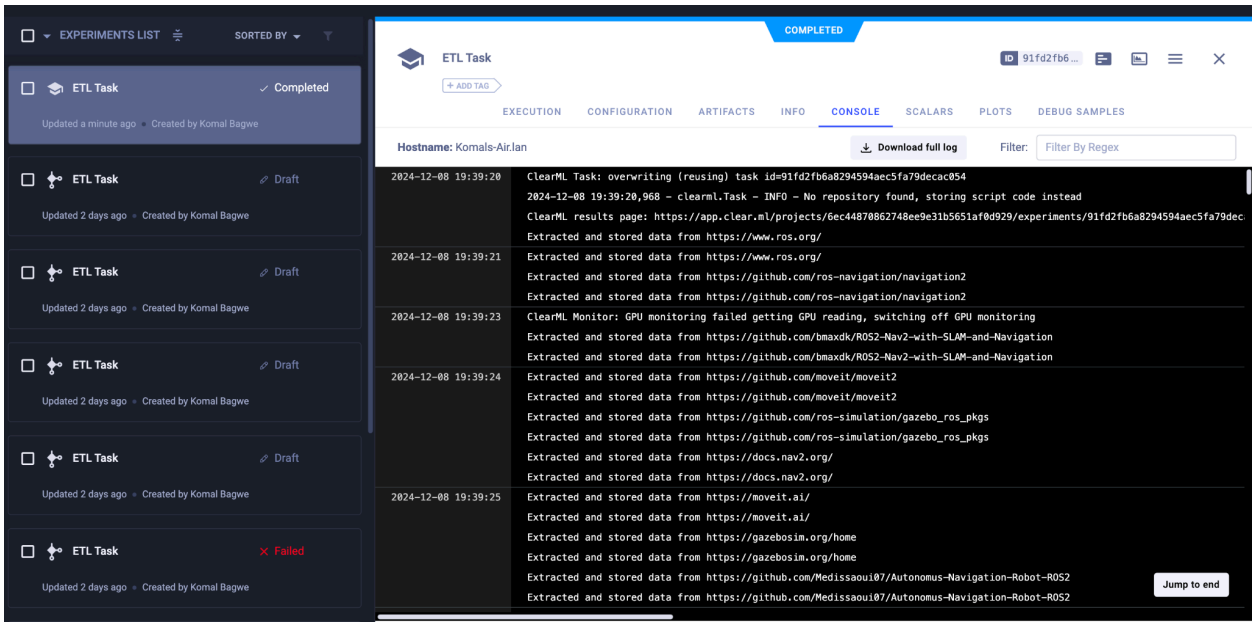
ETL Pipeline:

```python
from clearml import Task
from pymongo import MongoClient
from bs4 import BeautifulSoup
import requests
from youtube_transcript_api import YouTubeTranscriptApi
import logging

# MongoDB connection
mongo_client = MongoClient('mongodb://llm_engineering:llm_engineering@127.0.0.1:27017')
db = mongo_client['rag_system']
collection = db['raw_data']

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def etl_pipeline():
    # Initialize the task with the correct project name
    task = Task.init(project_name="RAG_System", task_name="ETL Task")
    task_logger = task.get_logger()

    def extract_ros2_docs():
        ros2_docs_urls = [
        "https://www.ros.org/",
        "https://github.com/ros-navigation/navigation2",
        "https://github.com/bmaxdk/ROS2-Nav2-with-SLAM-and-Navigation",
        "https://github.com/moveit/moveit2",
        "https://github.com/ros-simulation/gazebo_ros_pkgs",
        "https://docs.nav2.org/",
        "https://moveit.ai/",
        "https://gazebosim.org/home",
        "https://github.com/Medissaoui07/Autonomus-Navigation-Robot-ROS2",
        "https://github.com/ros-navigation/navigation2",
        "https://github.com/moveit/moveit2",
        "https://github.com/AndrejOrsula/ign_moveit2_examples/blob/master/docs/README.md",
        "https://github.com/oKermorgant/gz_moveit2_examples",
        "https://medium.com/schmiedeone/getting-started-with-ros2-part-1-d4c3b7335c71",
        "https://medium.com/@tetraengnrng/a-beginners-guide-to-ros2-29721dcf49c8",
```

```python
    def extract_ros2_docs():
        "https://medium.com/robotics-zone/understanding-moveit2-and-ros-2-for-robotics-61d74832cd2a",
        "https://medium.com/ros2-basics/building-your-own-ros2-navigation-stack-from-scratch-cc9c6b6a32e6",
        "https://medium.com/robotics-zone/introducing-ros2-and-gazebo-simulation-for-robotics-3e4054d4f8f8",
        "https://medium.com/robotics-zone/using-ros2-and-gazebo-to-simulate-robots-in-a-vibrant-world-34a6a4f35b28"

]

        for url in ros2_docs_urls:
            try:
                response = requests.get(url)
                soup = BeautifulSoup(response.content, 'html.parser')
                text = soup.get_text()
                data = {'source': 'github', 'url': url, 'content': text}
                collection.insert_one(data)
                task_logger.report_text(f"Extracted and stored data from {url}")
                print(f"Extracted and stored data from {url}")  # Use standard logger
            except Exception as e:
                task_logger.report_text(f"Failed to extract data from {url}: {e}")
                print(f"Failed to extract data from {url}: {e}")  # Use standard logger

    def extract_youtube_videos():
        youtube_video_ids = ['7TVWlADXwRw', 'rtrGoGsMVlI&list=PLgG0XDQqJckkSJDPhXsFU_RIqEh08nG0V',"sVUKeHMBtpQ","Xbij9Tst-WA","jkoGkAd0GYk","QI

        for video_id in youtube_video_ids:
            try:
                transcript = YouTubeTranscriptApi.get_transcript(video_id)
                transcript_text = ' '.join([t['text'] for t in transcript])
                data = {'source': 'youtube', 'url': f"https://www.youtube.com/watch?v={video_id}", 'content': transcript_text}
                collection.insert_one(data)
                task_logger.report_text(f"Extracted and stored data from YouTube video {video_id}")
                print(f"Extracted and stored data from YouTube video {video_id}")
            except Exception as e:
                task_logger.report_text(f"Failed to extract data from YouTube video {video_id}: {e}")
                print(f"Failed to extract data from YouTube video {video_id}: {e}")
```

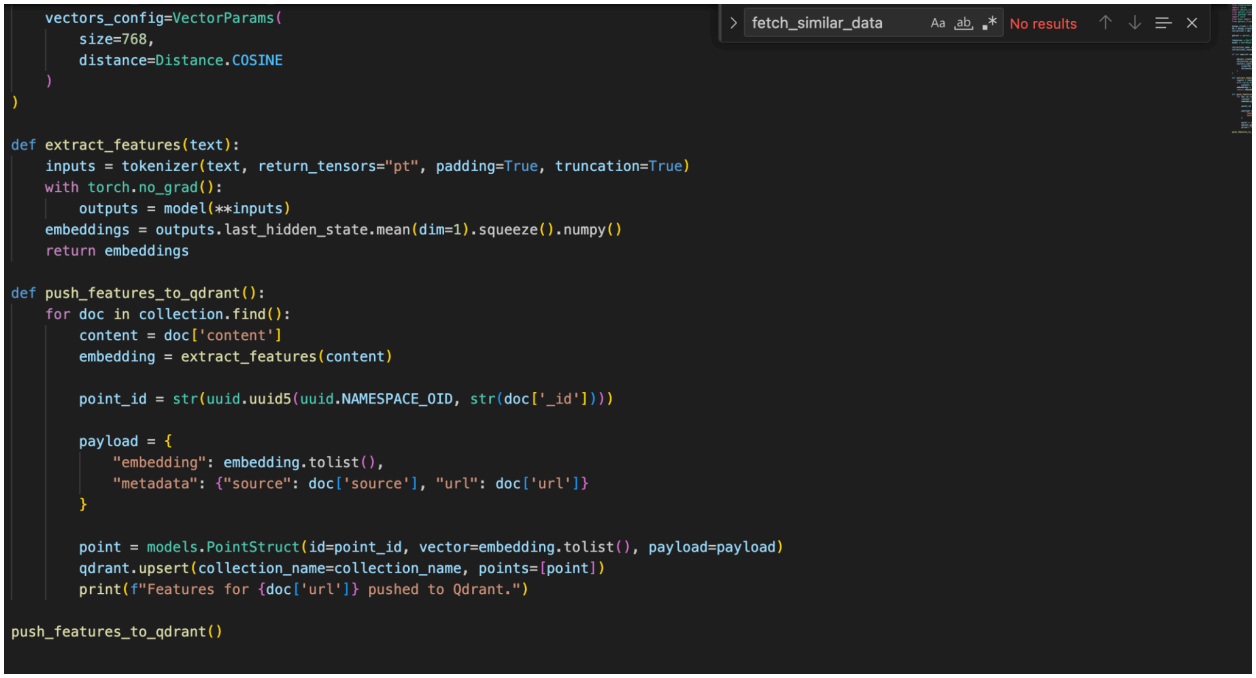
```
    def extract_ros2_docs():
        "https://medium.com/robotics-zone/understanding-moveit2-and-ros-2-for-robotics-61d74832cd2a",
        "https://medium.com/ros2-basics/building-your-own-ros2-navigation-stack-from-scratch-cc9c6b6a32e6",
        "https://medium.com/robotics-zone/introducing-ros2-and-gazebo-simulation-for-robotics-3e4054d4f8f8",
        "https://medium.com/robotics-zone/using-ros2-and-gazebo-to-simulate-robots-in-a-vibrant-world-34a6a4f35b28"

]

        for url in ros2_docs_urls:
            try:
                response = requests.get(url)
                soup = BeautifulSoup(response.content, 'html.parser')
                text = soup.get_text()
                data = {'source': 'github', 'url': url, 'content': text}
                collection.insert_one(data)
                task_logger.report_text(f"Extracted and stored data from {url}")
                print(f"Extracted and stored data from {url}")  # Use standard logger
            except Exception as e:
                task_logger.report_text(f"Failed to extract data from {url}: {e}")
                print(f"Failed to extract data from {url}: {e}")  # Use standard logger

    def extract_youtube_videos():
        youtube_video_ids = ['7TVWlADXwRw', 'rtrGoGsMVlI&list=PLgG0XDQqJckkSJDPhXsFU_RIqEh08nG0V',"sVUKeHMBtpQ","Xbij9Tst-WA","jkoGkAd0GYk","QI

        for video_id in youtube_video_ids:
            try:
                transcript = YouTubeTranscriptApi.get_transcript(video_id)
                transcript_text = ' '.join([t['text'] for t in transcript])
                data = {'source': 'youtube', 'url': f"https://www.youtube.com/watch?v={video_id}", 'content': transcript_text}
                collection.insert_one(data)
                task_logger.report_text(f"Extracted and stored data from YouTube video {video_id}")
                print(f"Extracted and stored data from YouTube video {video_id}")
            except Exception as e:
                task_logger.report_text(f"Failed to extract data from YouTube video {video_id}: {e}")
                print(f"Failed to extract data from YouTube video {video_id}: {e}")
```

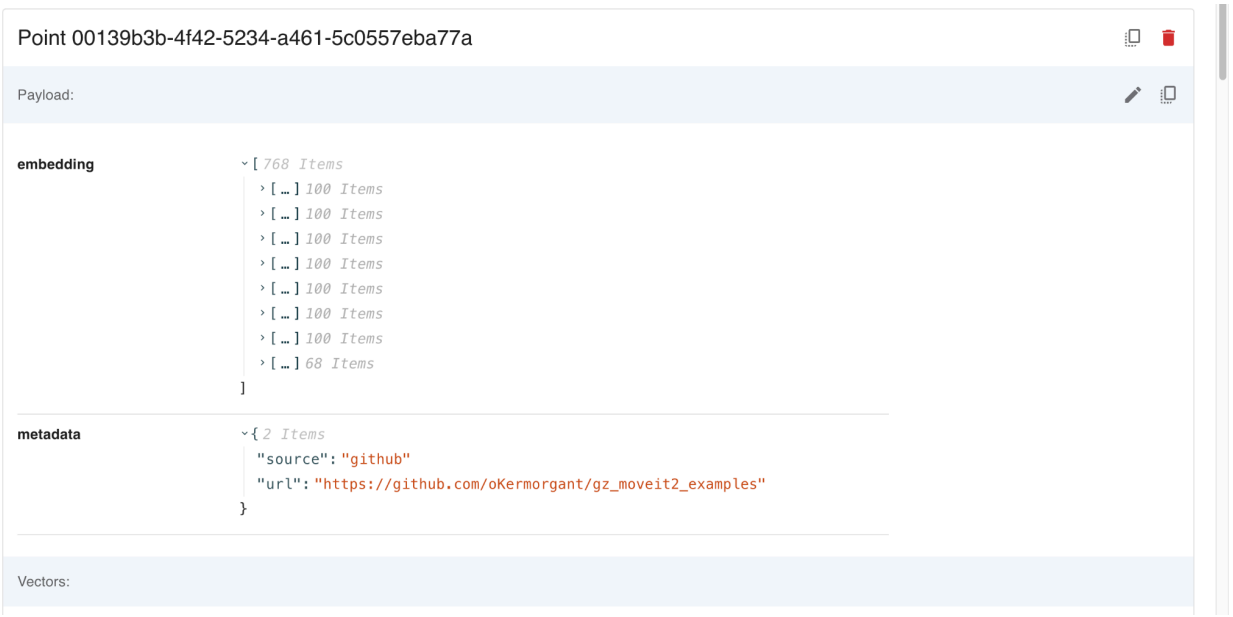
```
def etl_pipeline():
    def extract_youtube_videos():
                print(f"Failed to extract data from YouTube video {video_id}: {e}")


    def print_ingested_urls():
        urls = [doc['url'] for doc in collection.find()]
        print("Ingested URLs:")
        print("Ingested URLs:")
        for url in urls:
            logger.info(url)
            print(url)

    # Execute ETL steps
    extract_ros2_docs()
    extract_youtube_videos()
    print_ingested_urls()

    task.close()

# Directly execute the ETL pipeline locally
if __name__ == "__main__":
    etl_pipeline()
```



Feature Engineering:

```
fetch_similar_data  Aa  ab  .*  No results

from transformers import BertTokenizer, BertModel
import torch
import qdrant_client
from qdrant_client import models
from pymongo import MongoClient
import uuid
from bson import ObjectId
from qdrant_client.models import VectorParams, Distance

mongo_client = MongoClient('mongodb://llm_engineering:llm_engineering@127.0.0.1:27017')
db = mongo_client['rag_system']
collection = db['raw_data']

qdrant = qdrant_client.QdrantClient("http://127.0.0.1:6333")

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

collection_name = "raw_data"
collections_response = qdrant.get_collections()

if not any(col.name == collection_name for col in collections_response.collections):

    qdrant.create_collection(
    collection_name=collection_name,
    vectors_config=VectorParams(
        size=768,
        distance=Distance.COSINE
    )
)

def extract_features(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
    with torch.no_grad():
        outputs = model(**inputs)
    embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
    return embeddings
```

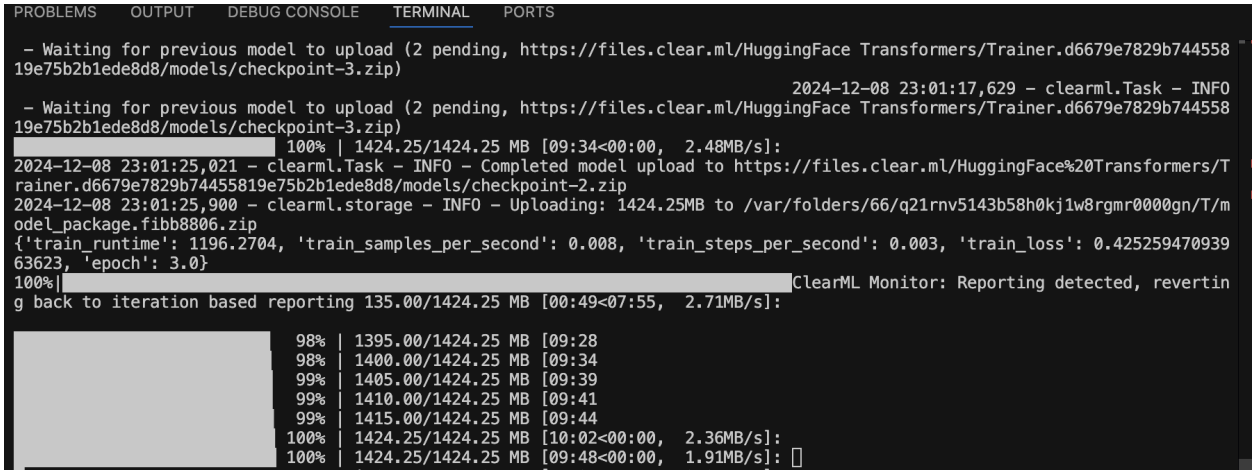

fetch_similar_data
No results
vectors_config=VectorParams(
size=768,
distance=Distance.COSINE
)
)
def extract_features(text):
inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
with torch.no_grad():
outputs = model(**inputs)
embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
return embeddings
def push_features_to_qdrant():
for doc in collection.find():
content = doc['content']
embedding = extract_features(content)
point_id = str(uuid.uuid5(uuid.NAMESPACE_OID, str(doc['_id'])))
payload = {
"embedding": embedding.tolist(),
"metadata": {"source": doc['source'], "url": doc['url']}
}
point = models.PointStruct(id=point_id, vector=embedding.tolist(), payload=payload)
qdrant.upsert(collection_name=collection_name, points=[point])
print(f"Features for {doc['url']} pushed to Qdrant.")
push_features_to_qdrant()


Point 00139b3b-4f42-5234-a461-5c0557eba77a
Payload:
embedding
[768 Items
[…] 100 Items
[…] 100 Items
[…] 100 Items
[…] 100 Items
[…] 100 Items
[…] 100 Items
[…] 100 Items
[…] 68 Items
]
metadata
{ 2 Items
"source": "github"
"url": "https://github.com/oKermorgant/gz_moveit2_examples"
}
Vectors:

FineTuning:

```
    vectors_config=VectorParams(
        size=768,
        distance=Distance.COSINE
    )
)

def extract_features(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
    with torch.no_grad():
        outputs = model(**inputs)
    embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
    return embeddings

def push_features_to_qdrant():
    for doc in collection.find():
        content = doc['content']
        embedding = extract_features(content)

        point_id = str(uuid.uuid5(uuid.NAMESPACE_OID, str(doc['_id'])))

        payload = {
            "embedding": embedding.tolist(),
            "metadata": {"source": doc['source'], "url": doc['url']}
        }

        point = models.PointStruct(id=point_id, vector=embedding.tolist(), payload=payload)
        qdrant.upsert(collection_name=collection_name, points=[point])
        print(f"Features for {doc['url']} pushed to Qdrant.")

push_features_to_qdrant()
```

```
    learning_rate=2e-5,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=16,
    num_train_epochs=3,
    weight_decay=0.01,
    save_strategy="epoch",
    logging_dir="./logs",
    logging_steps=10,
    fp16=False,
    report_to="clearml",
    optim="adamw_torch",
    max_grad_norm=0.3,
)

torch.cuda.empty_cache()

model.gradient_checkpointing_enable()

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    data_collator=data_collator,
)

try:
    trainer.train()

    model.save_pretrained("./fine_tuned_llama2")
    tokenizer.save_pretrained("./fine_tuned_llama2")
    print("Fine-tuned model saved successfully.")
except RuntimeError as e:
    print(f"An error occurred during training: {e}")
    print("Consider further reducing batch size, sequence length, or using a smaller model.")
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
– Waiting for previous model to upload (2 pending, https://files.clear.ml/HuggingFace Transformers/Trainer.d6679e7829b74455819e75b2b1ede8d8/models/checkpoint-3.zip)
2024-12-08 23:01:17,629 – clearml.Task – INFO
– Waiting for previous model to upload (2 pending, https://files.clear.ml/HuggingFace Transformers/Trainer.d6679e7829b74455819e75b2b1ede8d8/models/checkpoint-3.zip)
100% | 1424.25/1424.25 MB [09:34<00:00, 2.48MB/s]:
2024-12-08 23:01:25,021 – clearml.Task – INFO – Completed model upload to https://files.clear.ml/HuggingFace%20Transformers/Trainer.d6679e7829b74455819e75b2b1ede8d8/models/checkpoint-2.zip
2024-12-08 23:01:25,900 – clearml.storage – INFO – Uploading: 1424.25MB to /var/folders/66/q21rnv5143b58h0kj1w8rgmr0000gn/T/model_package.fibb8806.zip
{'train_runtime': 1196.2704, 'train_samples_per_second': 0.008, 'train_steps_per_second': 0.003, 'train_loss': 0.42525947093963623, 'epoch': 3.0}
100%|
ClearML Monitor: Reporting detected, reverting back to iteration based reporting 135.00/1424.25 MB [00:49<07:55, 2.71MB/s]:
98% | 1395.00/1424.25 MB [09:28
98% | 1400.00/1424.25 MB [09:34
99% | 1405.00/1424.25 MB [09:39
99% | 1410.00/1424.25 MB [09:41
99% | 1415.00/1424.25 MB [09:44
100% | 1424.25/1424.25 MB [10:02<00:00, 2.36MB/s]:
100% | 1424.25/1424.25 MB [09:48<00:00, 1.91MB/s]: