RAG Lab Setup Guide/Manual

Noah Opela

05/07/2025

1 Introduction

Contents

2 How to Install and Use Chainlit and VLLM

2.1 Chainlit Install

1. How to install chainlit:

   ```
   pip install chainlit
   ```

2. To test if it worked type the following command in the terminal:

   ```
   chainlit hello
   ```

3. This will automatically bring up a webpage with Chainlit AI working. You might have to refresh the page a time or two since it activates before the AI fully works.

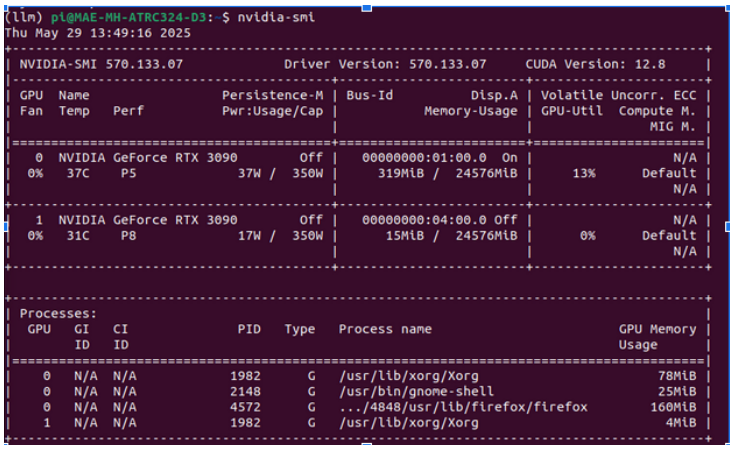
2.2 VLLM Install

1. How to install VLLM: Use either `conda` or `uv` (here, conda is used).
2. Commands:
   - `conda create -n (example_name) python=3.12 -y`
   - `conda activate (example_name)`
   - `pip install --upgrade uv`
   - `uv pip install vllm --torch-backend=auto`
3. It will sign into conda automatically.
4. To configure your GPU (Linux instructions):
   - Open your application search (for example, by pressing the Windows button or your favorite macro), type `Activities`, and select `Software and Updates`.
   - Go to the `Driver` tab and select the appropriate NVIDIA driver (e.g., `nvidia-driver-570`).

5. Restart your computer. After booting, in the terminal run:

   ```
   nvidia-smi
   ```

   This should display your NVIDIA GPU information.

   ```
   (llm) pi@MAE-MH-ATRC324-D3:~$ nvidia-smi
   Thu May 29 13:49:16 2025
   +-----------------------------------------------------------------------------------------+
   | NVIDIA-SMI 570.133.07             Driver Version: 570.133.07     CUDA Version: 12.8     |
   |-----------------------------------------+------------------------+----------------------+
   | GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
   | Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
   |                                         |                        |               MIG M. |
   |=========================================+========================+======================|
   |   0  NVIDIA GeForce RTX 3090        Off |   00000000:01:00.0  On |                  N/A |
   |  0%   37C    P5             37W /  350W |     319MiB /  24576MiB |     13%      Default |
   |                                         |                        |                  N/A |
   +-----------------------------------------+------------------------+----------------------+
   |   1  NVIDIA GeForce RTX 3090        Off |   00000000:04:00.0 Off |                  N/A |
   |  0%   31C    P8             17W /  350W |      15MiB /  24576MiB |      0%      Default |
   |                                         |                        |                  N/A |
   +-----------------------------------------+------------------------+----------------------+

   +-----------------------------------------------------------------------------------------+
   | Processes:                                                                              |
   |  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
   |        ID   ID                                                               Usage      |
   |=========================================================================================|
   |    0   N/A  N/A             1982      G   /usr/lib/xorg/Xorg                       78MiB |
   |    0   N/A  N/A             2148      G   /usr/bin/gnome-shell                     25MiB |
   |    0   N/A  N/A             4572      G   .../4848/usr/lib/firefox/firefox        160MiB |
   |    1   N/A  N/A             1982      G   /usr/lib/xorg/Xorg                        4MiB |
   +-----------------------------------------------------------------------------------------+
   ```

   This is what mine looked like.

6. Install (or upgrade) PyTorch libraries with proper CUDA support:

   ```
   pip install --upgrade torch torchvision torchaudio --index-url
   https://download.pytorch.org/whl/cuXXX
   ```

   Replace `cuXXX` with the most up-to-date version (e.g., `cu128` for CUDA 12.8).

7. When coding your Chainlit `app.py`, specify the client to connect to VLLM:

   ```
   client = AsyncOpenAI(base_url="http://localhost:8080/v1",
   api_key="token-ATRC324-IHL")
   ```

   This connects Chainlit to the VLLM API for information retrieval.

2.3 How to Code Chainlit's Config `app.py`

1. Chainlit's website provides a template. See the link:
   `https://docs.chainlit.io/integrations/openai`

2. There are several modifications required in the template.

3. Update the client configuration to:

   ```
   client = AsyncOpenAI(base_url="http://localhost:8080/v1",
   api_key="token-ATRC324-IHL")
   ```

4. Change the settings to specify your desired model:

   ```
   settings = {"model": "Qwen/Qwen2.5-Coder-1.5B-Instruct"}
   ```

   Use the appropriate model name installed on VLLM.

2.4 How to Run Chainlit and VLLM

1. Start the VLLM server. Ensure it is hosting on a port such as `localhost:8080` (and note: it must be different than Chainlit's default `localhost:8000`).

   ```
       vllm serve Qwen/Qwen2.5-Coder-1.5B-Instruct --port 8080
       --api-key token-ATRC324-IHL --gpu-memory-utilization .9
   ```

2. Next, run Chainlit by navigating to the directory containing your `app.py` and typing:

   ```
   chainlit run app.py -w
   ```

   The `-w` flag will open your web browser automatically. If not, navigate manually to `localhost:8000`.

2.5 Common Problems

- One common issue involves sending data to the wrong endpoint.
- Ensure that the VLLM URL in your client includes the full path. For example, use:

  ```
  http://localhost:8080/v1
  ```

  not just `http://localhost:8080`, which causes a 404 error.
- If Chainlit cannot locate `app.py`, verify that you are in the correct directory when running the command.

3 RAG Setup

This section shows how to integrate Retrieval–Augmented Generation (RAG) into your Chainlit application for document-based file retrieval.

3.1 Dependencies Installation

Install all required packages in one go. These provide llama-index core, OpenAI-style LLM wrappers, embeddings, and Langfuse integration:

Listing 1: Install RAG-related Python packages via pip

```
pip install \
  llama_index.core \
  llama_index.llms.openai_like \
  llama_index.embeddings.openai_like \
  langfuse.llama_index
```

3.2 Python Imports and Global Settings

Import the core modules, set your global defaults, and explain each:

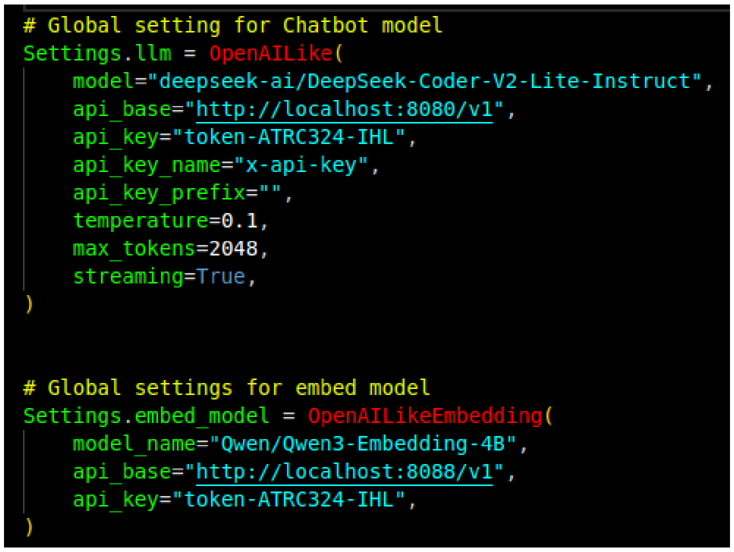
Listing 2: Key Python imports for RAG

```
# Core index and storage utilities
from llama_index.core import (
    settings,                  # global configuration for llama_index
    StorageContext,            # handles persistence directories
    SimpleDirectoryReader,     # reads files from a directory
    load_index_from_storage
)

# Vector store index builder
from llama_index.core import VectorStoreIndex

# OpenAI-like LLM and embedding wrappers
from llama_index.llms.openai_like import OpenAILike
from llama_index.embeddings.openai_like import OpenAILikeEmbedding

# Callback handling for Langfuse integration
from llama_index.core.callbacks import CallbackManager
from langfuse.llama_index import LlamaIndexCallbackHandler
```

Next, configure your global defaults once at startup (replace keys with your own):

Listing 3: Global LLM and Embedding Settings

```
settings.configure_llm(OpenAILike, api_key="YOUR_OPENAI_KEY")
settings.configure_embed(OpenAILikeEmbedding, api_key="YOUR_OPENAI_KEY")
```



```
# Global setting for Chatbot model
Settings.llm = OpenAILike(
    model="deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct",
    api_base="http://localhost:8080/v1",
    api_key="token-ATRC324-IHL",
    api_key_name="x-api-key",
    api_key_prefix="",
    temperature=0.1,
    max_tokens=2048,
    streaming=True,
)


# Global settings for embed model
Settings.embed_model = OpenAILikeEmbedding(
    model_name="Qwen/Qwen3-Embedding-4B",
    api_base="http://localhost:8088/v1",
    api_key="token-ATRC324-IHL",
)
```

Figure 1: Global defaults for the OpenAI-like LLM and embedding model in llama_index.

3.3 Storage Directory Setup

Define where raw data lives and where the index will be persisted:

Listing 4: Data and persistence directories

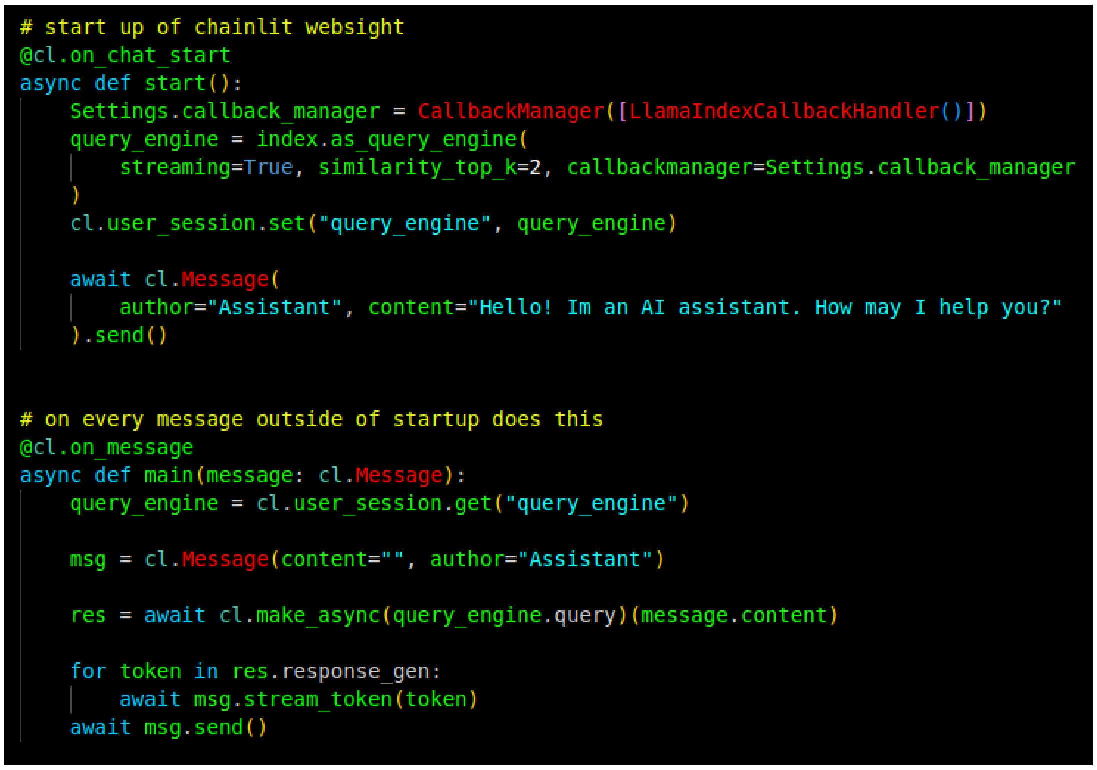
```
DATA_DIR    = "./data"     % Directory containing your source documents
PERSIST_DIR = "./storage"  % Directory for saving the built index
```

3.4 Index Loading with Reindex on Change

Use a try/except to reuse the existing index or rebuild if storage is missing or outdated:

Listing 5: Load or rebuild the VectorStoreIndex

```
try:
    # Attempt to load an existing index
    storage_context = StorageContext.from_defaults(persist_dir=PERSIST_DIR)
    index = load_index_from_storage(storage_context)

except (FileNotFoundError, FileExistsError):
    # If no index exists yet, read raw docs and build one
    docs = SimpleDirectoryReader(DATA_DIR).load_data(show_progress=True)
    index = VectorStoreIndex.from_documents(docs)
    index.storage_context.persist()
```



```
# start up of chainlit websight
@cl.on_chat_start
async def start():
    Settings.callback_manager = CallbackManager([LlamaIndexCallbackHandler()])
    query_engine = index.as_query_engine(
        streaming=True, similarity_top_k=2, callbackmanager=Settings.callback_manager
    )
    cl.user_session.set("query_engine", query_engine)

    await cl.Message(
        author="Assistant", content="Hello! Im an AI assistant. How may I help you?"
    ).send()


# on every message outside of startup does this
@cl.on_message
async def main(message: cl.Message):
    query_engine = cl.user_session.get("query_engine")

    msg = cl.Message(content="", author="Assistant")

    res = await cl.make_async(query_engine.query)(message.content)

    for token in res.response_gen:
        await msg.stream_token(token)
    await msg.send()
```

Figure 2: Example Chainlit chat settings—customize styles, message layouts, etc.

4 VLLM RAG Settings

Chainlit can orchestrate multiple AI backends via vllm. Below are two sample config files—one for embeddings, one for the main model.

4.1 Embedding Host Configuration

Listing 6: vllm config for the embedding service

```
host: "127.0.0.1"
port: 8088
api_key: "token-ATRC324-IHL"
gpu_memory_utilization: 0.15
tensor_parallel_size: 2
max_model_len: 15000
```

4.2 Main LLM Configuration

Listing 7: vllm config for the primary LLM

```
host: "127.0.0.1"
port: 8080
api_key: "token-ATRC324-IHL"
gpu_memory_utilization: 0.75
tensor_parallel_size: 2
max_model_len: 20000
trust_remote_code: true
```

4.3 Starting vllm Services

Launch each service with:

Listing 8: Run vllm serving commands

```
vllm serve embedding_model_name --config embedding_config.yaml
vllm serve main_model_name      --config model_config.yaml
```

This will spin up separate endpoints for embeddings and completion, which Chainlit will then consume in your RAG pipeline.

4.4 RAG Reranking

There are a few methods you can use, but the one used in this app is based off of `LLMRerank` module found in `llama_index.core.postprocessor`.

This reranker works by using the llm itself to generate answeres using the documents with highest top_k then ask itself how it would rate its own answeres then the top_k answers from that are then used to help it answer the question. You can then go from there and train it by giving it specific phrases and answers as a grading guide for future questions.

A few rerankers to look into are:

```
Cross-Encoding - Higher accuracy but slower.
Multi-Vector - Efficient with massive documentation not as accurate.
```

4.5 Where to go From Here

4.5.1 Fine-tuning

The llm process and llm to give us a more accurate answer This can be done by modifying the llm, and the reranker, so that way they will produce answers allong the lines of what both the user and hoster want.

4.5.2 Optimization

This allows for the process aas a whole to run more efficient and faster. This can be done by changing models, and how data is handles.