Working with LLMs

Local LLMs

LLM Utility

I'm a big fan of Simon Willison's llm package. It works nicely with llama-cpp.

Installing `llm`

I didn't get on well with pipx in this use case so I used conda to create a virtual environments for LLM and then installed it in there.

Since I have an NVIDIA card I pass in CMAKE flags to have it build support for cuda:

```
conda create -y -n llm python=3.10
conda activate llm
pip install llm llm-llama-cpp
CMAKE_ARGS="-DLLAMA_CUBLAS=ON -DCMAKE_CUDA_COMPILER=/usr/local/cuda/bin/nvcc" FORCE_CMAKE=1
llm install llama-cpp-python

# alternatively if no NVIDIA support is available, this works well
# CMAKE_ARGS="-DLLAMA_OPENBLAS=on" FORCE_CMAKE=1 llm install llama-cpp-python
```

LangChain

LangChain is a FOSS library for chaining together prompt-able language models. I've been using it for building all sorts of cool stuff.

LangChain and Zephyr

Zephyr is pretty powerful and it will quite happily use tools if you prompt it correctly.

Zephyr uses the following prompt template (as explained here):

```
<|system|>
</s>
<|user|>
{prompt}</s>
<|assistant|>
```

The system prompt is defined, followed by a user query/request and then we use `<|assistant|>` to prompt the model to start generating its own output.

Tool Prompt

Here is a tool prompt that I've managed to get working with Zephyr based on the original guide here and corresponding langchainhub prompt here. The interesting and key thing seems to be reminding the model to consider the inputs for the next action on line 23. Without that it would always try to run an action without any inputs.

```
<|system|>

Respond to the human as helpfully and accurately as possible. You have access to the following
tools:

{tools}

Use a json blob to specify a tool by providing an action key (tool name) and an action_input
key (tool input).

Valid "action" values: "Final Answer" or {tool_names}

Provide only ONE action per $JSON_BLOB, as shown:

```
{{
```
```

```
  "action": $TOOL_NAME,
  "action_input": $INPUT
}}
```

Follow this format:

Question: input question to answer
Thought: consider previous and subsequent steps. Consider inputs needed for next action.
Action:
```
$JSON_BLOB
```
Observation: action result
... (repeat Thought/Action/Observation N times)
Thought: I know what to respond
Action:
```
{{
  "action": "Final Answer",
  "action_input": "Final response to human"
}}
```

Begin! Reminder to ALWAYS respond with a valid json blob of a single action.

Use tools if necessary.

Respond directly if appropriate.

always pass appropriate values for `action_input` based on the tools defined above.

Format is Action:```$JSON_BLOB```then Observation

Previous conversation history:
{chat_history}
</s>
Question: {input}
{agent_scratchpad}

Embeddings and Llama.cpp

SQLite VSS - Lightweight Vector DB

SQLite VSS is a SQLite extension that adds vector search on top of SQLite. It's based on FAISS[1]

There are some examples of how to use Pure SQLite VSS on the blog post here

LangChain

You can use SQLite VSS with Langchain which makes it easier to use. The documentation is here for sqlite-vss and here for using llama for embedding.

You need to install `sqlite-vss` python package to use it via `pip install sqlite-vss`

Zephyr embeddings

Load the zephyr model with long contet and set gpu layers up.

```
llama = LlamaCppEmbeddings(model_path="/path/to/models/zephyr-7b-alpha.Q5_K_M.gguf",
            n_batch=512,
            verbose=True, # Verbose is required to pass to the callback manager
            n_ctx=16000,
            n_gpu_layers=32)
```

NB: I found that Zephyr isn't actually very good for generating embeddings - I suppose this is likely because it is fine-tuned for chatting rather than for embedding.

It actually turns out that the default MiniLM that comes with sentence-transformers does a pretty reasonable job:

```
embedding_function = SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-v2")
```

PyLLMCore

PyLLMCore is a python library for working with a variety of LLM models and it supports both OpenAI and Local models.

Setup on Linux

Install the `llama-cpp-python` library first so that you can ensure that the nvidia dependencies are all pre-configured.

```
CMAKE_ARGS="-DLLAMA_CUBLAS=ON -DCMAKE_CUDA_COMPILER=/usr/local/cuda/bin/nvcc" pip install
llama-cpp-python
pip install py-llm-core
```

Put models in the correct location

The library seems quite fussy about model location. They must be in the `~/.cache/py-llm-core/models/` folder inside your user profile. Since I am already using SimonW's LLM (as described here) I symlink the zephyr model from there:

```
ln -s ~/.config/io.datasette.llm/llama-cpp/models/zephyr-7b-alpha.Q5_K_M.gguf\
 ~/.cache/py-llm-core/models/zephyr-7b-alpha.Q5_K_M.gguf
```

I realised I had done this wrong because I passed in a full filename to a model elsewhere and got an error like this:

```
Traceback (most recent call last):
  File "/home/james/workspace/raf/llmcore.py", line 24, in <module>
    book = parser.parse(text)
  File "/home/james/miniconda3/envs/raf/lib/python3.10/site-packages/llm_core/parsers.py",
line 20, in parse
    completion = self.model_wrapper.ask(
  File "/home/james/miniconda3/envs/raf/lib/python3.10/site-
packages/llm_core/llm/llama_cpp_compatible.py", line 65, in ask
    self.sanitize_prompt(prompt=prompt, history=history, schema=schema)
  File "/home/james/miniconda3/envs/raf/lib/python3.10/site-packages/llm_core/llm/base.py",
line 29, in sanitize_prompt
    required_ctx_size = len(codecs.encode(complete_prompt, self.name))
```

```
LookupError: unknown encoding: /home/james/.config/io.datasette.llm/llama-cpp/models/zephyr-7b-alpha.Q5_K_M.gguf
```