STEVENS
INSTITUTE OF TECHNOLOGY
1870

FE524
Week 12

Week 11 Recap: Prompting

- We talked about open LLMs and how to run them locally by:
  - downloading model weights (and converting them if needed)
  - building llama.cpp, a tool for running model weights
  - using llama.cpp's llama-cli with the model weights to run individual prompts at the command line
  - running llama-cli from Python using the subprocess package

Downloading Model Files

- To download model weights or the llama.cpp source we'll use git
  - For model weights, we first need to set up Git LFS:

    ```
    git lfs install
    ```
  - To download:

    ```
    git clone <url>
    ```
- You'll need to request approval for gated models like the Llama models and authenticate with a Hugging Face account when you `git clone`
  - When you run `git clone`, you'll be prompted for a username and "password"
  - The username is your Hugging Face username
  - The "password" is a Hugging Face token

llama.cpp, Build

- `git clone https://github.com/ggml-org/llama.cpp.git`
- Building:
  - https://github.com/ggml-org/llama.cpp/blob/master/docs/build.md
    - cmake -B build
    - cmake --build build --config Release
- When building software, you may need to debug the build process, for example on MacOS:
  - https://github.com/ggml-org/whisper.cpp/issues/2841
- Pre-built llama.cpp binaries can be found on their GitHub page
  - https://github.com/ggml-org/llama.cpp/releases

llama.cpp, Setup

- At this point, we've downloaded our model files (typically .safetensors) and have a running version of llama.cpp
- We need to convert our models from .safetensors to .gguf
  - The llama.cpp comes with a script to do this, but it has dependencies that you need to install and a requirements.txt in which those dependencies are stored
  - We'll run, in order:
    - `python -m pip install –r requirements.txt`
    - `python convert_hf_to_gguf.py /path/to/model/directory`
  - It may be helpful to abstract your dependencies into a virtual environment, in which case you'd run the following before the above two commands:
    - `$ python -m venv /path/to/virtual/env/dir`
    - `$ . /path/to/virtual/env/dir/bin/activate`
    - `(venv) $ …do stuff…`
    - `(venv) $ deactivate`
    - `$ _`

llama.cpp: Using llama-cli and llama-server

- At this point, we have a .gguf model file and a running version of llama.cpp
- I'll use Llama-3.2-1B-Instruct as an example – it's a small model that I can easily run on my laptop (Macbook Air M1)
- Last class we ran a prompt using llama.cpp's CLI tool:
  - `./build/bin/llama-cli -m models/Llama-3.2-1B-Instruct/Llama-3.2-1B-Instruct-F16.gguf -p "How are you?"`
- `llama.cpp` also has a server you can run locally to stand up a web server with an API compatible with OpenAI's Chat Completions API
  - `./build/bin/llama-server -m models/Llama-3.2-1B-Instruct/Llama-3.2-1B-Instruct-F16.gguf --port 8080`

subprocess

- In Homework 11, we used the `subprocess` package from Python's Standard Library to call `llama-cli` and run a prompt
- `subprocess` lets us shell commands or binaries from Python
- Like the command line, `subprocess` prints output by default
- When `capture_output=True`, `subprocess.run(…)` will return a object with `stdout` and `stderr` attributes that contain the standard output and any error output, respectively
  - We can also use the `stdout` and `stderr` parameters in `run()` to redirect output to a file
  - We can also pass a timeout duration in seconds with the `timeout` parameter
    - To cleanly using a timeout, you could use:

      ```
      try: … except subprocess.TimeoutExpired: pass
      ```

llama.cpp: Using llama-server

- `./build/bin/llama-server -m models/Llama-3.2-1B-Instruct/Llama-3.2-1B-Instruct-F16.gguf --port 8080`
  - After running this at the command line, a server will be listening for connections on localhost:8080
  - "localhost" is hostname that will resolve to a loopback IP address, which means we can connect to it from other processes, like a running Python script, on our device
  - The code that we used early in the semester to call the GPT API using `requests` can be run against `llama-server` with minimal modification
    - We need to change the root URL from `https://api.openai.com` to `http://localhost:8080` – note the change in protocol from HTTPS to HTTP!
    - We should remove any use of `load_dotenv()` or our OpenAI API key
  - Any code that uses the `openai` package should first be converted to use `requests`

Late Homework Note

- If there are homework assignments that you haven't submitted, there is an opportunity to make them up
- This is the first time we're running this course, and our aim to be empathetic
- Please contact me through Canvas or by email to work out the logistics of submitting late homework assignments
- (Do send `.py` files by email – Outlook will block them.)

HW12: Local Open LLM APIs

- Install llama.cpp
- Download appropriately small model weights from Hugging Face or another source
- Convert the model weights to .gguf
- Launch a llama-server instance
- Convert your code from Homework 6 to run against the llama-server API
- Don’t worry too much about the correctness of the output – just experiment and work to get the output to at least be a valid CSV
- Submit both the script and a screenshot of its invocation and output, preferably as a .png file

STEVENS
INSTITUTE OF TECHNOLOGY
1870

END