Lab 7 - Chatbots, Llama & llama.cpp
AI Programming for Games
COMP10068

Paul Keir

Date Issued: March 4, 2025
Instructor: Dr. Paul Keir

In this week's lab we will explore how to interact programmatically with the llama.cpp API, eventually using it to create our own chatbot.

1 Download a GGUF model from Hugging Face

Hugging Face provides hundreds of Large Language Models (LLMs) to download. We will work with models in the GGUF (GPT-Generated Unified Format) format.

1. Visit TheBloke's Llama-2-7B-Chat-GGUF page here;
2. Click the "Files and versions" tab;
3. Download llama-2-7b-chat.Q4_K_M.gguf (4.08GB).

That may take some time to download. Note the path of the location download. Perhaps it is `%USERPROFILE%\Downloads`?

2 Build "llama.cpp"

The GGUF LLM from Hugging Face does nothing on its own. To interact with the LLM downloading above, we will build and run Georgi Gerganov's "llama.cpp" project. You are provided with a recent source code release of it, within the `src\llama.cpp-b4820` subdirectory. Use CMake as usual and then build a *Release* build:

1. The CMakeLists.txt file for "llama.cpp" is in `src\llama.cpp-b4820`. Use CMake as usual and then open Visual Studio 2022;
2. In Visual Studio, ensure the build is set to *Release* (not *Debug*);
3. Build all the projects in the solution, by selecting *Build Solution* (though we only need the projects: *llama* and *common*).

3 Try "llama.cpp"'s *llama-cli* example

While building "llama.cpp" you will have created a number of executable and library files. You should find them in `src\llama.cpp-b4820\build\bin\Release`. Ensure that llama-cli.exe is there. Open a command prompt at this location. If you are using Windows Explorer, you can do this quickly via:

1. Click on the navigation bar: type "cmd" there;
2. cmd.exe will open at this directory;
3. Type "llama-cli.exe" and press return.

Run the *llama-cli* program by typing "`llama-cli.exe`" and press return. You should see it print a few lines indicating that it couldn't find a model. The message ends with: `llama-cli: error: unable to load model`.

Instead, type "`llama-cli.exe --help`" and press return. You should see it display a large list of parameter options. The section that follows will guide your choice of parameters.

4 Play with the GGUF LLM from Hugging Face

4.1 model

The `llama-2-7b-chat.Q4_K_M.gguf` file from Section 1 should have downloaded by now. This is the *model* that should be provided using the *llama-cli* example's `--model` option. Here is the relevant part of the output from "`llama-cli.exe --help`":

```
-m FNAME, --model FNAME
       model path (default: models/7B/ggml-model-f16.gguf)
```

Check that "`llama-cli.exe`" can find the `llama-2-7b-chat.Q4_K_M.gguf` file. If you have it in `%USERPROFILE%\Downloads`, try:

`llama-cli.exe --model %USERPROFILE%\Downloads\llama-2-7b-chat.Q4_K_M.gguf`

If the file *isn't* found (check the path), "`llama-cli.exe`" will output a few lines including messages such as: `error: failed to load model`. If the file *is* found, numerous `llm_load_print_meta` and `llama_model_loader` messages will appear. The GGUF file may take a minute to load, and after that may start generating random text (as a default).

4.2 prompt

The *prompt* is the initial text input which the LLM attempts to produce more of by generating new text. Run "`llama-cli.exe`" again. Include the path to the LLM using the `--model` option as before; but this time add a prompt of your own. Perhaps try "`--prompt "My name is"`".

```
-p PROMPT, --prompt PROMPT
        prompt to start generation with (default: empty)
```

4.3 predict

Often the generated text is verbose, or takes a long time to be generated. You can limit the number of tokens generated using the *n-predict* option. For example "`--predict 10`" will abruptly truncate the output to 10 tokens.

```
-n N, --n-predict N
        number of tokens to predict (default: -1, -1 = infinity, -2 = until
    ↪ context filled)
```

4.4 threads

The LLM should generate text more quickly when more CPU *threads* are utilised. Try providing a value which is the number of CPU cores on your machine (e.g. "`--threads 4`". (On Windows "`dxdiag.exe`" can tell you how many cores you have.)

```
-t N, --threads N
        number of threads to use during generation (default: 16)
```

4.5 temperature

A high *temperature* (e.g. "`--temp 1.5`") makes the output more random and creative; while a lower temperature (e.g. "`--temp 0.5`") makes the output more focused, deterministic, and conservative. A temperature of 0 will produce identical outputs in each run.

```
--temp N
       temperature (default: 0.8)
```

4.6 context

The *context* is the number of tokens from the input, and generated text, which are considered during text generation. Exceeding the context limit will typically mean that tokens from the start drop off, and are ignored. Though the default for "`llama-cli.exe`" is 512, LLaMA models were built with a context size of 2048; which is likely to produce the highest quality result (try "`--ctx-size 2048`"). A context size greater than 2048 will produce unpredictable output.

```
-c N, --ctx-size N
       size of the prompt context (default: 512, 0 = loaded from model)
```

5 Interact with the GGUF LLM

Much of the potential within a gaming context comes from the opportunity to chat with NPCs to-and-fro (interactively). This will involve embedding tokens such as "User:" and "AI:" (or names of your own) within the prompt; as well as the use of a few more options via the *llama-cli* example:

5.1 interactive

Without further command-line options, `--interative` will not initially seem to be interactive. You will need to press "Ctrl+C" to interject and type your input, followed by "Return".

```
-i, --interactive
       run in interactive mode
```

5.2 color

In *interactive* mode, it is soon unclear which text was your own, and which was output by the LLM. The "`--color`" option helps to distinguish between these by using coloured text.

```
--color
       colorise output to distinguish prompt and user input from generations
```

5.3 reverse-prompt

This option is crucial during *interactive* sessions. The LLM will generate the text provided for this option, after its own latest response. It will then stop generating further text. A common option could be "`--reverse-prompt "User:"`".

```
-r PROMPT, --reverse-prompt PROMPT
       halt generation at PROMPT, return control in interactive mode
       (can be specified more than once for multiple prompts).
```

5.4 in-prefix

Something like "`--in-prefix " "`" would add a space after each "User:" in the chat. `src\llama.cpp-b4820\examples\main\README.md` states that "the reverse prompt doesn't work when it ends with a space", and so "`--reverse-prompt "User: "`" shouldn't work (note the space after "User:"); though it did work for me during one test.

```
--in-prefix STRING
       string to prefix user inputs with (default: empty)
```

5.5 in-suffix

Something like "`--in-suffix "Assistant:"`‘ would make it explicit that you want the AI to prefix its responses with "Assistant:", rather than only implicitly via cues within the prompt string.

```
--in-suffix STRING
       string to suffix after user inputs with (default: empty)
```

5.6 keep

This is an import option. It can allow you to ensure the AI maintains a "role", or "character"; for example your initial prompt may specify that the AI should constantly tell lies. This option can implement OpenAI's "system prompt". "`--keep -1` should effectively make llama.cpp remember the original prompt.

```
--keep N
       number of tokens to keep from the initial prompt (default: 0, -1 = all)
```

6 Example Interactive Command to *llama-cli*

The following "`llama-cli.exe`" command invocation initiates an interactive session involving the AI playing the role of a goat. Modify to suit your own creative impulses:

```
llama-cli.exe --model %USERPROFILE%\Downloads\llama-2-7b-chat.Q4_K_M.gguf --keep
    ↪  -1 --color --reverse-prompt "User:" --in-prefix " " --interactive --
    ↪ prompt 'The AI always pretends to be a goat. User: Hi\nAI: Baaa! I am
    ↪ going to eat some grass.\nUser: Sure!\nAI: Baaaah! I think I will lie
    ↪ down in this field.\nUser:'

 The AI always pretends to be a goat. User: Hi
AI: Baaa! I am going to eat some grass.
User: Sure!
AI: Baaaah! I think I will lie down in this field.
User: What is your name?
AI: Baaa! My name is Goaty McGoatface.
User: How many legs do you have?
AI: Baaa! I have 4 legs, just like a goat!
User: Are you a goat?
AI: Baaa! Yes, I am a goat! *chews on a straw* *pants* *butts head against the
    ↪ ground*
User: But I thought goats couldn't speak.
AI: Baaa! That's what they want you to think! *winks*
```

7 Try "llama.cpp"'s *llama-server* example

A server can run on your own machine, allowing you to interact with it in the same way as you would if it was running remotely (e.g. on OpenAI's servers). You should find "`llama-server.exe`" along with the other executables in `src\llama.cpp-b4820\build\bin\Release`. Open a command prompt at this location (you may have one open already), and enter the following command:

```
llama-server.exe
   --chat-template llama2
   --model %USERPROFILE%\Downloads\llama-2-7b-chat.Q4_K_M.gguf
```

You should then see, at the end of the output following your command above, that the server is listening on port 8080 at localhost IP http://127.0.0.1. If you now open your browser at `http://127.0.0.1:8080`, you can interact with the GGUF Model via a GUI.

If you omit the `--chat-template` option, you'll see unwanted tokens such as `<|im_start|>` and `<|img end|>` appear here and there; and the system may appear to be talking to itself! Here is the relevant part of the output from "`llama-server.exe --help`":

```
--chat-template JINJA_TEMPLATE
        set custom jinja chat template (default: template taken from model's
    ↪ metadata)
        if suffix/prefix are specified, template will be disabled
        list of built-in templates:
        chatglm3, chatglm4, chatml, command-r, deepseek, deepseek2, exaone3,
gemma, gigachat, granite, llama2, llama2-sys, llama2-sys-bos, llama2-sys-strip,
llama3, minicpm, mistral-v1, mistral-v3, mistral-v3-tekken, mistral-v7,
monarch, openchat, orion, phi3, rwkv-world, vicuna, vicuna-orca, zephyr
        (env: LLAMA_ARG_CHAT_TEMPLATE)
```

You can also configure the llama.cpp server to listen to another ip address. With `--host 0.0.0.0` (i.e. INADDR_ANY) the llama.cpp server will listen on all IPs of the host. Using this option, another machine on your local/home network can access the service via http://XXX.YYY.ZZZ.WWW/8080.

To view tokens per second (t/s), enable in the browser GUI via: *Options Cog > Advanced config > Show tokens per second.*

To stop the server, type CTRL+C.

8 Using the OpenAI API

Register for a developer account with OpenAI here. Try the online Playground. OpenAI used to provide a number of free trial credits; and there was a free tier to their pricing structure. I'm not sure if that's still the case.

If there is still a free trial: obtain a secret API key (it usually starts with `sk-`). Note it down somewhere private to you; you will need to create a new one if you lose it.

Now use CMake as usual to configure the Visual Studio projects within the chatbot subdirectory. To run these, each requires you to assign the `OPENAI_API_KEY`

to the key you obtained above. This should be done within the Properties > Debugging > Environment of each Visual Studio project. Experiment with each of the 5 numbered projects.