LLM Guide & Local Setup Manual

Background Knowledge

LLM, Text Generation, Chat Bot

Large language model(LLM) is a computational model that predict the probability of tokens based on their context. For example, here is an example which can be input to a large language model:

> I have two things an [1]

This is an incomplete sentence with a blank. The blank denotes the target token position where large language model predicts the probability of token based on the context, which is the other part of the incomplete sentence(I have two things an). On receipt of the input, a large language model will calculate the probabilities of each of its producible token, given the context. The producible tokens may vary by the large language model type(specifically, by the tokenizer of the large language model), and the producible tokens can be considered as a list of word that large language model is possible to choose to constitute an output. To put it in a metaphor, if the large language model is a native speaker of English, then the list of producible tokens can be the entry of the Oxford Dictionary or the Merriam Webster Dictionary. For an hypothetical instance, if a large language model only knows three fruits here: apple, pear and orange, a output can be the following:

Token	Probability
apple	0.7
pear	0.2
orange	0.1

A typical output of large language model takes the format of a list of token-probability pairs, as can be observed above. Also as discussed above, what tokens are there in the list is decided by the LLM(specifically, tokenizer). However, a discussion is still needed in what sources the large language model refers to when it predicts the probability.(Although I would be very like to also discuss how the large language model reach the output step by step, formula by formula, this may be too technical at this moment and this is within the modular confinement, meaning that, without knowing such won't hinder users'(your) usage and application of models).

Given the task of large language models, if this task is performed recursively, through a streak of token position, we can get a probability tree and, through searching through the probability tree via certain rules, we can extract a streak of tokens which can forms a text. This process is text generation. For example, for the context, and four consecutive token positions at the end of the text:

> I have two things an [1] [2] [3] [4]

and we set the generating rules to greedy search(search only for the max probability, the generating process can be described as follows:

LLM predicts the probability of token at the first blank, and get the following results:

Token	Probability
apple	0.7
pear	0.2
orange	0.1

Since the searching rule is set to be greedy search. Apple is chosen as the output token at the first blank. Now, the context become:

> I have two things an apple [2] [3] [4]

and the LLM predicts the second blank, and get the probability herewith:

Token	Probability
and	0.9
but	0.01
or	0.01

Again, because the rule is greedy search, the token and is chosen as the output token at the second blank. The same process takes place for the third blank and the fourth blank, and subsequentially we have:

> I have two things an apple and a pear

And this is text generation. Text generation works as a "continuation of writing" based on the context. However, this "continuation of writing" does not guarantee the output to be aligned with the instruction provided by user. A clear example can be seen here:

If the large language model is instructed to generate the text based on the following context:

> You are a pirate and you are going to introduce the history of the printing press in 300 words.

The large language model would typically respond:

> It was a chilly winter evening in 1632 when the unthinkable happened. The streets of Strasbourg, Germany, were abuzz with the sound of drums and the smell of gunpowder wafted through the air. A pirate ship had sailed up the Rhine River, its black sails billowing in the wind.

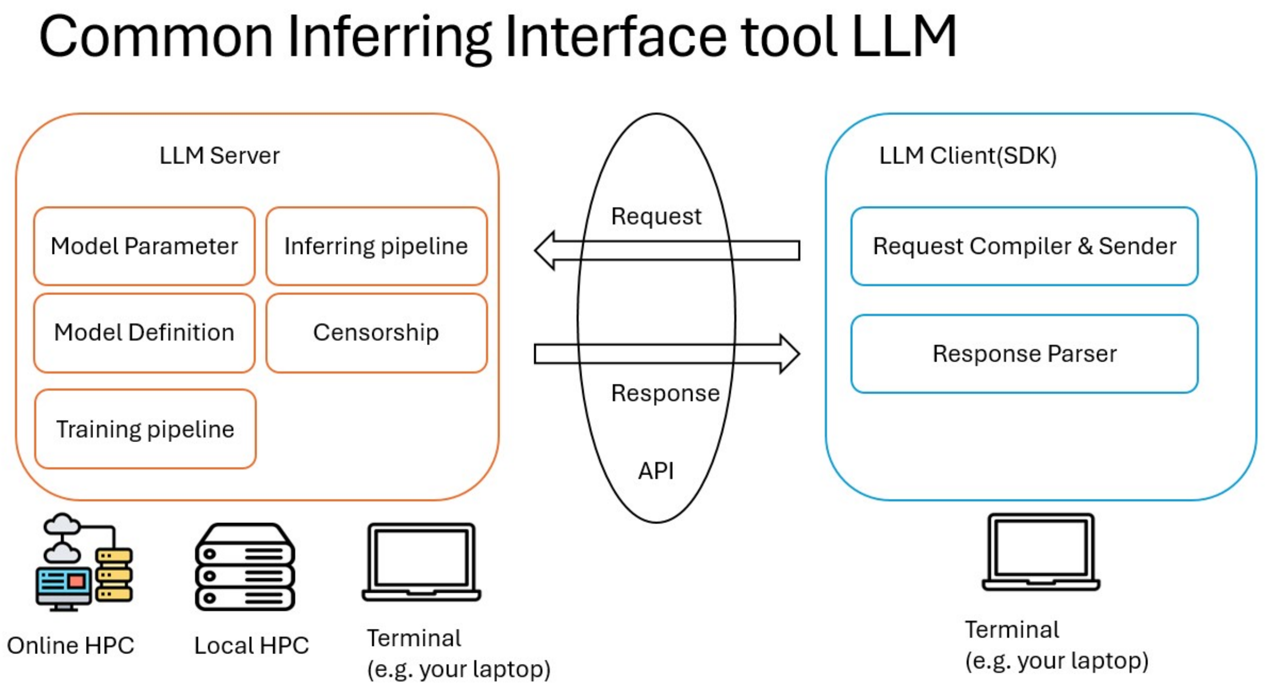
Although the large language model does say something about the history of printing press based on the context provided. it fails to suit itself in a identity of a pirate, and it ignores the style a pirate should have during the output. This shows that the large language model under text generation could not follows the instruction from the user.

To make the large language model follows instructions from user, a large language model is needed to be fine-tuned that so it would be aligned with the task of question answering. Question answering is similar to the text generation; however, question answering emphasize on instruction following -- that is, to answer the question, rather than to write something right after the question. The fine-tuning of large language model to the task of question answering is typically implemented with instruction-based supervised fine-tuning(SFT) or reinforcement learning from human feedback(RLHF) or both. And this fine-tuning makes a large language model chat bot, like the ChatGPT, Claude, Llama and Deepseek.

Precisely speaking, these models shall be called as chat bot, because they no longer perform the task of the language modeling, as originally defined at the beginning of this chapter. However, when mentioning these models, people blurred the idea of language model and chat bot, and the models like ChatGPT, Claude, Llama and Deepseek are also addressed as the large language models.

Pipeline of LLM

Typically, user interacts with large language model(LLM) through a server-client structure herewith:



Open-parameters and Closed-parameters LLM

Parameters of LLM(sometime also addressed as weight and bias of LLM) is the numerical representation of knowledge which LLM obtained through training. They are of vital importance during LLM inferring, because LLM needs the knowledge represented in the parameters to make effective response. Based on the accessibility of parameters, the large language models can be further divided as open-parameters LLMs and closed-parameters LLMs.

Closed-parameters LLMs are perhaps the LLMs that newbie users are most familiar with. Examples of closed-parameters LLMs includes ChatGPT, Claude, and Gemini. These LLMs are served to the user via a Graphical User Interface(GUI) or Application Programming Interface(API), so the users do not need to worry about how to host the LLM. In fact, the AI companies who develops these model host the model for users , and users are prohibited to host their own closed-parameters LLM technically, because they are closed-parameters, which means that their parameters can not be access in open internet. In simple word, you cannot download a ChatGPT, Claude and Gemini, or any other model of this kind. Moreover, users do not have full control on the model as well as the data input, because user need to upload the data via GUI or API to the hosting server at these AI companies. Although most End User License Agreement(EULA) would emphasize the privacy protection, the data input is needed to be leaked to at least these AI companies before a response of LLM can be made to the user.

In contrast, open-parameters LLMs have their parameters openly available on the internet. Their parameter can be downloaded, loaded on your local PC, fine-tuned or tweaked(e.g. activation manipulation) for users' own purposes. Although these models can also be found on their respectively publication repository, most of today's high-performing open-parameters LLMs can be found at huggingface.co, which is the largest and the most famous website for AI model repo hosting.

Using the open-parameters LLMs means that user has total control of the model. If needed, user can confine, inspect, manipulate any details of any steps of large language model at their own will. These models can be used locally, so an internet would not be necessary after a model is downloaded, and the text a user provide to a model would be strictly confined to your own PC, which maximize the privacy. This is especially useful if the data are not to be published due to various reasons, ethical or privilege-related.

However, using open-parameters LLMs comes at a price. User need to find their own hosting server or use software library to start the inferring process. This complicates the use of LLM(compared with closed-parameters model). Moreover, if the LLM is hosted locally, there would be high technical specification requirements for the hosting computer. Typically, abundant memory(>8GB) and a video card(graphic card) would be necessary to ensure the smooth use of the smallest open-parameters LLM like Llama 1B version.

Ollama Setup

Ollama is an open-sourced LLM server programs that allow you to host LLMs locally at your PC/Laptop. Its installation binary is compiled for on OSX, Windows and Linux so no matter what OS you use, Ollama is available for you.

Ollama can be used in terminal(command line, cmd), via an graphical user interface(GUI) and via an API. The Ollama at API is shown in the notebook "PGTip: IntroLLMAPI: Ollama Version". So before you run the notebook, it is essential that you install Ollama.

Installation

To install Ollama, please first go to its official website:

> https://ollama.com,

and following the system specific instruction

Windows

To install Ollama on windows, you have two options:

Option 1: go to this page:

> Download Ollama on Windows

And click the download button. After the installation program is downloaded, double-click to start the program and follow the instruction there.

Option 2: press Ctrl + r to start the run interactive window. Input

> cmd

And click OK.(If you have admin privilege or if you are the admin or if you are the only user of the PC, click OK when you hold Ctrl+Shfit to acquire admin privilege for following process.)

This would start a windows command line for you.

In the command line window, input:

> winget install ollama

Press Enter, and the installation should start. Please note, you may be prompted for EULA(End User License Agreement), if so, type y(es) to proceed.

Linux

To install the Ollama on Linux, press Ctrl+Alt+T to start terminal. Then in terminal, do:

> curl -fsSL https://ollama.com/install.sh | sh

This should automatically start downloading and installing Ollama. However, if you are prompted for admin privilege, please do the following instead:

> sudo curl -fsSL https://ollama.com/install.sh | sh

Note: if you are using this command, you may be prompted for your admin password.

OSX

To install Ollama on OSX, go to

> Download Ollama on macOS

and follow the instruction.

Ollama Command Line Interface(CLI)

After you install Ollama successfully, you will be able to use Ollama in various way, among which the command line interface(CLI) is the most fundamental one.

The Ollama CLI would let you interact with Ollama, check its status, download and run models, and manage the models.

To use the Ollama CLI, you need a command line(cmd in windows), or terminal(in OSX and Linux).

To start a command line in windows, simply press Ctrl + r, input

```
cmd
```

And then click run.

To start a terminal in Linux, press Ctrl+Alt+T .

To start a terminal in OSX, run the application called "terminal" in your launchpad or app finder.

After you start cmd or terminal you may use ollama by calling

```
ollama
```

And this should return a brief document on what sub-command you can use under Ollama. They would look like:

```
Usage: ollama [flags] ollama [command]

Available Commands: serve Start ollama create Create a model from a Modelfile show
Show information for a model run Run a model stop Stop a running model pull Pull a
model from a registry push Push a model to a registry list List models ps List running
models cp Copy a model rm Remove a model help Help about any command

Flags: -h, --help help for ollama -v, --version Show version information

Use "ollama [command] --help" for more information about a command.
```

If you don't find this as the output, go back and double check your installation of ollama, and if this problem persists, please drop me an email.

In the following part I will walk you through the basic usages of ollama. It is recommended that you go through this with me and try this command when you learn.

Start Ollama

Typically, Ollama should start together with your OS, or right after the installation. If you find Ollama is not started, you may use the following command to start:

```
ollama serve
```

Note: Only one ollama instance is allowed at a time. If you see the following outputs:

```
Error: listen tcp 127.0.0.1:11434: bind: address already in use
```

typically the ollama has already started. (if you are an advanced user, it can also be the case that the port 11434 is used by another program, but this should be rare because this is not an usual port that is used by a program).

Download a model

To use a model, you need to first download it. To do so, first, please check here for a full list of model:

> Ollama Search

If you are using a laptop, a home PC with low tech specs, it is highly recommended that you choose a model with low parameter size, such as 1b, or 3b, but if you are working on a workstation or a server(e.g. at bluebear), you are good to go with model bigger than 3b.

And after you select the model you would like to use, run

```
ollama pull [MODEL_NAME]
```

to download the model.

In the PGTip session as demonstration, the model llama3.2:1b is planned to be used. To download it, run

```
ollama pull llama3.2:1b
```

Check downloaded model

To check downloaded model, use the following command:

```
ollama list
```

You may check a list of models and their information.

Run a model

To run a model that has already been downloaded, run:

```
ollama run [MODEL_NAME]
```

For example

> ollama run llama3.2:1b

The model should be running soon. After you see

> > > > send a message (/? for help)

You can start typing your prompt to the model. Any prompt you put here would be sent to the model for inference as a user message. And the model would response in the same command line window or terminal window.

To exit, type the following as chat

> /bye

But please note that /bye DOES NOT stop the model from running. It only exit the dialogue with model and allow you to use your command line window/terminal again. To stop your model you need to check the following.

Stop a model

To stop a model, please run

> ollama stop [MODEL_NAME]

And the model should be stopped.

For example

> ollama stop llama3.2:1b

Check model that is running

Sometime you would forget the name of the model that is in running, or sometime you would be not aware of a model that is initiated by another program, but for whatever reason, if you need to check what model is still running, please run:

> ollama ps

This should return you with a list of model that is currently running.

Ollama with GUI

Beyond the CLI, a mort elegant and efficient way to use Ollama is via a graphical user interface(GUI). A GUI would allow you to interact with model hosted by Ollama in the same(or similar style) of ChatGPT.

To use Ollama with GUI, please ensure that a model is running. Here, I would use llama3.2:1b as an example, if you need to use another model, please replace llama3.2:1b with the name of the model you selected.

To run the model:

> ollama run llama3.2:1b

After the model is started and running, you may go to your web browser and install the related extension.

In either Firefox, or Chrome, there is an extension named "Page Assist".

Firefox Page Assist link: https://addons.mozilla.org/en-US/firefox/addon/page-assist/

Chrome Page Assist link: https://chromewebstore.google.com/detail/page-assist-a-web-ui-for/jfgfiigpkhlkbnfnbobbkinehhfdhndo (or search Page Assist this in chrome web store)

After you installed the extension, you should see the page assist in your extension list(pin it is recommended if you would like to use it repeatedly). Click on the extension icon to start it, and it would automatically connect to the local ollama and the model running in it.

Now, you are equipped yourself with a local LLM that is owned 100% by yourself. It is hosted in a PC you have full control which means that all data you sent towards it remains in your full control, and this also means that the process is more transparent, free of censorship applied outside the model, and also more reliable.