

# Nvidia 驅動程式、LLaMA 與 ChatGPT

LLaMA（大型語言模型 Meta AI）是由 Meta AI 自 2023 年 2 月起發布的一系列大型語言模型（LLMs）。

最近，我組裝了一台配備 Nvidia GPU 的電腦。你可以在這裡查看如何組裝電腦：<https://lzwjava.github.io/computer>。

之後，我開始運行 LLaMA 項目。LLaMA 項目的 GitHub 網址是：<https://github.com/facebookresearch/llama>。

## 安裝 Nvidia 驅動

當你運行以下命令時：

```
torchrun --nproc_per_node 1 example_text_completion.py \
--ckpt_dir llama-2-7b/ \
--tokenizer_path tokenizer.model \
--max_seq_len 128 --max_batch_size 4
```

會出現錯誤：“RuntimeError: Distributed package doesn’t have NCCL built in”。讓我們來了解一下 NCCL。

NVIDIA 集體通信庫（NCCL）實現了針對 NVIDIA GPU 和網絡優化的多 GPU 和多節點通信原語。我參考了以下網站來安裝 NVIDIA 驅動：

- CUDA Toolkit 12.2 Update 1 下載，<https://developer.nvidia.com/cuda-downloads>
- NVIDIA NCCL，<https://developer.nvidia.com/nccl>
- NVIDIA 深度學習 NCCL 文檔，<https://docs.nvidia.com/deeplearning/nccl/install-guide/index.html>
- NVIDIA CUDA Linux 安裝指南，<https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html>
- 安裝 Ubuntu 後遇到 Perform MOK Management，<https://www.cnblogs.com/yutian-blogs/p/13019226.html>
- Ubuntu 22.04 深度學習，<https://gist.github.com/amir-saniyan/b3d8e06145a8569c0d0e030af6d60bea>
- Ubuntu 22.04 筆記，<https://github.com/kmcminn/thinkpad/tree/main/extreme3g>

當我們成功安裝了顯卡的 NVIDIA 驅動後，使用 `nvidia-smi` 命令顯示其詳細信息，可以看到以下內容：

```
(base) lzw@lzw-MS-7E01:~$ nvidia-smi
Thu Aug 17 04:15:43 2023
+-----+
| NVIDIA-SMI 535.86.10      Driver Version: 535.86.10    CUDA Version: 12.2 |
|-----+-----+-----+
| 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 4070        On  | 00000000:01:00.0  On |                   N/A | | |
|  0%   34C     P8            9W / 215W |    666MiB / 12282MiB |    15%     Default |
|          |                               |              |           | N/A |
+-----+-----+-----+
+-----+
| Processes:
| GPU  GI  CI          PID  Type  Process name             GPU Memory |
|       ID  ID
|=====+
|  0  N/A  N/A        1926    G  /usr/lib/xorg/Xorg          381MiB |
|  0  N/A  N/A        2065    G  /usr/bin/gnome-shell        120MiB |
|  0  N/A  N/A        3482    G  gnome-control-center        2MiB |
|  0  N/A  N/A        3803    G  ...irefox/2987/usr/lib/firefox/firefox  149MiB |
+-----+
```

實際上，達到這個階段並不容易。請仔細參考這裡的鏈接，Ubuntu 22.04 筆記，<https://github.com/kmcminn/thinkpad/tree/main/extreme3g>。

## 學習 LLaMA

下載模型後，嘗試運行命令時，我們會遇到以下錯誤：

```
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 86.00 MiB (GPU 0;
11.69 GiB total capacity; 9.70 GiB already allocated; 64.81 MiB free; 9.70 GiB reserved in
total by PyTorch) If reserved memory is » allocated memory try setting max_split_size_mb to
avoid fragmentation.
```

由於我們的顯卡內存只有 12GB，而 llama-2-7b 模型的大小約為 13GB，因此我們無法在顯卡上運行它。

我們嘗試使用另一個項目，open-llama-3b，[https://huggingface.co/openlm-research/open\\_llama\\_3b](https://huggingface.co/openlm-research/open_llama_3b)。

我們遇到了以下錯誤。

```
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (when checking argument for argument index in method wrapper_CUDA__index_select)
```

我們向 ChatGPT 詢問了這個問題。

ChatGPT 紿出了一個非常漂亮的解決方案。我們需要添加以下代碼。

```
input_ids = input_ids.to(model.device)
```

最後，我們可以成功運行它。

```
(llama) lzw@lzw-MS-7E01:~/Projects/open_llama_3b$ python run.py
Q: What is the largest animal?
A: The blue whale.
Q: What is the largest animal?
A: The blue whale. It is the largest animal on Earth. It is also the
```

我們還嘗試了項目 llama2.c，<https://github.com/karpathy/llama2.c>。

```
(base) lzw@lzw-MS-7E01:~/Projects/llama2.c$ ./run stories15M.bin
One day, a big lion and a little rabb
it were in the woods. The big lion said, "Let's race to the river and drink some water." The little rab
The lion was fast, but he was careful not to hurt the little rabbit. The rabbit was getting close to the
The little rabbit saw the lion in the water and thought, "I should tell the lion not to drink." He said
The little rabbit tried to help the lion, but he couldn't. The lion drank too much and his tummy started
achieved tok/s: 185.527329
```

## 通過 ChatGPT 理解代碼

Zhiwei: 你能解釋一下下面的代碼嗎？

```

def generate(
    self,
    prompt_tokens: List[List[int]],
    max_gen_len: int,
    temperature: float = 0.6,
    top_p: float = 0.9,
    logprobs: bool = False,
    echo: bool = False,
) -> Tuple[List[List[int]], Optional[List[List[float]]]]:
    params = self.model.params
    bsz = len(prompt_tokens)
    assert bsz <= params.max_batch_size, (bsz, params.max_batch_size)

    min_prompt_len = min(len(t) for t in prompt_tokens)
    max_prompt_len = max(len(t) for t in prompt_tokens)
    assert max_prompt_len <= params.max_seq_len
    total_len = min(params.max_seq_len, max_gen_len + max_prompt_len)

    pad_id = self.tokenizer.pad_id
    tokens = torch.full((bsz, total_len), pad_id, dtype=torch.long, device="cuda")
    for k, t in enumerate(prompt_tokens):
        tokens[k, :len(t)] = torch.tensor(t, dtype=torch.long, device="cuda")
    if logprobs:
        token_logprobs = torch.zeros_like(tokens, dtype=torch.float)

    prev_pos = 0
    eos_reached = torch.tensor([False] * bsz, device="cuda")
    input_text_mask = tokens != pad_id
    for cur_pos in range(min_prompt_len, total_len):
        logits = self.model.forward(tokens[:, prev_pos:cur_pos], prev_pos)
        if logprobs:
            token_logprobs[:, prev_pos + 1 : cur_pos + 1] = -F.cross_entropy(
                input=logits.transpose(1, 2),
                target=tokens[:, prev_pos + 1 : cur_pos + 1],
                reduction="none",
                ignore_index=pad_id,
            )

```

```

        )

    if temperature > 0:
        probs = torch.softmax(logits[:, -1] / temperature, dim=-1)
        next_token = sample_top_p(probs, top_p)

    else:
        next_token = torch.argmax(logits[:, -1], dim=-1)

    next_token = next_token.reshape(-1)

    # only replace token if prompt has already been generated
    next_token = torch.where(
        input_text_mask[:, cur_pos], tokens[:, cur_pos], next_token
    )

    tokens[:, cur_pos] = next_token
    eos_reached |= (~input_text_mask[:, cur_pos]) & (
        next_token == self.tokenizer.eos_id
    )

    prev_pos = cur_pos
    if all(eos_reached):
        break

    if logprobs:
        token_logprobs = token_logprobs.tolist()
        out_tokens, out_logprobs = [], []
        for i, toks in enumerate(tokens.tolist()):
            # cut to max gen len
            start = 0 if echo else len(prompt_tokens[i])
            toks = toks[start : len(prompt_tokens[i]) + max_gen_len]
            probs = None
            if logprobs:
                probs = token_logprobs[i][start : len(prompt_tokens[i]) + max_gen_len]
            # cut to eos tok if any
            if self.tokenizer.eos_id in toks:
                eos_idx = toks.index(self.tokenizer.eos_id)
                toks = toks[:eos_idx]
                probs = probs[:eos_idx] if logprobs else None
            out_tokens.append(toks)

```

```

        out_logprobs.append(probs)
    return (out_tokens, out_logprobs if logprobs else None)

```

ChatGPT: 當然，以下是對提供代碼的解釋：

```

def generate(
    self,
    prompt_tokens: List[List[int]],
    max_gen_len: int,
    temperature: float = 0.6,
    top_p: float = 0.9,
    logprobs: bool = False,
    echo: bool = False,
) -> Tuple[List[List[int]], Optional[List[List[float]]]]:

```

這個函數名為 `generate`，接受幾個參數：

- `prompt_tokens` 是一個包含提示令牌序列的列表。
- `max_gen_len` 是生成文本的最大長度。
- `temperature` 是用於控制生成過程中隨機性的參數。
- `top_p` 是用於控制採樣過程中多樣性的概率閾值，範圍在 0 到 1 之間。
- `logprobs` 是一個布爾值，指示是否記錄生成過程的對數概率。
- `echo` 是一個布爾值，指示在生成過程中是否回顯輸入。

```

params = self.model.params
bsz = len(prompt_tokens)
assert bsz <= params.max_batch_size, (bsz, params.max_batch_size)

```

這裡，函數獲取模型參數並計算提示的批次大小。它確保批次大小不大於模型參數中指定的最大批次大小。

```

min_prompt_len = min(len(t) for t in prompt_tokens)
max_prompt_len = max(len(t) for t in prompt_tokens)
assert max_prompt_len <= params.max_seq_len
total_len = min(params.max_seq_len, max_gen_len + max_prompt_len)

```

這些行計算提示的最小和最大長度，並確保最大提示長度不超過模型參數中指定的最大序列長度。總長度是根據最大生成長度和最大提示長度計算的。

```

pad_id = self.tokenizer.pad_id
tokens = torch.full((bsz, total_len), pad_id, dtype=torch.long, device="cuda")
for k, t in enumerate(prompt_tokens):
    tokens[k, : len(t)] = torch.tensor(t, dtype=torch.long, device="cuda")

```

這部分初始化了 `tokens` 張量，並用提示的令牌序列填充它。

```

if logprobs:
    token_logprobs = torch.zeros_like(tokens, dtype=torch.float)

```

如果 `logprobs` 為 `True`，則創建一個張量 `token_logprobs` 來存儲對數概率。

```

prev_pos = 0
eos_reached = torch.tensor([False] * bsz, device="cuda")
input_text_mask = tokens != pad_id

```

初始化變量 `prev_pos` (前一個位置)、`eos_reached` (是否遇到序列結束令牌) 和 `input_text_mask` (未填充填充的位置)。

以下循環從最小提示長度開始生成令牌，直到總長度。

```

for cur_pos in range(min_prompt_len, total_len):
    logits = self.model.forward(tokens[:, prev_pos:cur_pos], prev_pos)
    if logprobs:
        token_logprobs[:, prev_pos + 1 : cur_pos + 1] = -F.cross_entropy(
            input=logits.transpose(1, 2),
            target=tokens[:, prev_pos + 1 : cur_pos + 1],
            reduction="none",
            ignore_index=pad_id,
        )
    if temperature > 0:
        probs = torch.softmax(logits[:, -1] / temperature, dim=-1)
        next_token = sample_top_p(probs, top_p)
    else:
        next_token = torch.argmax(logits[:, -1], dim=-1)

    next_token = next_token.reshape(-1)
    next_token = torch.where(

```

```

        input_text_mask[:, cur_pos], tokens[:, cur_pos], next_token
    )
    tokens[:, cur_pos] = next_token
    eos_reached |= (~input_text_mask[:, cur_pos]) & (
        next_token == self.tokenizer.eos_id
    )
    prev_pos = cur_pos
    if all(eos_reached):
        break

```

在這個循環中，模型為下一個令牌生成概率分佈 logits。如果 logprobs 為 True，則使用交叉熵計算對數概率。根據 temperature，它使用 softmax 或 argmax 來選擇下一個令牌。然後更新 tokens 張量並檢查序列結束令牌。

```

if logprobs:
    token_logprobs = token_logprobs.tolist()

```

如果 logprobs 為 True，則將對數概率張量轉換為 Python 列表。

```

out_tokens, out_logprobs = [], []
for i, toks in enumerate(tokens.tolist()):
    ...

```

在這個循環中，對於每個輸入樣本，它生成輸出令牌序列及其相關的對數概率。

```

return (out_tokens, out_logprobs if logprobs else None)

```

最後，函數返回生成的令牌序列及其對應的對數概率（如果請求）。