理解神經網絡

解釋以下代碼：

```
class Llama:
    @staticmethod
    def build(
        ckpt_dir: str,
        tokenizer_path: str,
        max_seq_len: int,
        max_batch_size: int,
        model_parallel_size: Optional[int] = None,
    ) -> "Llama":
        if not torch.distributed.is_initialized():
            torch.distributed.init_process_group("nccl")
        if not model_parallel_is_initialized():
            if model_parallel_size is None:
                model_parallel_size = int(os.environ.get("WORLD_SIZE", 1))
            initialize_model_parallel(model_parallel_size)

        local_rank = int(os.environ.get("LOCAL_RANK", 0))
        torch.cuda.set_device(local_rank)

        # seed must be the same in all processes
        torch.manual_seed(1)

        if local_rank > 0:
            sys.stdout = open(os.devnull, "w")

        start_time = time.time()
        checkpoints = sorted(Path(ckpt_dir).glob("*.pth"))
        assert len(checkpoints) > 0, f"no checkpoint files found in {ckpt_dir}"
        assert model_parallel_size == len(
            checkpoints
        ), f"Loading a checkpoint for MP={len(checkpoints)} but world size is {model_parallel_size}"
        ckpt_path = checkpoints[get_model_parallel_rank()]
```

```
        checkpoint = torch.load(ckpt_path, map_location="cpu")
        with open(Path(ckpt_dir) / "params.json", "r") as f:
            params = json.loads(f.read())
```

錯誤日誌：

```
% 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
NOTE: Redirects are currently not supported in Windows or MacOs.
Traceback (most recent call last):
  File "/Users/lzwjava/projects/llama/example_text_completion.py", line 55, in <module>
    fire.Fire(main)
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/fire/core.py", line 141, in Fire
    component_trace = _Fire(component, args, parsed_flag_args, context, name)
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/fire/core.py", line 475, in _Fire
    component, remaining_args = _CallAndUpdateTrace(
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/fire/core.py", line 691, in _CallAndUpdat
    component = fn(*varargs, **kwargs)
  File "/Users/lzwjava/projects/llama/example_text_completion.py", line 18, in main
    generator = Llama.build(
  File "/Users/lzwjava/projects/llama/llama/generation.py", line 62, in build
    torch.distributed.init_process_group("nccl")
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/distributed_c10d.py", li
    default_pg = _new_process_group_helper(
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/distributed_c10d.py", li
    raise RuntimeError("Distributed package doesn't have NCCL " "built in")
RuntimeError: Distributed package doesn't have NCCL built in
ERROR:torch.distributed.elastic.multiprocessing.api:failed (exitcode: 1) local_rank: 0 (pid: 23273) of l

Traceback (most recent call last):
  File "/Users/lzwjava/anaconda3/bin/torchrun", line 33, in <module>
    sys.exit(load_entry_point('torch==2.0.1', 'console_scripts', 'torchrun')())
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/elastic/multiprocessing/
    return f(*args, **kwargs)
```

```
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/run.py", line 794, in ma
    run(args)
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/run.py", line 785, in ru
    elastic_launch(
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/launcher/api.py", line
    return launch_agent(self._config, self._entrypoint, list(args))
  File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/launcher/api.py", line
    raise ChildFailedError(
torch.distributed.elastic.multiprocessing.errors.ChildFailedError:
============================================================
example_text_completion.py FAILED
------------------------------------------------------------
Failures:
  <NO_OTHER_FAILURES>
------------------------------------------------------------
Root Cause (first observed failure):
[0]:
  time      : 2023-08-03_15:35:37
  host      : 1.0.0.127.in-addr.arpa
  rank      : 0 (local_rank: 0)
  exitcode  : 1 (pid: 23273)
  error_file: <N/A>
  traceback : To enable traceback see: https://pytorch.org/docs/stable/elastic/errors.html
============================================================
```

錯誤分析：

錯誤日誌顯示，代碼在嘗試初始化分佈式進程組時失敗，原因是 NCCL 後端未在 PyTorch 中啟用。NCCL 是 NVIDIA 提供的用於多 GPU 通信的庫，通常用於分佈式訓練。錯誤信息表明，當前的 PyTorch 安裝不支持 NCCL，這可能是因為 PyTorch 安裝時未啟用 CUDA 支持，或者是在非 NVIDIA GPU 的環境中運行。

解決方案：

1. **檢查 PyTorch 安裝**：確保安裝的 PyTorch 版本支持 CUDA 和 NCCL。可以通過以下命令檢查：

   ```
   python -c "import torch; print(torch.cuda.is_available())"
   ```

如果返回 `False`，則需要重新安裝支持 CUDA 的 PyTorch 版本。

2. **重新安裝 PyTorch**：如果當前安裝不支持 CUDA，可以通過以下命令安裝支持 CUDA 的版本：

   ```
   pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
   ```

   請根據你的 CUDA 版本選擇合適的安裝命令。

3. **在 CPU 上運行**：如果無法使用 GPU，可以修改代碼以在 CPU 上運行。將 `torch.distributed.init_process_gr` 改為 `torch.distributed.init_process_group("gloo")`，因為 `gloo` 是 CPU 上的分佈式後端。

解釋以下代碼：

```
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM

model_path = '.'
# model_path = 'openlm-research/open_llama_7b'

tokenizer = LlamaTokenizer.from_pretrained(model_path)
model = LlamaForCausalLM.from_pretrained(
    model_path, torch_dtype=torch.float16, device_map='auto',
)

prompt = 'Q: What is the largest animal?\nA:'
input_ids = tokenizer(prompt, return_tensors="pt").input_ids
generation_output = model.generate(
    input_ids=input_ids, max_new_tokens=32
)
print(tokenizer.decode(generation_output[0]))
```

錯誤日誌：

```
/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py:1445: User
  warnings.warn(
Traceback (most recent call last):
```

```
  File "/home/lzw/Projects/open_llama_3b/run.py", line 17, in <module>
    generation_output = model.generate(
                        ^^^^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/utils/_contextlib.py", line 1
    return func(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py", l
    return self.greedy_search(
           ^^^^^^^^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py", l
    outputs = self(
              ^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", line 15
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/models/llama/modeling_l
    outputs = self.model(
              ^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", line 15
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/models/llama/modeling_l
    inputs_embeds = self.embed_tokens(input_ids)
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", line 15
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/sparse.py", line 10
    return F.embedding(
           ^^^^^^^^^^^^
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/functional.py", line 2210,
    return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and
```

解決方案：

錯誤信息表明，input_ids 和模型不在同一個設備上。模型在 GPU (cuda:0) 上，而 input_ids 在 CPU 上。為了解決這個問題，需要將 input_ids 移動到與模型相同的設備上。

修改後的代碼如下：

```
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM

model_path = '.'
# model_path = 'openlm-research/open_llama_7b'

tokenizer = LlamaTokenizer.from_pretrained(model_path)
model = LlamaForCausalLM.from_pretrained(
    model_path, torch_dtype=torch.float16, device_map='auto',
)

prompt = 'Q: What is the largest animal?\nA:'
input_ids = tokenizer(prompt, return_tensors="pt").input_ids

# 將 input_ids 移動到與模型相同的設備上
input_ids = input_ids.to(model.device)

generation_output = model.generate(
    input_ids=input_ids, max_new_tokens=32
)
print(tokenizer.decode(generation_output[0]))
```

在生成文本之前，將 input_ids 移動到與模型相同的設備上，這樣可以避免設備不匹配的問題。

解釋以下代碼：

```
   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)
```

代碼解釋：

這段代碼是一個文本生成方法，用於根據輸入的提示（prompt）生成文本。以下是代碼的主要功能：

1. **參數說明**：
    - `prompt_tokens`：輸入的提示文本，已經被轉換為 token ID 列表。
    - `max_gen_len`：生成文本的最大長度。
    - `temperature`：控制生成文本的隨機性，值越大生成結果越隨機。
    - `top_p`：控制生成文本的多樣性，值越大生成結果越多樣。
    - `logprobs`：是否返回生成 token 的對數概率。
    - `echo`：是否在生成結果中包含輸入的提示文本。
2. **初始化**：
    - 獲取模型的參數，並檢查輸入的 batch size 是否超過模型的最大 batch size。
    - 計算輸入提示的最小和最大長度，並確定生成文本的總長度。
3. **生成文本**：
    - 使用一個循環逐個生成 token，直到達到最大長度或生成結束標記（EOS）。
    - 根據 `temperature` 和 `top_p` 參數選擇下一個 token。
    - 如果 `logprobs` 為 `True`，則計算生成 token 的對數概率。
4. **處理生成結果**：
    - 根據 `echo` 參數決定是否包含輸入提示。
    - 如果生成結果中包含 EOS 標記，則截斷生成結果。
5. **