ModelTC / **lightllm** Public

LightLLM is a Python-based LLM (Large Language Model) inference and serving framework, notable for its lightweight design, easy scalability, and high-speed performance.

Apache-2.0 license

2.7k stars 218 forks Branches Tags Activity

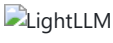
Star Notifications

Code Issues 64 Pull requests 8 Discussions Actions Projects Security Insights

main 101 Branches 1 Tag Go to file Go to file Code

hiworldwzj udpate mla decode attention. (#696) ✓ 6ede09e · yesterday

.github	add docker clean (#555)	3 months ago
assets	init lightllm repo	2 years ago
demos	fix (#526)	4 months ago
docs	misc: update python requirements (#680)	2 weeks ago
format_out	fix bug for format out (#546)	4 months ago
lightllm	udpate mla decode attention. (#696)	yesterday
test	better cuda graph mla decode kernel. (#694)	2 days ago
tools	Modify readme about container multiple GPUs …	2 years ago
.gitignore	add deepseek2 support (#472)	6 months ago
.pre-commit-config.yaml	[Feature] Dynamic prompt cache (#356)	10 months ago
CONTRIBUTING.md	Code Style Improvement (#282)	last year
Dockerfile	misc: update python requirements (#680)	2 weeks ago
LICENSE	Initial commit	2 years ago
README.md	misc: update python requirements (#680)	2 weeks ago
benchmark.md	init lightllm repo	2 years ago
build_and_upload_docker.sh	Some API changes (#125)	2 years ago
format.py	Revert "Merge internal (#164)" (#165)	2 years ago
requirements.txt	add kernel config tuning way to get better perf…	2 weeks ago
setup.py	add kernel config tuning way to get better perf…	2 weeks ago



LightLLM is a Python-based LLM (Large Language Model) inference and serving framework, notable for its lightweight design, easy scalability, and high-speed performance. LightLLM harnesses the strengths of numerous well-regarded open-source implementations, including but not limited to FasterTransformer, TGI, vLLM, and FlashAttention.

English Docs | 中文文档

Features

- Tri-process asynchronous collaboration: tokenization, model inference, and detokenization are performed asynchronously, leading to a

README Apache-2.0 license

disparities.
- Dynamic Batch: enables dynamic batch scheduling of requests
- FlashAttention: incorporates FlashAttention to improve speed and reduce GPU memory footprint during inference.
- Tensor Parallelism: utilizes tensor parallelism over multiple GPUs for faster inference.
- Token Attention: implements token-wise's KV cache memory management mechanism, allowing for zero memory waste during inference.
- High-performance Router: collaborates with Token Attention to meticulously manage the GPU memory of each token, thereby optimizing system throughput.
- Int8KV Cache: This feature will increase the capacity of tokens to almost twice as much. only llama support.

Supported Model List

- BLOOM
- LLaMA
- LLaMA V2
- StarCoder
- Qwen-7b
- ChatGLM2-6b
- InternLM-7b
- InternVL-Chat
- Qwen-VL
- Qwen-VL-Chat
- Qwen2-VL
- Llava-7b
- Llava-13b
- Mixtral
- Stablelm
- MiniCPM
- Phi-3
- CohereForAI
- DeepSeek-V2-Lite
- DeepSeek-V2

> When you start Qwen-7b, you need to set the parameter '--eos_id 151643 --trust_remote_code'.

> ChatGLM2 needs to set the parameter '--trust_remote_code'.

> InternLM needs to set the parameter '--trust_remote_code'.

> InternVL-Chat(Phi3) needs to set the parameter '--eos_id 32007 --trust_remote_code'.

> InternVL-Chat(InternLM2) needs to set the parameter '--eos_id 92542 --trust_remote_code'.

> Qwen2-VL-7b needs to set the parameter '--eos_id 151645 --trust_remote_code', and use 'pip install git+https://github.com/huggingface/transformers' to upgrade to the latest version.

> Stablelm needs to set the parameter '--trust_remote_code'.

> Phi-3 only supports Mini and Small.

> DeepSeek-V2-Lite and DeepSeek-V2 need to set the parameter '--data_type bfloat16'

Get started

Requirements

The code has been tested with Pytorch>=1.3, CUDA 12.4, and Python 3.9. To install the necessary dependencies, please refer to the provided **requirements.txt** and follow the instructions as

```
# for cuda 12.4
pip install -r requirements.txt --extra-index-url https://download.pytorch.org/whl/cu124
```

NOTE: If you are using torch with cuda 11.x instead, run `pip install nvidia-nccl-cu12==2.20.5` to support torch cuda graph.

Container

You can use the official Docker container to run the model more easily. To do this, follow these steps:

- Pull the container from the GitHub Container Registry:

```
docker pull ghcr.io/modeltc/lightllm:main
```

- Run the container with GPU support and port mapping:

```
docker run -it --gpus all -p 8080:8080                \
        --shm-size 1g -v your_local_path:/data/       \
        ghcr.io/modeltc/lightllm:main /bin/bash
```

- Alternatively, you can build the container yourself:

```
docker build -t <image_name> .
docker run -it --gpus all -p 8080:8080                \
        --shm-size 1g -v your_local_path:/data/       \
        <image_name> /bin/bash
```

- You can also use a helper script to launch both the container and the server:

```
python tools/quick_launch_docker.py --help
```

- Note: If you use multiple GPUs, you may need to increase the shared memory size by adding `--shm-size` to the `docker run` command.

Installation

- Install from the source code by

```
python setup.py install
```

- Install Triton Package

The code has been tested on a range of GPUs including V100, A100, A800, 4090, and H800. If you are running the code on A100, A800, etc., we recommend using triton==3.0.0.

```
pip install triton==3.0.0 --no-deps
```

If you are running the code on H800 or V100., you can try triton-nightly to get better performance.

```
pip install -U --index-url https://aiinfra.pkgs.visualstudio.com/PublicPackages/_packaging/Triton-Nightly/pypi/simple/ triton-n
```

RUN LLaMA

With efficient Routers and TokenAttention, LightLLM can be deployed as a service and achieve the state-of-the-art throughput performance.

Launch the server:

```
python -m lightllm.server.api_server --model_dir /path/llama-7B     \
                                     --host 0.0.0.0                 \
                                     --port 8080                    \
                                     --tp 1                         \
                                     --max_total_token_num 120000
```

The parameter `max_total_token_num` is influenced by the GPU memory of the deployment environment. You can also specify --mem_faction to have it calculated automatically.

```
python -m lightllm.server.api_server --model_dir /path/llama-7B     \
                                     --host 0.0.0.0                 \
                                     --port 8080                    \
                                     --tp 1                         \
                                     --mem_faction 0.9
```

To initiate a query in the shell:

```
curl http://127.0.0.1:8080/generate     \
    -X POST                             \
    -d '{"inputs":"What is AI?","parameters":{"max_new_tokens":17, "frequency_penalty":1}}' \
    -H 'Content-Type: application/json'
```

To query from Python:

```
import time
import requests
import json

url = 'http://localhost:8080/generate'
headers = {'Content-Type': 'application/json'}
data = {
    'inputs': 'What is AI?',
    "parameters": {
        'do_sample': False,
        'ignore_eos': False,
        'max_new_tokens': 1024,
    }
}
response = requests.post(url, headers=headers, data=json.dumps(data))
if response.status_code == 200:
    print(response.json())
else:
    print('Error:', response.status_code, response.text)
```

RUN Multimodal Models

Run QWen-VL

```
python -m lightllm.server.api_server \
    --host 0.0.0.0                 \
    --port 8080                    \
    --tp 1                         \
    --max_total_token_num 12000    \
    --trust_remote_code            \
    --enable_multimodal            \
    --cache_capacity 1000          \
    --model_dir /path/of/Qwen-VL or /path/of/Qwen-VL-Chat
```

Run Llava

```
python -m lightllm.server.api_server \
    --host 0.0.0.0                 \
    --port 8080                    \
    --tp 1                         \
    --max_total_token_num 12000    \
    --trust_remote_code            \
    --enable_multimodal            \
    --cache_capacity 1000          \
    --model_dir /path/of/llava-v1.5-7b or /path/of/llava-v1.5-13b
```

Query From QWen-VL

```
import time
import requests
import json
import base64
```

```
url = 'http://localhost:8080/generate'
headers = {'Content-Type': 'application/json'}

uri = "/local/path/of/image" # or "/http/path/of/image"
if uri.startswith("http"):
    images = [{"type": "url", "data": uri}]
else:
    with open(uri, 'rb') as fin:
        b64 = base64.b64encode(fin.read()).decode("utf-8")
    images=[{'type': "base64", "data": b64}]

data = {
    "inputs": "<img></img>Generate the caption in English with grounding:",
    "parameters": {
        "max_new_tokens": 200,
        # The space before <|endoftext|> is important, the server will remove the first bos_token_id, but QWen tokenizer does n
        "stop_sequences": [" <|endoftext|>"],
    },
    "multimodal_params": {
        "images": images,
    }
}

response = requests.post(url, headers=headers, data=json.dumps(data))
if response.status_code == 200:
    print(response.json())
else:
    print('Error:', response.status_code, response.text)
```

Query From QWen-VL-Chat

```
import json
import requests
import base64

def run_once(query, uris):
    images = []
    for uri in uris:
        if uri.startswith("http"):
            images.append({"type": "url", "data": uri})
        else:
            with open(uri, 'rb') as fin:
                b64 = base64.b64encode(fin.read()).decode("utf-8")
            images.append({'type': "base64", "data": b64})

    data = {
        "inputs": query,
        "parameters": {
            "max_new_tokens": 200,
            # The space before <|endoftext|> is important, the server will remove the first bos_token_id, but QWen tokenizer do
            "stop_sequences": [" <|endoftext|>", " <|im_start|>", " <|im_end|>"],
        },
        "multimodal_params": {
            "images": images,
        }
    }

    # url = "http://127.0.0.1:8080/generate_stream"
    url = "http://127.0.0.1:8080/generate"
    headers = {'Content-Type': 'application/json'}
    response = requests.post(url, headers=headers, data=json.dumps(data))
    if response.status_code == 200:
        print(" + result: ({})".format(response.json()))
    else:
        print(' + error: {}, {}'.format(response.status_code, response.text))

"""
multi-img, multi-round:

<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user
<img></img>
<img></img>
上面两张图片分别是哪两个城市？请对它们进行对比。<|im_end|>
```

```
<|im_start|>assistant
根据提供的信息，两张图片分别是重庆和北京。<|im_end|>
<|im_start|>user
这两座城市分别在什么地方？<|im_end|>
<|im_start|>assistant
"""
run_once(
    uris = [
        "assets/mm_tutorial/Chongqing.jpeg",
        "assets/mm_tutorial/Beijing.jpeg",
    ],
    query = "<|im_start|>system\nYou are a helpful assistant.<|im_end|>\n<|im_start|>user\n<img></img>\n<img></img>\n上面两张图片
)
```

Query From Llava

```
import time
import requests
import json
import base64

url = 'http://localhost:8080/generate'
headers = {'Content-Type': 'application/json'}

uri = "/local/path/of/image" # or "/http/path/of/image"
if uri.startswith("http"):
    images = [{"type": "url", "data": uri}]
else:
    with open(uri, 'rb') as fin:
        b64 = base64.b64encode(fin.read()).decode("utf-8")
    images=[{'type': "base64", "data": b64}]

data = {
    "inputs": "A chat between a curious human and an artificial intelligence assistant. The assistant gives helpful, detailed, a
    "parameters": {
        "max_new_tokens": 200,
    },
    "multimodal_params": {
        "images": images,
    }
}

response = requests.post(url, headers=headers, data=json.dumps(data))
if response.status_code == 200:
    print(response.json())
else:
    print('Error:', response.status_code, response.text)
```

> Additional lanuch parameters: `--enable_multimodal` , `--cache_capacity` , larger `--cache_capacity` requires larger `shm-size`

> Support `--tp > 1` , when `tp > 1` , visual model run on the gpu 0

> The special image tag for Qwen-VL is `<img></img>` (`<image>` for Llava), the length of `data["multimodal_params"]["images"]` should be the same as the count of tags, The number can be 0, 1, 2, ...

> Input images format: list for dict like `{'type': 'url'/'base64', 'data': xxx}`

Performance

Service Performance

We compared the service performance of LightLLM and vLLM==0.1.2 on LLaMA-7B using an A800 with 80G GPU memory.

To begin, prepare the data as follows:

```
wget https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_spli
```

Launch the service:

```
python -m lightllm.server.api_server --model_dir /path/llama-7b --tp 1 --max_total_token_num 121060 --tokenizer_mode auto
```

Evaluation:

```
cd test
python benchmark_serving.py --tokenizer /path/llama-7b --dataset /path/ShareGPT_V3_unfiltered_cleaned_split.json --num-prompts 
```

The performance comparison results are presented below:

vLLM	LightLLM
Total time: 361.79 s Throughput: 5.53 requests/s	Total time: 188.85 s Throughput: 10.59 requests/s

Static inference performance

For debugging, we offer static performance testing scripts for various models. For instance, you can evaluate the inference performance of the LLaMA model by

```
cd test/model
python test_llama.py
```

FAQ

- The LLaMA tokenizer fails to load.
  - consider resolving this by running the command `pip install protobuf==3.20.0`.
- `error : PTX .version 7.4 does not support .target sm_89`
  - launch with `bash tools/resolve_ptx_version python -m lightllm.server.api_server ...`

Projects using lightllm

If you have a project that should be incorporated, please contact via email or create a pull request.

1. ▶ LazyLLM: Easyest and lazyest way for building multi-agent LLMs applications.

Community

For further information and discussion, join our discord server.

License

This repository is released under the Apache-2.0 license.

Acknowledgement

We learned a lot from the following projects when developing LightLLM.

Releases

1 tags

Packages 1

lightllm

Contributors 32