딥러닝/클라우드

Chapter 20

Speed up inference

오 세 종

DANKOOK UNIVERSITY

Contents

Summary

- LLM의 문제
  - 질의에 대한 response time 이 길다.
  - Realtime application 개발시 문제

- Speed up inference
  - Text Generation Interface(tgi) :
    - Huggingface document (link)
    - medium.com document (link)

  - vLLM : https://github.com/vllm-project/vllm

- Speed up methods:
  - Document (link)

1. Text Generation Interface

- 개요
  - Developed by hugging face
  - Text Generation Inference (TGI) is a toolkit for deploying and serving Large Language Models (LLMs).
  - TGI enables high-performance text generation for the most popular open-source LLMs, including Llama, Phi3, gemma, Mistral, BLOOM, GPT-NeoX, and T5.
  - Text Generation Inference implements many optimizations and features.
  - Text Generation Inference is used in production by multiple projects, such as
    - Hugging Chat, an open-source interface for open-access models, such as Open Assistant and Llama
    - OpenAssistant, an open-source community effort to train LLMs in the open
    - nat.dev, a playground to explore and compare LLMs.
  - Supported GPU : Nividia GPU (link), AMD GPU (link), Intel GPU (link), Gaudi, Inferentia

1. Text Generation Interface

- Setup : TGI on an Nvidia GPU

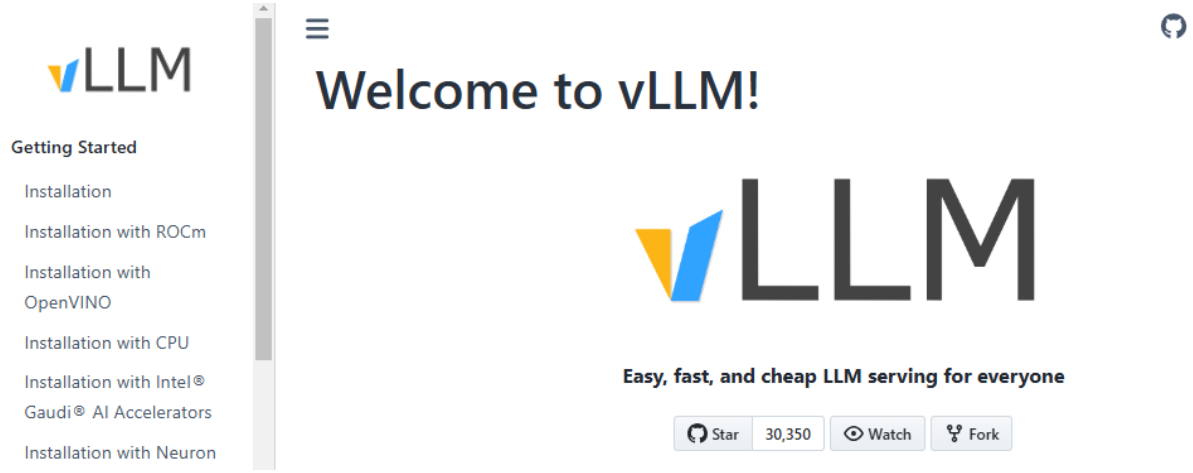
```
model=teknium/OpenHermes-2.5-Mistral-7B
volume=$PWD/data # share a volume with the Docker container to avo

docker run --gpus all --shm-size 1g -p 8080:80 -v $volume:/data \
    ghcr.io/huggingface/text-generation-inference:2.4.1 \
    --model-id $model
```

- inference

```
import requests

headers = {
    "Content-Type": "application/json",
}

data = {
    'inputs': 'What is Deep Learning?',
    'parameters': {
        'max_new_tokens': 200,
    },
}

response = requests.post('http://127.0.0.1:8080/generate', headers=headers,
                         json=data)
print(response.json())
```

2. vLLM

- https://docs.vllm.ai/en/latest/
  - PagedAttention 등의 기법을 활용하여 문장 생성 속도를 비약적으로 높임
  - 허깅페이스 대비 최대 24배까지 성능을 높일 수 있다고 알려짐



2. vLLM

- vLLM is fast with:
  - State-of-the-art serving throughput
  - Efficient management of attention key and value memory with **PagedAttention**
  - Continuous batching of incoming requests
  - Fast model execution with CUDA/HIP graph
  - Quantization: GPTQ, AWQ, INT4, INT8, and FP8
  - Optimized CUDA kernels, including integration with FlashAttention and FlashInfer.
  - Speculative decoding
  - Chunked prefill

  - OpenAI-compatible API server
  - Support NVIDIA GPUs, AMD CPUs and GPUs, Intel CPUs, Gaudi® accelerators and GPUs, PowerPC CPUs, TPU, and AWS Trainium and Inferentia Accelerators.

2. vLLM

- 지원 모델
  - https://docs.vllm.ai/en/latest/models/supported_models.html

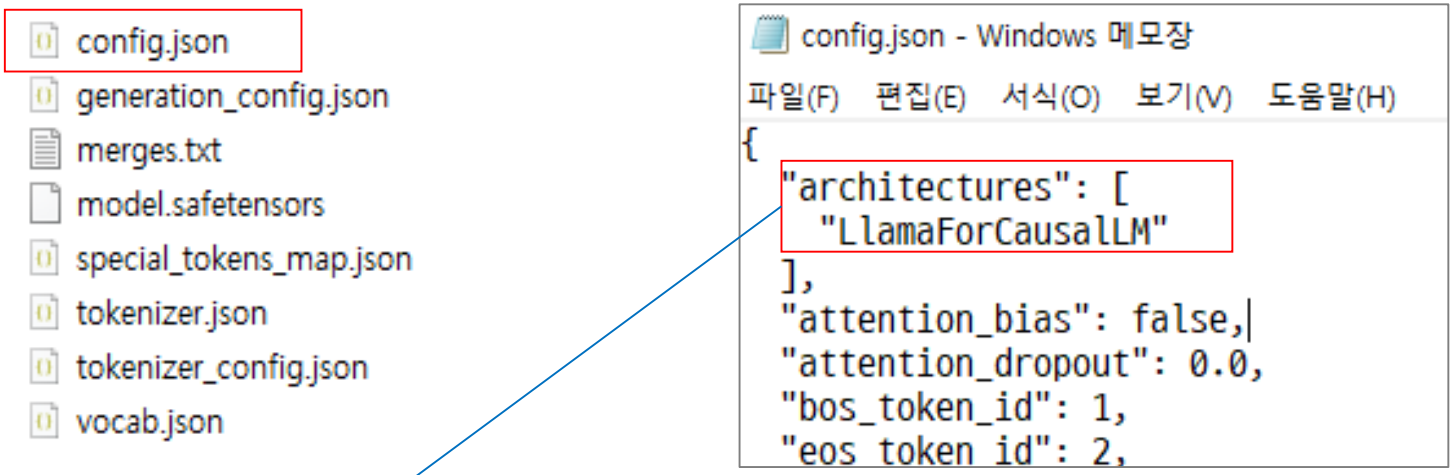
Text Generation

Architecture	Models	Example HF Models	LoRA	PP
`AquilaForCausalLM`	Aquila, Aquila2	`BAAI/Aquila-7B`, `BAAI/AquilaChat-7B`, etc.	✅	✅
`ArcticForCausalLM`	Arctic	`Snowflake/snowflake-arctic-base`, `Snowflake/snowflake-arctic-instruct`, etc.		✅
`BaiChuanForCausalLM`	Baichuan2, Baichuan	`baichuan-inc/Baichuan2-13B-Chat`, `baichuan-inc/Baichuan-7B`, etc.	✅	✅
`BloomForCausalLM`	BLOOM, BLOOMZ, BLOOMChat	`bigscience/bloom`, `bigscience/bloomz`, etc.		✅
`BartForConditionalGeneration`	BART	`facebook/bart-base`, `facebook/bart-large-cnn`, etc.		

- For other models, you can check the config.json file inside the model repository. If the "architectures" field contains a model architecture listed below, then it should be supported in theory.

2. vLLM

- 지원 모델
  - 다운로드한 HuggingFaceTB/SmolLM2-360M-Instruct 모델을 vllm 이 지원하는지 확인



`LlamaForCausalLM`	Llama 3.1, Llama 3, Llama 2, LLaMA, Yi	`meta-llama/Meta-Llama-3.1-405B-Instruct`, `meta-llama/Meta-Llama-3.1-70B`, `meta-llama/Meta-Llama-3-70B-Instruct`, `meta-llama/Llama-2-70b-hf`, `01-ai/Yi-34B`, etc.

2. vLLM

- Installation
  - 기본적으로 linux OS만 지원
  - Window 에서 사용시에는 WSL 환경에서 사용할 수 있음
  - Vs code 는 WSL 과 쉽게 연동됨

2. vLLM

- Install GPU version (cuda)
  - https://docs.vllm.ai/en/latest/getting_started/installation.html

vLLM is a Python library that also contains pre-compiled C++ and CUDA (12.1) binaries.

Requirements

- OS: Linux
- Python: 3.9 – 3.12
- GPU: compute capability 7.0 or higher (e.g., V100, T4, RTX20xx, A100, L4, H100, etc.)

```
# (Recommended) Create a new conda environment.
conda create -n myenv python=3.10 -y
conda activate myenv

# Install vLLM with CUDA 12.1.
pip install vllm
```

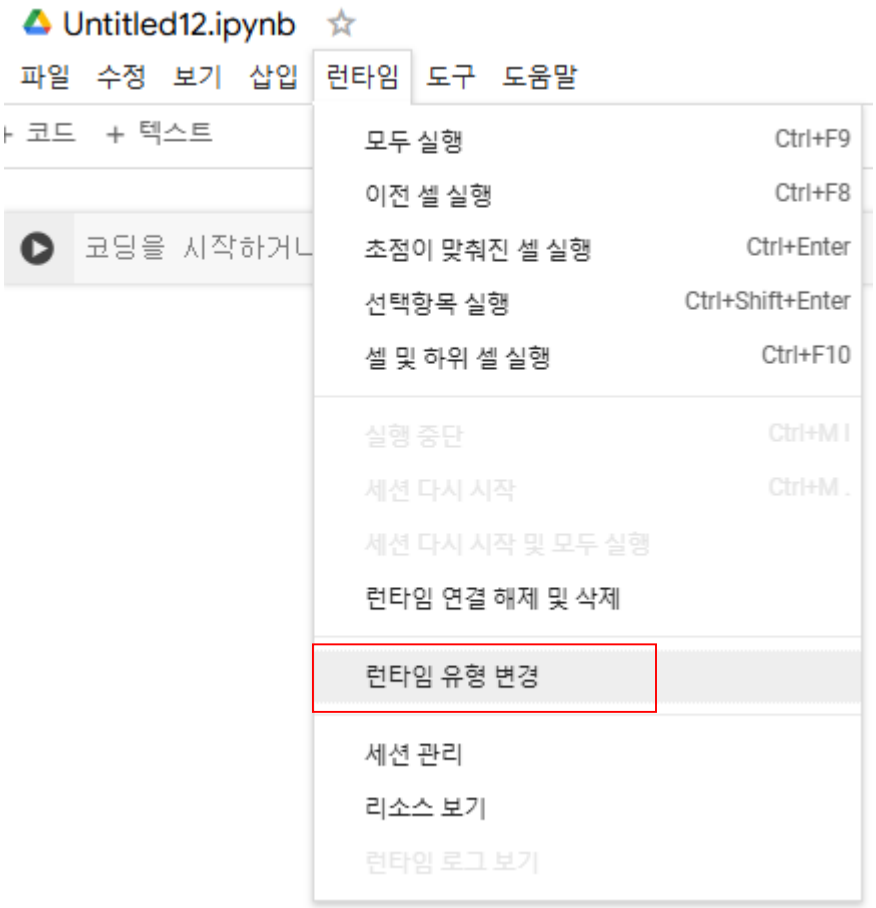
2. vLLM

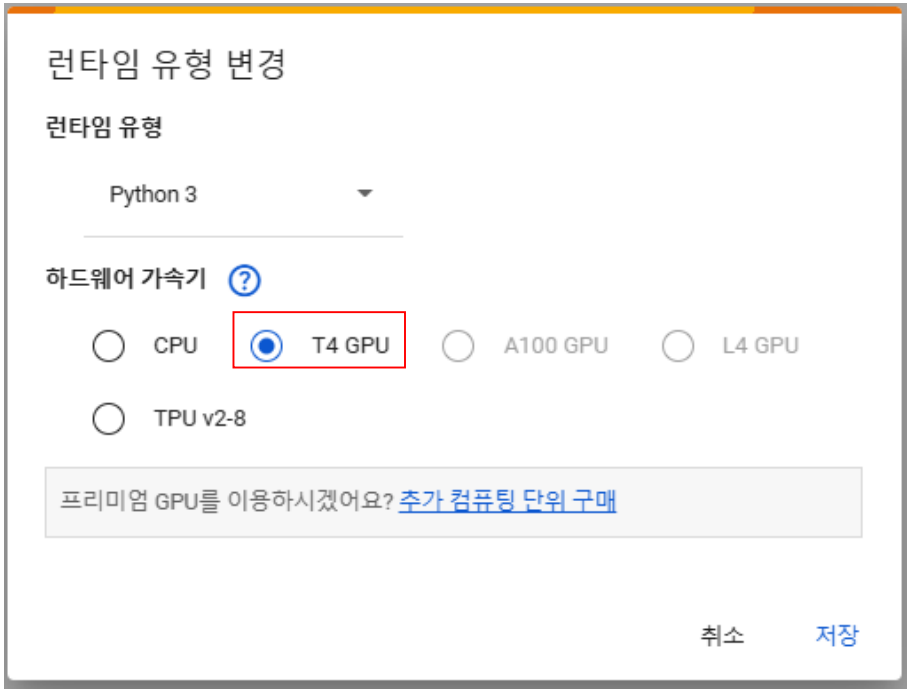
- Install CPU version
  - https://docs.vllm.ai/en/latest/getting_started/cpu-installation.html
  - Requirement:
    - OS: Linux
    - Compiler: gcc/g++>=12.3.0 (optional, recommended)
    - Instruction set architecture (ISA) requirement: AVX512 (optional, recommended)

```
sudo apt-get update  -y
sudo apt-get install -y gcc-12 g++-12 libnuma-dev
sudo update-alternatives --install /usr/bin/gcc gcc
/usr/bin/gcc-12 10 --slave /usr/bin/g++ g++ /usr/bin/g++-12

pip install --upgrade pip
pip install cmake>=3.26 wheel packaging ninja "setuptools-
scm>=8" numpy
pip install -v -r requirements-cpu.txt --extra-index-url
https://download.pytorch.org/whl/cpu

VLLM_TARGET_DEVICE=cpu python setup.py install
```

2. vLLM

- vLLM with colab
  - 사전에 GPU 를 사용하도록 설정





** 세션이 새로 시작될 때 마다 매번 설정 필요

2. vLLM

- Text generation example (1)

20_vllm_text_generation_1.py

```
!pip install vllm
!pip install triton

from vllm import LLM, SamplingParams

model = LLM(model="facebook/opt-125m")
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)

prompts = [
  "The future of AI is",
]

outputs = model.generate(prompts, sampling_params)
answer  = outputs[0].outputs[0].text
print(answer)
```

```
Processed prompts: 100%|██████████| 1/1 [00:00<00:00,  7.12it/s, est. speed input: 43.14 toks/s, output: 115.02 toks/s] cloud computing

A recent report by Arvind Kejriwal suggests AI will be on
```

2. vLLM

- Text generation example (2)

18_vllm_text_generation_2.py

```
!pip install vllm
!pip install triton

from vllm import LLM, SamplingParams

model = LLM(model="microsoft/Phi-3-mini-4k-instruct", dtype="float16")
sampling_params = SamplingParams(temperature=0.8, top_p=0.95,
                                 max_tokens=500)

prompts = [
  "show me python example for histogram chart",
]

outputs = model.generate(prompts, sampling_params)
answer  = outputs[0].outputs[0].text
print(answer)
```

2. vLLM

```
[10] outputs = model.generate(prompts, sampling_params)
     answer  = outputs[0].outputs[0].text
     print(answer)
```

```
Processed prompts: 100%|██████████| 1/1 [00:13<00:00, 13.21s/it, est. speed input: 0.61 toks/s, output: 30.59 toks
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Assuming 'data' is a dictionary containing 'distance' and 'frequency'
data = {
    'distance': np.array([10, 20, 30, np.nan, 40, 50]),  # Example data with NaN
    'frequency': np.array([5, 10, 15, np.nan, 20, 25])
}

# Filter out NaN values
mask = ~np.isnan(data['distance']) & ~np.isnan(data['frequency'])
filtered_distance = data['distance'][mask]
filtered_frequency = data['frequency'][mask]

# Plotting the histogram
plt.hist(filtered_distance, bins=5, range=(10, 50), color='blue', edgecolor='black', weights=filtered_frequency)
plt.title('Histogram of Distances')
plt.xlabel('Distance')
plt.ylabel('Frequency')

# Set y-axis limit
plt.ylim(0, 30)

plt.show()
```

This code snippet will create a histogram that ignores any NaN values and limits the y-axis from 0 to 30. Is there

2. vLLM

- Note. Colab에 영구적으로 패키지 설치하기
  - 새로운 세션을 시작하거나 런타임을 초기화하는 경우 패키지 설치를 반복적으로 수행해야한다는 번거로움
  - https://dacon.io/codeshare/4200

```
import os, sys
from google.colab import drive
drive.mount('/content/drive')

my_path = '/content/package'
save_path = '/content/drive/MyDrive/Colab Notebooks/package' ## 패키지가 저장될 경로

os.symlink(save_path, my_path)
sys.path.insert(0, my_path)
```

```
!pip install --target=$my_path selenium
```

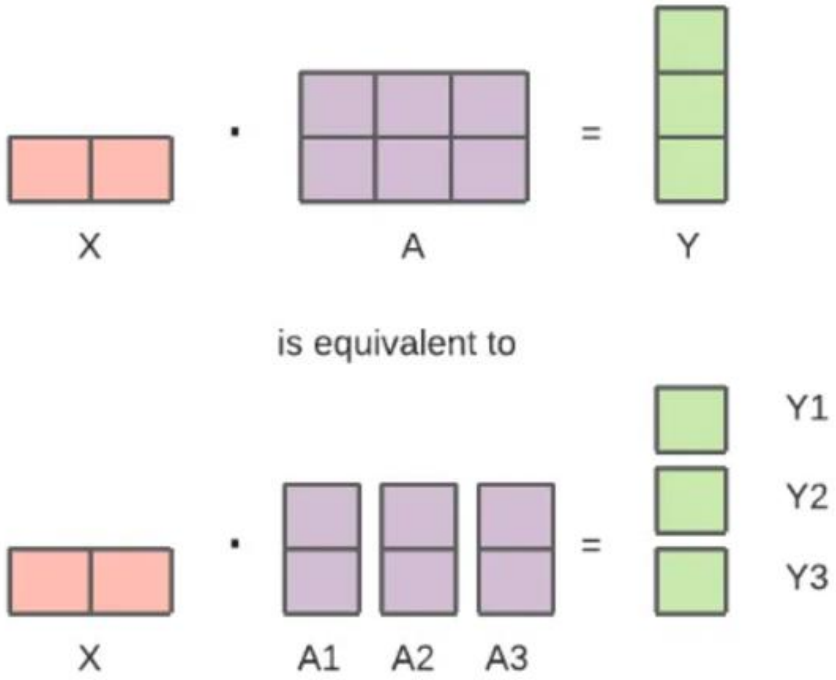
2. vLLM

- 영구 설치한 package import 하기

```
import os, sys
from google.colab import drive
drive.mount('/content/drive')

my_path = '/content/package'
save_path = '/content/drive/MyDrive/Colab Notebooks/package'

os.symlink(save_path, my_path)
sys.path.insert(0, my_path)

import selenium
```

3. Speed up Methods

- Tensor Parallelism
  - weights, gradient 그리고 optimizer 상태와 같은 텐서를 여러 chunk로 분산하여 다중 GPU 장비에서 분산 저장하고 처리



Reference : Transformers — Efficient Training on Multiple GPUs

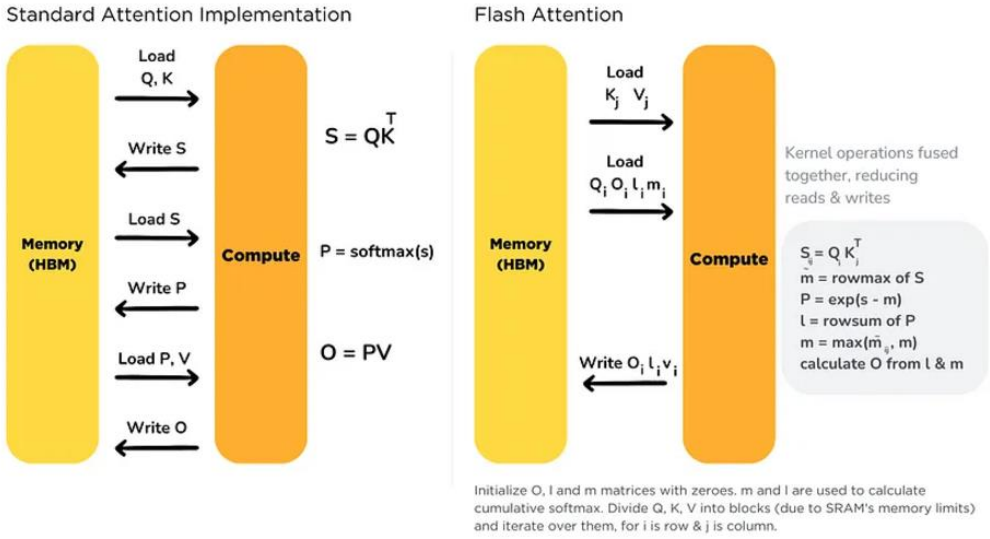
3. Speed up Methods

- Quantization with bitsandbytes or gptq
  - **Quantization(양자화)**
    - 실수형 변수를 정수형 변수로 변환하는 과정
    - 주로 딥러닝에서 FP32(32-bit floating point)는 사용하여 가중치를 표현하는데, 이를 FP16(16-bit floating point)이나 INT8(8-bit integer)로 변환하여 모델의 크기를 줄일 수 있게 된다.
  - **Bitsandbytes**
    - QLoRA를 논문으로 발표한 Tim Dettmers에 의해 배포된 양자화 라이브러리
  - **GPTQ**
    - LLM을 양자화하는 또다른 방법
    - Post-Training Quantization(PTQ)를 통해 32비트에서 3~4비트로 정수 양자화를 수행
  - 양자화는 추론시에만 적용 (학습시에는 X)

3. Speed up Methods

- Optimized transformers code for inference using flash-attention
  - **Flash Attention**은 Google Research Brain 팀에 의해서 개발된 메커니즘
  - N을 시퀀스 길이라고 할 때, 전통적인 Attention은 $O(N^2)$의 메모리 복잡성
  - 이로 인해 모델의 효율과 속도가 떨어지는 문제가 발생.
  - Flash Attention은 메모리 복잡성을 선형 복잡도인 O(N)까지 줄임.
  - 이를 통해 훈련과 추론 속도를 가속화

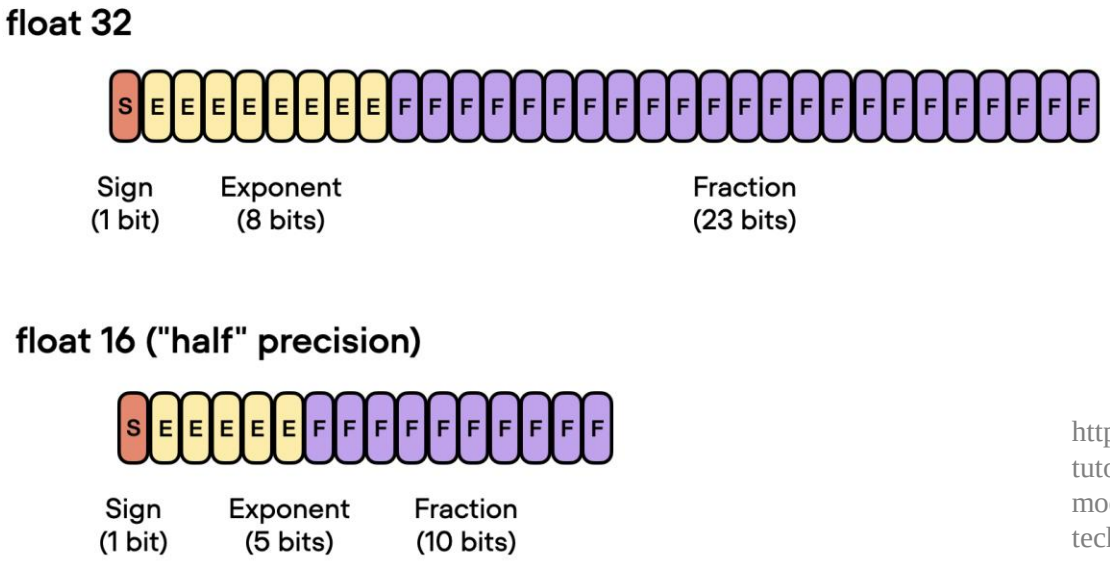
  - 다른 최적화 기법과는 달리, Flash Attention은 하드웨어적으로 성능을 개선
    - GPU 내 메모리 계층에 따라 용량과 속도가 다르다는 것에서 착안



https://huggingface.co/docs/text-generation-inference/conceptual/flash_attention

3. Speed up Methods

- Precision reduction
  - PyTorch는 32-bit floats를 디폴트로 사용
  - 이를 float16 or bfloat16로 변환하여 사용
  - 모델을 20%까지 빠르게 해주고, 메모리 소비를 2배 줄이는 효과

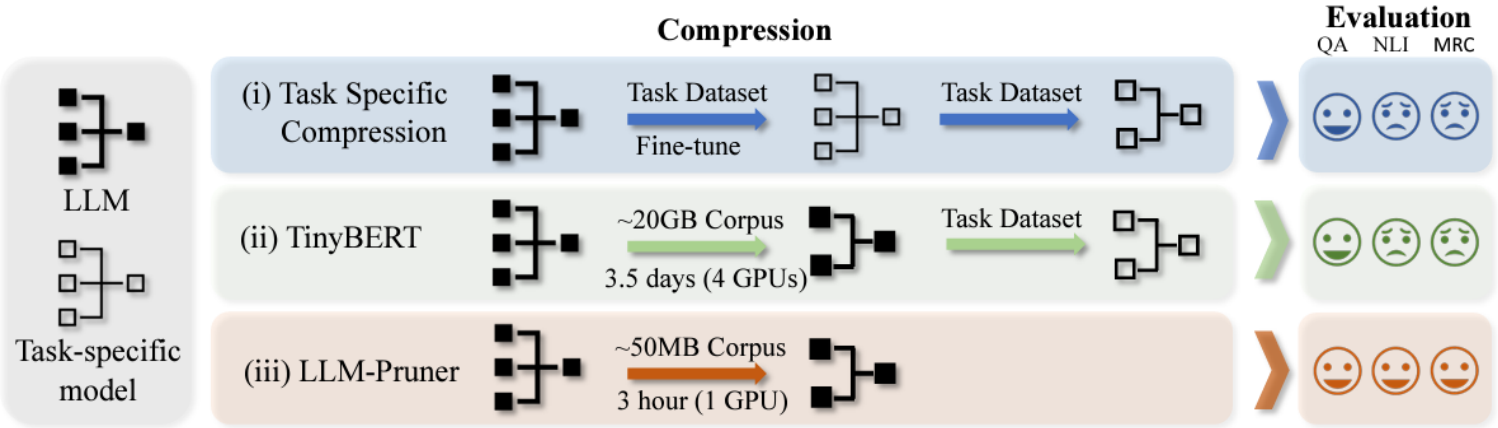


https://lightning.ai/pages/community/tutorial/accelerating-large-language-models-with-mixed-precision-techniques/

- Mixed-precision training
  - 학습 동안에, 계속 16-bit을 쓰지 않고 필요에 따라 16-32를 왔다갔다함.
  - 이렇게 하면 accuracy와 stability를 유지하면서 학습이 가능

3. Speed up Methods

- Pruning
  - 모델 학습 시 중요한 파라미터(결과에 유의미한 영향을 끼치는)는 살리고 아닌건 가지치기하는 경량화 방법.
  - Pruning은 추론속도를 높여주고(아예 파라미터가 줄어들기 때문에), 모델의 복잡도를 줄여서 Robust하게 만들어준다.
  - 원하는 정도만큼 prune이 가능 (50%까지)
  - https://arxiv.org/pdf/2305.11627

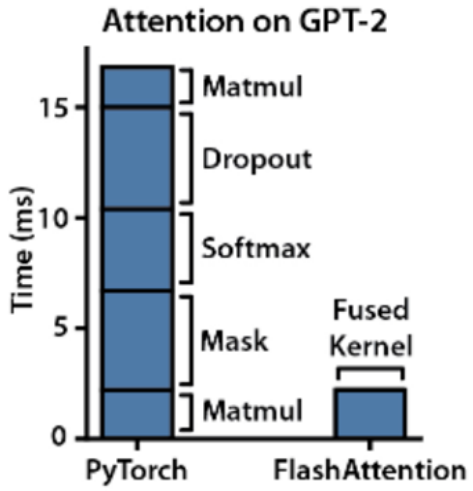


3. Speed up Methods

- Batch inference
  - https://dytis.tistory.com/59
  - ML training 에서의 batch 와 유사 개념
  - Inference 에서의 batch : 모델이 동시에 처리하는 입력 데이터의 묶음
  - 여러 입력을 한번에 모델에 전달하여 병렬처리 -> 성능향상
  - 여러 종류의 batch inference 방법이 존재

3. Speed up Methods

- Kernel fusion
  - GPU는 주로 다양한 수학적 연산, 예를 들면 MatMul, Softmax, BatchNorm 등을 수행
  - GPU에서 연산이 실행되면, 필요한 데이터를 메모리로 불러와 연산을 수행한 후, 결과를 다시 메모리에 저장
  - kernel fusion은 기본적으로 연속된 독립적인 계산 작업들을 단일 하드웨어 작업으로 통합
  - 결과적으로 여러 독립적인 계산을 하나로 통합하여 메모리 이동을 최소화



https://github.com/Dao-AILab/flash-attention

FlashAttention에서 제공하는 fused Kernel의 성능 향상을 시각적으로 보여줌