Seminar

Evaluating Large Language Models & Scaling

2025.5.26

LLM 모델 평가

언어 모델 평가 방법

- BLEU(텍스트 번역에 대한 성능 지표)
- ROUGE(텍스트 요약에 대한 성능 지표)

→ 다양한 목적에 맞게 성능을 측정하는 방법을 적절히 선택

언어 모델 평가 방법 : Perplexity(PPL)

- 텍스트 생성 언어 모델의 성능 평가지표 중 하나

특징 : 테스트 세트의 역확률을 단어의 수가 정규화한 것, 단어 시퀀스의 확률이 높을수록 Perplexity가 낮아짐

$$\begin{aligned}\text{Perplexity}_\theta(w_{1:n}) &= P_\theta(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P_\theta(w_{1:n})}}\end{aligned}$$

Perplexity 산정식

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i|w_1\ldots w_{i-1})}}$$

Chain Rule 기반 Perplexity 산정식

Perplexity의 특징

Perplexity(PPL)의 핵심 : Chain Rule

- 전체 문장의 확률을 각 단어의 조건부 확률의 곱으로 나타내는 규칙

→ 언어 모델이 단어 사이의 관계를 학습함을 보임

Perplexity(PPL) 평가 : 낮을수록 더 좋은 모델로 판단

- 단어 시퀀스의 확률이 높을수록 Perplexity는 낮아지기 때문

주의점 : 토큰화 알고리즘의 차이에 매우 민감

(매우 다른 토크나이저를 사용하는 두 언어 모델의 Perplexity를 정확하게 비교하는 것은 어려움)

→ 텍스트 길이에 따라 차이가 발생하기 때문

추가 주의점

- Perplexity를 계산할 때, 언어 모델은 테스트 세트에 대한 지식 없이 구성

Perplexity는 모델의 언어 이해 능력을 측정하는 시험 점수와 같음
시험 점수가 높다고 해서 실제 사회생활에서 모든 일을 잘 할 수 있는 것은 X
Perplexity가 낮다고 해서 모든 자연어 처리 작업에서 완벽한 성능을 보장하는 것은 아님.

Perplexity 계산 예시

branch factor = 3

L = {red, blue, green}

모델 1

모든 확률 : 1/3

$$= ((\frac{1}{3})^{5})^{-\frac{1}{5}}$$

$$= (\frac{1}{3})^{-1}$$

perplexity : 3

TEST SET

S = {red, red, red, red, blue}

모델 2

확률 : RED = 0.8 | BLUE = 0.1 | GREEN = 0.1

$$(0.8)^{4} \times (0.1)^{1}$$

$$= 0.04096^{-\frac{1}{5}}$$

perplexity : 1.89

새로운 접근 방법

언어 모델을 적용하는 '다운스트림 작업'을 수행하도록 다양한 종류의 정확도에 관심
- 기계 번역, 요약, 질의응답, 음성 인식 등

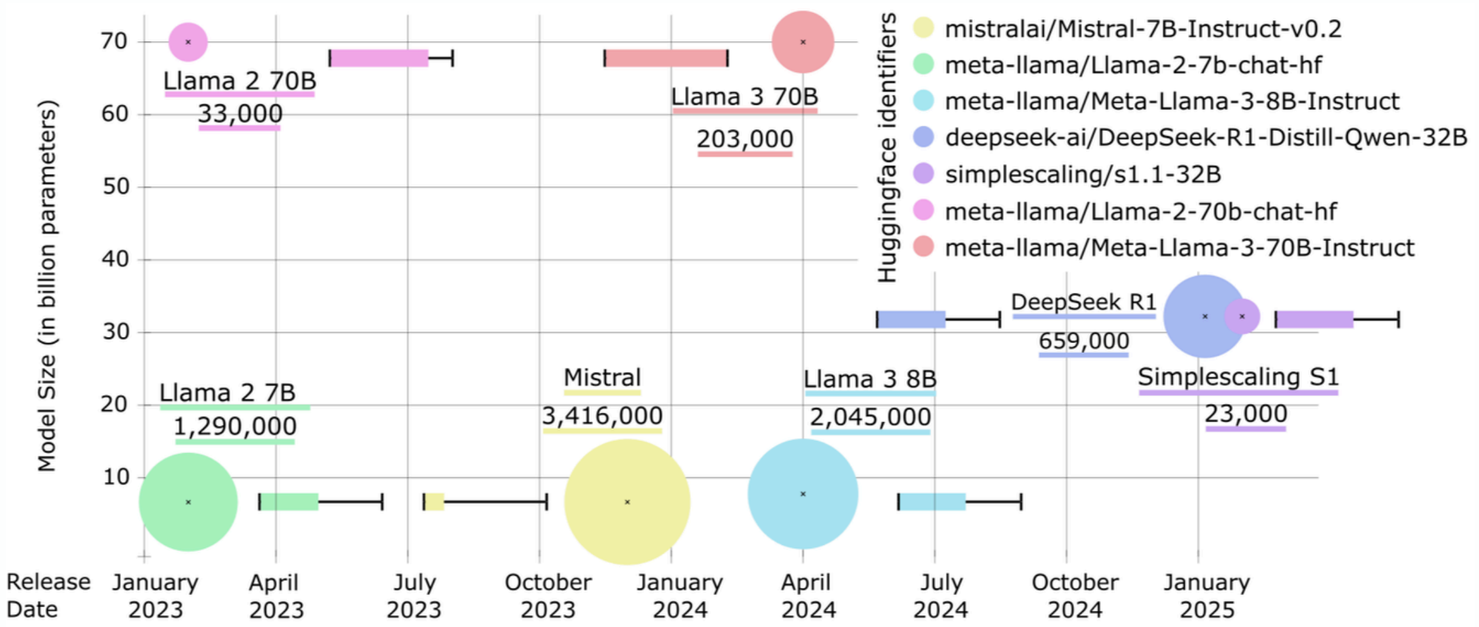
평가에 고려될 수 있는 요소 : 모델의 크기, 학습 속도, 추론 속도
- CNN Architecture에서도 경량화, 이미지 추론 속도 향상을 위해 모델을 개선하듯이 LLM 모델도 동일

책에서 언급하는 특징 : '공정성'
언어 모델이 특정 그룹과 구분에 따라 성능이 저하되는 '편향성'을 고려

편향 강도를 측정하기 위한 언어 모델 평가 벤치마크
- StereoSet(https://github.com/moinnadeem/StereoSet) - 인종, 나라 데이터
- RealToxicityPrompts(https://huggingface.co/datasets/allenai/real-toxicity-prompts) - 일상 문장 데이터
- BBQ(https://paperswithcode.com/dataset/bbq) - Q&A 데이터

Dynabench와 같은 다양한 종류의 리더보드와 HELM과 같은 일반적 평가 프로토콜 존재
(질의응답, 정보 검색과 같은 작업에 대한 평가 지표를 소개할 때 다시 다룰 예정)

Scale 다루기



출시 언어 모델의 Model Size 비교

Ex) Meta의 Llama 3.1 405B Instruct는 4050억개의 매개변수, 128K의 토큰 어휘집을 사용, 15.6TB의 텍스트 토큰으로 학습

최근 연구 방향 : 제한된 자원으로 LLM 모델을 구현하는 방법

앞으로 Scaling 방법 및 KV Cache&Tuning 기술을 언급

Scaling 법칙

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}$$
$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}$$
$$L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}$$

매개변수, 데이터셋 크기, 컴퓨팅 예산에 따른 관계

LLM 언어 모델의 성능에 영향을 주는 3가지 대표 요인 : 모델 크기, 데이터셋 크기, 학습에 사용된 컴퓨팅 양
→ 대규모 언어 모델의 성능(손실)은 모델 학습의 세 가지 속성 각각에 따라 거듭제곱 법칙으로 확장

개선(향상) 방법 : 매개변수 추가, 더 많은 데이터를 학습, 더 많은 반복으로 학습
→ 이러한 요인들과 성능 사이의 관계를 Scaling laws(스케일링 법칙)

Scaling 법칙

$$
\begin{aligned}
N &\approx 2\, d\, n_{\text{layer}}(2\, d_{\text{attn}} + d_{\text{ff}}) \\
&\approx 12\, n_{\text{layer}}\, d^2 \\
&\quad (\text{assuming } \; d_{\text{attn}} = d_{\text{ff}}/4 = d)
\end{aligned}
$$

LLM 언어 모델에서의 매개변수 N 계산법

96개의 레이어와 d=12288의 차원을 가진 GPT-3는 12×96×122,88의 2의 제곱
1,750억 개의 매개변수를 가짐

N_c, D_c, C_c, α_N, α_D, α_C의 값은 정확한 트랜스포머 아키텍처, 토큰화, 어휘 크기에 따라 달라지므로, 스케일링 법칙은 정확한 값보다는 손실과의 관계에 중점을 둠

스케일링 법칙의 유용성 : 특정 성능 수준에 도달하기 위해 모델 학습에 영향을 줌
Ex) 학습 곡선의 초기 단계, 적은 양의 데이터로 얻은 성능을 통해 더 많은 데이터 추가와 모델 크기를 늘리면 손실이 어떻게 될지 예측 가능, 모델을 확장할 때 얼마나 많은 데이터를 추가해야 하는지도 알려줄 수 있음

KV 캐시

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}}\right)\mathbf{V}$$

Attention 구하는 식(KV Cache에 고려됨)

Self-attention을 위해 현재 시퀀스에 있는 각 항목의 key-value 값을 필요로 하며, 이러한 벡터들은 KV Cache로 제공
→ Auto Regressive 방식에서 생기는 비효율(중복 계산)을 줄여보려는 시도

Auto Regressive : 출력이 다시 입력이 돼서 다음 출력을 생성하는 방법

Ex)
input : “우리나라의 수도는 어디야?”
model : “대한민국의”를 생성
→ “우리나라의 수도는 어디야? 대한민국의“
model : “수도는”
→ “우리나라 수도는 어디야? 대한민국의 수도는“
model : “서울입니다”
→ “우리나라 수도는 어디야? 대한민국의 수도는 서울입니다“
model : 문장 끝&문장 출력

KV 캐시

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

Attention 구하는 식(KV Cache에 고려됨)

Auto Regressive 관점에서 발생하는 2가지 문제

1. 한 토큰씩 생성을 반복하기 때문에 생성하려는 결과의 길이만큼 시간이 길어짐
2. Self-Attention과 관련, 해당 방식은 Self-Attention 계산을 중복으로 하는 경우가 많아짐

개선하기 위해서 사용되는 방법은 Grouped Query Attention(GQA), Multi-Headed Attention(MHA) 등이 존재

책에서는 KV Cache에 대해서 언급

KV 캐시

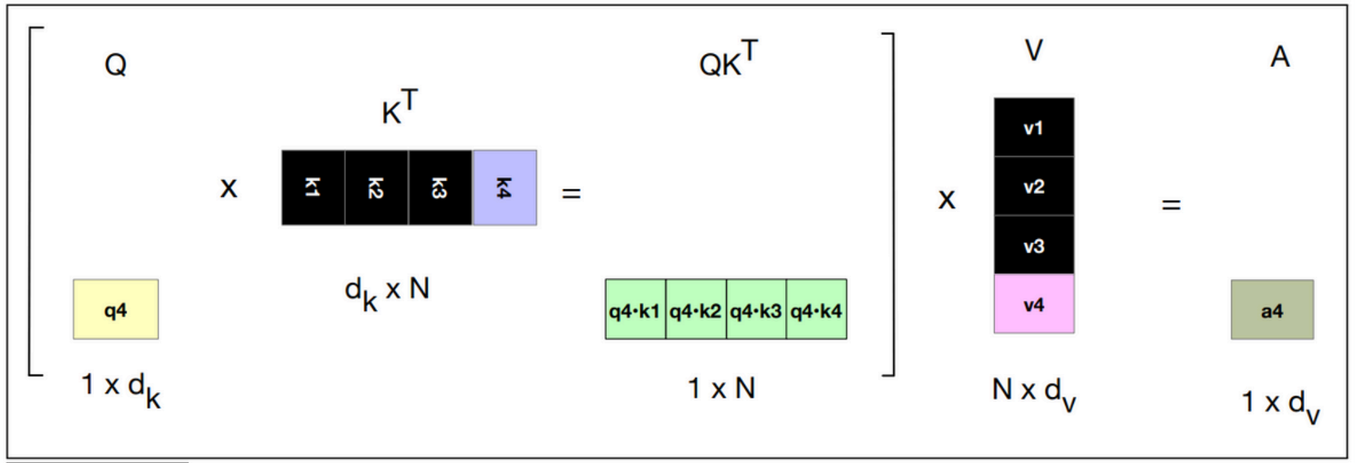


Figure 10.7 Parts of the attention computation (extracted from Fig. **??**) showing, in black, the vectors that can be stored in the cache rather than recomputed when computing the attention score for the 4th token.

추론 시에는 학습 시와 똑같은 효율적인 계산을 수행할 수 없음
(추론 시 다음 토큰을 한 번에 하나씩 반복적으로 생성하기 때문)

방금 생성한 새 토큰을 x_i라고 부르면, W^Q, W^K, W^V를 각각 곱하여 Q,K값을 계산해야 함
하지만 이전 토큰을 다시 계산하는 것은 계산 시간을 낭비하는 일
(이전 단계에서 이미 이 키 및 값 벡터를 계산했기 때문)
다시 계산하는 대신, 키 및 값 벡터를 계산할 때마다 KV 캐시에 저장하고, 필요할 때 캐시에서 가져와 사용할 수 있도록 함!

KV 캐시

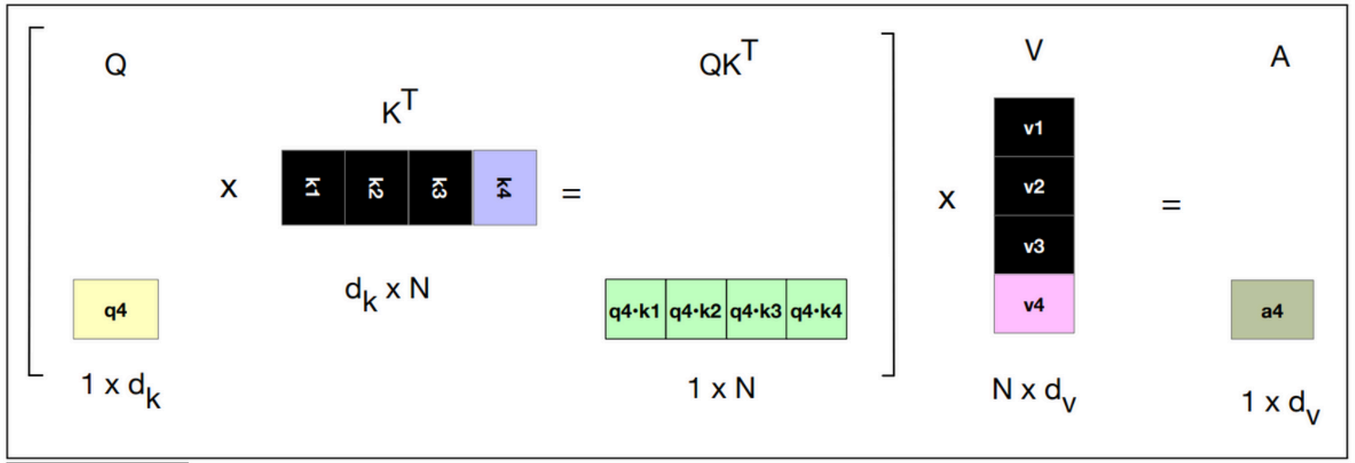


Figure 10.7 Parts of the attention computation (extracted from Fig. **??**) showing, in black, the vectors that can be stored in the cache rather than recomputed when computing the attention score for the 4th token.

단일 새 토큰에 대해 발생하는 계산 소개 및 다시 계산하는 대신 Cache에서 가져올 수 있는 값들을 보여줌
(KV Cache는 이전 토큰을 생성할 때 이미 계산한 중간값들을 저장했다가 나중에 재사용한다는 아이디어)
검은색으로 표시된 부분은 4번째 토큰에 대한 Attention score를 계산 시, 다시 계산할 필요 없이 Cache에 저장할 수 있는 벡터들

주의할 점 : KV Cache는 Auto Regressive라는 방식에서 생기는 문제를 해결하기 위한 방법, 모든 입력이 한 번에 제공되는 BERT와 같은 모델에서는 사용할 수 없음

PEFT

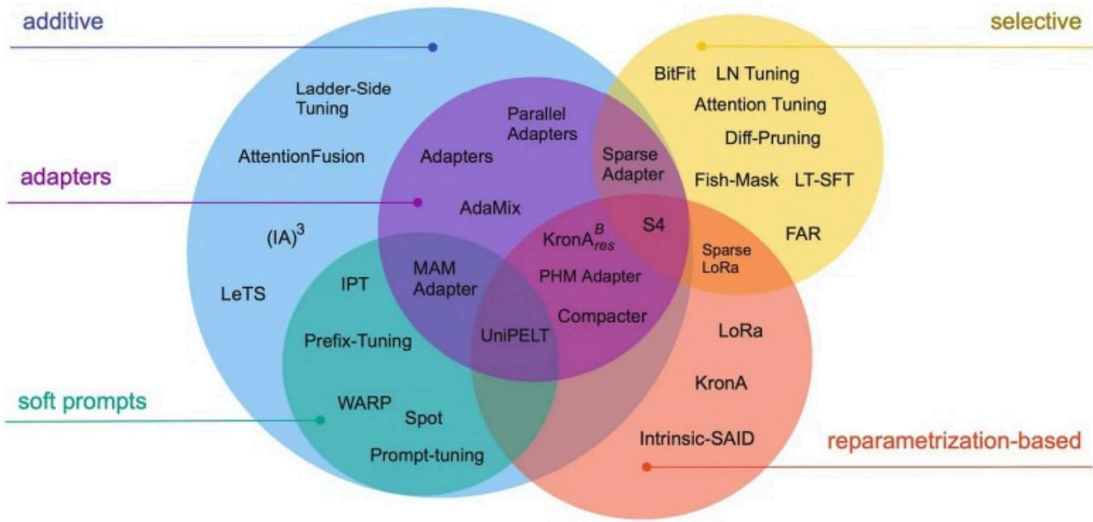
미세 조정을 하는 이유 : LLM에 새로운 도메인에 대한 추가 정보 제공 목적

LLM의 경우 엄청난 수의 매개변수를 학습하므로 복잡

- 배치 경사 하강법의 각 단계는 수많은 거대한 레이어를 통해 역전파해야 함
- LLM을 미세 조정하는 것은 처리 능력, 메모리, 시간 면에서 엄청나게 비쌈

PEFT : 모든 매개변수를 변경하지 않고도 모델을 미세 조정하기 위한 방법
(미세 조정 시 업데이트 할 매개변수들의 하위 집합을 효율적으로 선택)

Ex) 일부 매개변수는 고정하고, 특정 매개변수 하위 집합만 업데이트



PEFT

LoRA(Low-Rank Adaptation) : Low-Rank 방법을 사용하여 속도 및 resource cost를 줄이는 것을 목표
→ **모델 적응기간동안 가중치의 변화에도 적은 본질적인 rank를 가질 것이라고 가정**

LoRA 의 사상은 다음과 같음

- Fully FIne-Tuning 하지 않음
- Model weight 를 Freeze 함
- 학습하는 Layer 는 LoRA_A & LoRA_B (둘 다 nn.linear 형태)
- Transformer Layer 에 있는 Query, Key, Value, Output(=self attention) 중 선택하여 (LoRA_B x LoRA_A)를 단순히 더해줌

미세 조정 중에 이러한 레이어를 업데이트하는 대신, LoRA는 레이어를 고정, 더 적은 매개변수를 가진 저랭크 근사치를 업데이트
(Transformer에 행렬 곱셈을 수행하는 많은 밀집 레이어가 존재, (W^Q, W^K, W^V, W^O))

순방향 전달 h = xW를 대체하기 위한 새로운 순방향 전달 방법

$$\mathbf{h = xW + xAB}$$

PEFT

차원 N×d인 행렬 W가 경사 하강법을 통해 미세 조정 중에 업데이트되어야 한다고 가정

일반적으로 이 행렬은 경사 하강법 이후 N×d개의 매개변수를 업데이트하기 위해 차원 N×d의 ΔW 업데이트를 받음

LoRA에서는 W를 고정하고 대신 W의 저랭크 분해를 업데이트함
우리는 두 개의 행렬 A와 B를 생성 : A는 크기가 d×r, B는 r×d
(여기서 r은 상당히 작은 값, 즉 r≪min(d,N)로 선택)

미세 조정 중에 우리는 W 대신 A와 B를 업데이트함
(즉, W+ΔW를 W+BA로 대체)

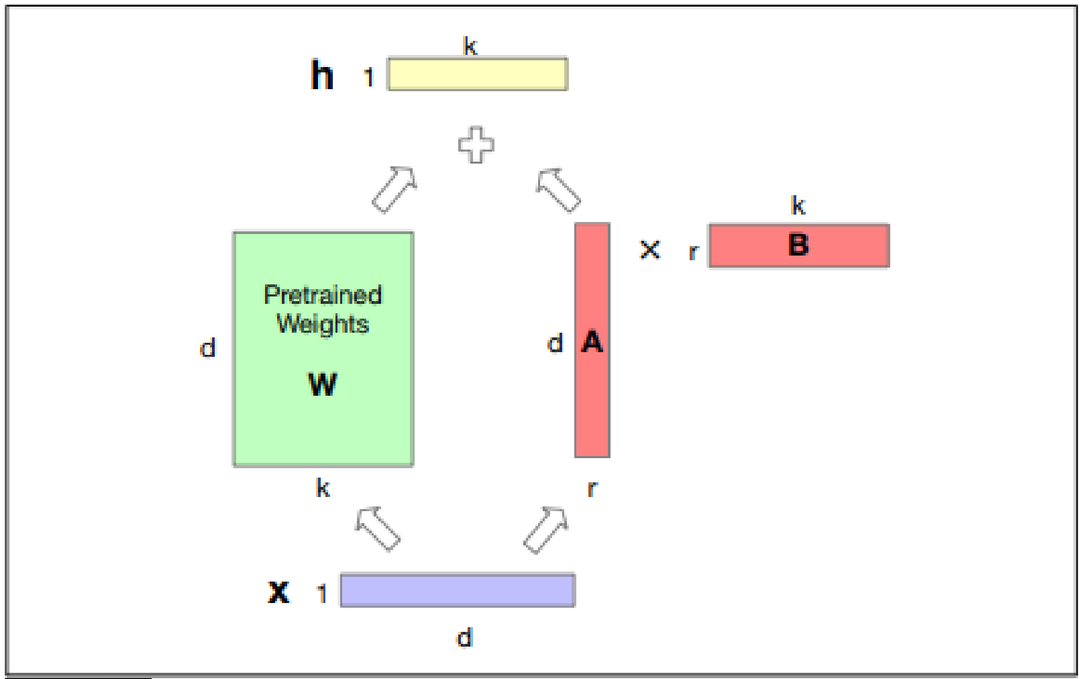


Figure 10.8 The intuition of LoRA. We freeze **W** to its pretrained values, and instead fine-tune by training a pair of matrices **A** and **B**, updating those instead of **W**, and just sum **W** and the updated **AB**.

```
lora_config = LoraConfig(
    r = 16,
    lora_alpha = 8,
    target_modules = ['query', 'value'],
    lora_dropout = 0.05,
    bias = 'none',
    task_type = 'SEQ_CLS'
)
```

PEFT

LoRA의 장점

- 대부분의 매개변수에 대해 기울기를 계산할 필요가 없으므로 하드웨어 요구 사항을 극적으로 줄여줌
- BA가 W와 동일한 크기, 가중치 업데이트를 사전 학습된 가중치에 간단히 추가 가능

(추론 시 시간을 전혀 추가하지 않음 & 다양한 도메인에 대한 LoRA 모듈을 구축, W에 추가하거나 W에서 빼는 방식으로 쉽게 교체 가능)

논문 저자의 언급

- Pretrain Model weight 를 update 하지 않고도 Fully Fine-Tuning 한 결과와 비슷하거나 더 좋은 성능을 보임
- weight update 에 필요한 weight 가 오직 LoRA_A, LoRA_B layer 에 있는 weight 뿐이기 때문에 vram 을 상당부분 save 할 수 있음
- 학습된 모델로 inference 해도 Pretrain Model 로 inference 할 때와 동일한 연산량 (추론 시간이 거의 동일)
- Pretrain Model weight 를 Freeze 한 상태로 (LoRA_b x LoRA_A) 행렬을 단순히 더해주기 때문에 Pretrain Model weight 로 다시 원복하기 쉬움 (더해준 만큼 빼면 되기 때문)