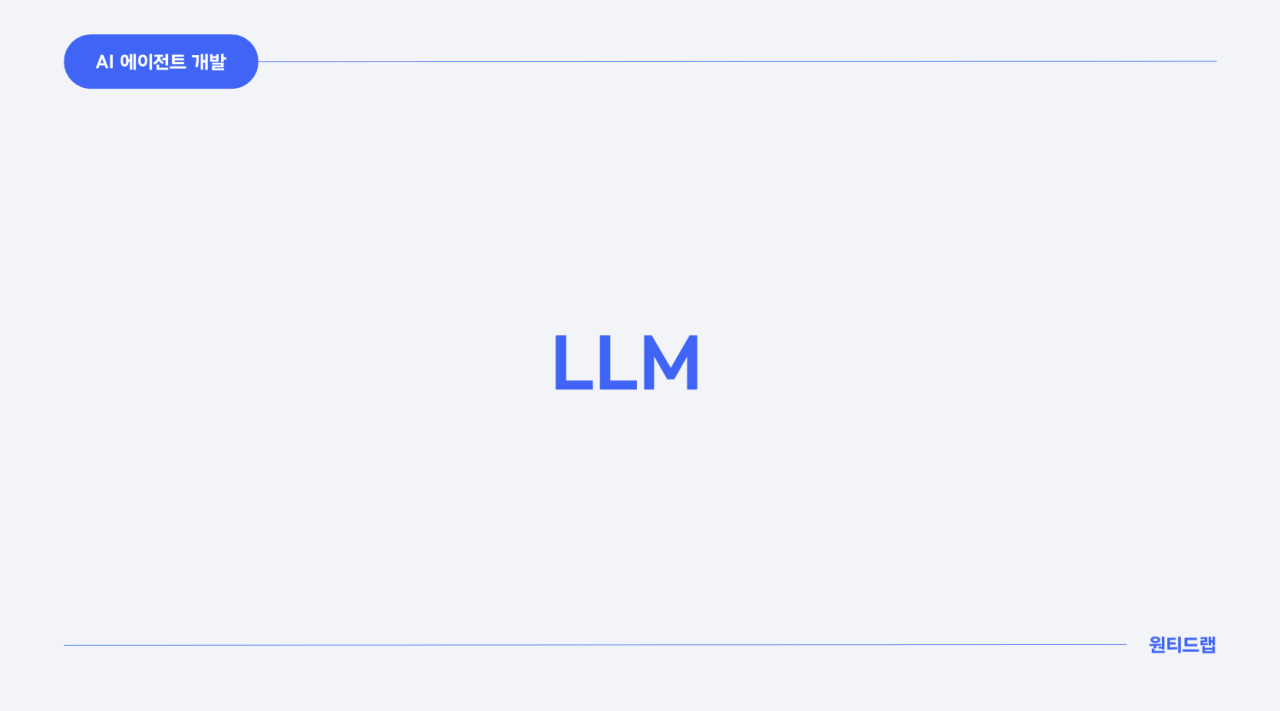
05 LLM 파인튜닝



1. 파인튜닝의 목적

- **Base Fine-tuning**
  - 모델이 아직 모르는 지식을 새로 학습시키는 과정
  - 예: 의료 논문, 법률 문서, 특정 기업 내부 문서
  - 순수 텍스트를 넣고 언어모델링(Language Modeling, 다음 단어 예측)을 통해 새로운 도메인 지식을 흡수하게 함
- **Instruct Fine-tuning (SFT)**
  - 모델이 이미 알고 있는 지식을 특정 **말투, 포맷, 지시**에 맞춰 표현하도록 학습하는 과정
  - 예: "답변은 항상 JSON으로", "공손한 어투로 대답해라"
  - 프롬프트(prompt, 입력 지시문)와 응답(output) 쌍을 만들어 지도 학습(Supervised Fine-tuning)으로 진행함

2. 파인튜닝 방법

1) Full Fine-tuning

- 모델의 모든 파라미터(parameter, 가중치)를 업데이트함
- 장점: 가장 성능이 높음
- 단점: GPU 메모리와 시간이 많이 필요함

2) PEFT(Parameter-Efficient FT)

- 원래 모델(베이스 모델)은 그대로 두고, 그 옆에 **어댑터(Adapter)**라는 작은 장치를 붙여서 새로운 지식이나 패턴을 저장
- 학습할 때는 어댑터만 업데이트하고, 원래 모델 가중치는 바꾸지 않음

LoRA (Low-Rank Adaptation)

- 아이디어: 전체를 바꾸는 대신, W에 "작은 변화(ΔW)"만 주자
- LoRA(Low-Rank Adaptation)는 어댑터를 수학적으로 구현한 방식

- 큰 가중치 W는 그대로 두고, 작은 행렬 A와 B를 붙여서 "변화량 ΔW"만 표현함
- 추론할 때는 $W + (A \times B)$로 동작함
- 학습은 A, B만 업데이트하므로 **매우 효율적**
  - A: 입력을 **아주 낮은 차원으로 압축**하는 역할 (정보를 간단히 요약)
  - B: 압축된 정보를 다시 **원래 크기로 확장**하는 역할 (요약본을 펼쳐줌)

QLoRA

> ⓘ **양자화(Quantization)**
> 숫자를 더 적은 비트(bit)로 표현하는 것

- LoRA와 양자화(Quantization)를 결합한 방식
- 비트(bit): 컴퓨터가 수를 저장하는 최소 단위 (0 또는 1)

- 32비트: 큰 수를 정밀하게 표현 → 메모리 많이 사용.
- 4비트: 적은 수만 표현 → 메모리 절약
- QLoRA는 모델의 큰 가중치를 4비트로 줄여 GPU 메모리를 절약하고, LoRA 어댑터만 16비트로 학습.
- 결과적으로 Colab이나 개인 GPU에서도 대형 모델 파인튜닝이 가능해짐

3. 학습 과정

1) Base FT

(1) 학습 데이터의 이해

```
text = "안녕하세요. 파인튜닝을 시작합니다."
tokens = tok(text, return_tensors="pt")
print(tokens)

{
  'input_ids': tensor([[   2, 1234, 5678, 910, 111, 2222, 3333,    3]]),
  'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1]])
}
```

- `input_ids`: 텍스트를 숫자로 바꾼 결과 (모델이 실제로 보는 데이터)
- `attention_mask`: 패딩을 구분하는 마스크 (1: 실제 토큰, 0: 패딩)

(2) 학습 과정

```
1. 데이터셋 불러오기
2. 토크나이저로 텍스트를 숫자 토큰으로 변환
3. 여러 문장을 하나로 이어붙여 BLOCK_SIZE 단위로 나눔
4. DataCollatorForLanguageModeling을 사용해 배치 단위 데이터 생성
5. Trainer를 설정하고 학습 실행
```

> ⓘ DataCollator
>
> **단일 샘플들을 하나의 배치(batch)**로 묶는 단계에서 **패딩/마스킹/라벨 셋업**을 담당하는 구성 요소임.
> - 샘플 간 길이가 다를 때 **동적 패딩(dynamic padding)**으로 배치 텐서 크기를 정렬함.
> - 필요 시 **labels**를 생성하거나 패딩 위치를 -100으로 채워 **loss에서 제외**함.
> - (모델/태스크별) 추가 전처리를 수행함.

2) Instruct FT

(1) 학습 데이터의 이해

`tokenizer.apply_chat_template`를 사용하면 모델이 이해할 수 있는 프롬프트 문자열을 자동으로 생성

```
messages = [
    {"role": "user", "content": "서울 여행 코스를 추천해줘"},
    {"role": "assistant", "content": "경복궁, 남산타워, 홍대 일대를 추천함."}
]

출력 예시:
<start_of_turn>user
서울 여행 코스를 추천해줘
<end_of_turn>
<start_of_turn>assistant
```

```
경복궁, 남산타워, 홍대 일대를 추천함.
<end_of_turn>
```

(2) 학습 과정

```
1. 데이터셋을 불러오기
2. instruction, input, output을 chat template에 맞춰 변환
3. 데이터셋을 train/validation으로 분리
4. SFTTrainer를 설정
5. Trainer.train()으로 학습 실행
```