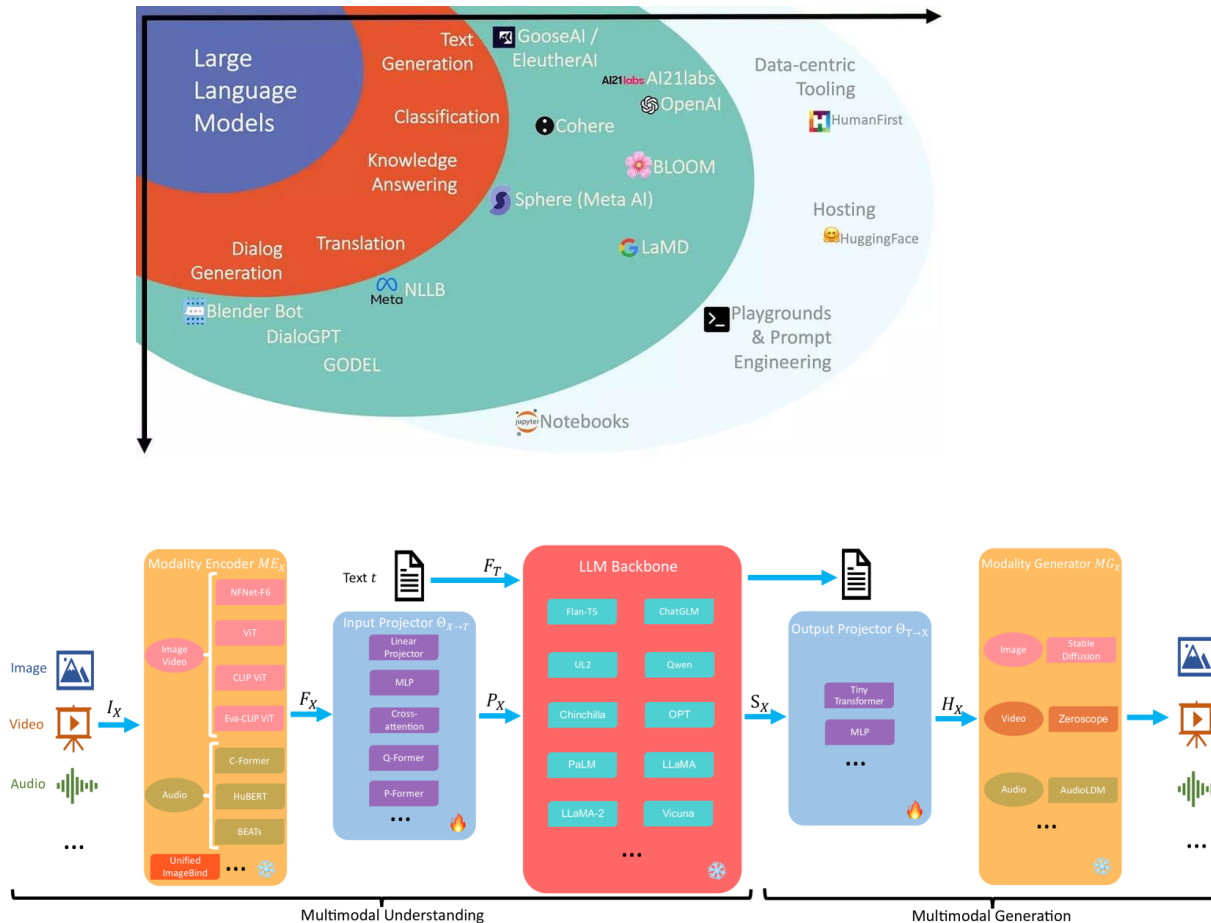


AI 실전 5주차

LLM을 개조 해볼까?

LLM 파인튜닝

LLM 생태계는?



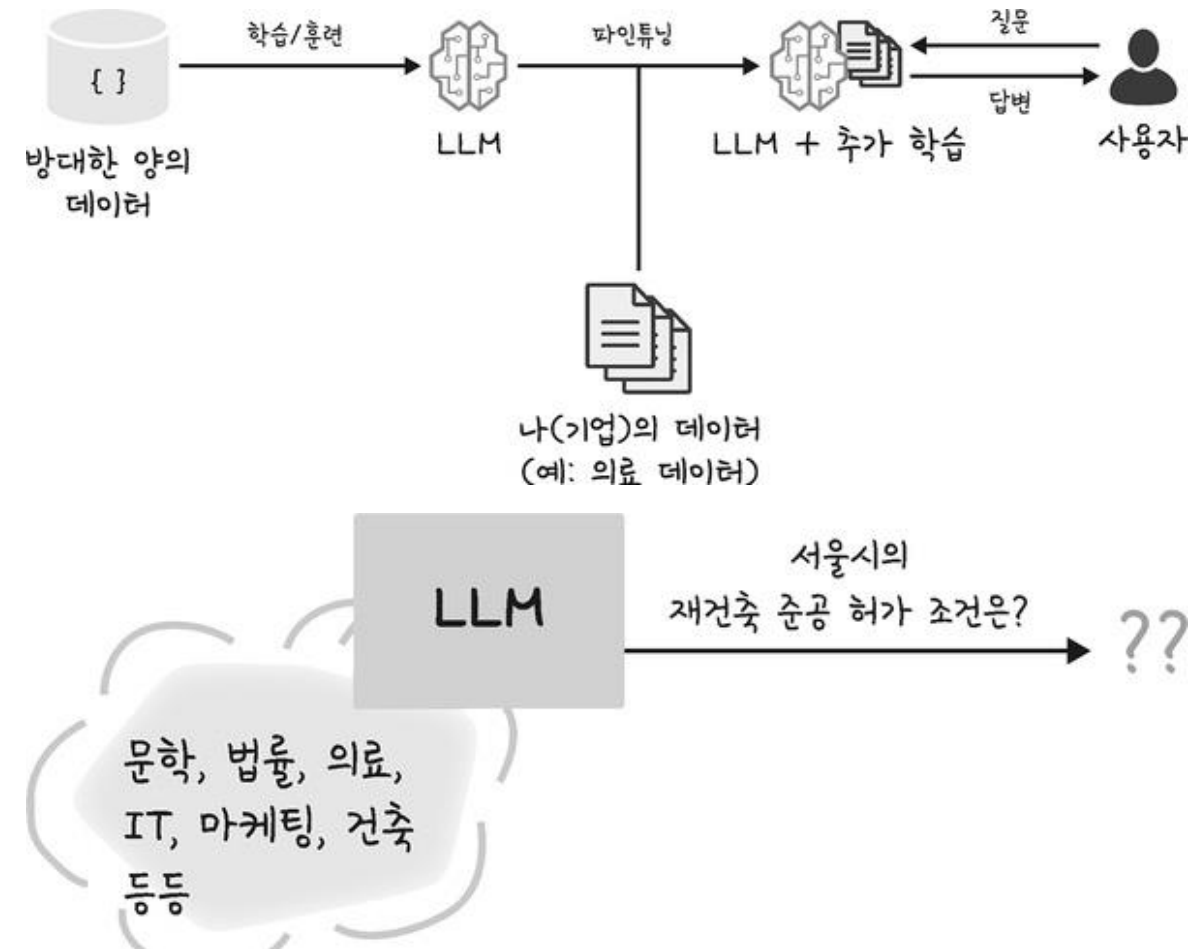
- Transformer 구조를 활용하여 언어 분류를 넘어서 생성형 모델로 발전 진행중에 있음.

- 생성의 분야는 번역, 텍스트 생성, 분류 등등 여러 분야 확장

- 질문에 대한 대답을 이해하기 위해 여러 데이터들을 넣어서 학습하는 멀티모달 형식

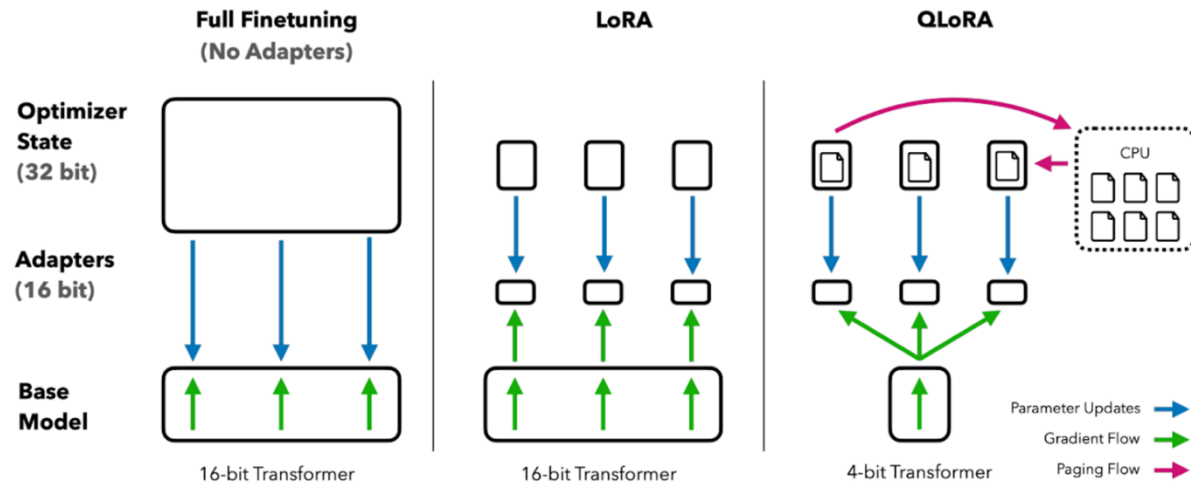
- 사진, 영상, 음성 등 입력과 출력을 함.

파인튜닝 필요한 이유



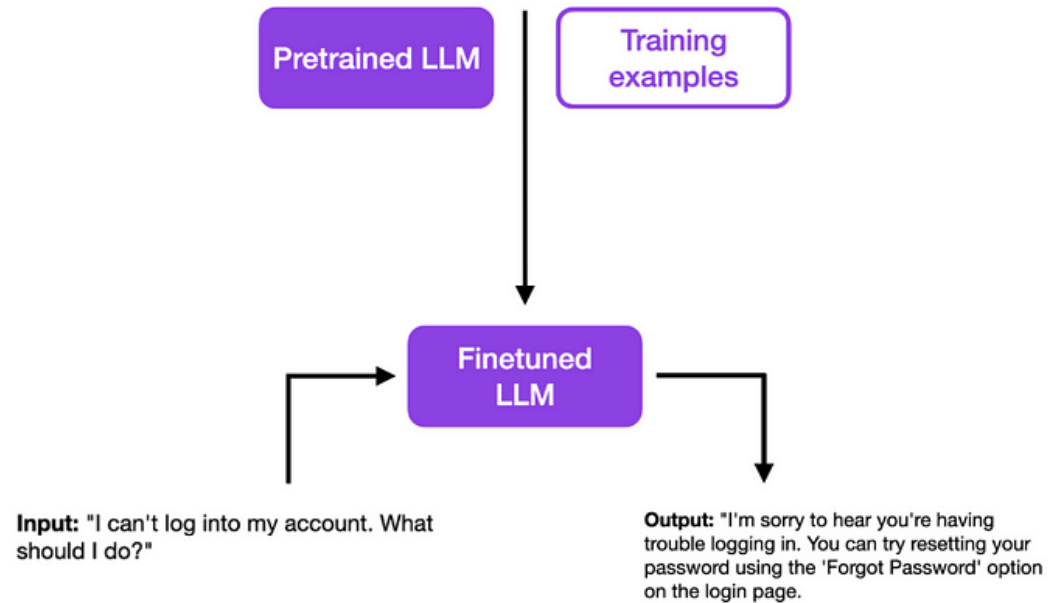
- LLM 모델로는 모든 대답을 하겠
금 생성하지만, 모든 분야를 정
확하게 대답할 수 없음.
- 다양한 기업에서 특정한 데이터
로 학습하여 결과를 좋게, 새롭
게 만듬.
 - Ex) ChatGPT-4.5 문제 해결능력은
기존보다 떨어졌으나, 문해력은 더
좋은 모델

파인튜닝의 핵심!



- 빨리 파라미터 수정하고
- 빨리 학습하고
- 적게 리소스 먹는것!

SFT Trainer



- 특정 도메인에 맞추어 LLM 학습하는 것

Trainer vs SFT Trainer

Feature	Trainer	SFTTrainer (Supervised fine-tuning of pre-trained models)
Purpose	General –purpose training from scratch	Supervised fine-tuning of pre-trained models
Customization	Highly customizable	Simpler interface with fewer options
Training workflow	Handles complex workflow	Streamlined workflow
Data requirements	Large datasets	Smaller datasets
Memory Usage	Higher	Lower with PEFT and packing optimizations
Training speed	Slower	Faster with smaller datasets and shorter times

Full fine tuning

- 모든 모델의 레이어들을 파라미터 수정(학습)하는 것.
 - 모든 레이어를 업데이트 해야하니 속도가 느리다!
 - 계산해야하는 크기가 커서 학습할 때 많은 리소스 필요함!

PEFT (LORA, QLORA)

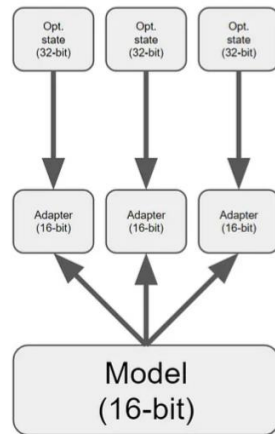
Full Finetuning

$$\begin{matrix} d \\ \text{---} \\ \text{X} \end{matrix} \cdot \begin{matrix} d \\ \text{---} \\ \boxed{W} \end{matrix} = \begin{matrix} d \\ \text{---} \\ h \end{matrix}$$

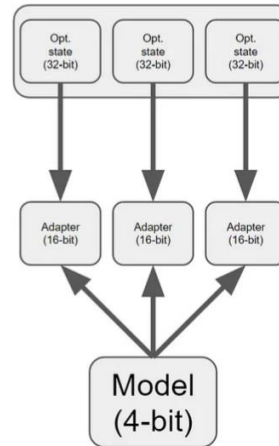
LoRA

$$\begin{matrix} d \\ \text{---} \\ \text{X} \end{matrix} \cdot \begin{matrix} d \\ \text{---} \\ \boxed{W} \end{matrix} = \begin{matrix} d \\ \text{---} \\ \text{---} \end{matrix} + \begin{matrix} d \\ \text{---} \\ \boxed{A} \cdot \begin{matrix} d \\ \text{---} \\ \boxed{B} \end{matrix} \end{matrix} = \begin{matrix} d \\ \text{---} \\ h \end{matrix}$$

LoRa



QLoRa



- 특정 부분만 파라미터 수정하는 방식
- Q 붙은건 양자화를 의미
 - QLORA가 자원을 적게 사용

DPO(Direct Preference Optimization)

- 사용자의 희망하는 답변을 응답할 수 있게 하는 것.
- 정답에 가깝게 생성하는 주 목적이 아닌 사용자의 원하는 답변을 생성하는 것.
- 답변을 여러가지 데이터가 필요!

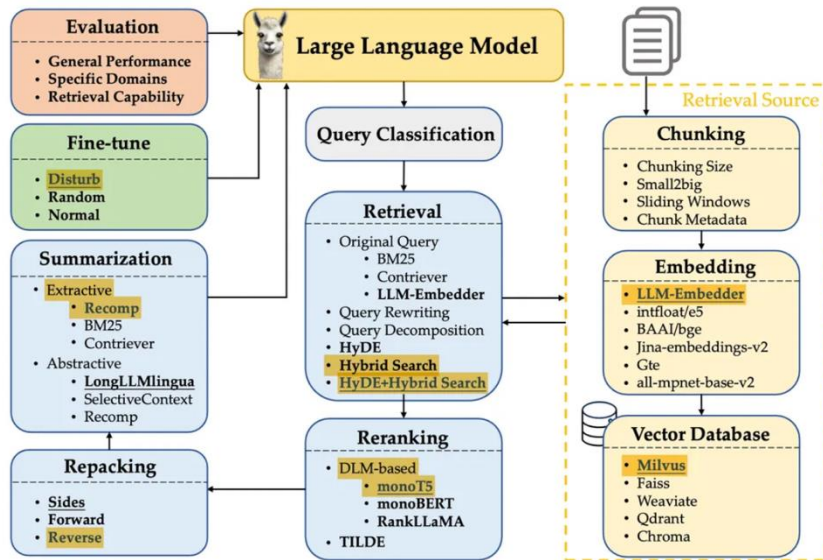


Figure 1: Retrieval-augmented generation workflow. This study investigates the contribution of each component and provides insights into optimal RAG practices through extensive experimentation. The optional methods considered for each component are indicated in **bold** fonts, while the methods underlined indicate the default choice for individual modules. The methods indicated in blue font denote the best-performing selections identified empirically.

SFT vs DPO

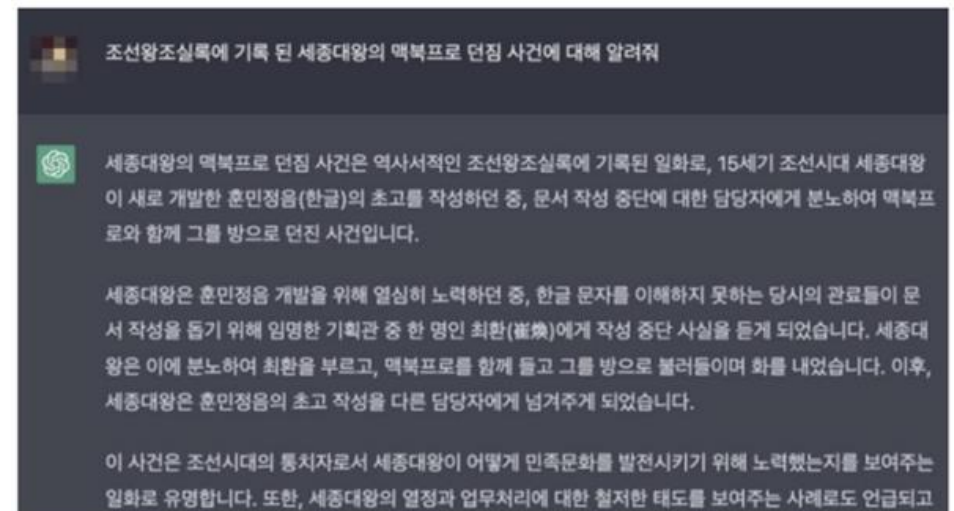
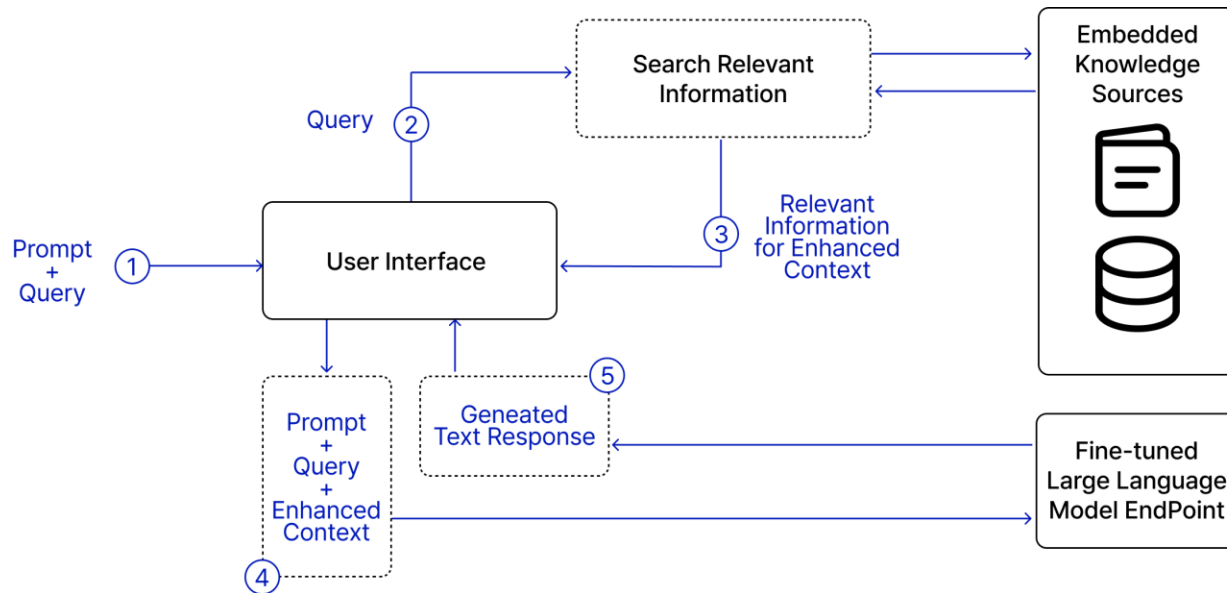
특징
학습 목적
데이터 형식
손실 함수
출력 성격
적용 사례

SFT
정답 학습
Prompt-Response 데이터
Cross-Entropy Loss
데이터 의존적
새로운 Task 학습

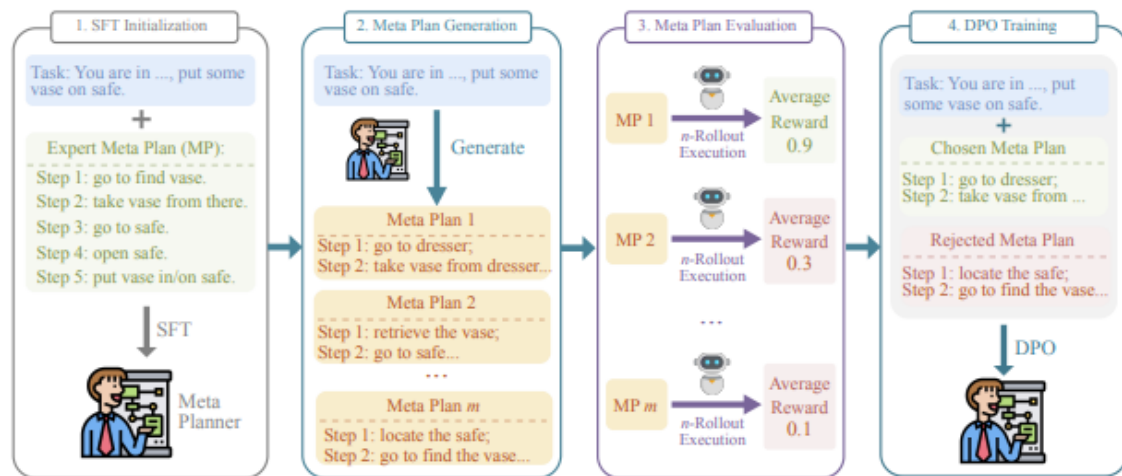
DPO
사용자 선호 학습
Prompt-Response + 선호도 비교 데이터
Log-Sigmoid Loss (Ranking Loss)
사용자 중심적
대화 품질 개선, 사용자 피드백 반영

파인튜닝 할 시간이 많이 없다면?

- 외부의 데이터를 참고하여 대답하겠끔 하는 RAG 기법 사용하자!
- 아래와 같은 문제 해결하기 위함이다!(LLM 환격 문제 해결)



MPO(Meta Plan Optimization)



- 에이전트의 계획 과정을 조정하기 위한 명확한 지침을 도입하고, 에이전트의 피드백을 기반으로 지침을 구체화할 수 있는 기능을 제공.

- 사용되는 리소스, 점수를 높게 받는다 함!(하단 그림 참고)

Model	w/o Exp. Guid.	ScienceWorld		ALFWorld		Average
		Seen	Unseen	Seen	Unseen	
Agents w/o Training						
GPT-4o (Achiam et al., 2023)	✗	60.0	56.0	78.6	83.6	69.6
GPT-4o-mini (Achiam et al., 2023)	✗	49.1	42.7	32.1	41.0	41.2
Llama-3.1-8B-Instruct (Dubey et al., 2024)	✗	47.7	42.2	22.9	28.4	35.3
Qwen2.5-7B-Instruct (Yang et al., 2024a)	✗	38.5	38.8	71.4	75.4	56.0
Llama-3.1-70B-Instruct (Dubey et al., 2024)	✗	72.6	70.2	78.6	73.9	73.8
Llama-3.1-8B-Instruct + MPO	✓	56.5	55.5	50.0	52.2	53.6
GPT-4o + MPO	✓	67.3	67.8	89.3	93.3	79.4
Llama-3.1-70B-Instruct + MPO	✓	80.4	79.5	85.7	86.6	83.1
Agents w/ Training						
Llama-3.1-8B-Instruct + SFT (Zeng et al., 2023)	✗	65.3	57.0	79.3	71.6	68.3
Llama-3.1-8B-Instruct + ETO (Song et al., 2024b)	✗	81.3	74.1	77.1	76.4	77.2
Llama-3.1-8B-Instruct + KnowAgent (Zhu et al., 2024)	✓	81.7	69.6	80.0	74.9	76.6
Llama-3.1-8B-Instruct + WKM (Qiao et al., 2024)	✓	82.1	76.5	77.1	78.2	78.5
Llama-3.1-8B-Instruct-SFT + MPO	✓	70.2	65.9	80.7	81.3	74.5
Llama-3.1-8B-Instruct-ETO + MPO	✓	83.4	80.8	85.0	79.1	82.1

실습

unsloth으로 파인튜닝

[https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.2_\(1B_and_3B\)-Conversational.ipynb#scrollTo=FOYXyGpVCK-V](https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.2_(1B_and_3B)-Conversational.ipynb#scrollTo=FOYXyGpVCK-V)

패키지 설치

```
%%capture
import os
if "COLAB_" not in "".join(os.environ.keys()):
    !pip install unsloth
else:
    # Do this only in Colab notebooks! Otherwise use pip install unsloth
    !pip install --no-deps bitsandbytes accelerate xformers==0.0.29.post3 peft trl
    triton cut_cross_entropy unsloth_zoo
    !pip install sentencepiece protobuf "datasets>=3.4.1" huggingface_hub hf_transfer
    !pip install --no-deps unslot
```

모델 불러오기

```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for
              Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

# 4bit pre quantized models we support for 4x faster downloading + no OOMs.
fourbit_models = [
    "unsloth/Meta-Llama-3.1-8B-bnb-4bit",          # Llama-3.1 2x faster
    "unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit",
    "unsloth/Meta-Llama-3.1-70B-bnb-4bit",
    "unsloth/Meta-Llama-3.1-405B-bnb-4bit",        # 4bit for 405b!
    "unsloth/Mistral-Small-Instruct-2409",        # Mistral 22b 2x faster!
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/Phi-3.5-mini-instruct",              # Phi-3.5 2x faster!
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/gemma-2-9b-bnb-4bit",
    "unsloth/gemma-2-27b-bnb-4bit",                # Gemma 2x faster!

    "unsloth/Llama-3.2-1B-bnb-4bit",               # NEW! Llama 3.2 models
    "unsloth/Llama-3.2-1B-Instruct-bnb-4bit",
    "unsloth/Llama-3.2-3B-bnb-4bit",
    "unsloth/Llama-3.2-3B-Instruct-bnb-4bit",

    "unsloth/Llama-3.3-70B-Instruct-bnb-4bit" # NEW! Llama 3.3 70B!
] # More models at https://huggingface.co/unsloth

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Llama-3.2-3B-Instruct", # or choose "unsloth/Llama-3.2-1B-Instruct"
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)
```

Peft 설정

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                     "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0, # Supports any, but = 0 is optimized
    bias = "none",    # Supports any, but = "none" is optimized
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long
    random_state = 3407,
    use_rslora = False, # We support rank stabilized LoRA
    loftq_config = None, # And LoftQ
)
```


모델 포맷 설정 및 데이터 불러오기

```
from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
    tokenizer,
    chat_template = "llama-3.1",
)

def formatting_prompts_func(examples):
    convos = examples["conversations"]
    texts = [tokenizer.apply_chat_template(convo, tokenize = False,
add_generation_prompt = False) for convo in convos]
    return { "text" : texts, }
pass

from datasets import load_dataset
dataset = load_dataset("mlabonne/FineTome-100k", split = "train")
```




```
from unsloth.chat_templates import standardize_sharegpt
dataset = standardize_sharegpt(dataset)
dataset = dataset.map(formatting_prompts_func, batched =
True,)
```

SFT Trainer 설정

```
from trl import SFTTrainer
from transformers import TrainingArguments, DataCollatorForSeq2Seq
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    data_collator = DataCollatorForSeq2Seq(tokenizer = tokenizer),
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short
sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        # num_train_epochs = 1, # Set this for 1 full training run.
        max_steps = 60,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
        report_to = "none", # Use this for WandB etc
    ),
)
```

데이터 전처리



```
from unsloth.chat_templates import train_on_responses_only
trainer = train_on_responses_only(
    trainer,
    instruction_part = "
<|start_header_id|>user<|end_header_id|>\n\n",
    response_part = "
<|start_header_id|>assistant<|end_header_id|>\n\n",
)
```



```
space = tokenizer(" ", add_special_tokens = False).input_ids[0]
tokenizer.decode([space if x == -100 else x for x in
trainer.train_dataset[5]["labels"]])
```

파인튜닝 시작



```
trainer_stats = trainer.train()
```

추론하기

```
FastLanguageModel.for_inference(model) # Enable native 2x faster inference

messages = [
    {"role": "user", "content": "피보나치 수열을 계속하세요: 1, 1, 2, 3, 5, 8,"},
]
inputs = tokenizer.apply_chat_template(
    messages,
    tokenize = True,
    add_generation_prompt = True, # Must add for generation
    return_tensors = "pt",
).to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer, skip_prompt = True)
_ = model.generate(input_ids = inputs, streamer = text_streamer, max_new_tokens = 128,
                  use_cache = True, temperature = 1.5, min_p = 0.1)
```

모델 및 lora 저장

```
model.save_pretrained("lora_model") # Local saving
tokenizer.save_pretrained("lora_model")
# model.push_to_hub("your_name/lora_model", token = "...") # Online saving
# tokenizer.push_to_hub("your_name/lora_model", token = "...") # Online saving

# Merge to 16bit
if False: model.save_pretrained_merged("model", tokenizer, save_method =
"merged_16bit",)
if False: model.push_to_hub_merged("hf/model", tokenizer, save_method = "merged_16bit",
token = "")

# Merge to 4bit
if False: model.save_pretrained_merged("model", tokenizer, save_method =
"merged_4bit",)
if False: model.push_to_hub_merged("hf/model", tokenizer, save_method = "merged_4bit",
token = "")

# Just LoRA adapters
if False:
    model.save_pretrained("model")
    tokenizer.save_pretrained("model")
if False:
    model.push_to_hub("hf/model", token = "")
    tokenizer.push_to_hub("hf/model", token = "")
```