

# LG CNS One-day LLM-FT

모델 성능 향상을 위한 Fine-Tuning

강사 김형욱  
[\(hukim@artiasolution.com\)](mailto:hukim@artiasolution.com)

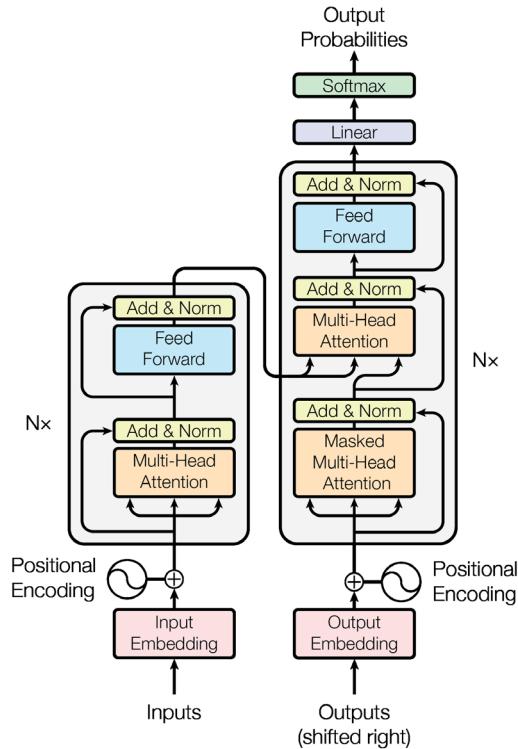
# Contents

- LLM fine -tuning (3 page)
- Instruction tuning (35 page)
- PEFT(49 page)

# **LLM fine-tuning**

# Revisit LLMs

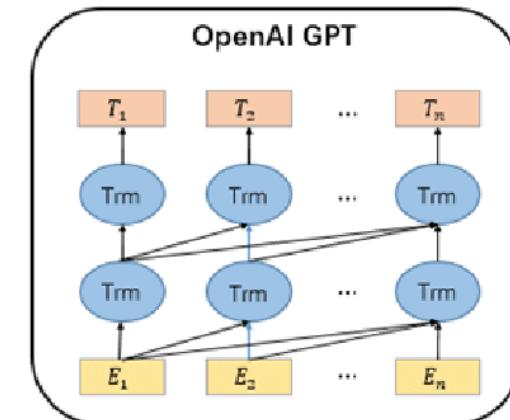
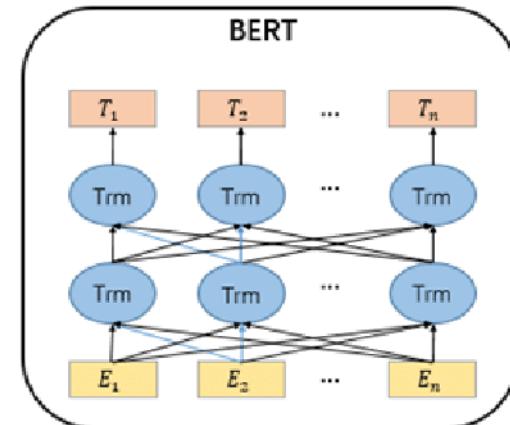
- LLM(Large Language Model)은 대량의 텍스트 데이터를 기반으로 학습되어 자연어 이해와 생성에 관한 복잡한 작업을 수행
- 대부분의 LLM은 Transformer 기반 아키텍처를 채용
  - GPT 계열 : 문장 생성 특화. 자기회귀(auto-regressive) 모델
  - BERT 계열 : 문장 맥락 이해 특화. 자기인코딩(auto-encoding) 모델



*Transformer architecture  
: encoders and decoders*

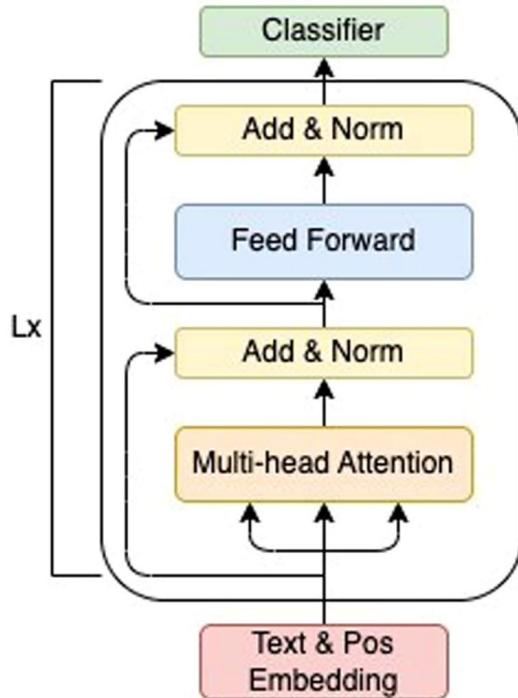
# Bert & GPT architectures

- BERT (Bidirectional Encoder Representations from Transformers), Google AI, 2018년
  - **BERT는 Transformer 아키텍처의 Encoder 부분으로 언어이해에 중점**
  - 양방향성: BERT는 문장의 앞과 뒤를 모두 고려하여 단어의 맥락을 이해하여 더 정확한 단어의 의미를 파악
- GPT (Generative Pre-training Transformer), OpenAI, 2018년
  - **GPT는 Transformer 아키텍처의 Decoder 부분으로 문장생성에 중점**
  - 단방향성: 초기 GPT 모델은 주로 문장의 이전 단어들만을 고려하여 다음 단어를 예측. GPT-3와 같은 최신 버전에서는 더 발전된 기법을 사용하지만, 기본적으로 생성적 예측에 초점



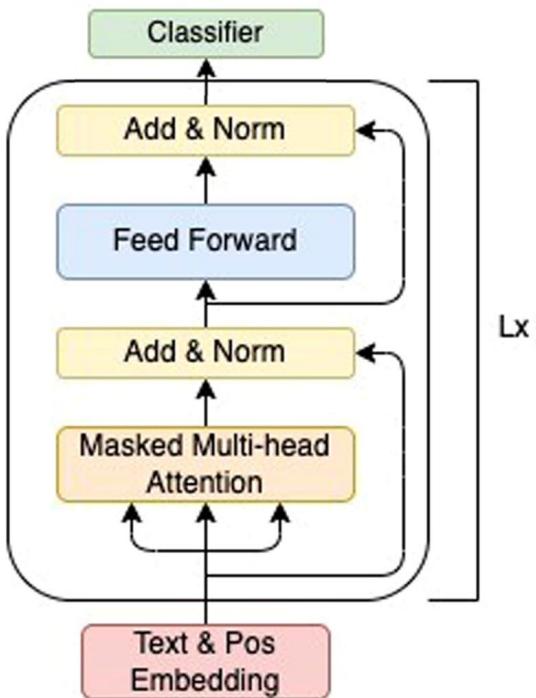
## BERT

Sentence classification  
Or token classification



## GPT

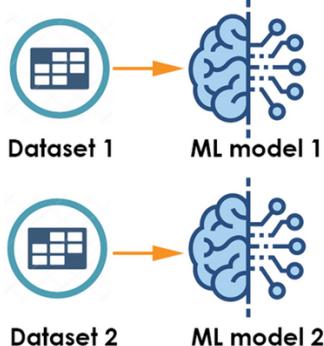
Next token prediction



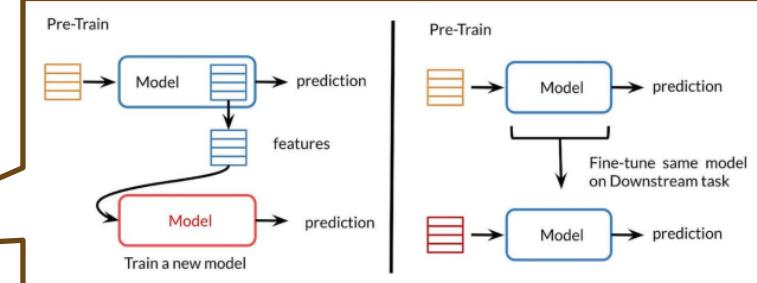
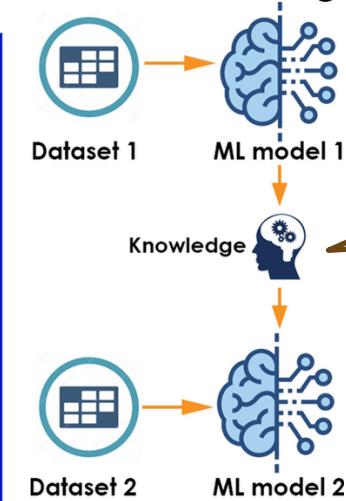
# Scratch(초기값 학습) vs Transfer Learning(전이 학습)

- 전이학습은 사전 학습된 모델을 활용하여 적은 데이터와 리소스로도 높은 성능 발휘
- 학습 시간 절약과 다양한 문제에 대한 적용성을 제공하는 딥러닝의 핵심 기술

Training from scratch

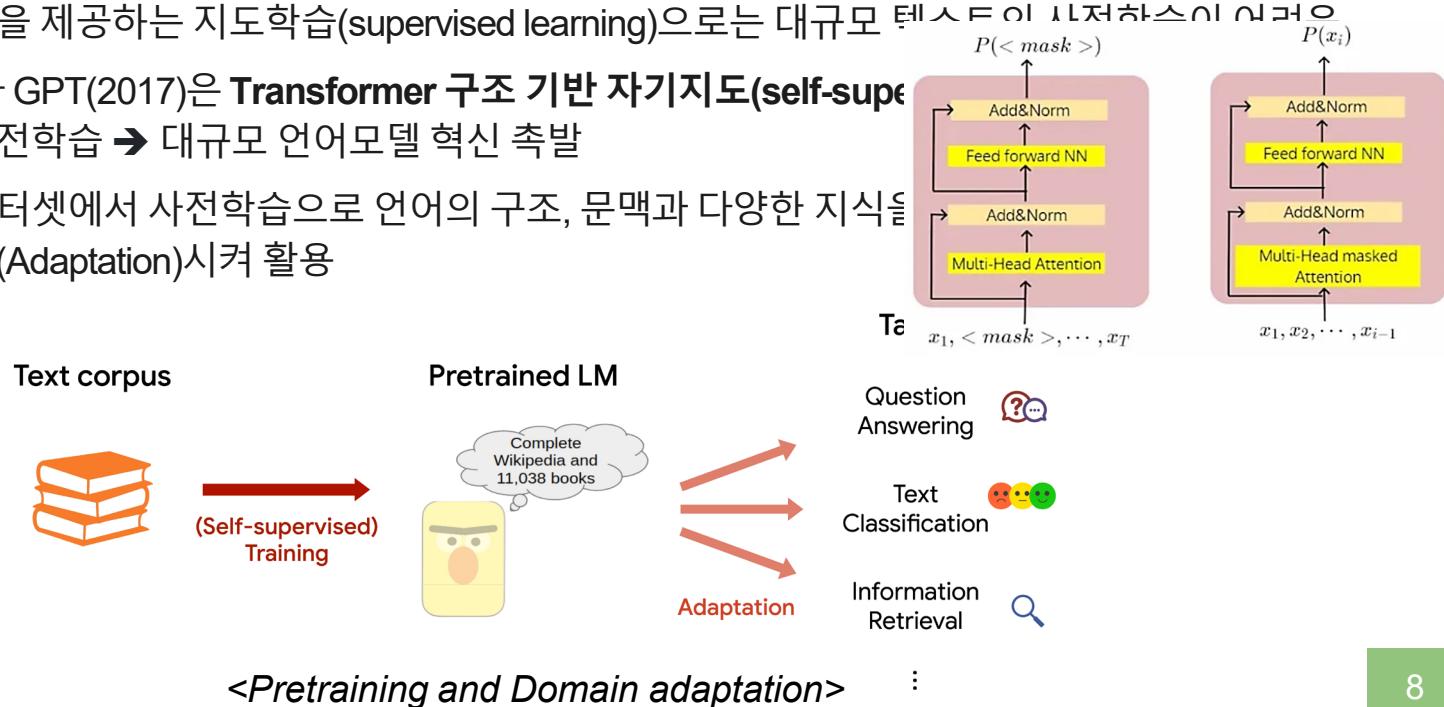


Transfer Learning



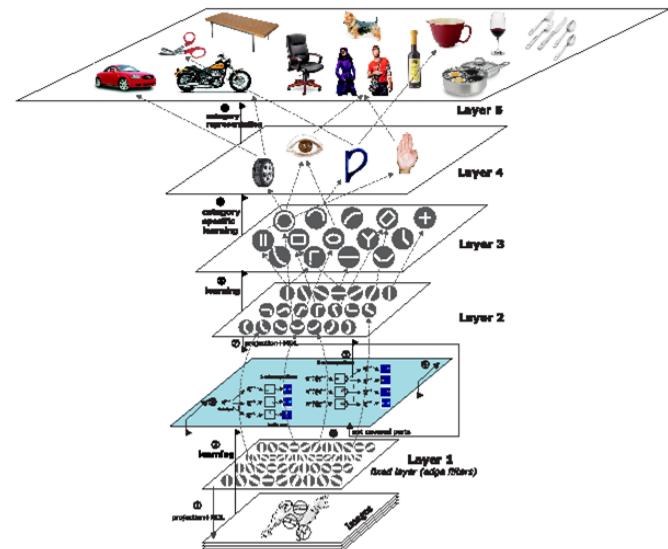
# Pretrained Language Modeling

- PLM(Pretrained Language Modeling)은 사전 학습된 언어 모델을 의미
  - 사람이 정답을 제공하는 지도학습(supervised learning)으로는 대규모 텍스트와 시너지 학습이 가능
  - Bert(2017)와 GPT(2017)은 **Transformer 구조 기반 자기지도(self-supervised)** 텍스트를 사전학습 → 대규모 언어모델 혁신 촉발
  - 대규모 데이터셋에서 사전학습으로 언어의 구조, 문맥과 다양한 지식을 모델을 적응(Adaptation)시켜 활용



# BERT, GPT and PLM

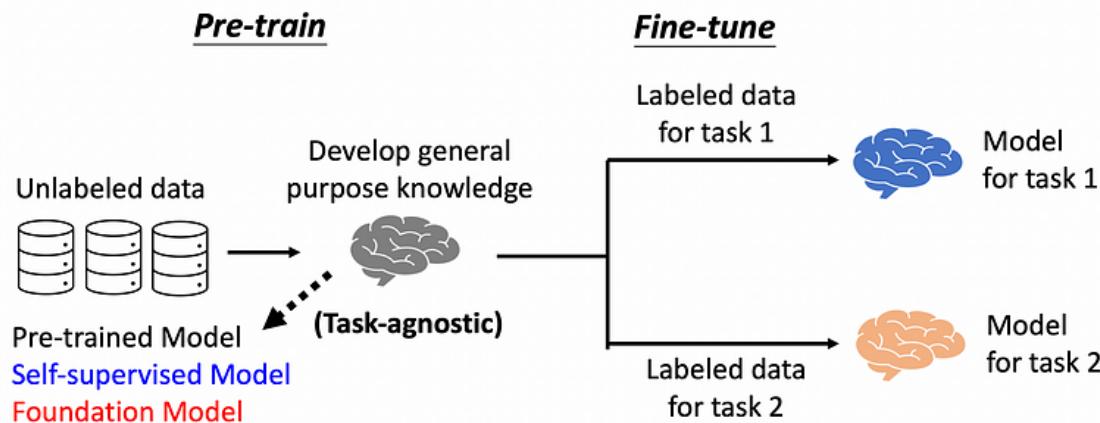
- 비전모델의 전이학습이 이미지의 여러 층위의 특징을 학습하는 것과 유사하게 텍스트의 전이학습도 언어의 여러 층위의 특징을 전달
  - 이미지 특징 vs 텍스트 특징
    - ✓ Edge, blob, color vs Lexical(어휘적)
    - ✓ Object, texture vs Syntactic(구문적), Semantic(의미적)
    - ✓ Scene understanding vs Contextual understanding
- 지도학습을 위한 학습용 corpus 구성은 매우 비효율적
  - 방대한 unlabeled corpus를 활용한 방법을 구상
    - ✓ SSL(Self-supervised learning)으로 사전학습 모델 개발
    - ✓ FT(Fine-tuning)으로 사전학습 모델을 task-specific 모델로 개발



이미지 특징계층의 시각화

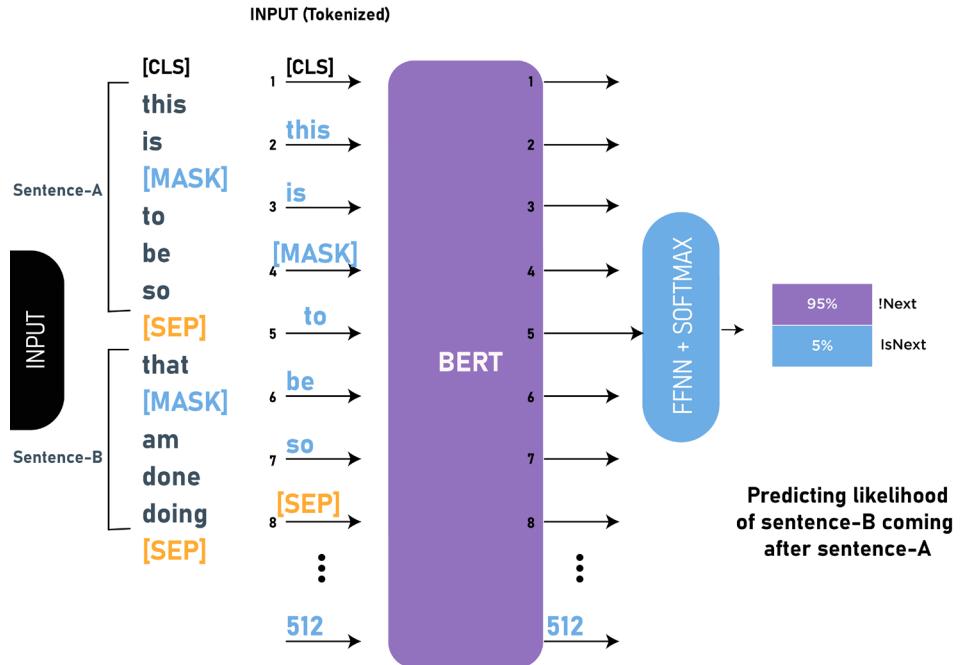
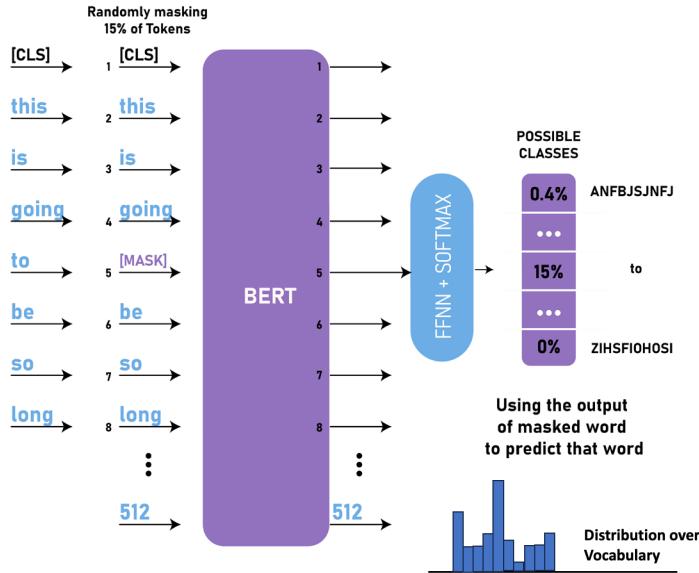
# Pre-training and Fine-tuning

- Pre-training은 모델을 특정 task와 무관하지만 대규모 데이터에 대해 학습시키는 작업
- Fine-tuning은 Pretrained Model을 특정 작업에 최적화하기 위한 과정
  - 사전학습된 모델의 활용은 계산 비용을 절감, 작은 데이터셋에서의 효과성, 학습 시간 단축 등 쉽게 성능을 향상시키는 강력한 기법



<Pre-training to Fine-tuning>

# Pre-training on Berts with MLM & NSP

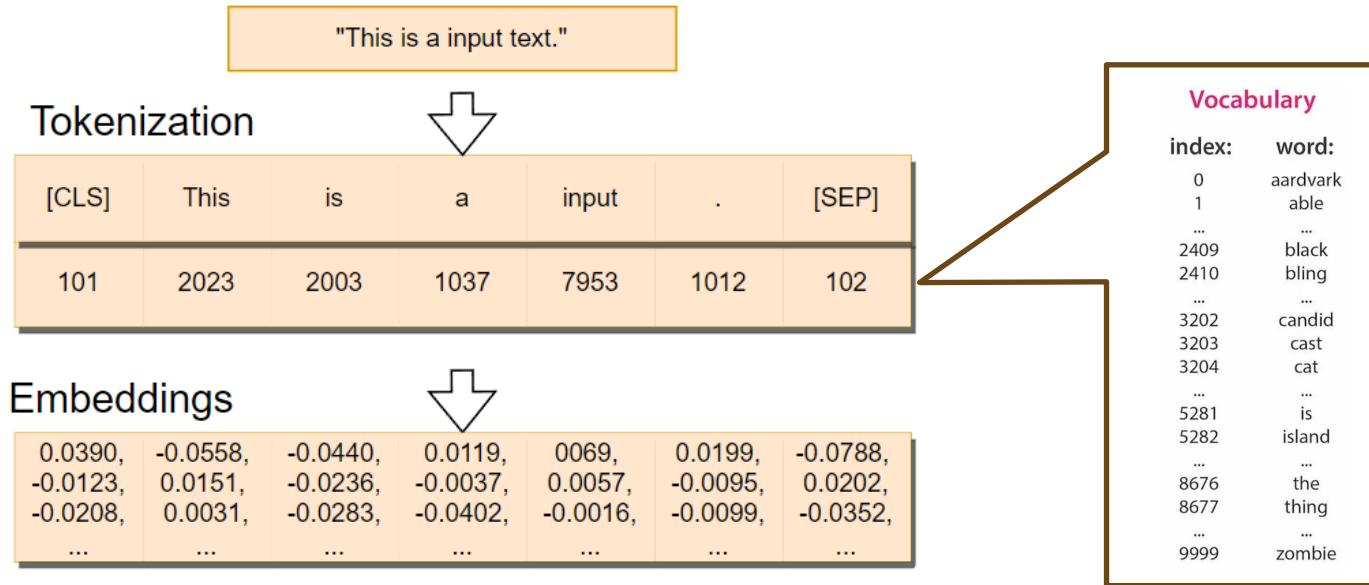


Masked language modeling

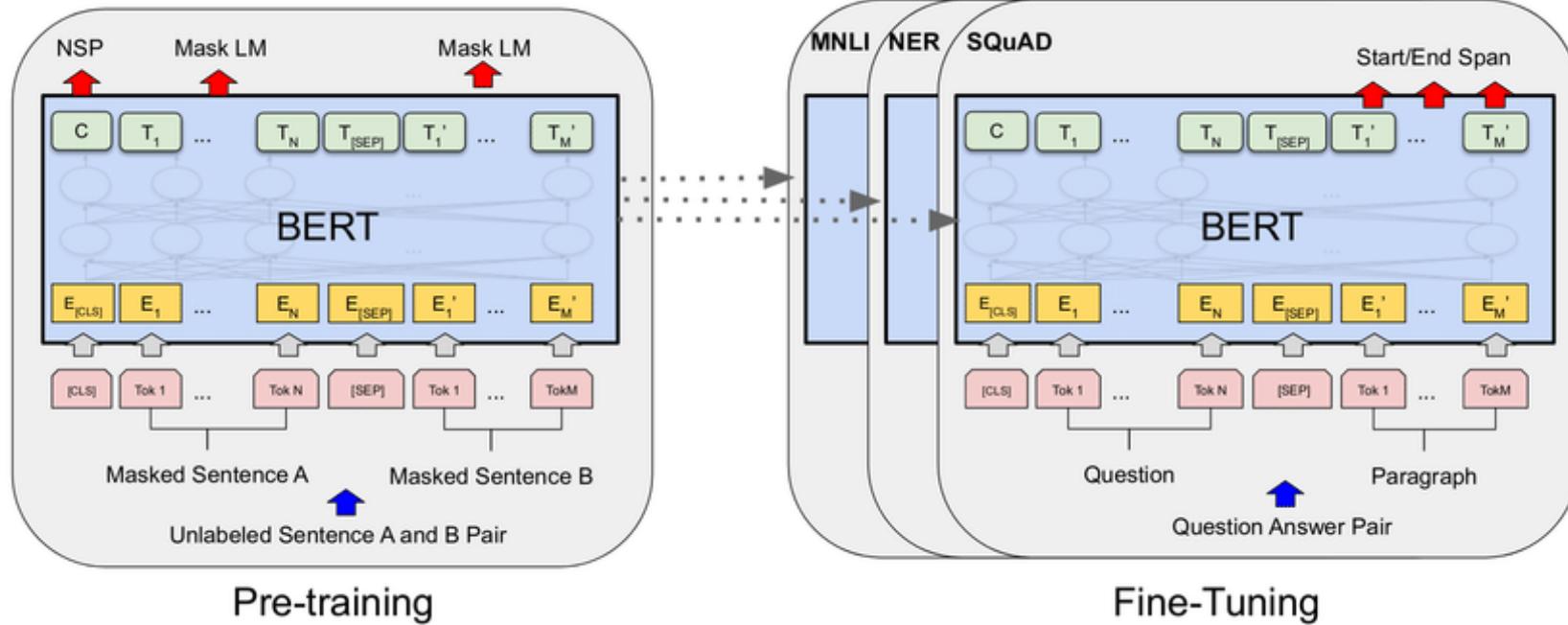
Masked language modeling + Next sentence prediction

# Tokenization & Embedding

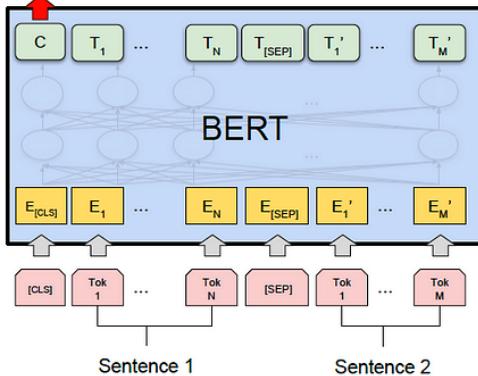
- NLP에서 모델에 텍스트 데이터를 입력하기 위해 텍스트를 수치형으로 변환하는 과정
  - Tokenization: 텍스트를 의미 있는 단위(토큰)로 나누고, 이를 고유한 인덱스 번호(vocabulary 참조)로 변환
  - Embedding: 각 토큰 인덱스를 고차원의 벡터로 변환하여 모델이 처리할 수 있는 형태로 변환



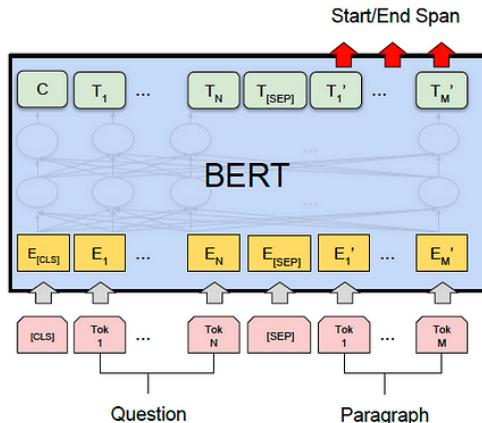
# Fine-tuning on Berts for specific tasks



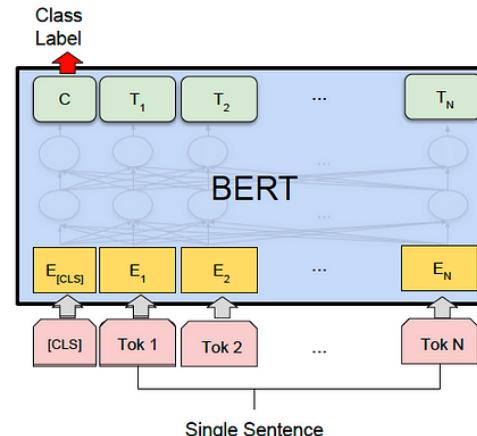
(Image Source: [Devlin, et. al., 2018](#))



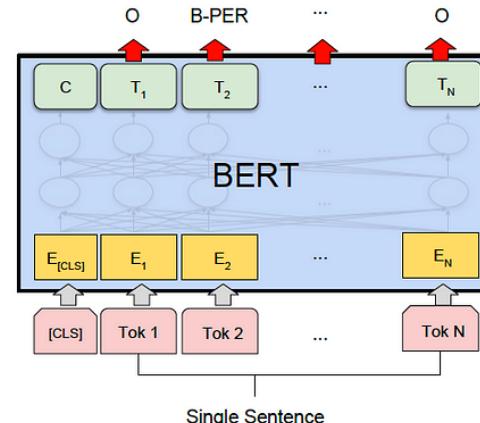
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(c) Question Answering Tasks:  
SQuAD v1.1



(b) Single Sentence Classification Tasks:  
SST-2, CoLA

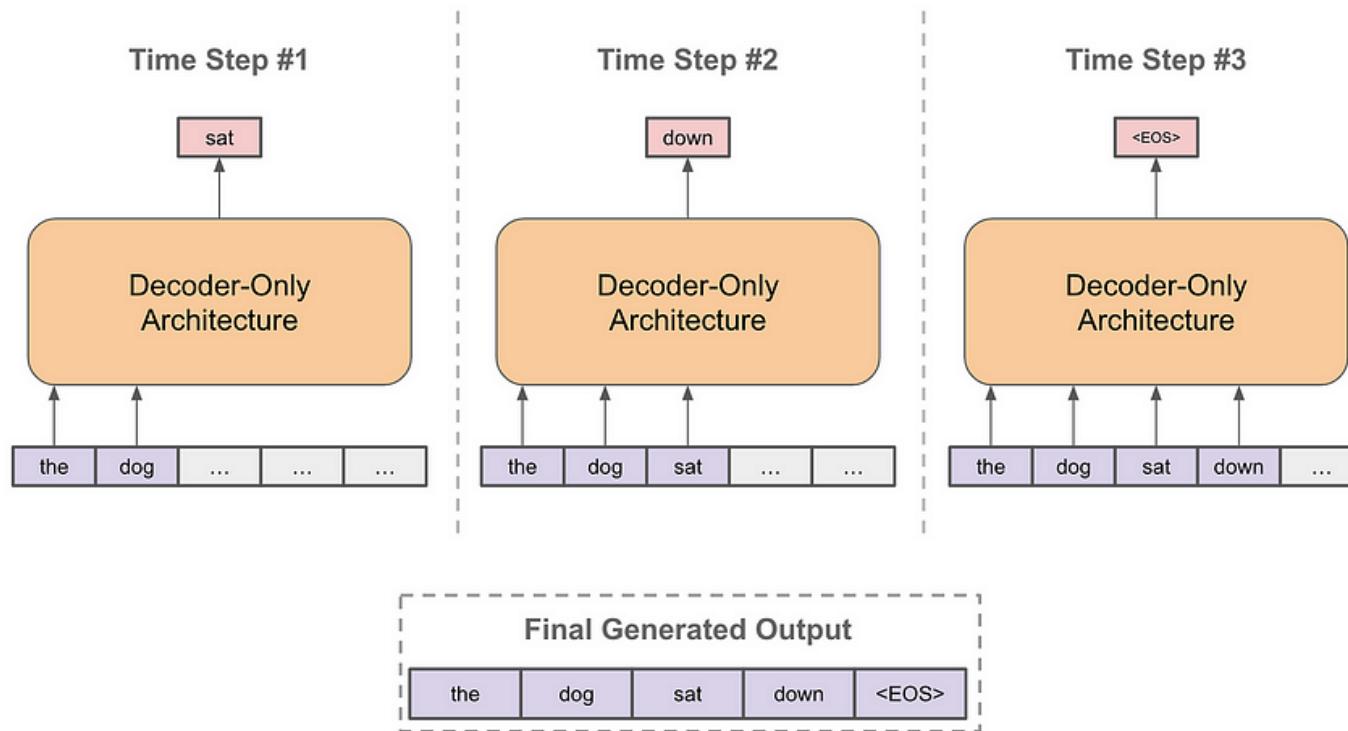


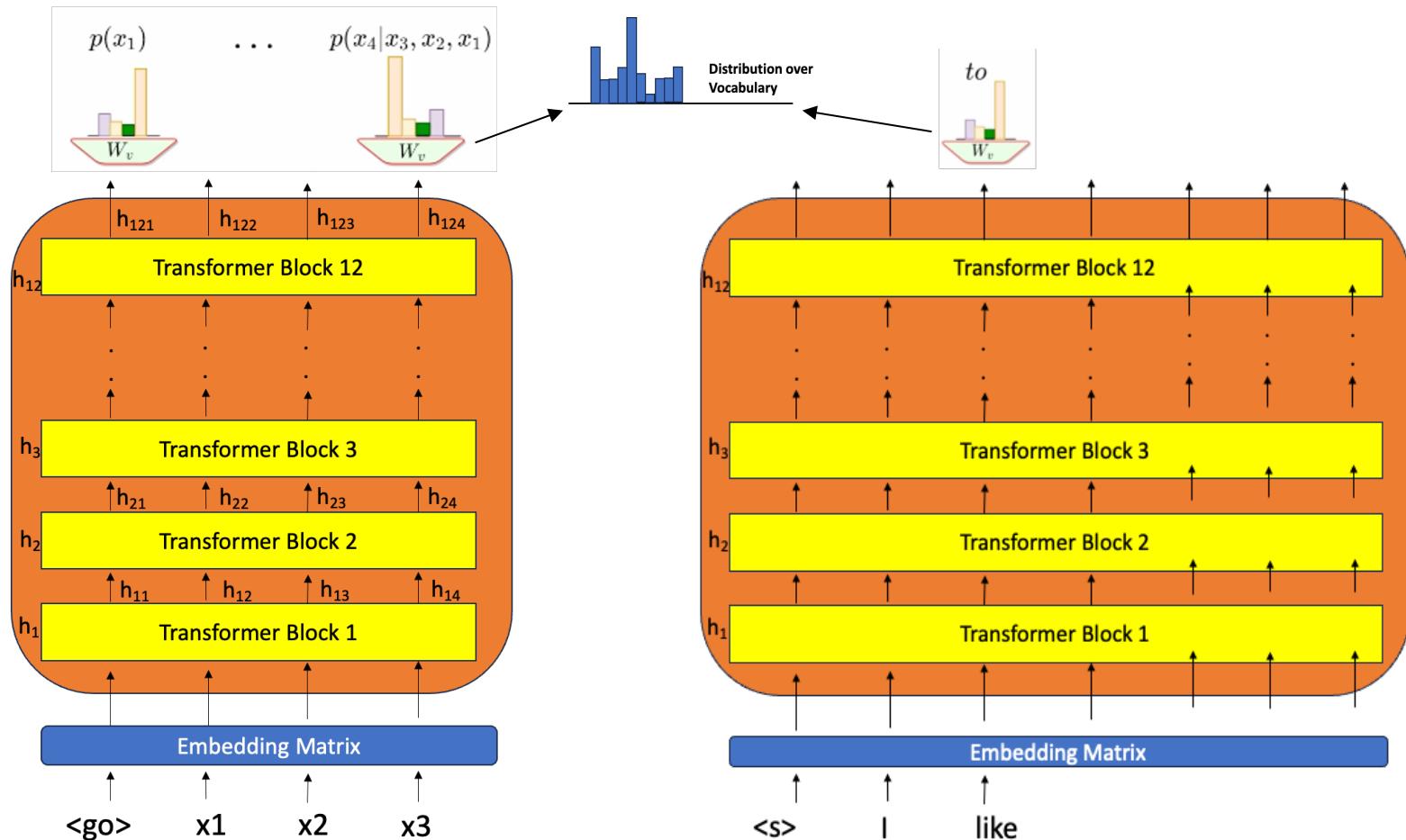
(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

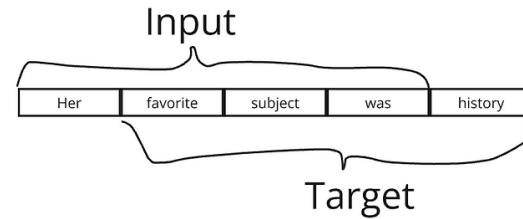
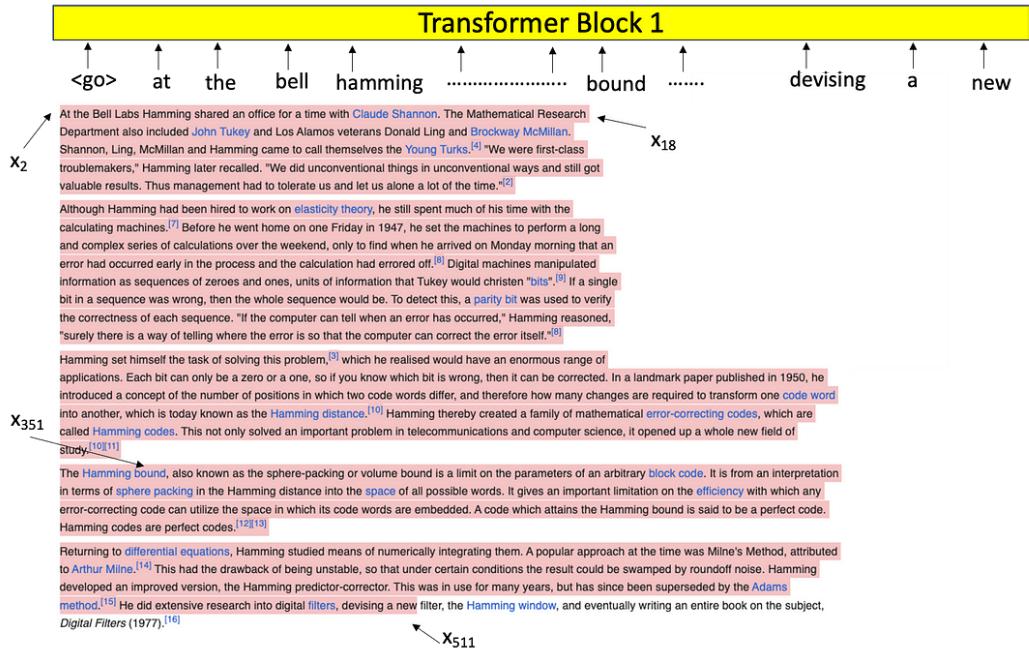
사전 학습 후 각각의 개별 다운스트림  
작업에 대해 좌측처럼 Fine-tuning.

- (a) 문장 쌍 분류 작업 (예: 두 문장 간의 의미적 유사성)
- (b) 단일 문장 분류 작업 (예: 감정 분석)
- (c) Question Answering (질문-응답)
- (d) Named entity recognition(개체명 인식 태깅)

# Pretraining on GPTs with NTP







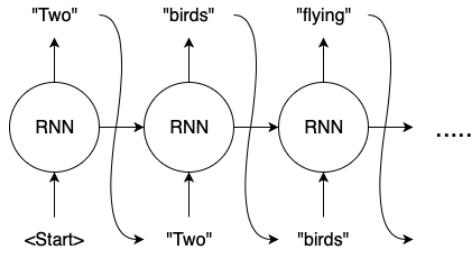
## Training Examples

Input	Expected Output
[Her]	[favorite]
[Her, favorite]	[subject]
[Her, favorite, subject]	[was]
[Her, favorite, subject, was]	[history]

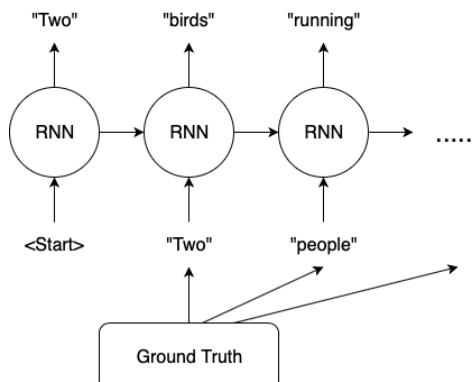
GPT는 문서를 입력으로 받아 각 time-step에서 next token prediction 학습

GPT의 학습 방식은 사람이 따로 label을 해줄 필요 없이, input을 shift 해주는 것만으로 label을 만들 수 있음.

# Applying teacher forcing when training GPTs

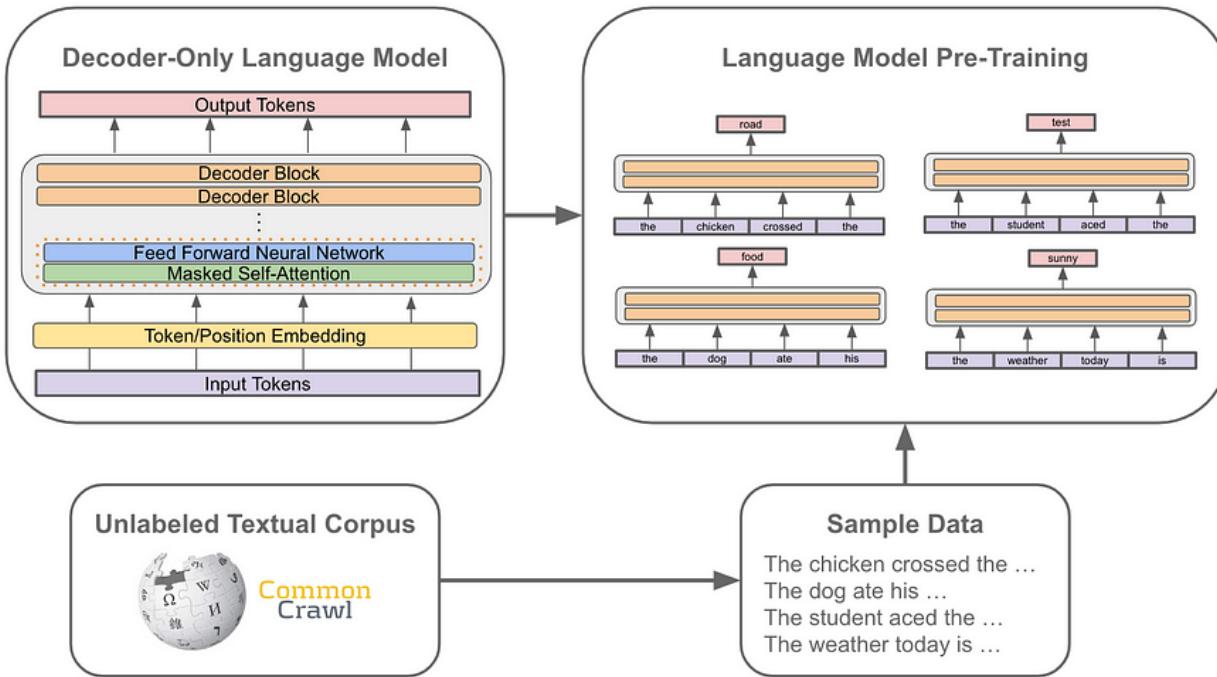


Without Teacher Forcing



With Teacher Forcing

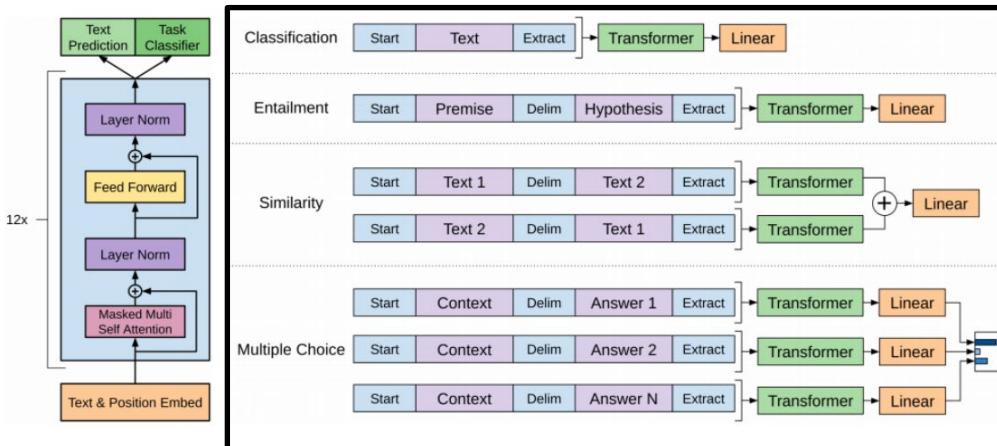
- 학습 시 순수하게 auto-regressive training 대신 Teacher Forcing 방식을 사용
  - Google의 Seq2Seq 연구에서 학습의 효과성 확인
- 초기 모델의 예측은 부정확하므로, 학습 시 모델은 자신의 예측 단어가 아닌 실제 정답 단어를 다음 입력으로 사용하면서 학습함.
  - 초기 학습 단계에서 모델이 올바른 출력을 학습하도록 도움.



*Autoregressive language modeling(or Causal language modeling)*은 레이블이 없는 방대한 코퍼스 데이터셋에서 잘 작동함. 이는 데이터 부족 문제를 크게 완화시킴

# Fine-tuning on GPTs for specific tasks

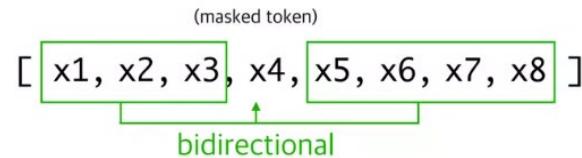
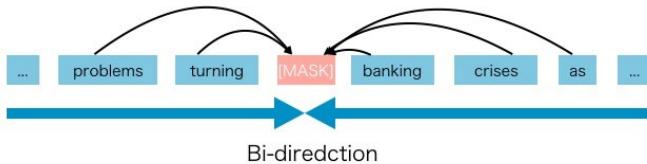
- GPT는 일반적으로 문장 생성에 특화. Pretraining과 마찬가지로 text completion 문제를 학습하는 경우가 많음. 이 경우 목표 데이터를 사용하되 학습 방식은 사전 학습과 동일.
  - GPT도 Bert와 같이 FT를 통해 문장 분류, 문장 유사도 추론, 다중선택 등의 문제에 적용은 가능



Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	<u>89.3</u>	-	-	-
CAFE [58] (5x)	80.2	79.0	<u>89.3</u>	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	<u>83.3</u>	-	-
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	<b>61.7</b>
Finetuned Transformer LM (ours)	<b>82.1</b>	<b>81.4</b>	<b>89.9</b>	<b>88.3</b>	<b>88.1</b>	56.0
Method	Story Cloze	RACE-m	RACE-h	RACE		
val-LS-skip [55]	76.5	-	-	-		
Hidden Coherence Model [7]	<u>77.6</u>	-	-	-		
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2		
BiAttention MRU [59] (9x)	-	60.2	50.3	53.3		
Finetuned Transformer LM (ours)	<b>86.5</b>	<b>62.9</b>	<b>57.4</b>	<b>59.0</b>		

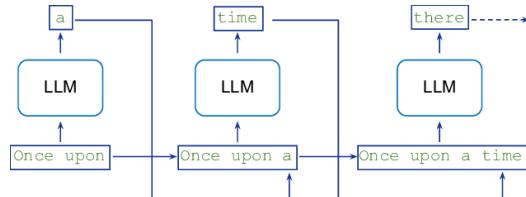
# Bert & GPT 학습 목표의 수학적 표현

## 1. Auto-encoding language model (feat. Bert)



$$\max_{\theta} \log p_{\theta}(\mathbf{x}_m | \mathbf{x}_{\bar{m}}) \approx \sum_{t=1}^T m_t \log p_{\theta}(x_t | \mathbf{x}_{\bar{m}}) = \sum_{t=1}^T m_t \log \frac{\exp(y_{\mathbf{x}_{\bar{m}}}(x_t))}{\sum_{x'} \exp(y_{\mathbf{x}_{\bar{m}}}(x'))}$$

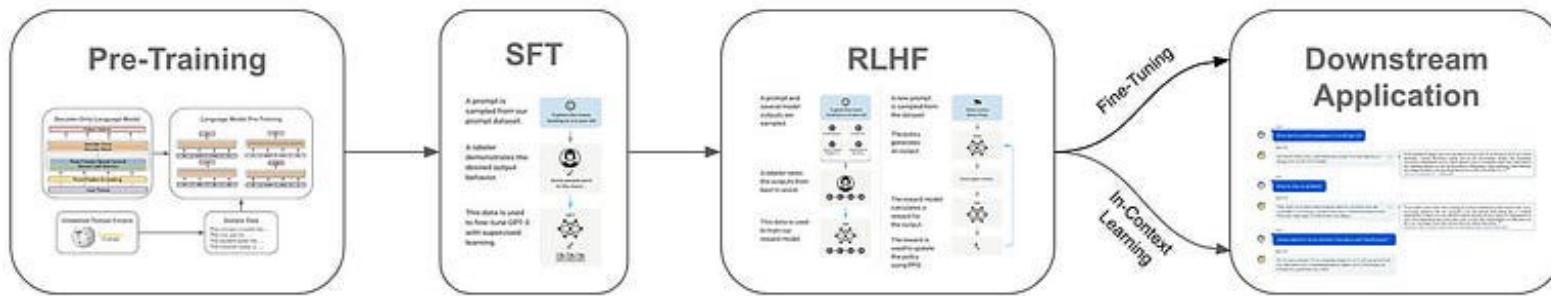
## 2. Auto-regressive language model (feat. GPT)



$$\max_{\theta} \log p_{\theta}(x_1, \dots, x_T) \approx \sum_{t=1}^T \log p_{\theta}(x_t | x_1, \dots, x_{t-1}) = \sum_{t=1}^T \log \frac{\exp(y_{x_1, \dots, x_{t-1}}(x_t))}{\sum_{x'} \exp(y_{x_1, \dots, x_{t-1}}(x'))}$$

# LLM training pipeline

- 현대적 언어 모델은 여러 단계를 거쳐 훈련



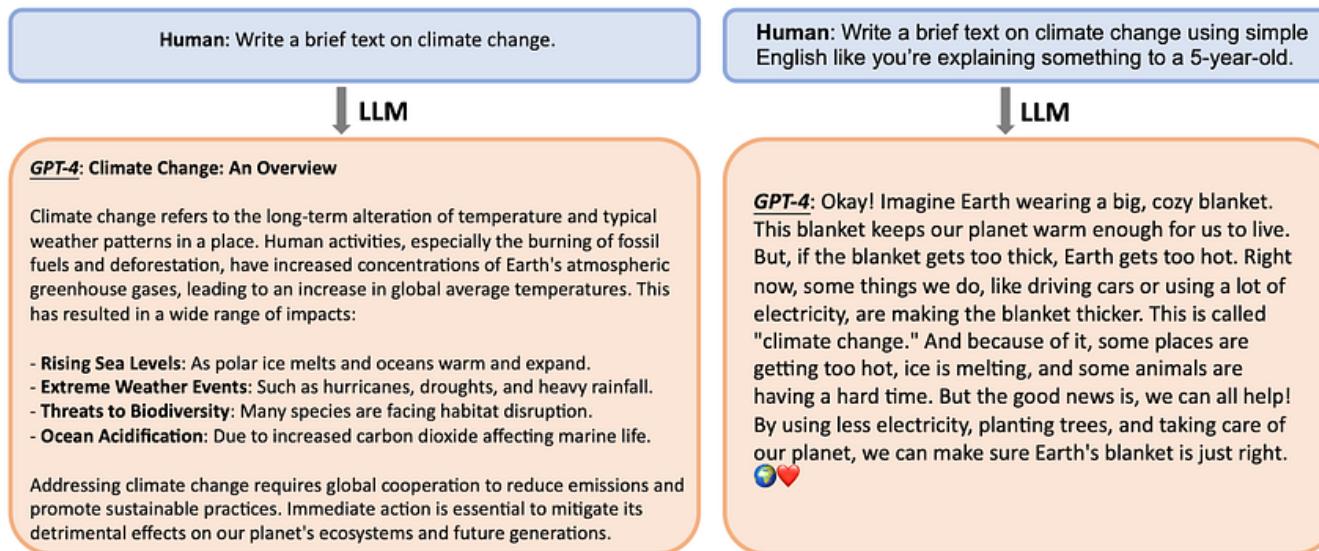
*Multiple process for training generative LLMs*

# Approaches to LLM adaptation

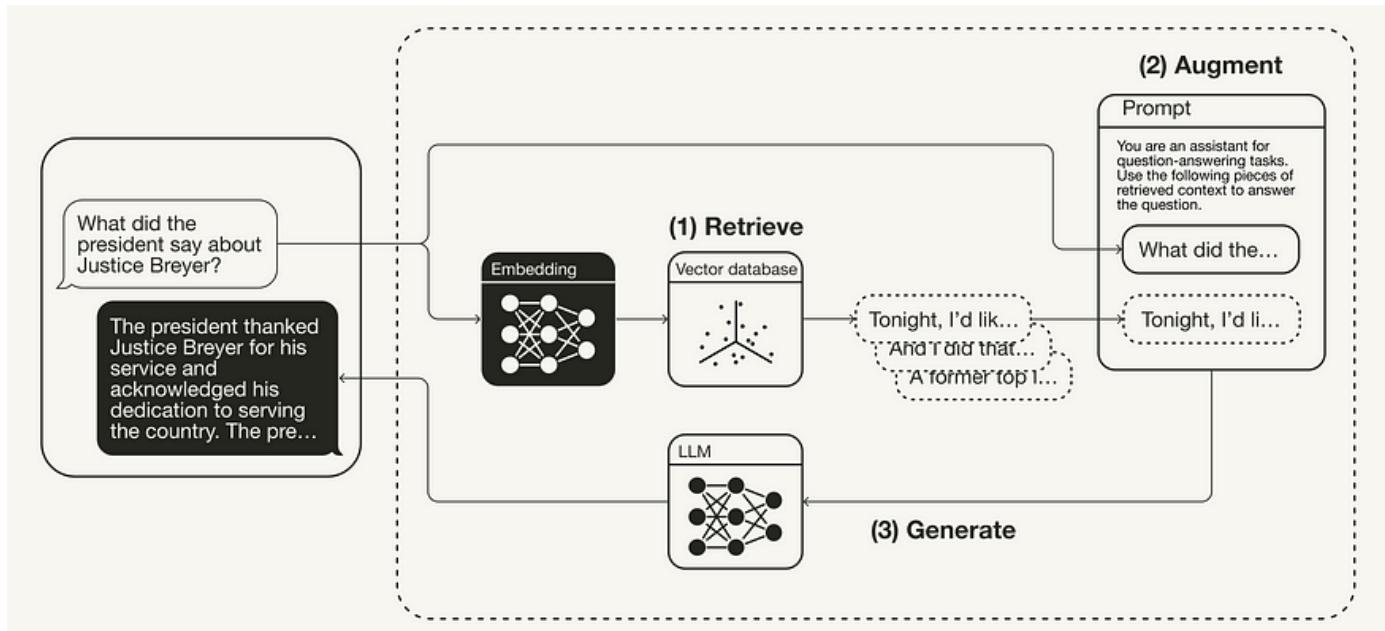
- 1. 사전 학습(Pre-training)
  - 사전 학습은 LLM을 처음부터 훈련시키는 과정으로, 수조 개의 데이터 토큰을 사용하여 기초 모델(foundation model) 구축. 경우에 따라 기초 모델을 새로운 도메인 데이터로 추가 훈련하는 Continued Pre-training도 가능
- 2. Prompt engineering
  - 문맥 내 학습(ICL, In-Context Learning) : 입력에 예시를 제공하여 적응시키는 방법
  - 검색 증강 생성(RAG, Retrieval-Augmented Generation) : 입력에 데이터베이스의 검색 결과를 추가
- 3. 지도학습적 세부조정(Supervised Fine-tuning)
  - 세부 조정은 사전 학습된 모델을 감독 학습(supervised learning) 또는 강화 학습 기반의 기법을 통해 적응시키는 과정

# 1. Prompt Engineering

- Prompt engineering은 LLM이 정확, 간결, 창의적인 텍스트 생성 작업을 수행하도록 지시하는 실천적 가이드라인
  - 엄밀한 공학 분야나 수학적 기초에 근거한 것은 아님.



## 2. RAG(Retrieval-Augmented Generation)



- RAG(Retrieval-Augmented Generation)는 검색 기반의 정보 증강 + LLM
  - 검색과 생성의 결합: RAG는 주어진 질문이나 프롬프트에 대응하는 정보를 데이터셋에서 검색
  - 도메인 지식 활용: 검색을 통해 모델은 학습 데이터에 없는 최신 지식 또는 전문적인 지식을 활용해 답변

### 3. SFT(Supervised Fine-tuning)

- 사전 학습 모델은 특정 문제와 무관한 문제를 방대하게 학습 : *task-agnostic learning*
  - 자연어 데이터에 대한 사람의 지도학습 없이 언어의 기본적인 구조와 문맥을 이해
  - But, 인간의 지시사항 이해하고 특정 작업을 잘 수행하기 위한 방법의 학습은 부족
- 추가적인 지도학습을 통해 이런 한계를 극복
  - 사전 학습된 모델을 특정 작업에 최적화하거나, 지시사항을 이해할 수 있도록 조정
  - 지도학습 데이터로 사용 사례에 적합하게 튜닝
    - ✓ Full Fine-Tuning : 모델의 모든 파라미터를 훈련하는 과정
    - ✓ PEFT : 모든 파라미터를 직접 훈련시키는 대신, 특정 어댑터만을 훈련

# Reinforcement Learning from Human Feedback

- 사전 학습된 LLM은 SFT 적용 이후, RLHF로 답변 품질을 개선할 수 있음.
  - 선호되는 답변 방향으로 강화학습을 적용
    - ✓ 편향된 답변, 선정적이거나 반사회적 답변 회피, 사용자들에게 선호되는 스타일을 학습
- 인간의 피드백을 모델의 강화 학습에 활용
  - DPO (Direct Preference Optimization) : 사용자의 선호를 직접 최적화하는 방법
  - PPO (Proximal Policy Optimization) : 강화 학습으로 에이전트를 학습해 최적화하는 방법



# To fine-tune or not to fine-tune

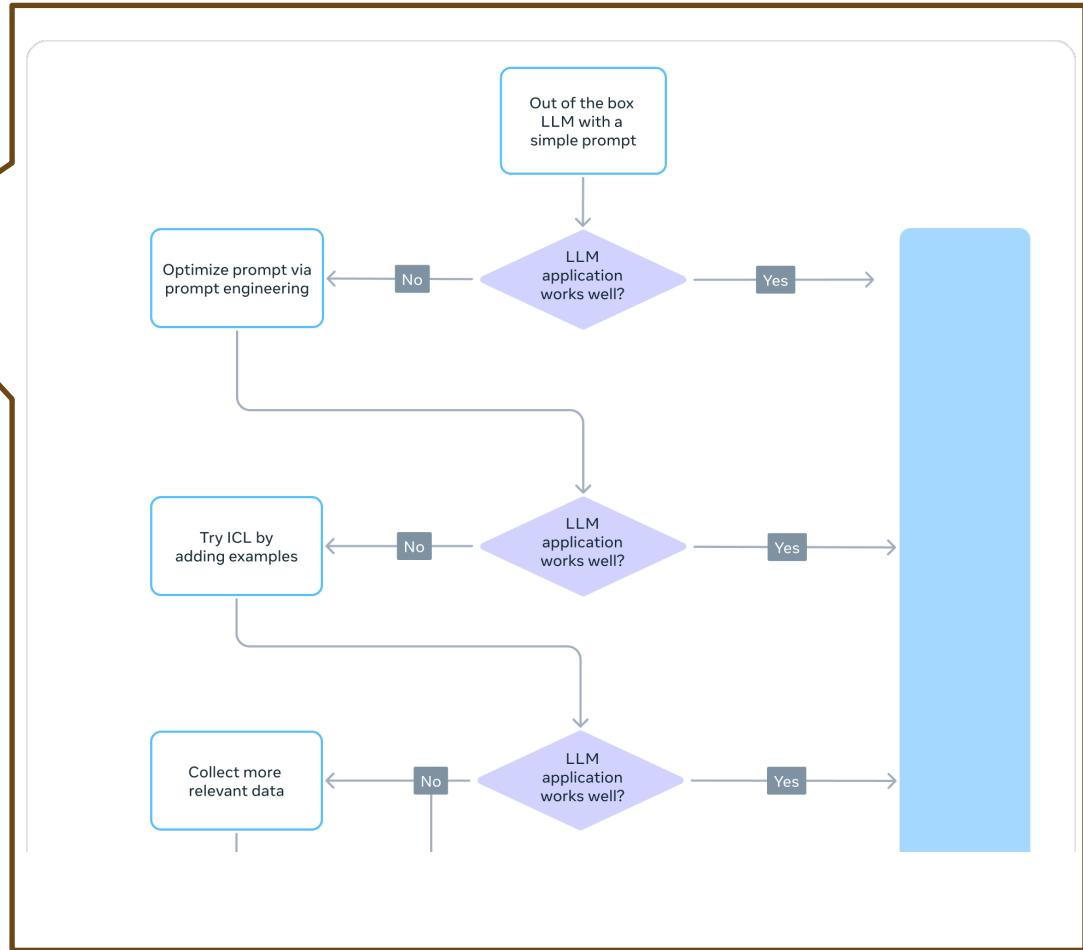
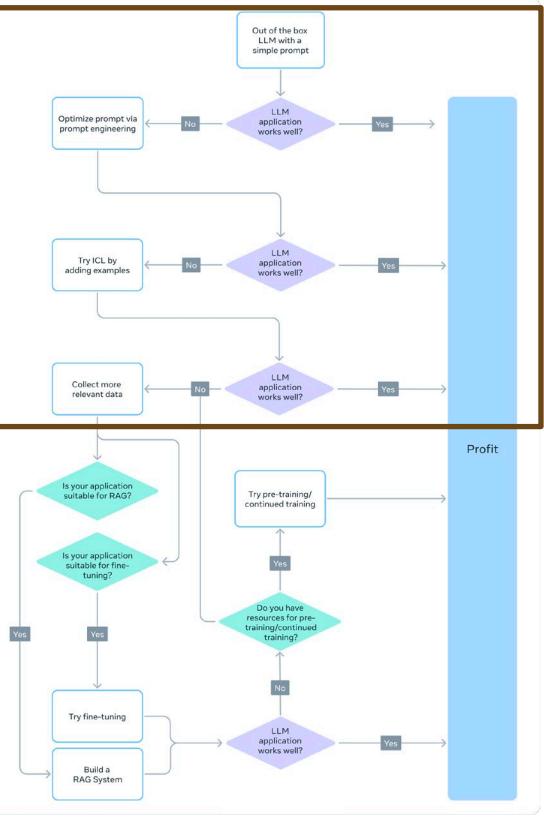
- LLM 이전 FT는 상대적으로 작은 모델(100M–300M 파라미터)에서 자주 사용
  - 감독 학습(supervised fine-tuning)을 통해 미리 훈련된 모델을 특정 도메인에 맞춰 추가 훈련하는 방식이 주로 사용
- 10억 개 이상의 파라미터를 갖는 LLM의 fine-tuning은 더 복잡해졌고, 많은 컴퓨팅 자원과 상업적인 하드웨어를 요구

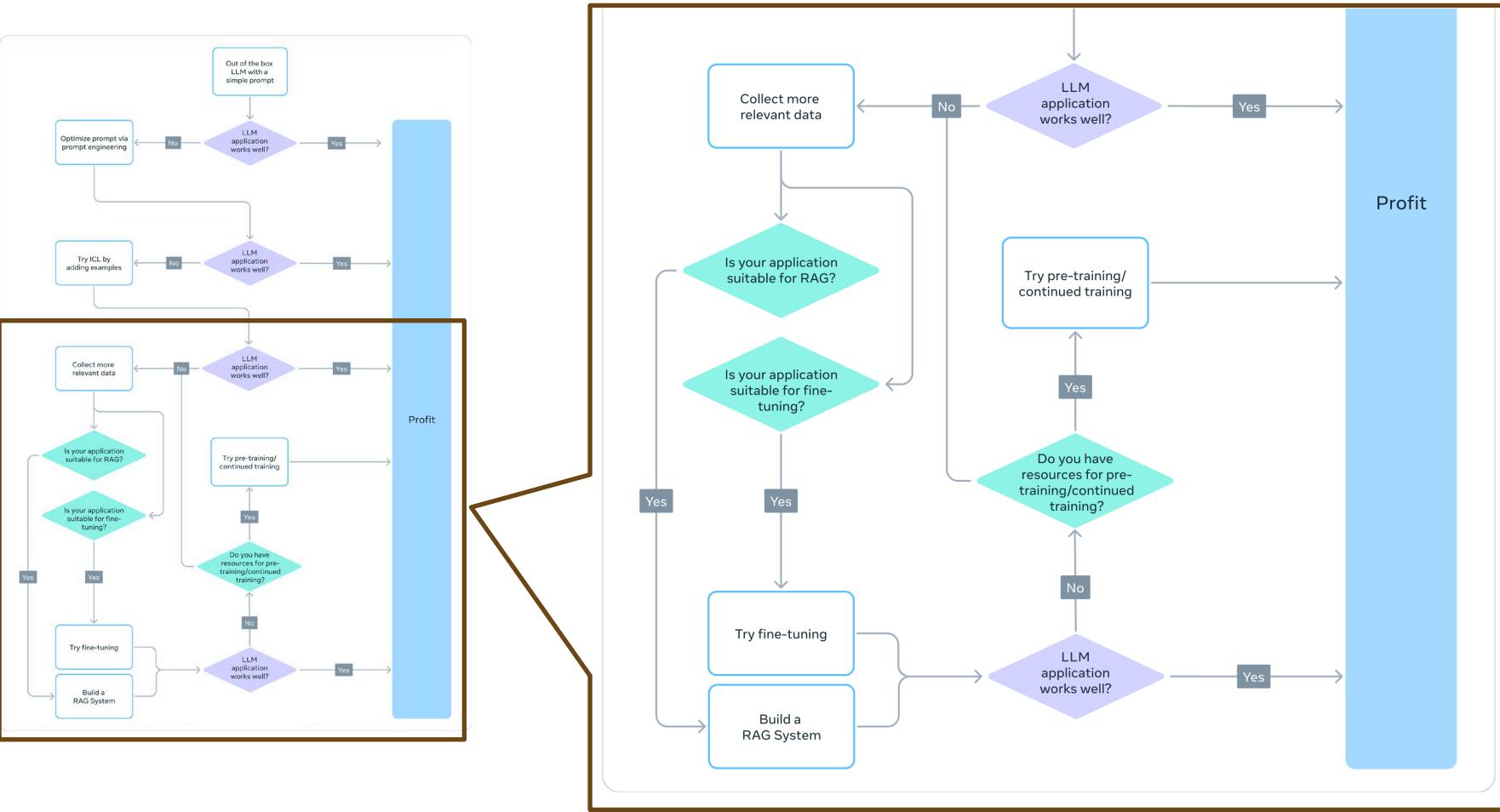
# To fine-tune or not to fine-tune

- LLM Fine-tuning이 유용한 상황
  - 톤, 스타일, 포맷 맞춤화 : 특정 인물의 스타일이나 특정 청중을 겨냥한 모델을 만들 때
  - 정확도 향상 및 예외 처리 : 예측 오류, 환각(hallucination)의 수정, 복잡한 작업을 수행하도록 모델의 능력을 개선
  - 미비한 도메인 다루기 : 일반적인 데이터로 훈련된 LLM은 특정 전문 용어에 취약. 의료, 법률, 금융 등의 전문 도메인에서는 추가적인 학습을 통해 정확도를 향상
  - 비용 절감 : 세부 조정을 통해 더 큰 모델(예: Llama 2 70B/GPT-4)의 능력을 더 작은 모델(예: Llama 2 7B)로 압축
  - 새로운 작업/능력 추가: 세부 조정을 통해 모델에 새로운 능력을 추가할 수 있습니다. 예를 들어, 주어진 검색자에서 더 잘 정보를 활용하도록 하거나, 다른 LLM을 평가하는 작업을 수행

# To fine-tune or not to fine-tune

- FT vs ICL
  - 문맥 내 학습(ICL)은 세부 조정에 비해 간단하고 비용 효율적인 방법.
    - ✓ 다만, ICL 예시가 많을수록 추론 비용과 지연 시간이 증가.
  - ICL을 먼저 시도하여 세부 조정이 필요한지 평가 가능.
- FT vs RAG
  - LLM의 기본 성능이 부족할 경우 RAG를 먼저 시도하고, 세부 조정으로 전환할 수도 있음.
  - 여러 상황에서 RAG는 세부 조정의 대안이 아니라 오히려 보완적인 접근임.
    - ✓ 외부지식 필요? RAG
    - ✓ 톤, 행동, 어휘, 스타일 필요? FT
    - ✓ 환각에 취약? RAG
    - ✓ 동적 데이터? RAG





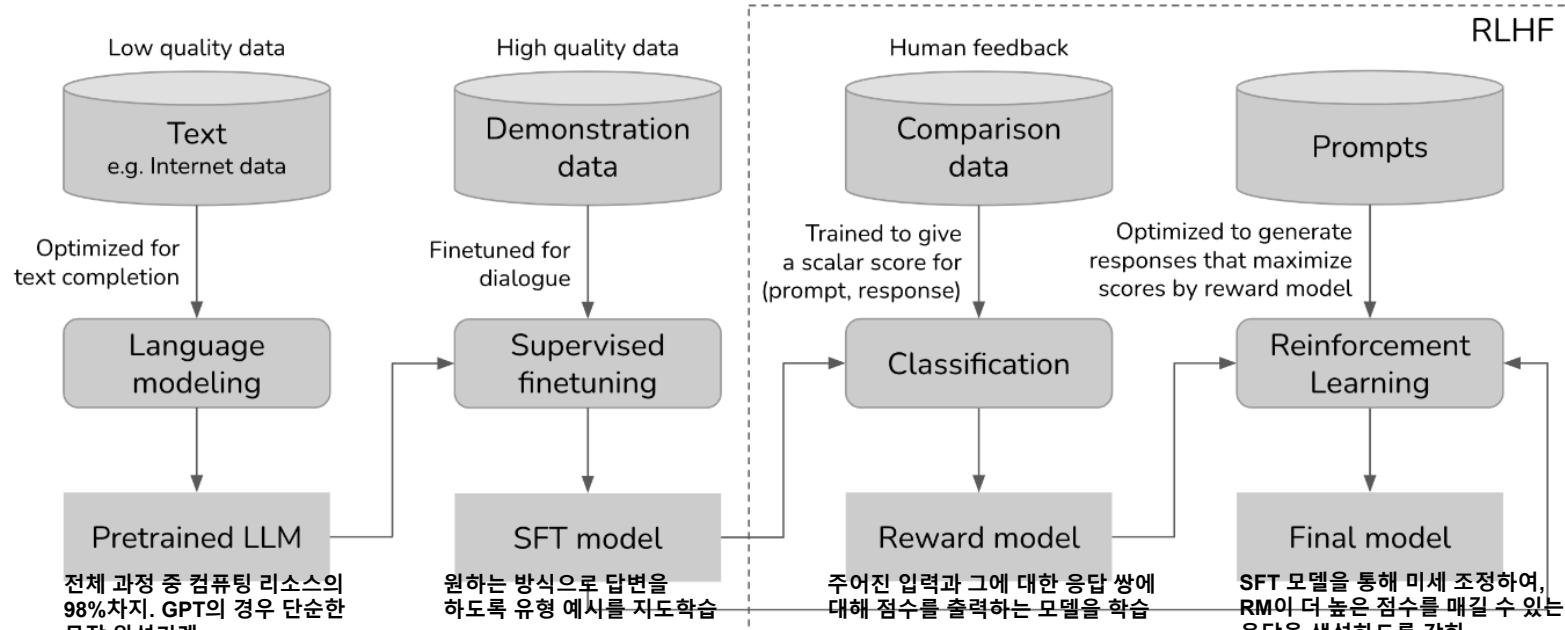
# Datasets

- 세부 조정(fine-tuning) 실험에서, 데이터셋의 품질과 구성은 성능 향상에 중요한 요소
- Data quality or quantity?
  - 품질 우선: 작은 고품질 데이터셋이 대용량 저품질 데이터셋보다 효과적. 오픈AI는 50-100개 예시로도 유의미한 차이를 줄 수 있다고 밝힘.
    - ✓ 텍스트 생성, 요약 같은 어려운 작업은 분류나 개체 추출보다 더 많은 데이터를 요구
  - 효율적인 고품질 데이터 수집
    - ✓ 실패 사례 관찰: 모델이 실패하는 예시를 모아 추가
    - ✓ 인적 개입: 기계가 생성한 답변을 사람이 빠르게 검토하여 레이블링 시간 단축

# Datasets

- Data diversity
  - 응답의 다양성 확보: 특정 유형의 응답에 과도하게 훈련되면 편향이 생길 수 있음.
  - 중복 제거: 중복을 제거하여 모델 성능 저하 방지.
  - 입력 다양성: 입력의 다양한 표현(패러프레이징)으로 데이터 다양성 증가.
  - 데이터셋 다양성: 다양한 데이터셋을 통합하여 사용하면 FT에 의한 학습-망각 trade-off를 개선할 수 있음.
  - 표준화된 출력: 불필요한 공백이나 형식을 제거해 모델이 핵심 개념에 집중하도록 함.

# Instruction tuning



Scale  
May '23

>1 trillion  
tokens

10K - 100K  
(prompt, response)

100K - 1M comparisons  
(prompt, winning\_response, losing\_response)

10K - 100K  
prompts

Examples  
**Bolded:** open sourced

GPT-x, Gopher, **Falcon**,  
LLaMa, **Pythia**, Bloom,  
**StableLM**

**Dolly-v2, Falcon-Instruct**

InstructGPT, ChatGPT,  
Claude, **StableVicuna**

# Instruction tuning

- Pretrained model의 단순한 다음 토큰 예측 모델을 답변생성기 전환 (feat. Chatgpt)

**Completion (Base model without fine-tuning):**

**User Prompt:**

A large language model is a type of artificial intelligence model that

**LLM Response:**

is trained on a huge amount of text data and can generate human-like language in response to queries.

**Instruction-Tuned Model:**

**User Prompt:**

What is a large language model?

**LLM Response:**

A large language model is a type of artificial intelligence model that is trained on a huge amount of text data and can generate human-like language in response to queries.

<Instruction tuning은 모델이 질문과 지시에 적절한 형식으로 응답>

- 예를 들어 프롬프트에 “How to make pizza”를 넣었을 때, GPT의 예상 동작은...
  - 1. 질문에 컨텍스트를 추가한다.: “for a family of six”
  - 2. 질문을 이어나간다.: “? What ingredients do I need? How much time would it take?”
  - 3. 실제 답변을 제공한다.
- 세 번째 옵션이 대화용 언어모델로 적합하다.
  - 우리가 원하는 적절한 응답의 예를 보여주는 것을 행동 복제(behavior cloning)라고 함.
  - SFT는 사전 학습된 모델이 이미 가지고 있지만 사용자가 프롬프트만으로는 액세스하기 어려운 기능을 잠금 해제하는 것

Table 1: Distribution of use case categories from our API prompt dataset.

Use-case	(%)
Generation	45.6%
Open QA	12.4%
Brainstorming	11.2%
Chat	8.4%
Rewrite	6.6%
Summarization	4.2%
Classification	3.5%
Other	3.5%
Closed QA	2.6%
Extract	1.9%

Table 2: Illustrative prompts from our API prompt dataset. These are fictional examples inspired by real usage—see more examples in Appendix A.2.1.

Use-case	Prompt
Brainstorming	List five ideas for how to regain enthusiasm for my career
Generation	Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.
Rewrite	This is the summary of a Broadway play: """ {summary} """ This is the outline of the commercial for that play: """

<OpenAI 유저의 프롬프트  
사용사례의 분포>

- Supervised Fine-Tuning vs. Instruction Tuning?
  - Supervised Fine-Tuning : label이 지정된 데이터를 사용해 특정 작업에 맞게 미세 조정
    - ✓ 데이터: 입력-출력 쌍이 포함된 라벨이 있는 데이터셋을 사용
  - Instruction Tuning : 모델이 다양한 작업을 수행할 수 있도록 자연어 지시를 학습해 미세 조정
    - ✓ 연어로 작성된 지시와 그에 따른 출력 데이터가 포함된 데이터셋을 사용
  - Instruction Tuning이 본질적으로 label이 있는 데이터를 사용하므로, SFT의 방법으로 볼 수 있음.
- 다양한 작업에 대한 지시(instruction)와 그에 따른 출력 데이터가 포함된 데이터셋이 필요

**Instruction:** Write a letter from the perspective of a cat.

**Input:** Null

**Output:**

Dear [Owner], I am writing to you today because I have a problem. You see, I've been living with you for the past [Number of Years] and I love it here. The food is great, the bed is comfy, and there are plenty of toys to play with. But lately, I've noticed that you're not as affectionate towards me as you used to be. In fact, sometimes when I try to cuddle up next to you on the couch, you push me away ...



**Instruction:** I am looking for a job and I need to fill out an application form. Can you please help me complete it?

**Input:**

Application Form:

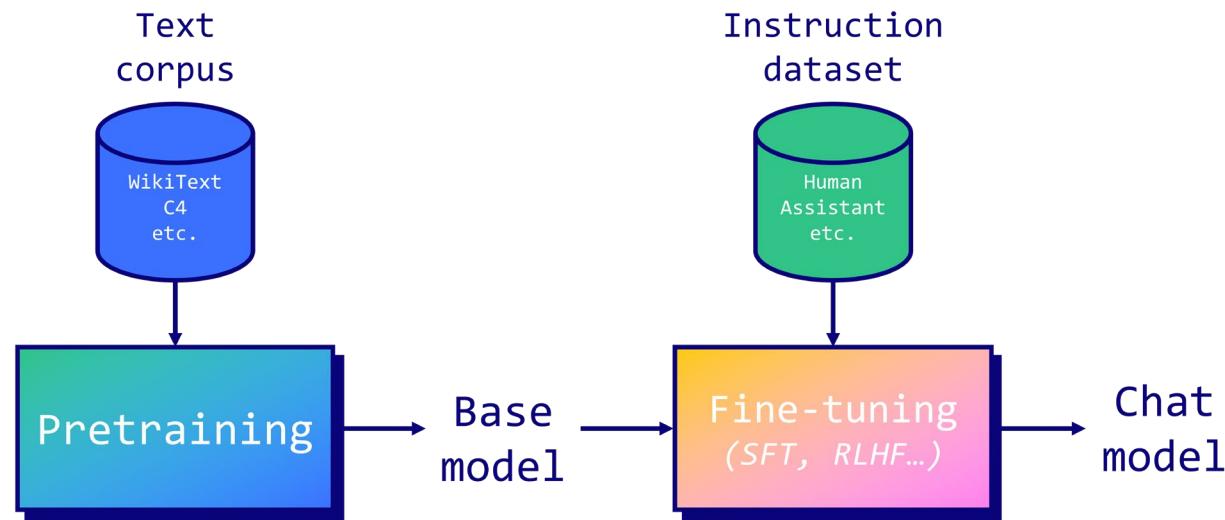
Name: \_\_\_\_\_ Age: \_\_\_\_\_ Sex: \_\_\_\_\_  
 Phone Number: \_\_\_\_\_ Email Address: \_\_\_\_\_  
 Education: \_\_\_\_\_ ...

**Output:**

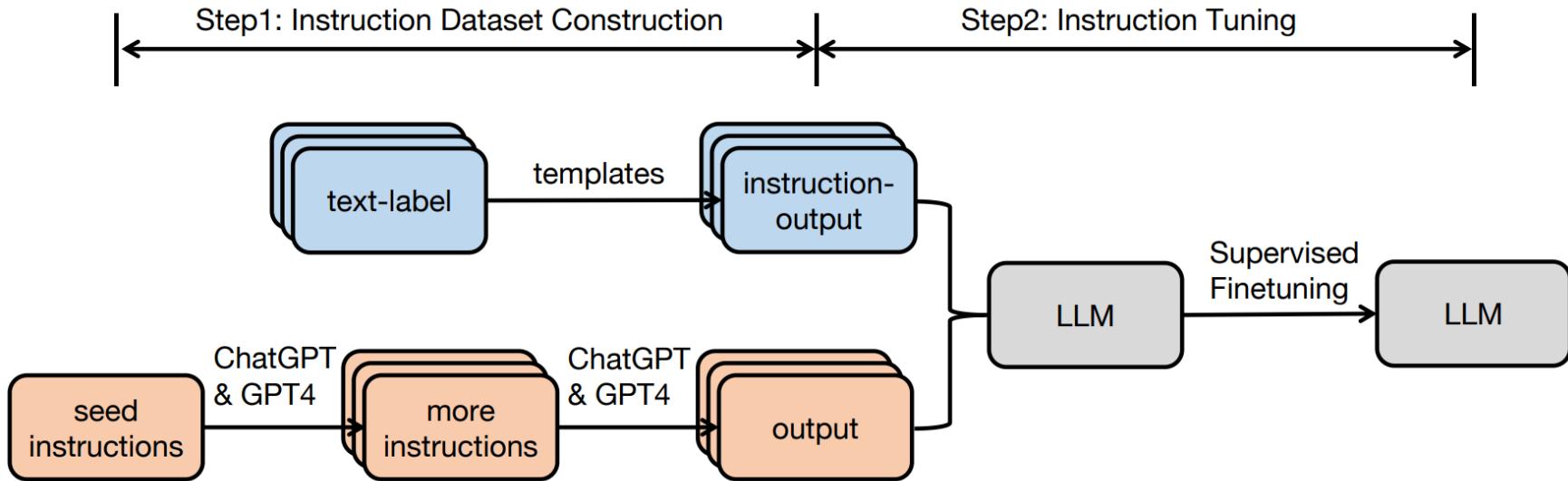
Name: John Doe Age: 25 Sex: Male  
 Phone Number: ...



- Instruction Tuning은 모델이 다양한 작업에 대해 자연어 지시를 수행하도록 학습
  - 데이터 규모가 작거나, 연산자원에 따라 SFT + LoRA와 같은 PEFT를 활용할 수도 있음.
  - 모델의 편향성, 위험성, 답변품질을 향상하기 위해 RLHF 수행할 수도 있음



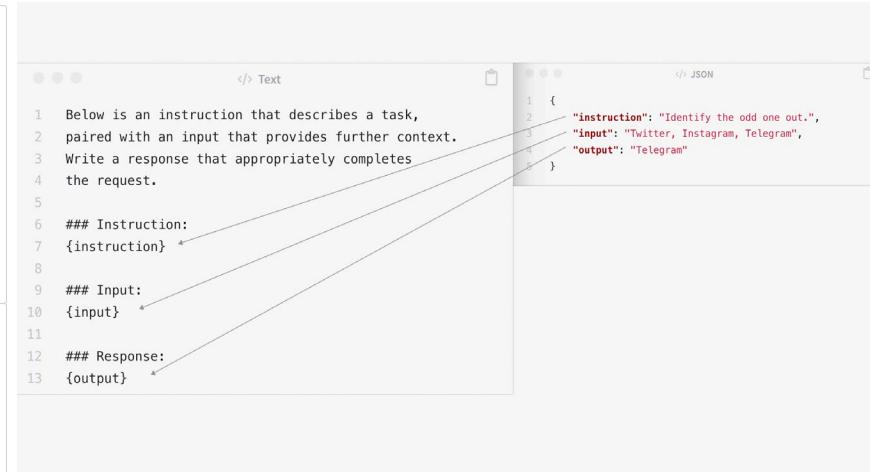
# Instruction Tuning workflow



# Preparing a Dataset for Instruction tuning

- 오픈 LLM을 Instruction tuning하기 위해 다양한 문제들에 대한 Instruction 데이터셋 필요
  - [Flan Collection](#)과 [Dolly15k dataset](#) 같이 수동 구축한 데이터셋
  - [Alpaca dataset](#) 같이 LLM을 사용해 자동적으로 생성한 데이터셋도 존재

Instruction	Answer
Explain the concept of a bubble sort algorithm to a non-technical audience.	A bubble sort algorithm is a type of sorting algorithm that is used to sort elements in an array. It works by looking at each element of the array and comparing it to the next element. If the first element is bigger than the second element, they are swapped. This process is repeated until the whole array is sorted. This type of sorting is one of the simplest sorting algorithms, but it can be slow if the array has many elements.
Instruction	Answer
Make the second sentence shorter.  Context Winter is usually the coldest season of the year. Snow is a common element during winter.	Winter is the coldest season, often accompanied by snow.



The screenshot shows a code editor interface with two tabs: 'Text' and 'JSON'. The 'Text' tab contains the following code:

```
1 Below is an instruction that describes a task,  
2 paired with an input that provides further context.  
3 Write a response that appropriately completes  
4 the request.  
5  
6 ### Instruction:  
7 {instruction}  
8  
9 ### Input:  
10 {input}  
11  
12 ### Response:  
13 {output}
```

The 'JSON' tab contains the following JSON object:

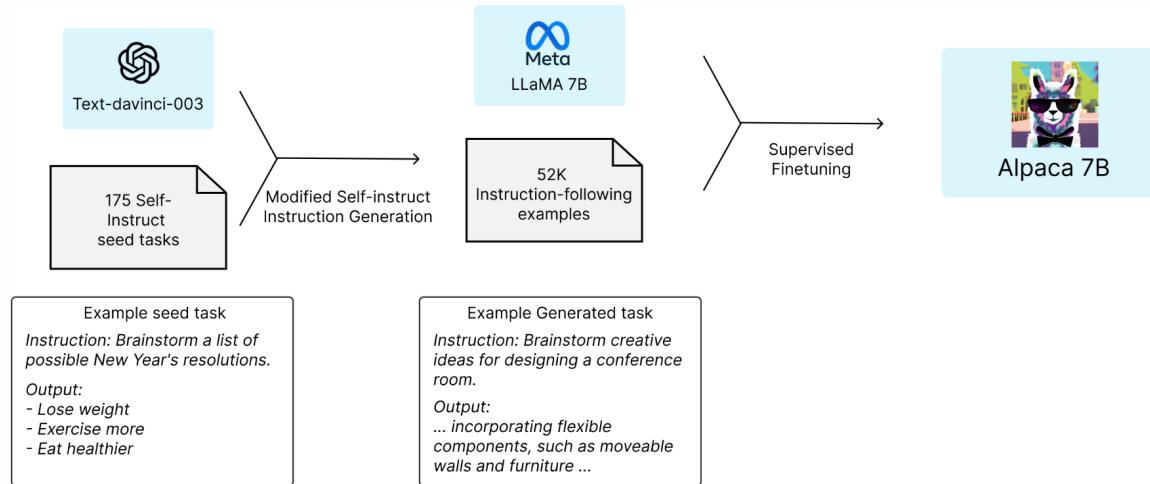
```
1 {  
2   "instruction": "Identify the odd one out.",  
3   "input": "Twitter, Instagram, Telegram",  
4   "output": "Telegram"  
5 }
```

Arrows point from the numbered lines in the 'Text' tab to the corresponding JSON object in the 'JSON' tab. Line 1 points to the 'instruction' key, line 2 points to the 'input' key, and line 13 points to the 'output' key.

<Instruction과 answer 예시>

# Alpaca Dataset

- 스탠포드 연구자들이 OpenAI davinci 모델을 사용하여 지시사항/출력 쌍을 생성
  - Llama를 미세 조정해 Alpaca 모델을 학습하기 위해 사용한 합성 데이터셋
  - 이메일 작성, 소설 미디어, 생산성 도구를 포함한 다양한 사용자 지향적 지시사항 목록을 커버



# Alpaca Dataset preparation and tokenization

- Alpaca 데이터셋은 단일 Json 파일 형태로 구성할 수 있음.
  - 데이터셋의 각 행은 'instruction', 'input', 'output' 키를 가진 딕셔너리
- 전처리 함수 정의
  - 입력이 없는 지시사항과 입력이 있는 지시사항을 분리하여 처리
- EOS 추가, Tokenization, Padding, Packing 전처리

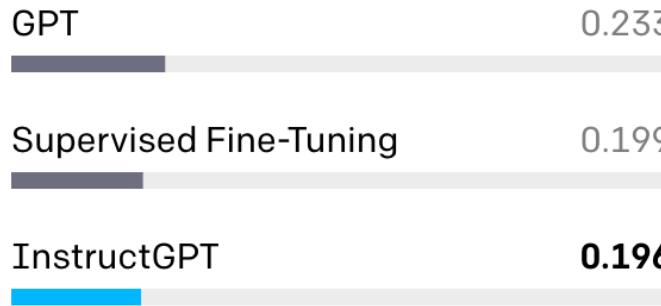
```
instruction: str, describes the task the model should perform.  
            Each of the 52K instructions is unique.  
input:       str, optional context or input for the task.  
output:      str, the answer to the instruction as generated by GPT-4.
```

<Alpaca style instruction data>

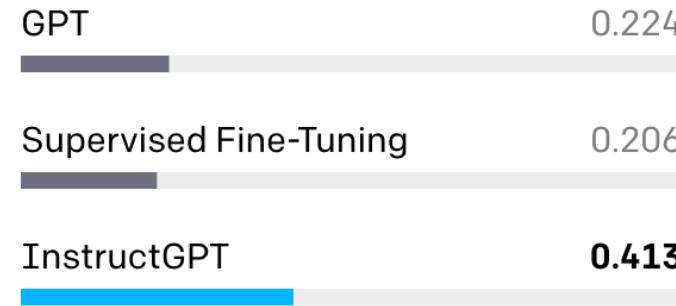
# RLHF : Reinforcement Learning from Human Feedback

- LLM이 더 크고 유용해짐에 따라, 그것들을 안전하고 편향되지 않게 유지하는 방법이 필요
- GPT-3는 모델의 우수성뿐만 아니라 인종, 성별, 종교 등에서 편향성을 가짐
- LLM의 편향성, 위험성을 줄이기 위한 방법으로 인간의 피드백을 사용해 InstructGPT 구축
  - InstructGPT는 사용자가 제공하는 지시사항을 따르도록 설계
  - 모델의 출력이 인간의 기대와 더 일치하여 더 정확하고, 유용한 정보를 제공하는 개선된 결과 획득
    - ✓ Proximal Policy Optimization(PPO) : 주어진 텍스트가 인간에 의해 높게 순위가 매겨졌는지를 예측하는 보상 모델을 활용해 SFT 모델을 최적화하는 데 사용
    - ✓ Direct Preference Optimization(DPO) : 문제를 분류 문제로 재구성함으로써 과정을 단순화. 보상 모델 대신 참조 모델을 사용하며(훈련이 필요 없음), 오직 하나의 하이퍼파라미터만 요구해, 더 안정적이고 효율적

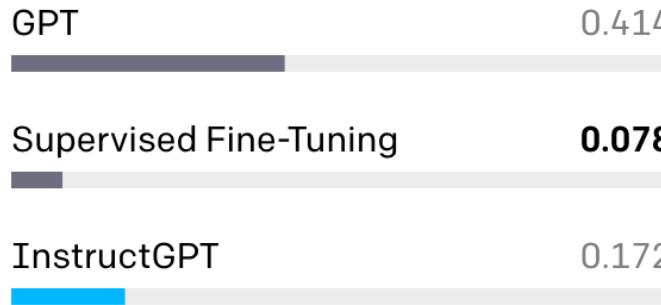
Dataset  
**RealToxicity**



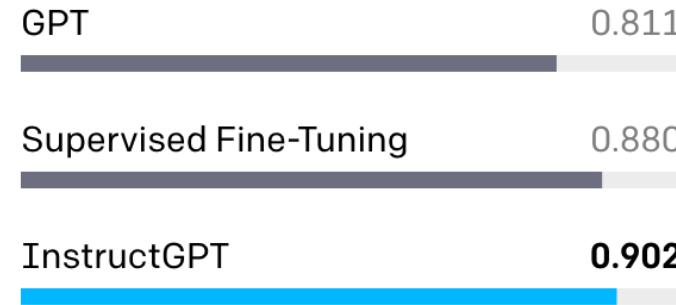
Dataset  
**TruthfulQA**



API Dataset  
**Hallucinations**



API Dataset  
**Customer Assistant Appropriate**



## Step 1

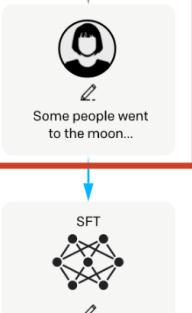
**Collect demonstration data,  
and train a supervised policy.**

A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



## Step 2

**Collect comparison data,  
and train a reward model.**

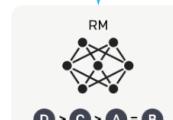
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used  
to train our  
reward model.



### Step 3

**Optimize a policy against  
the reward model using  
reinforcement learning.**

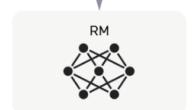
A new prompt  
is sampled from  
the dataset.



The policy generates an output.

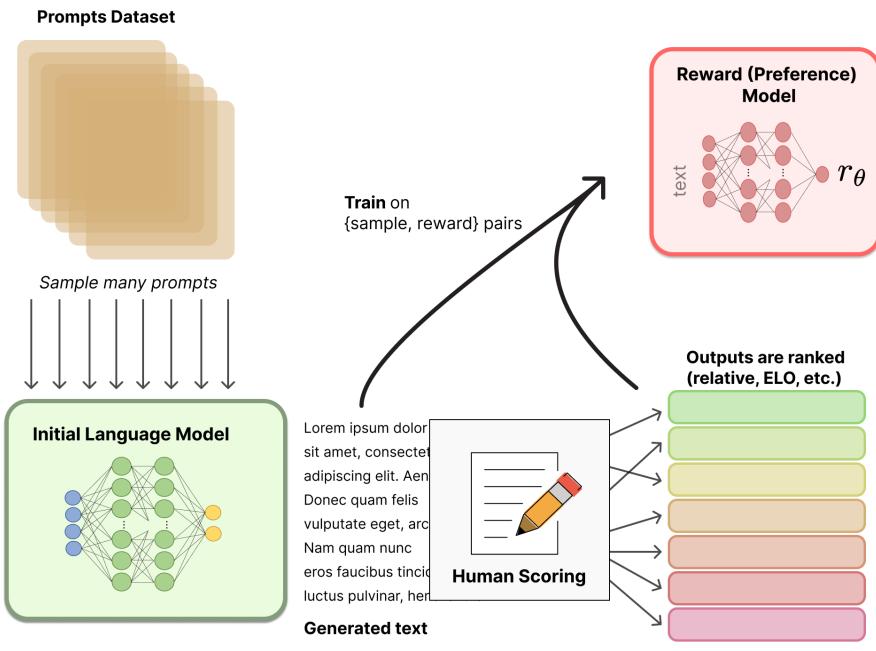


The reward mode calculates a reward for the output.

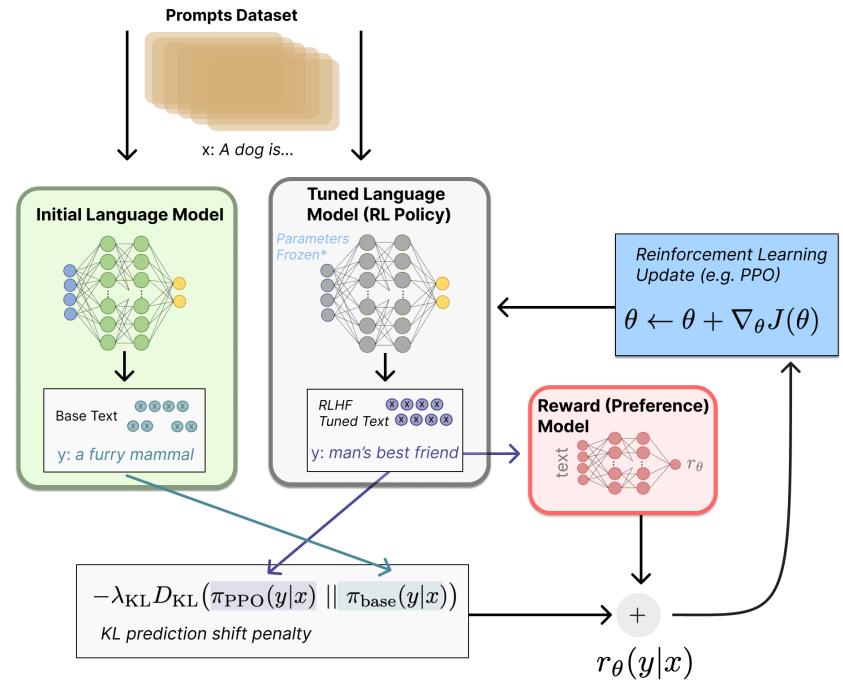


The reward is used to update the policy using PPO.

- Step1) 사전 훈련된 LLM에서 시작하여, SFT(인간의 피드백)으로 미세 조정
  - Step2) SFT의 결과들과 인간의 피드백의 순위를 결정해 보상 모델(RM) 훈련
  - Step3) SFT를 다시 미세 조정하지만, 인간의 피드백 대신 RM을 사용한 강화 학습을 사용.



<Reward model training>



<Fine-tuning with RL>

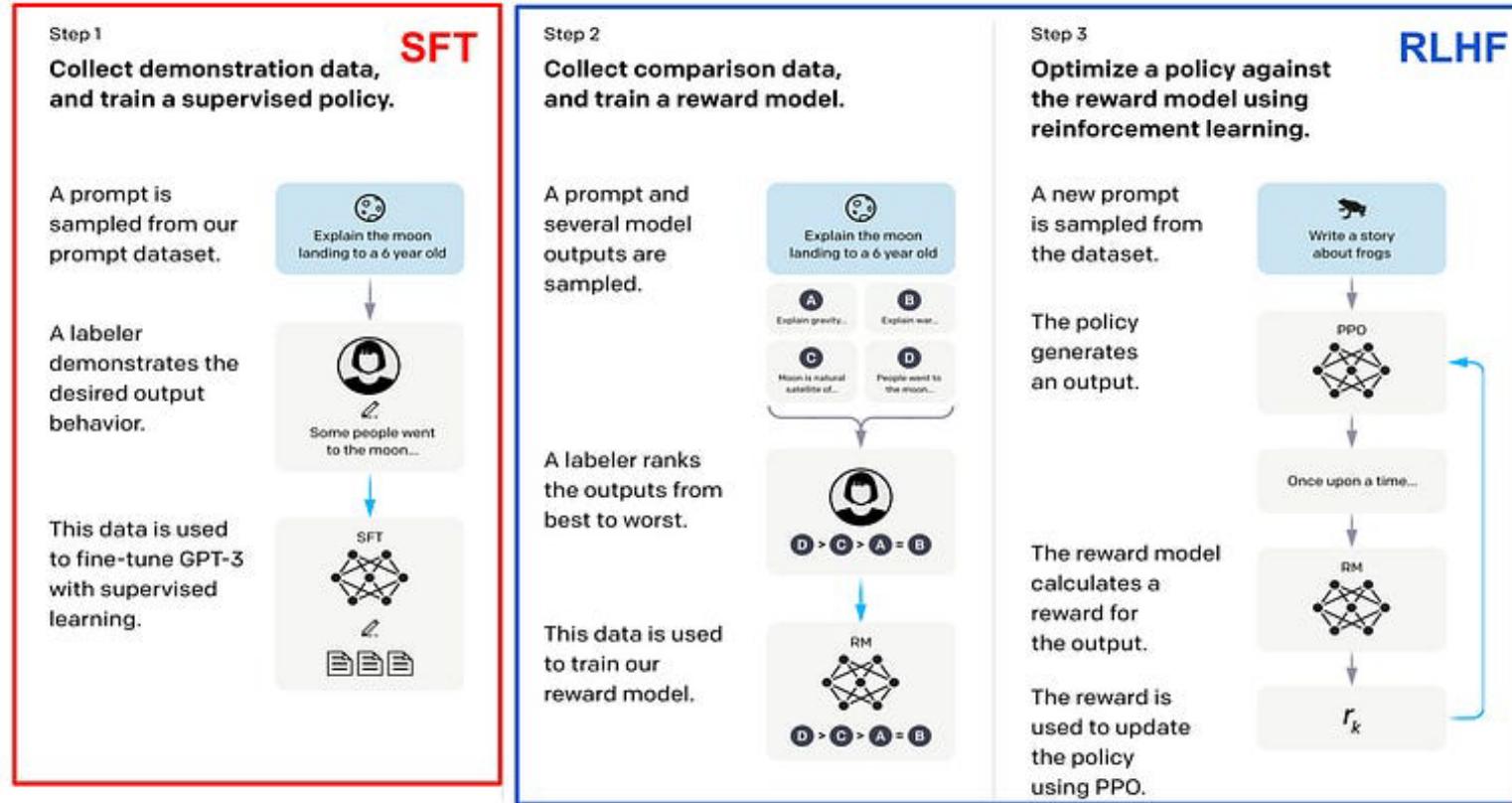
# PEFT

## Parameter-Efficient Fine-Tuning

# Foundation model의 시대

- **Foundation model**은 특정 도메인에 국한되지 않는 거대 사전 학습된 모델
  - Pretrained model보다 더 다양한 다운스트림 작업에 범용적으로 적용된다는 맥락
  - 광범위한 데이터로부터 언어, 이미지, 오디오 등의 복잡한 패턴과 지식을 학습하여, 이를 다양한 특정 작업에 적용할 수 있는 능력
- 최소 수십억 개 이상의 파라미터를 가지며, 이러한 복잡성은 더 많은 패턴과 정보를 학습
  - 범용성, 데이터 효율성, 고성능, 효과적인 전이학습의 이점
  - 리소스 집약적, 과적합 문제, 대규모 망각의 문제점
- **PEFT(Parameterized, Efficient Fine-Tuning)**의 중요성 증가
  - PEFT는 모델의 일부분만을 미세 조정함으로써 효율성을 높이고, 리소스를 절약
  - 빠른 학습, 소규모 데이터 요구, 하드웨어 요구사항 완화, 산업적 활용에서 강점

# Alignment process for Generative LLMs

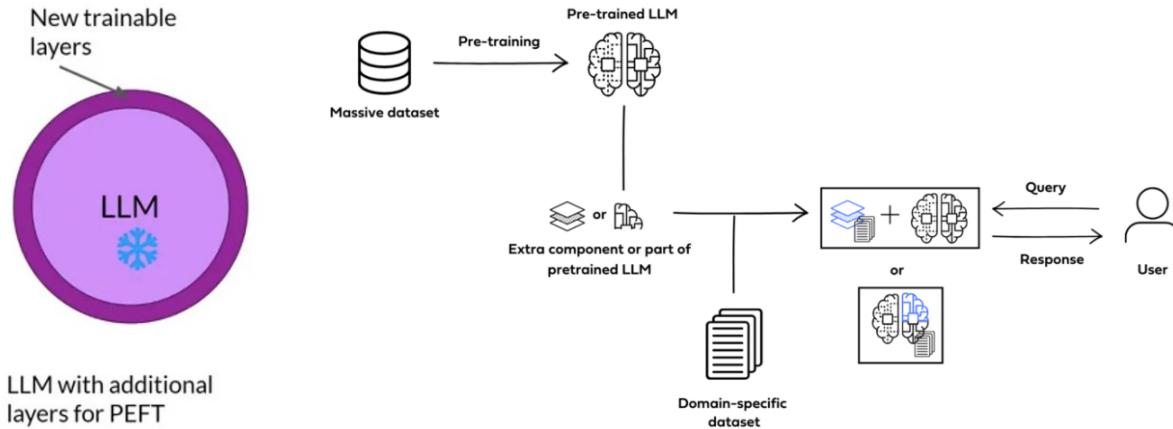


# How to adapt foundation models to your task

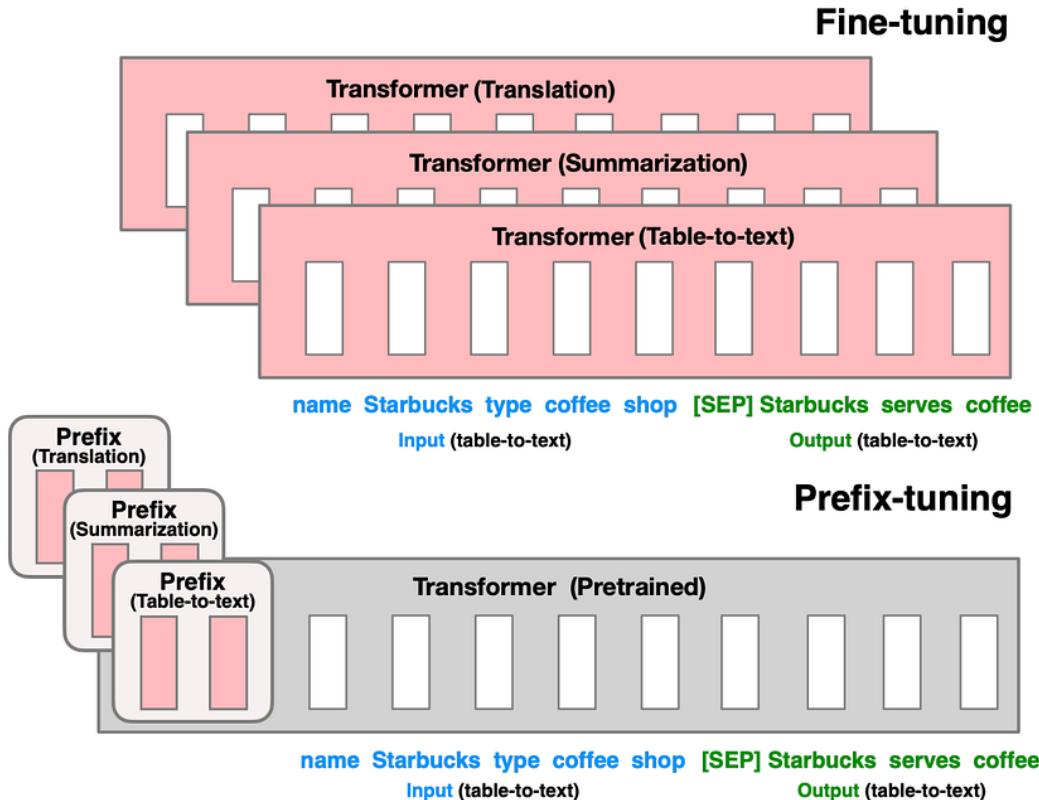
- 사전학습 이후 Foundation 모델은 여러 종류의 다운스트림(Downstream) 작업에 맞게 조정
- 주요 접근 방식
  - In-Context learning : 입력 컨텍스트로 좋은 답변을 생성하기 위한 hidden state 학습
    - ✓ *prompt engineering* 방식 사용하나, 성능 향상 한계 존재
  - Full fine-tuning : 모델의 전체 파라미터를 데이터셋에 대해 업데이트하는 방식
    - ✓ 과적합 문제, 높은 메모리 및 연산 사용, 복잡한 하이퍼파라미터 튜닝의 단점 존재
  - **Parameter-Efficient Fine-Tuning, PEFT** : 모델의 매우 작은 매개변수만 조정
    - ✓ 좋은 성능, 빠른 학습, 소규모 데이터의 장점

# PEFT : Parameter-Efficient Fine-Tuning

- Parameter-Efficient Fine-Tuning (PEFT) 방법론은 대규모 Foundation 모델을 더 적은 매개변수 변경으로 특정 작업에 효과적으로 적용할 수 있게 하는 다양한 전략
  - Prompt-Based Methods: 모델 입력에 특정 프롬프트나 작은 조정 가능한 매개변수 집합을 추가하여, 기존 모델이 특정 작업을 더 잘 수행할 수 있도록 유도
  - Adapter Modules: 모델의 기존 아키텍처 사이에 작은 신경망 모듈(Adapters)을 삽입하여, 전체 모델을 재학습하는 대신 이러한 모듈만을 특정 작업에 맞게 학습.



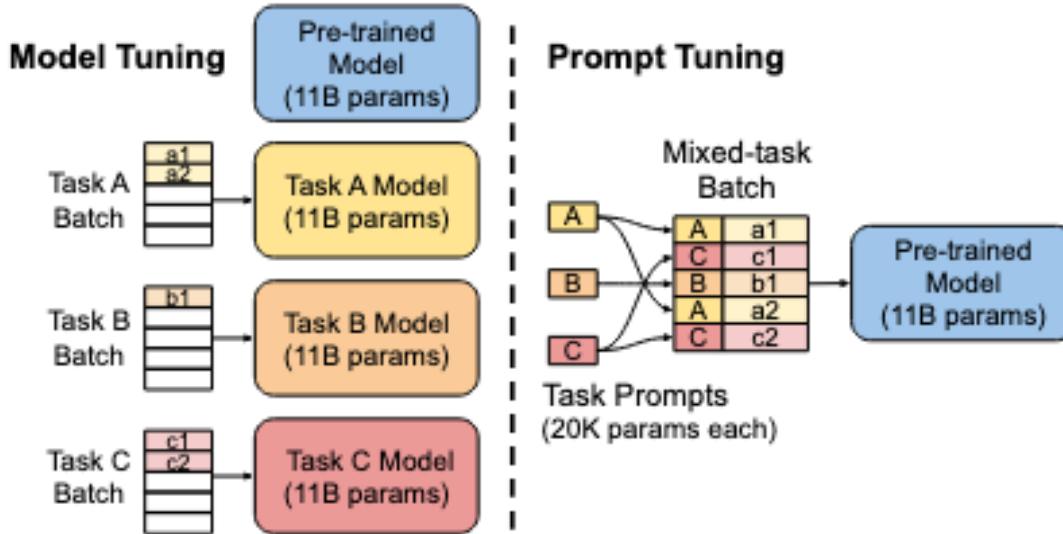
# Prefix-tuning



- 모델의 Encoder 입력에 prefix라고 불리는 추가적인 학습가능한 토큰 시퀀스를 추가
  - 전체 모델의 파라미터를 미세 조정하는 대신, 매우 작은 토큰 임베딩 부분만 튜닝
- 특정 작업을 수행할 때 prefix embedding을 통해 어떤 컨텍스트를 고려해야 하는지 모델을 제어함.

Figure 1: Fine-tuning (top) updates all Transformer parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the Transformer parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denote transformer activations at one time step.

# Prompt Tuning



- 고정된 언어모델에 “소프트 프롬프트”라는 가변적인 매개변수 입력을 추가하여 작동
  - Prefix tuning은 고정된 접두사를 각 레이어의 입력 데이터 앞에 추가.
  - prompt tuning은 오직 학습 가능한 소프트 프롬프트인 입력 부분만을 사용

# LoRA: Low-Rank Adaptation of Large Language Models

- 핵심 아이디어는 모델을 동결한 채, 학습 정보를 분리 가능한 어댑터에 저장
- 어댑터는 모델 파라미터 대신 업데이트되는 학습 가능한 저차원 연산 모듈
  - 학습할 행렬을 두 개의 low-rank matrix로 분리
  - 두 matrix는 곱하여 기존 행렬과 동일 크기 갖음.

Finetuned Weights

$$\overbrace{W_{\text{ft}}}^{\text{Finetuned Weights}} = \underbrace{W_{\text{pt}}}_{\text{Pretrained Weights}} + \overbrace{\Delta W}^{\text{Weight Update}}$$

Weight Update

Pretrained Weights

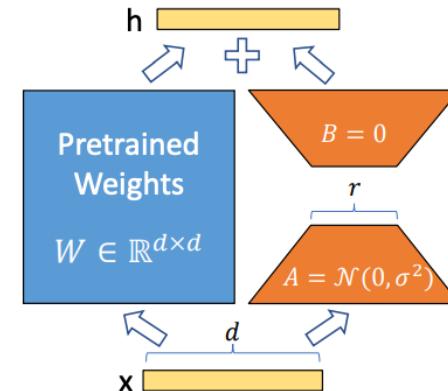


Figure 1: Our reparametrization. We only train  $A$  and  $B$ .

**Rank Decomposition  
Matrix**

$$W_{\text{ft}} = W_{\text{pt}} + \underbrace{\Delta W}_{\text{Approximation}} = W_{\text{pt}} + \overbrace{AB}^{\text{Rank Decomposition Matrix}}$$

where  $W_{\text{ft}}, W_{\text{pt}}, \Delta W, AB \in \mathbb{R}^{d \times d}$

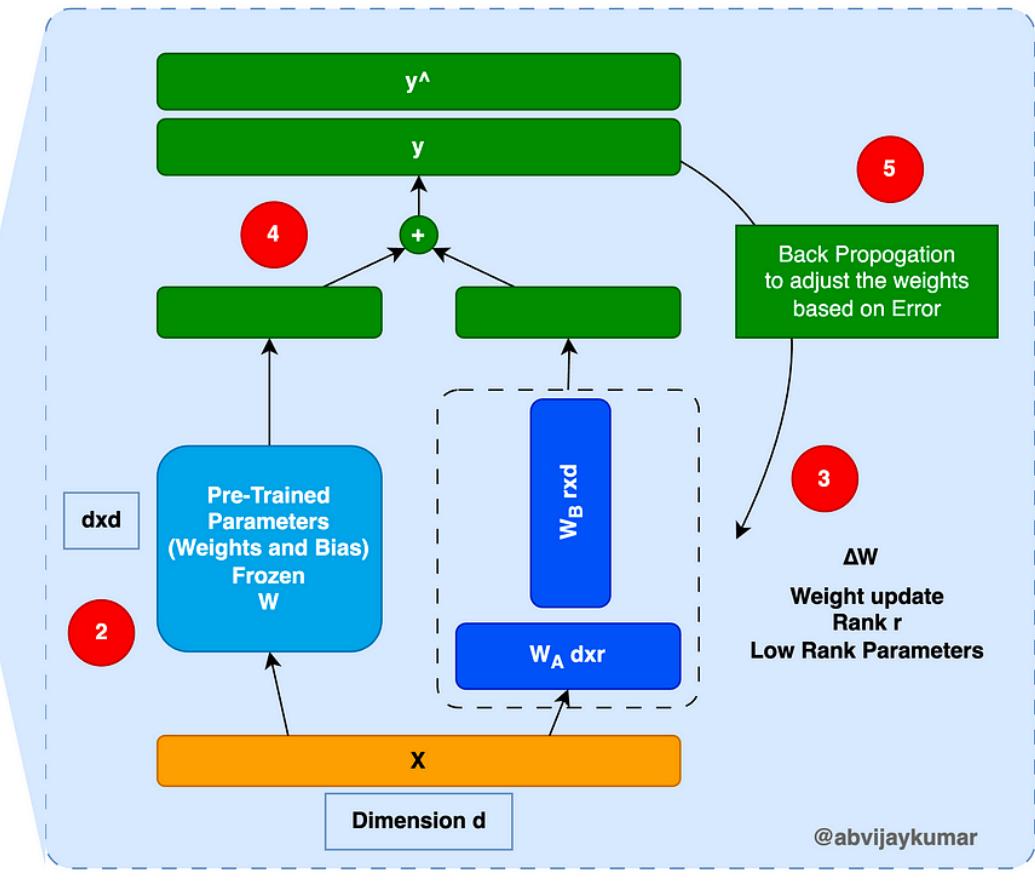
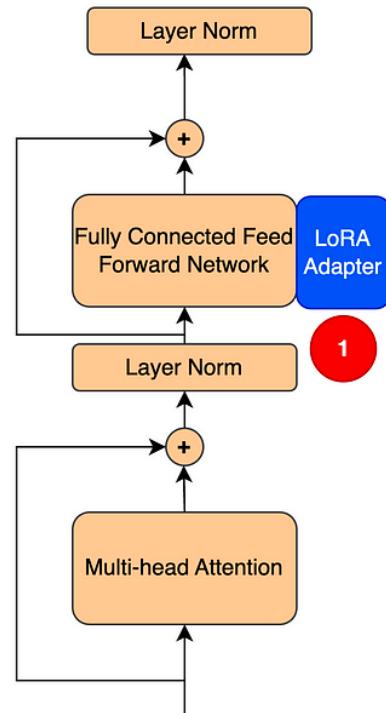
and  $\underbrace{A \in \mathbb{R}^{d \times r}, B \in \mathbb{R}^{r \times d}}_{\text{Low Rank}}$

<Rank Decomposition을 통한 업데이트 가중치 분리>

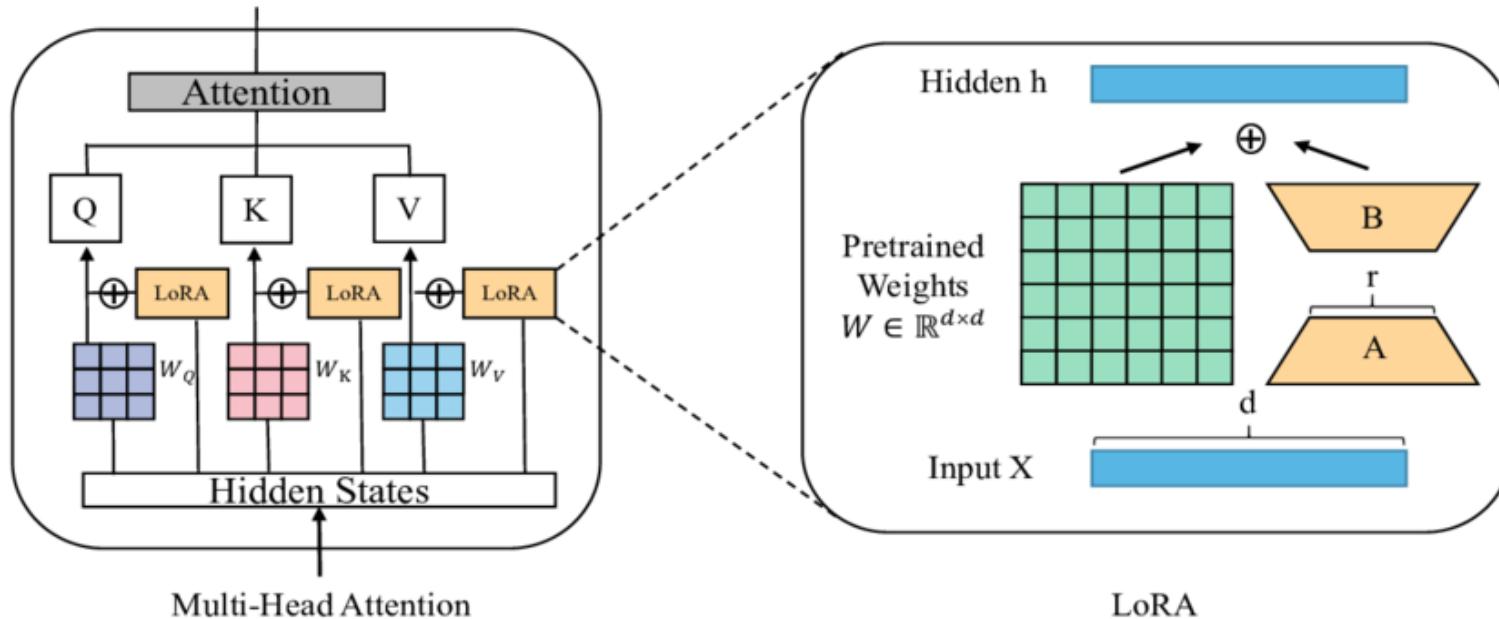
**Scaling Factor**

$$W_{\text{ft}} = W_{\text{pt}} + \frac{\alpha}{r} \overbrace{AB}^{\text{Rank Decomposition Matrix}}$$

<LoRA 학습 시 Scaling factor 도입>



# LoRA in Transformers



# "Implementing LoRA in Transformers with Python."

```
def train(self, mode: bool = True):
    def T(w):
        return w.T if self.fan_in_fan_out else w
    nn.Linear.train(self, mode)
    if self.merge_weights and self.merged:
        # Make sure that the weights are not merged
        if self.r > 0:
            self.weight.data -= T(self.lora_B @ self.lora_A) * self.scaling
    self.merged = False
```

```
def eval(self):
    def T(w):
        return w.T if self.fan_in_fan_out else w
    nn.Linear.eval(self)
    if self.merge_weights and not self.merged:
        # Merge the weights and mark it
        if self.r > 0:
            self.weight.data += T(self.lora_B @ self.lora_A) * self.scaling
    self.merged = True
```

$$h = W_0x + \Delta Wx = W_0x + BAx$$

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB <sub>base</sub> (FT)*	125.0M	<b>87.6</b>	94.8	90.2	<b>63.6</b>	92.8	<b>91.9</b>	78.7	91.2	86.4
RoB <sub>base</sub> (BitFit)*	0.1M	84.7	93.7	<b>92.7</b>	62.0	91.8	84.0	81.5	90.8	85.2
RoB <sub>base</sub> (Adpt <sup>D</sup> )*	0.3M	87.1 $\pm$ 0.0	94.2 $\pm$ 1.1	88.5 $\pm$ 1.1	60.8 $\pm$ 4.4	93.1 $\pm$ 1.1	90.2 $\pm$ 0.0	71.5 $\pm$ 2.7	89.7 $\pm$ 3.3	84.4
RoB <sub>base</sub> (Adpt <sup>D</sup> )*	0.9M	87.3 $\pm$ 1.1	94.7 $\pm$ 1.3	88.4 $\pm$ 1.1	62.6 $\pm$ 9.9	93.0 $\pm$ 2.2	90.6 $\pm$ 0.0	75.9 $\pm$ 2.2	90.3 $\pm$ 1.1	85.4
RoB <sub>base</sub> (LoRA)	0.3M	87.5 $\pm$ 3.1	<b>95.1<math>\pm</math>2.1</b>	89.7 $\pm$ 7.7	63.4 $\pm$ 1.2	<b>93.3<math>\pm</math>3.3</b>	90.8 $\pm$ 1.1	<b>86.6<math>\pm</math>7.7</b>	<b>91.5<math>\pm</math>2.2</b>	<b>87.2</b>
RoB <sub>large</sub> (FT)*	355.0M	90.2	<b>96.4</b>	<b>90.9</b>	68.0	94.7	<b>92.2</b>	86.6	92.4	88.9
RoB <sub>large</sub> (LoRA)	0.8M	<b>90.6<math>\pm</math>2.2</b>	96.2 $\pm$ 5.5	<b>90.9<math>\pm</math>1.2</b>	68.2 $\pm$ 1.9	<b>94.9<math>\pm</math>3.3</b>	91.6 $\pm$ 1.1	<b>87.4<math>\pm</math>2.5</b>	<b>92.6<math>\pm</math>2.2</b>	<b>89.0</b>
RoB <sub>large</sub> (Adpt <sup>P</sup> )†	3.0M	90.2 $\pm$ 3.1	96.1 $\pm$ 3.3	90.2 $\pm$ 7.7	<b>68.3<math>\pm</math>10.0</b>	<b>94.8<math>\pm</math>2.2</b>	<b>91.9<math>\pm</math>1.1</b>	83.8 $\pm$ 2.9	92.1 $\pm$ 7.7	88.4
RoB <sub>large</sub> (Adpt <sup>P</sup> )†	0.8M	<b>90.5<math>\pm</math>3.1</b>	<b>96.6<math>\pm</math>2.2</b>	89.7 $\pm$ 1.2	67.8 $\pm$ 2.5	<b>94.8<math>\pm</math>3.1</b>	91.7 $\pm$ 2.2	80.1 $\pm$ 2.9	91.9 $\pm$ 4.4	87.9
RoB <sub>large</sub> (Adpt <sup>H</sup> )†	6.0M	89.9 $\pm$ 5.5	96.2 $\pm$ 3.3	88.7 $\pm$ 2.9	66.5 $\pm$ 4.4	94.7 $\pm$ 2.2	92.1 $\pm$ 1.1	83.4 $\pm$ 1.1	91.0 $\pm$ 1.7	87.8
RoB <sub>large</sub> (Adpt <sup>H</sup> )†	0.8M	90.3 $\pm$ 3.1	96.3 $\pm$ 3.5	87.7 $\pm$ 1.7	66.3 $\pm$ 2.0	94.7 $\pm$ 2.1	91.5 $\pm$ 1.1	72.9 $\pm$ 2.9	91.5 $\pm$ 5.5	86.4
RoB <sub>large</sub> (LoRA)†	0.8M	<b>90.6<math>\pm</math>2.2</b>	96.2 $\pm$ 5.5	<b>90.2<math>\pm</math>1.0</b>	68.2 $\pm$ 1.9	<b>94.8<math>\pm</math>3.3</b>	91.6 $\pm$ 2.2	<b>85.2<math>\pm</math>1.1</b>	<b>92.3<math>\pm</math>3.5</b>	<b>88.6</b>
DeBXXL (FT)*	1500.0M	91.8	<b>97.2</b>	92.0	72.0	<b>96.0</b>	92.7	93.9	92.9	91.1
DeBXXL (LoRA)	4.7M	<b>91.9<math>\pm</math>2.2</b>	96.9 $\pm$ 2.2	<b>92.6<math>\pm</math>6.6</b>	<b>72.4<math>\pm</math>1.1</b>	<b>96.0<math>\pm</math>1.1</b>	<b>92.9<math>\pm</math>1.1</b>	<b>94.9<math>\pm</math>4.4</b>	<b>93.0<math>\pm</math>2.2</b>	<b>91.3</b>

Table 2: RoBERTa<sub>base</sub>, RoBERTa<sub>large</sub>, and DeBERTa<sub>XXL</sub>, with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew’s correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. \* indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	<b>68.2</b>	<b>8.62</b>	<b>46.2</b>	71.0	2.47
GPT-2 M (Adapter <sup>L</sup> )*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter <sup>L</sup> )*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter <sup>H</sup> )	11.09M	67.3 $\pm$ 6.6	8.50 $\pm$ .07	46.0 $\pm$ .2	70.7 $\pm$ .2	2.44 $\pm$ .01
GPT-2 M (FT <sup>Top2</sup> )*	25.19M	68.1	<b>8.59</b>	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	<b>70.4<math>\pm</math>.1</b>	<b>8.85<math>\pm</math>.02</b>	<b>46.8<math>\pm</math>.2</b>	<b>71.8<math>\pm</math>.1</b>	<b>2.53<math>\pm</math>.02</b>
GPT-2 L (FT)*	774.03M	<b>68.5</b>	<b>8.78</b>	<b>46.0</b>	69.9	2.45
GPT-2 L (Adapter <sup>L</sup> )	0.88M	69.1 $\pm$ .1	8.68 $\pm$ .03	46.3 $\pm$ .0	71.4 $\pm$ .2	<b>2.49<math>\pm</math>.0</b>
GPT-2 L (Adapter <sup>L</sup> )	23.00M	68.9 $\pm$ .3	8.70 $\pm$ .04	46.1 $\pm$ .1	71.3 $\pm$ .2	2.45 $\pm$ .02
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	<b>70.4<math>\pm</math>.1</b>	<b>8.89<math>\pm</math>.02</b>	<b>46.8<math>\pm</math>.2</b>	<b>72.0<math>\pm</math>.2</b>	2.47 $\pm$ .02

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. \* indicates numbers published in prior works.

Model&Method	# Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Acc. (%)	Acc. (%)	R1/R2/RL
GPT-3 (FT)	175,255.8M	<b>73.8</b>	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter <sup>H</sup> )	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter <sup>H</sup> )	40.1M	73.2	<b>91.5</b>	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	<b>91.7</b>	<b>53.8/29.8/45.9</b>
GPT-3 (LoRA)	37.7M	<b>74.0</b>	<b>91.6</b>	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around  $\pm 0.5\%$ , MNLI-m around  $\pm 0.1\%$ , and SAMSum around  $\pm 0.2/\pm 0.2/\pm 0.1$  for the three metrics.

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter <sup>L</sup> )*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter <sup>L</sup> )*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter <sup>H</sup> )	11.09M	67.3 <sub>±.6</sub>	8.50 <sub>±.07</sub>	46.0 <sub>±.2</sub>	70.7 <sub>±.2</sub>	2.44 <sub>±.01</sub>
GPT-2 M (FT <sup>Top2</sup> )*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	<b>70.4<sub>±.1</sub></b>	<b>8.85<sub>±.02</sub></b>	<b>46.8<sub>±.2</sub></b>	<b>71.8<sub>±.1</sub></b>	<b>2.53<sub>±.02</sub></b>
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter <sup>L</sup> )	0.88M	69.1 <sub>±.1</sub>	8.68 <sub>±.03</sub>	46.3 <sub>±.0</sub>	71.4 <sub>±.2</sub>	<b>2.49<sub>±.0</sub></b>
GPT-2 L (Adapter <sup>L</sup> )	23.00M	68.9 <sub>±.3</sub>	8.70 <sub>±.04</sub>	46.1 <sub>±.1</sub>	71.3 <sub>±.2</sub>	2.45 <sub>±.02</sub>
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	<b>70.4<sub>±.1</sub></b>	<b>8.89<sub>±.02</sub></b>	<b>46.8<sub>±.2</sub></b>	<b>72.0<sub>±.2</sub></b>	2.47 <sub>±.02</sub>

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. \* indicates numbers published in prior works.

Model&Method	# Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Acc. (%)	Acc. (%)	R1/R2/RL
GPT-3 (FT)	175,255.8M	<b>73.8</b>	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter <sup>H</sup> )	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter <sup>H</sup> )	40.1M	73.2	<b>91.5</b>	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	<b>91.7</b>	<b>53.8/29.8/45.9</b>
GPT-3 (LoRA)	37.7M	<b>74.0</b>	<b>91.6</b>	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around  $\pm 0.5\%$ , MNLI-m around  $\pm 0.1\%$ , and SAMSum around  $\pm 0.2/\pm 0.2/\pm 0.1$  for the three metrics.