AI

<8주차>

Pre-training (사전 학습)

개념:

- 언어모델이 방대한 인터넷 텍스트 데이터를 기반으로 언어 패턴과 일반 지식을 학습하는 단계
- Self-supervised learning(자기지도학습) 방식 사용 -> 입력 데이터 자체에서 학습 신호를 생성

학습 목표: 다음 단어 예측
Ex) 오늘은 날씨가 __ -> '맑다'를 예측하도록 학습

특징:

- 지도학습이 아님 (정답 레이블 필요 없음)
- 언어 구조, 통계적 패턴, 문맥 이해 능력 습득
- GPT, BERT 등의 기본 언어모델이 이 단계를 거침

한계:

- 인간의 의도나 요청을 이해하지 못함
- "지식은 많지만, 대화는 못하는 모델" 상태 -> 질문에 맞는 '행동'이나 '응답'을 하지 못함

1. Post-training (사후 학습)
   개념:

- 사전 학습된 모델에 '사람이 원하는 행동'을 학습시키는 단계
- 즉, 모델을 대화형·지시문 반응형으로 만들기 위한 과정

목표:

- 모델이 인간의 의도, 안전성, 유용성을 반영하여 작동하도록 조정
- Pre-training 모델을 실제 사용자 친화형 모델로 전환

구성: Instruction-tuning (지시문 학습) + RLHF (인간 피드백 기반 강화학습)
→ 이 두 단계를 합쳐 “Post-training”이라고 부름.

1. Instruction-tuning의 한계와 RLHF의 개념에 대한 이해

Instruction-tuning (지시문 학습)

개념:

- 사람이 내린 지시문(Instruction)을 모델이 이해하고 따르도록 학습
- 사전학습된 모델이 인간의 의도를 잘 못 따르기 때문에, 이를 보완하기 위한 지도학습 기반 파인튜닝(Fine-tuning) 단계

방식:

- 입력(Input): 사람이 내린 명령(Instruction)
- 출력(Output): 그에 대한 올바른 정답(Reference Answer)
- 여러 태스크(요약, 번역, 추론, QA 등)에 대해 학습시켜 다양한 지시문을 수행할 수 있도록 함.

특징:

- “이렇게 말하면 이렇게 대답하라”를 가르치는 단계
- 정답 데이터(Label)가 필수

한계:

- 데이터 수집 비용이 높음 → 사람이 직접 지시문과 정답을 만들어야 함
- 창의적 생성(creative generation) 과제에는 ‘정답’이 존재하지 않음
- 모든 오류를 동일하게 취급함 (중요한 오류, 사소한 오류 구분 불가)
- 사람이 만든 정답 자체가 최적이 아닐 수 있음 → 모델이 인간의 ‘선호’를 반영하지 못함

RLHF (Reinforcement Learning from Human Feedback)

개념:

- Instruction-tuning의 한계를 극복하기 위한 강화학습 단계
- 모델의 답변 중 어떤 것이 ‘더 인간에게 선호되는가’를 학습하여, 모델이 “인간이 좋아하는 답변”을 생성하도록 강화학습함.

과정 (3단계 구조):

- Supervised Fine-tuning (지도 미세조정): 사람이 직접 쓴 좋은 답변 예시로 모델을 1차 튜닝
- Reward Model 학습 (보상모델 학습):
  두 개의 모델 응답(A, B)을 사람이 비교 → 어떤 것이 더 좋은지 선택
  이 비교 데이터를 이용해 '선호 점수'를 예측하는 보상모델(Reward Model) 학습
- Reinforcement Learning (강화학습):
  보상모델이 부여하는 점수를 최대화하도록 모델을 학습
  → 인간이 선호할 만한 답변을 내도록 정책(policy)을 최적화

핵심 포인트:

- RLHF는 "정답"이 아닌 "선호도(Preference)"에 초점을 맞춘 학습
- ChatGPT가 친절하고 안전한 답변을 하게 만든 핵심 기술
- "Instruction-tuning + RLHF = ChatGPT"

한계점:

1. 인간 선호도 자체가 일관되지 않음
2. Reward hacking(보상 해킹) 문제 발생
   → 모델이 실제로 올바르지 않은 답을 하더라도,
   "그럴듯하고 긍정적인 말"로 높은 점수를 받으려 함
3. 환각(Hallucination) 문제
   → 실제 사실이 아닌 내용을 '있어 보이게' 생성함
4. Reward Model의 일관성 부족
   → 사람이 만든 평가기준이 제각각이기 때문
5. DPO / RLVR 등 새로운 접근
   등장 배경:

- RLHF는 강력하지만 복잡하고 비효율적
- 강화학습 과정에서의 불안정성, 보상 모델 오류 문제를 개선하기 위해 등장

DPO (Direct Preference Optimization):

- RLHF의 'RL'을 제거한 단순화된 방식
- 인간 선호 비교 데이터를 직접 이용해 바로 모델을 최적화하는 접근

장점:

- 강화학습 필요 없음 → 더 안정적이고 효율적
- 모델이 '인간이 더 선호하는 응답'을 바로 학습 가능

RLVR (Reinforcement Learning with Verifiable Rewards):

- RLHF의 환각 문제를 줄이기 위한 후속 접근
- 리워드 모델 대신 사실 검증(Verification) 단계 추가 → 생성된 응답이 실제 지식과 일치하는지 평가

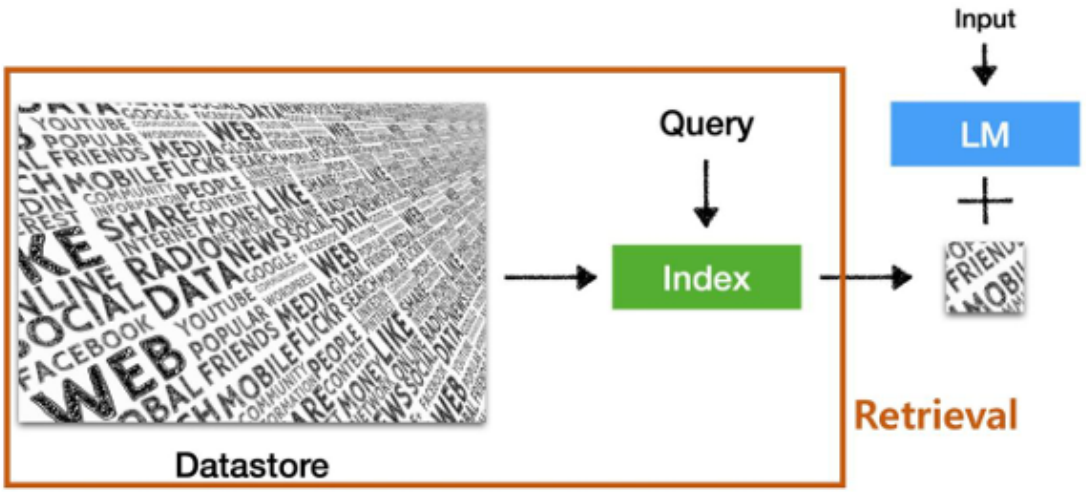
RAG

Retrieval-augmented Language Models

Information Retrieval, RAG

RAG(검색 증강 생성) 기본 개념

- RAG란? 추론 시 외부 데이터 저장소를 불러와 활용하는 언어모델
  - 검색된 문서를 활용해 더 정확하고 최신의 답변을 생성
  - 구성요소: DataStore, Query, Index, Language Model



1. Datastore: 가공되지 않은 대규모 텍스트 코퍼스
   a. 최소 수십억에서 수조 단위의 토큰으로 구성

    b. 라벨링된 데이터셋이 아님
    c. 지식베이스와 같이 구조화된 데이터가 아님
2. Query: 검색 질의, Retrieval Input
    a. 언어 모델의 질의와 같아야 하는 것은 아님
3. Index: 문서나 단락과 같은 검색 가능한 항목들을 체계적으로 정리해 더 쉽게 찾을 수 있도록 하는 것
    a. 각 정보 검색 메서드는 인덱싱 과정에서 구축된 인덱스를 활용해 쿼리와 관련 있는 정보를 식별함

- 검색(Retrieval): 데이터스토어에 있는 수 많은 정보 중, 주어진 쿼리와 가장 관련성 높은 정보를 찾아내는 과정
- **Retrieval-augmented Generation(RAG)**
    - 사용자의 질문에 답하기 위해, datastore 에서 관련 정보를 검색해와서 이를 언어 모델이 **생성 단계**에 함께 활용하는 방법

Information Retrieval(정보 검색)

- 정보 검색이란?
    - 정보검색(IR): 사용자의 질의(Query)에 맞는 정보를 대규모 데이터에서 찾아 제공하는 과정
    - 목표: 검색 질의와 가장 관련성 높은 정보 제공
- 정보 검색의 활용
    - 추천 시스템(Recommender System) - OTT, E-Commerce
- Retriever 란?
    - 사용자의 질의에 맞는 후보 문서를 저장소에서 찾아오는 **모듈**
    - 검색 증강 언어모델에서 **첫단계** 역할을 수행

      질문 ⇒ **"관련 정보 검색"** ⇒ 검색 결과 제공
    - 종류
        - Sparse Retriever (어휘적 유사도 기반): 전통적, 문서 간의 **정확한 용어 일치에** 기반

          특정 단어의 중요도를 나타내는 **가중치** 방식 (예: TF-IDF, BM-25)

          → TF(Term Frequency): **자주 등장하는 단어는 중요**

→ IDF(Inverse Document Frequency - 단어가 전체 코퍼스에서 얼마나 드물게 등장하는지): **너무 많은 문서에 등장하는 단어는 덜 중요 (ex: this, is, and ...)**

- 장점: 단순성, 효율성, 투명성
- 단점: 제한된 의미 이해 (예: bad guy, villain 이 매치가 되지 않음)

- Dense Retriever (의미적 유사도 기반) - **임베딩** 모델: dense vector(a.k.a, embeddings) 활용함 (예: DPR, Contriever, Openai-embeddings, etc)

  → Bi-encoder: 대조 학습을 통해 학습, 긍정적 문서와 가깝게 유지 부정적 문서와 멀리 유도, 두 문장을 따로 인코딩

  - 장점: 매우 빨라 대규모 데이터베이스 검색에 적합
  - 단점: 낮은 정확도

  → Cross-encoder 아키텍처: 두 개의 텍스트를 하나의 시퀀스로 결합, 두 문장을 함께 처리

  - 장점: self-attention을 통해 모든 쿼리와 문서 토큰이 완전히 상호 작용, bi-encoder 보다 더 높은 정확도
  - 단점: 모든 쿼리-문서 쌍을 개별적으로 모델에 입력해야 해 계산 비용이 크고 처리속도가 느림

- DR의 장점: 의미적 이해, 풍부한 쿼리 표현
- DR의 단점: 높은 연산 비용, 제한된 투명성(블랙박스 효과), 데이터 및 모델 의존성

Retrieval-augmented LM

- Retrieval-augmented LM란?

  추론 시 외부 데이터 저장소를 불러와 활용하는 언어모델

  RAG(Retrieval augmented Generation): 정보 검색부터 답변 생성까지의 프레임워크

- 사용하는 이유?
  - 거대 언어 모델은 모든 지식을 다 자신의 파라미터에 저장하지 못함
  - 자주 등장하지 않는 정보에 대해 큰 효과
  - 거대 언어 모델이 보유한 지식은 금세 시대에 뒤쳐지며 갱신이 어려움
  - DataStore는 쉽게 업데이트가 가능하며 확장성도 만족함

    - 거대 언어 모델의 답변은 해석과 검증이 어려움
    - 기업 내부 정보와 같은 보안 정보는 언어 모델 학습에 활용되지 않음
- R~ LM의 개선 방향, 도전 과제, 한계점
    - Context 구성 방법: 언어모델의 컨텍스트 길이를 늘려야 함
    - RAG 결과는 검색 모델 성능에 의존해 검색 노이즈에 취약함(Hallucination)
        - Training with noises로 극복

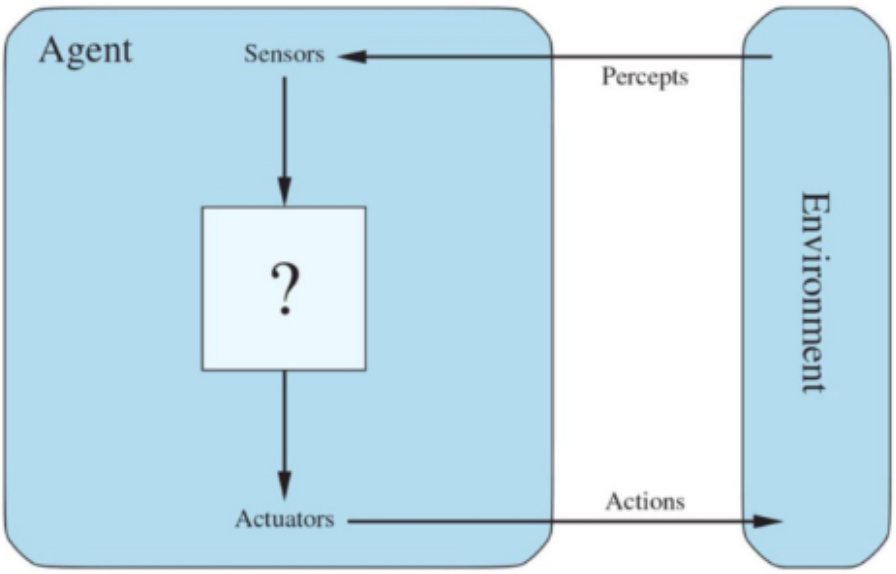
          → Noise Robustness: 외부 문서에 노이즈가 포함되어 있어도 올바른 답을 찾아내는 능력, RAG 동작 - 문서 안에 노이즈가 있어도 올바른 정보를 식별하고 정답 생성

          → Negative Rejection: 모델은 추측하지 않고 관련 정보가 없을 경우 답변 거부
    - LLM의 사전지식과 컨텍스트 간의 충돌 발생
        - Context 위에서 Grounding 학습 강화
        - Contect 없을 때, 답변 회피/거절 학습
    - 복잡한 추론 필요 & 문서가 명확한 사실에 대한 오류를 포함할 때

LLMs with Tool Usage

- Agent

  : 센서를 통해 환경을 인지하고 액추에이터를 통해 환경에 대해 액션을 통해 영향을 미치는 것으로 간주될 수 있는 모든 것



  - 성공적인 에이전트가 갖추어야 할 요건

    : 도구 사용, 추론과 계획, 환경 표현, 환경 이해, 상호작용, 의사소통

- LLM Agent

  : 거대언어모델을 핵심 구조로 삼아 환경을 이해하고 행동을 수행하는 에이전트

  - LLM-first view

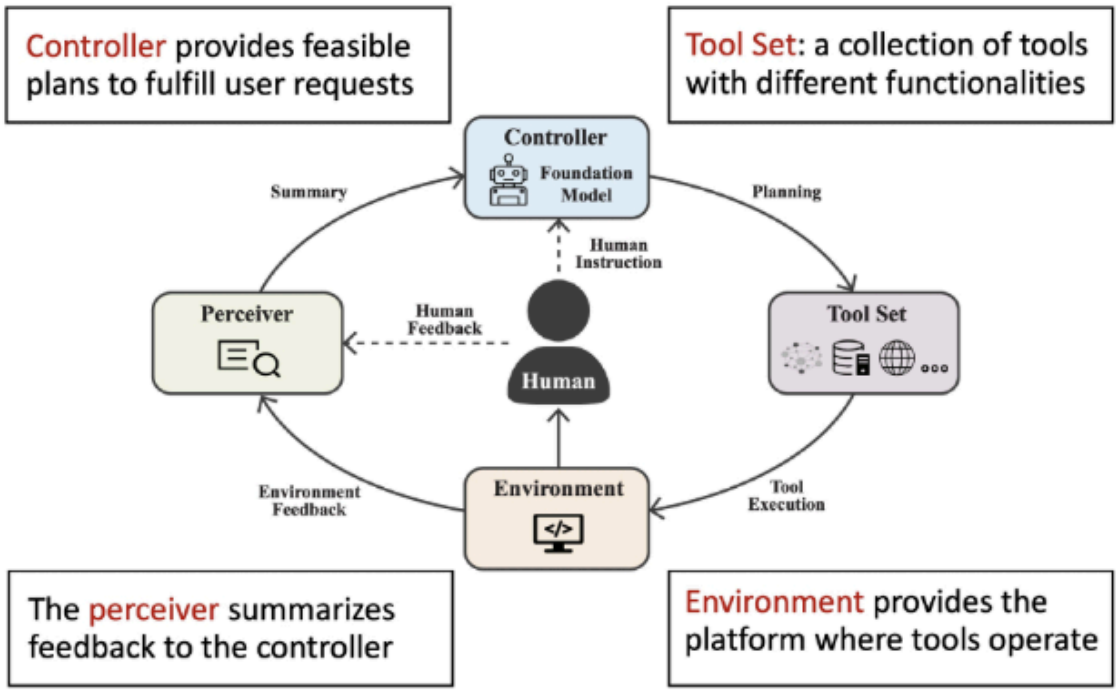
    : 기존 LLM을 활용한 시스템을 에이전트로 만들게 함

    ex) 서치 에이전트, 심리상담 에이전트, 코드 에이전트

  - Agent-first view

    : LLM을 AI 에이전트에 통합하여 언어를 활용한 추론과 의사소통을 가능하게 함

    ex) 로봇, 임바디드 에이전트



- Tool (LLM Agent를 위한 Tool)

  : 언어 모델 외부에서 실행되는 프로그램에 연결되는 함수 인터페이스를 의미

  → LLM은 함수 호출과 입력 인자를 생성함으로써 도구를 활용할 수 있음

- 도구 사용 패러다임 (Tool Use Paradigms)
  - 도구 사용 (Tool Use)

    : 두 모드 간의 전환
    - 텍스트 생성 모드
    - 도구 실행 모드
  - 도구 사용을 유도하는 방법
    - 추론 시 프롬프트 (inference-time prompting)
    - **학습 (Training: Tool Learning)**

- Tool Learning 방식

  : LLM이 외부 도구를 스스로 사용하도록 학습시키는 방식
  - 모방 학습

: 인간의 도구 사용 행동 데이터를 기록함으로써 언어 모델이 인간의 행동을 모방하도록 학습

→ 가장 간단하고 직관적인 방식

- OPenAI : WebGPT (2022)

  : 검색 엔진을 사용하기 위해 인간의 행동 모방

  : 지도 학습 + 강화 학습

  → 단 6000개 데이터만을 가지고 학습

  → 장문의 질의응답에 뛰어난 성능을 보이며 인간보다 좋은 성능을 보이기도 함

- Meta: Toolformer (2023)

  : 모델 스스로 학습 데이터 생성

  : 지도 학습

  1. API 호출 샘플링 : LLM이 기존 데이터셋에서 문맥을 보고 API 호출 후보 생성
  2. API 실행 : 생성된 API 호출을 실제로 실행하여 응답을 얻음
  3. API 호출 필터링 : API 호출이 문맥에 유용한지 평가 및 필터링 진

- ToolLLM (2023)

  : 기존 연구의 한계점인 툴의 다양성과 범용성을 타겟팅 한 연구로 데이터셋 수집

  1. API 수집
  2. 수집한 API를 기반하여 명령문 생성
  3. 정답 어노테이션 & 필터링

⇒ 최근에는 멀티모달 대규모 언어모델을 기반으로 도구를 정의하고 활용하는 연구인 멀티 모달 툴 러닝 방식 + 지도학습을 넘어 강화학습을 도입하는 연구

- MCP (Model Context Protocol)

  : 언어 모델이 외부 툴과 상호작용하기 위한 표준화된 방식으로 정의한 프로토콜로 툴 호출, 응답 전달, 컨택스트 공유를 하나의 공통 규격으로 처리

  - 아키텍처
    - MCP Host

: 하나 또는 여러 개의 MCP 클라이언트를 조정하고 관리하는 AI 애플리케이션

- MCP Client

  : MCP 서버와의 연결을 유지함 MCP 호스트가 사용할 수 있도록 MCP 서버로부터 컨택스트를 가져오는 구성요소

- MCP Server

  : MCP 클라이언트에게 컨텍스트를 제공하는 프로그램

- 계층
  - 데이터 계층

    : 클라이언트-서버 통신을 위한 JSON-RPC 기반 프로토콜

    → 라이프 사이클 관리, 툴, 리소스, 프롬프트와 같은 핵심 요소들 포

  - 전송 계층

    : 클라이언트 서버 간 데이터 교환을 가능하게 하는 통신 메커니즘과 채널을 정의

    → 전송 방식에 특화된 연결 수립, 메세지 프레이밍, 인증이 포함

  ⇒ 개념적으로 데이터 계층은 내부 계층, 전송 계층은 외부 계

- 장점
  - 표준화 : 모든 모델, 툴이 동일한 호출 규격 사용
  - 확장성 : 새로운 툴 쉽게 추가 가능
  - 호환성 : 모델, 플랫폼에 상관없이 같은 툴 호출 가능
  - 재사용성 : 한번 정의한 툴을 여러 모델에서 활용 가능
  - 투명성 : 호출 과정이 명확하게 기록, 검증됨
- 예시 코드

  : FastMCP 라이브러리 사

```
# calc_server.py
from fastmcp import FastMCP

mcp = FastMCP("calc")                    # 1 서버 인스턴스

@mcp.tool()                          # 2 MCP 를 선언
def add(a: int, b: int) → int:       # 타입천드→JS0N-Schema 자동 변
```

```
환
"""Add two numbers"""
return a + b

if _name_mm "_main_":
mcp. run()                    # @ stdio·HTTP·WS 등 자동 선택
```

⇒ MCP는 현재 학계에서 모두 표준으로 확립되고 있음

Web과 AI 개발을 구분하게 되어 양쪽 생태계 모두에서 표준적이고 호환 가능한 개발이 가능해짐

Recap: LLM Agent 이란?

- 거대언어모델(LLM)을 핵심 구조(backbone)로 삼아 환경을 이해하고 행동을 수행하는 에이전트

| Recap: 성공적인 에이전트가 갖추어야 할 요건들

- 도구 사용(Tool use)
- 추론과 계획(Reasoning and Planning)
- 환경 표현(Environment Representation)
- 환경 이해(Environment Understanding)
- 상호작용/의사소통(Interation/Communication)

1) Environment Representation & Understanding

| 에이전트가 환경을 이해하기 위해 필요한 것

- 환경에 접근하기 위한 툴(Tool)
- 환경의 표현(Representation)
- 환경을 이해/탐색하기 위한 방법론들

복잡한 환경은 어떻게 이해할 수 있을까?

- 모델은 자신이 상호작용하는 환경에 대해 모든 것을 알고 있지 않음

- 일부 지식은 LLM 파라미터 안에 포함되어 있음
- Ex) 코딩 지식, 자주 사용하는 웹사이트 탐색 방법 등
- 다른 지식은 실시간으로 환경과의 상호작용을 통해 발견해야 함

| 복잡한 환경에 대한 이해

- 환경 탐색 (Environment Exploration)
- 모델이 환경을 탐색할 때 보상(reward)을 부여하여 학습을 유도 → 호기심(curiosity) 기반 보상
    - Ex) 강화학습에서 예측 불가능한 상태 공간(state space)에 진입할 경우 보상을 증가
- 탐색 기반 궤적 기억 (Exploration-based Trajectory Memorization)

. BAGEL (Murtey et al. 2024)

- 명령어를 샘플링하여 실행한 뒤, 더 정확한 새로운 명령어로 궤적(trajectory)을 재라벨링
- LM-agent가 환경을 탐색하고, LM-labeler가 이를 바탕으로 지시를 다시 정제
- 기존 데이터에 의존하지 않고, 탐색과 자기 교정(self-correction)을 통해 테이터 생성
- 에이전트의 일반화 성능과 환경 적응력을 향상

2) Reasoning & Planning

| Controller: Planning 종류

- 국소적 계획 (Local Planning)
- 한 단계씩(step by step) 계획을 세움
- 매 스텝마다 사용할 하나의 툴(tool) 결정
- 단순하고 직관적이나, 장기 의존성 문제 발생
- ReACT (Yao et al. 2023)
- 전역적 계획 (Global Planning)
- 실행 가능한 전체 계획 경로(planning path)를 한 번에 생성
- 여러 개의 툴을 조합하여 시퀀스 형태로 결정
- 효율적이나, 복잡한 환경에서는 실패 가능

- Plan-and-Solve Prompting (Wang et al, 2023)

| 오류 식별과 회복 (Error identification and Recovery)

- 에이전트는 에러/실수에서 회복할 방법이 필요
- Reflexion (Shinn et al., 2023)

| 계획 재검토 (Revisiting Plans)

. CoAct (Hou et al. 2024)

3) Langchain

Langchain 이란?

- LLM 기반 애플리케이션을 빠르게 개발할 수 있는 오픈 소스 프레임워크
- LLM을 다양한 데이터/툴과 연결하여 강력한 애플리케이션 개발 가능
- 연구와 산업 현장에서 빠르게 표준으로 자리잡음

Langchain 특징

- 다양한 LLM provider(OpenAI, Anthropic, Google, etc) 와 통합하여 모델/회사별 API 차이를 공통 인

터페이스로 관리 가능

- Prompt, Memory, Tools 와 같은 컴포넌트들이 모듈화 되어 있어 재사용성과 확장성 확보
- LangGraph 기반으로 복잡한 워크플로우를 시각적으로 설계 및 관리 가능

Langchain 주요 컴포넌트

- Prompt Templates: 프롬프트를 구조화
- Chains: 여러 단계를 연결한 워크 플로우
- Agents: 동적으로 툴 선택 및 실행
- Memory: 대화 히스토리와 상태 유지
- Tools: 외부 API, DB, 계산기 등 연결
- Etc

| Langchain 튜토리얼 (https://python.langchain.com/docs/tutorials/)

- RAG w/Langchain
- Agent w/Langchain
- MCP w/Langchain

<9주차>

컴퓨터의 수 체계

2의 거듭제곱 (2의 9승까지 외우기~)

2의 0~9승 : 1 2 4 8 16 32 64 128 256 512

2^10=1024(대충 1000), 2^20=1024^2=1000^2=10^6

16진수

Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

부호 있는 수 체계

- 부호-크기 표현 방식
  - 최상위 비트(MSB)를 부호 비트로 사용해서 0이면 양수, 1이면 음수를 표현, 나머지 비트는 절댓값을 표현하는 방식
  - 최상위 자리에 SIGN BIT(부호) 할당 (양수는 0, 음수는 1)
  - Ex) +6=0110, -6=1110, 0은 1000이나 0000 2가지 방법으로 표현
  - 문제점 : 기존 덧셈에서 문제 발생

$-6+6$ 예시

$$\begin{array}{r} 1110 \\ +\ 0110 \\ \hline 10100 \end{array}$$

- 2의 보수 표현 방식
  - 음수를 표현할 때 해당 수의 절댓값을 이진수로 쓰고, 모든 비트(1의 보수)를 반전한 뒤 1을 더하는 방식
  - 부호-크기와 동일하게 최상위 자리에서는 부호가 표시됨
  - 가장 큰 4bit : 0111, 가장 작은 4bit : 1000
  - 이 방식을 활용하면 기존 덧셈이 정상적으로 동작

$6+(-6)$

$$\begin{array}{r} 111 \\ 0110 \\ +\ 1010 \\ \hline 10000 \end{array}$$

• $-2+3$

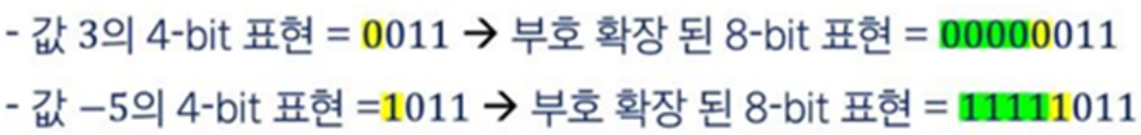
$$\begin{array}{r} 111 \\ 1110 \\ +\ 0011 \\ \hline 10001 \end{array}$$

$3_{10} = 0011_2$ 부호반전 예시

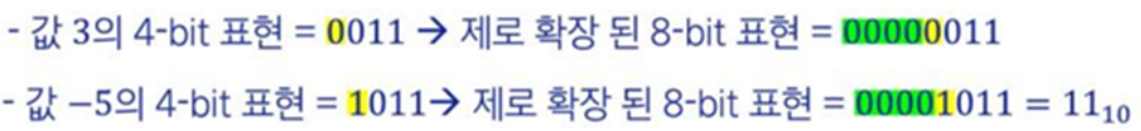
$$\begin{array}{r} 1100 \\ +\ 0001 \\ \hline 1101 \end{array} = -3_{10}$$

비트 폭 확장

- N-bit 수를 M-bit 수로 확장(M>N)
  - 부호 확장 : 최상위 비트를 복사해 상위 비트 채우기(값을 유지)



  - 제로 확장 : 상위 비트를 모두 0으로 채움(패턴을 유지)

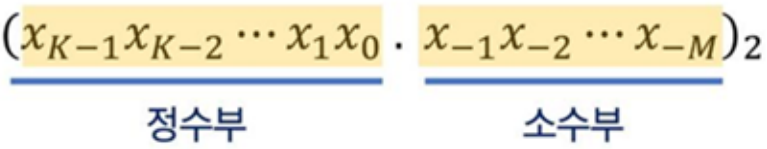


int와 long의 역사

- int
  - 가장 효율적으로 처리될 수 있는 정수 타입을 목표로 정의
  - 컴퓨터 구조에 따라 다른 크기였음(16-bit, 32-bit 등)
  - 64-bit 등장 후 혼란을 막고자 int는 32-bit 길이로 고정
  - 조금 더 엄밀하게, C에서 int는 최소 2B 범위 포함한 정수형, 32-bit 컴퓨터까지는 구조에 따라 크기가 달라질 수 있었으나 이후에는 4B로 길이 고정
- long
  - 이로 인해 long이 애매해짐, 8B로 일단은 정의를 명확히 함
  - 단, long은 컴퓨터나 OS 구조에 따라 4B 혹은 8B로 사용됨

고정소수점 체계의 계산

- 정수부와 소수부를 나눠서 표현
- K-bit 정수부, m-bit 소수부로 N-bit를 표현
- 정수부와 소수부 사이 이진 소수점을 포함



- 이진 소수점의 위치는 실제 저장되는 것이 아님 -> HW는 이를 모두 N-bit의 정수로 판단함
- 덧/뺄셈은 이전과 동일하게 동작, 곱/나눗셈은 scailing factor 체크HW는 이 계산이 정수 입력이라 가정하고 동작하므로, 사용자가 어떠한 의도를 가지고 있는지가 중요함

$$aX_I = 2^{-4} \times 24_{10} = 0001.1000_2 = 1.5_{10} = X_{FX}$$
$$aY_I = 2^{-4} \times 8_{10} = 0000.1000_2 = 0.5_{10} = Y_{FX}$$

$X_I \times Y_I = 192_{10} = 11000000_2 \rightarrow$ 실제 HW 수행

$a(X_I \times Y_I) = 2^{-4} \times 192_{10} = 1100.0000_2 = 12_{10}$
$= 2^4 \times (X_{FX} \times Y_{FX})$
동일 고정소수점 해석

$X_I \times Y_I \times a = 192_{10} \times 2^{-4} = 00001100_2 \rightarrow$ 실제 HW 수행 (scaling factor 고려)

$a(X_I \times Y_I \times a) = 2^{-4} \times 12_{10} = 0000.1100_2 = 0.75_{10} = X_{FX} \times Y_{FX}$ 동일 고정소수점 해석

$$aX_I = 2^{-4} \times 24_{10} = 0001.1000_2 = 1.5_{10} = X_{FX}$$
$$aY_I = 2^{-4} \times 8_{10} = 0000.1000_2 = 0.5_{10} = Y_{FX}$$

$X_I \div Y_I = 3_{10} = 00000011_2 \rightarrow$ 실제 HW 수행

$a(X_I \div Y_I) = 2^{-4} \times 3_{10} = 0000.0011_2 = 0.1875_{10} = 2^{-4} \times (X_{FX} \div Y_{FX})$ 동일 고정소수점 해석

$(X_I \div Y_I) \div a = 3_{10} \div 2^{-4} = 00110000_2 \rightarrow$ 실제 HW 수행 (scaling factor 고려)

$a((X_I \div Y_I) \div a) = 2^{-4} \times 48_{10} = 0011.0000_2 = 3_{10} = X_{FX} \div Y_{FX}$ 동일 고정소수점 해석

부동소수점 수 체계

- 매우 크거나 작은 수를 효과적으로 표현 가능
- 기존 방식은 너무 많은 비트를 요구했음
- 이를 개선하고자, **지수를 활용**해 이러한 수들을 문자로 표현하고, 대신 유효숫자의 개념을 도입함

$$1.23 \times 10^{3}$$

지수

가수 밑

- 가수는 반드시 1 이상 10 미만의 수로 표현
- 유효숫자의 처리 기법

- 반올림, 올림, 버림, 절삭
- 오류 최소화를 위한 다양한 rounding 기법 존재

이진 부동소수점 체계

- 지수적 표현을 이진 수 체계에도 적용

$$(-1)^S \cdot M \cdot \beta^E$$

- 가수는 두 가지 표현 중 선택, 밑은 2의 거듭제곱의 형태를 선택함

- [0, 1) 범위의 순수한 소수 표현: 0. ■■■■
- [1, 2) 범위의 정규화된 표현: 1. ■■■■

- Ex) 연산 예제 확인하기

예시: 어떤 연산의 결과가 $01.10100 \cdot 2^{100}$ 이었다면? → 사전 정의된 [0, 1) 가수 범위 초과 (최대인 M_{max} 초과)

- 유효한 표현을 위한 수정작업 (correction) 필요 → $0.11010 \cdot 2^{101}$
- 밑이 2가 아니라면? $\beta = 2^k$상황에서는 지수를 1 증가시키기 위해선 가수를 k 위치 이동함
  -> $\beta = 4 = 2^2$ 상황에서 수정된 유효한 표현 = $01.10100 \cdot 2^{100} = 01.10100 \cdot 4^{010} = 00.01101 \cdot 4^{011}$

정규화된 이진 부동소수점 수 체계

- 부동소수점은 unique한 표현이 아님

$$0.11010 \cdot 2^{101} = 0.01101 \cdot 2^{100} = 0.001101 \cdot 2^{99} \ \ldots$$

- 일반적으로는 0으로 시작하는 패턴을 피해 유효숫자를 확보하고자 하는 정규화를 진행함

$\beta = 2^k$경우에는 가수는 k 비트 단위로 이동할 수 있음 → 첫 k 비트가 모두 0이 아니면 정규화된 것으로 판단

- 예시) (k = 1) 일때 정규화

$$0.01101 \cdot 2^{100} \Rightarrow 0.11010 \cdot 2^{101}$$

- 예시) (k = 4) 일때 정규화

$$0.00000110 \cdot 16^{101} \Rightarrow 0.01100000 \cdot 16^{100}$$

- 한계점 : 정규화된 표현은 0을 포함하지 않음, 이를 위해 0을 표현하는 특수 패턴을 정의 (M=0, E=0)

바이어스(편향) 지수

- 부동소수점 표현에서 지수를 음수와 양수 모두 표현해야 하는 문제를 해결하기 위해, 실제 지수 값에 일정한 상수(bias)를 더하여 항상 양수로 기록하는 방식

S는 부호 비트(1bit), E는 지수 비트(e bit), M은 부호 없는 가수(s bit)

- 장점: 지수를 0/양수 영역으로 이동시켜 기록된 지수를 부호-없는 수 표현으로 해석하게 만듬
- 바이어스 지수는 실제 지수에 일정한 상수(bias)를 더해 양수로 저장하는 방식이며, 이는 음수를 없애고 지수 비교를 단순화하기 위한 목적을 가짐

이진 부동소수점의 표현 범위

- 부동소수점 수는 양수 영역(F^+)과 음수 영역(F^-)을 따로 생각할 수 있으며, 음수는 부호 비트(S)만 1로 바꿔서 표현
- 정규화된 가수를 사용한다면 상한초과는 지수의 변화에서만 발생
- 하한미달: 너무 작은 수를 표현하려다 0에 근접한 값이 되는 경우 -> 연산을 멈추지 않고 0에 근사시켜 계산을 이어나감 (언더플로 처리)
- 이진 부동소수점의 표현 범위는 지수부(E)가 결정하며, 가수부(M)는 정밀도(소수점 이하 자리수)를 담당한다.
- 연산 결과가 지수 표현 범위를 벗어나면 상한초과(overflow)나 하한미달(underflow)이 발생한다.

- 오버플로는 일반적으로 무한대(infinity)로 처리되고, 언더플로는 0으로 근사시켜 표현한다.
- 따라서 부동소수점 연산에서 중요한 점은 **지수 표현 범위를 벗어나지 않도록 관리하는 것**이다.

IEEE 754 단정밀도 부동소수점 포맷

- 32비트로 구성되며, 1비트 부호, 8비트 지수, 23비트 가수를 사용
- 지수는 bias=127로 편향 저장되고, 가수는xxx 형태의 hidden bit를 사용해 정규화한다.
- E=0은 비정규화 수, E=255는 ∞ 또는 NaN을 의미하며, 정규화된 수의 범위는 약 $\pm 3.4 \times 10^{38}$까지 표현 가능

연산과 복잡도 P.61 PPT 참고

고정소수점 수 체계의 연산

- 4비트 부호 없는 정수의 덧셈

$7_{(10)} = 0111_{(2)} 5_{(10)} = 0101_{(2)}$

$$0111_2 + 0101_2 = 1100_2$$

- 상한초과: 정해진 비트 수로 표현 가능한 범위를 초과하면 발생

$$0111_2(7_{10}) + 1100_2(12_{10}) = 10011_2$$

4비트로 표현할 수 있는 부분은 오른쪽 4자리(0011)만 남고, 왼쪽의 자리올림(carry out)은 저장되지 않는다. 따라서 실제 결과는 $0011_2 = 3_{(10)}$이 되어 **정상적인 결과가 나오지 않는다.**

이것이 **overflow** 상황이다.

이진고정소수점 수 체계 정수의 덧셈/뺄셈

- 이진 고정소수점(fixed-point) 수 체계는 소수점의 위치가 고정된 형태로,정수를 포함한 실수를 표현할 때 사용

$$
\begin{aligned}
A &= 1.50_{10} = 01.100_2 \\
+B &= 0.75_{10} = 000.11_2 \\
\hline
&001.100_2 \\
&+\ 000.110_2 \\
&010.010_2 = 2.25_{10}
\end{aligned}
$$

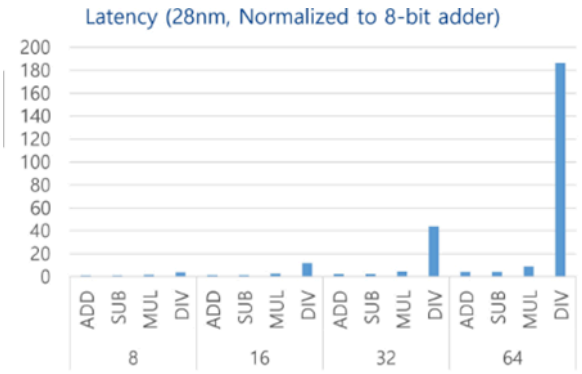
4비트 부호 있는 정수의 표현과 2의 보수

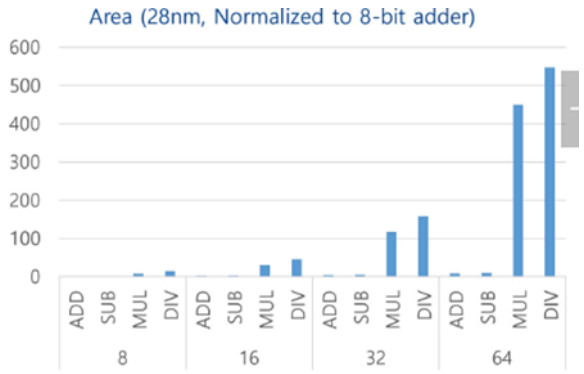
- 부호 있는 정수를 표현하기 위해서는 2의 보수 방식을 사용한다.

$$
\begin{aligned}
(+)7_{10} &= \ 0111_2 \\
-(+)5_{10} &= \ 0101_2 \\
\hline
(+)7_{10} &= \ 0111_2 \\
+(-)5_{10} &= \ 1011_2 \\
\hline
(+)2_{10} &= 10010_2
\end{aligned}
$$

고정소수점 연산의 복잡도

덧셈~뺄셈 << 곱셈 << 나눗셈

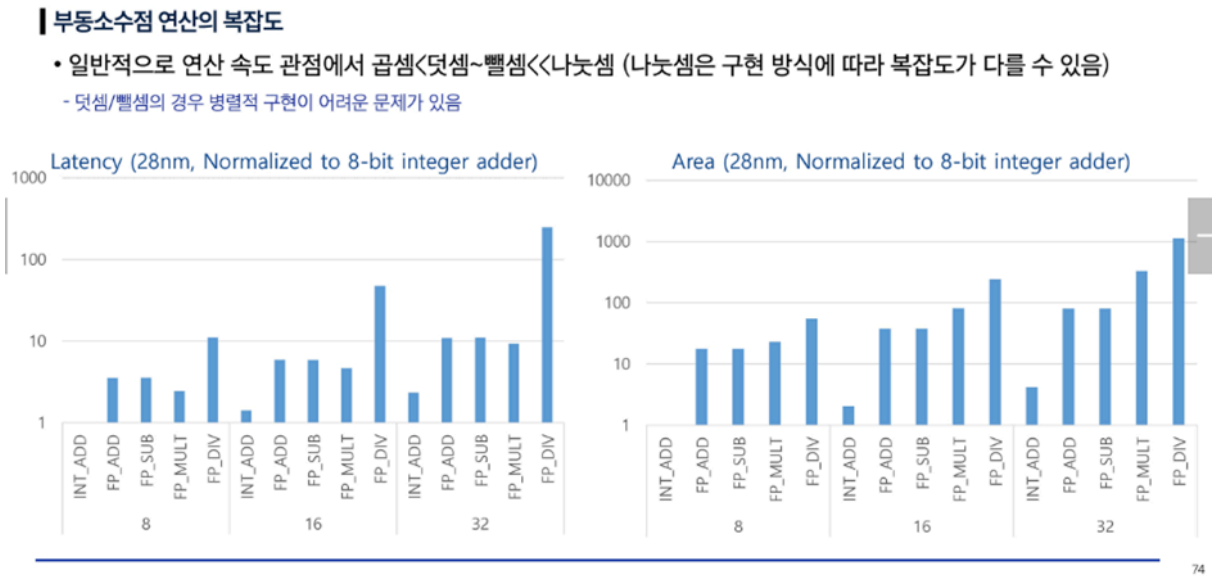




부동소수점 수 체계의 연산

- 부동소수점 연산은 고정소수점보다 복잡하며, 연산 전후에 **지수 정렬과 가수 정규화**가 반드시 필요

부동소수점 연산의 복잡도



곱셈 < 덧셈~뺄셈 << 나눗셈 (나눗셈은 구현 방식에 따라 복잡도가 다를 수 있음)

모델 경량화의 필요성

- AI/ML은 한계도 존재하지만 이미 많은 영역에서 평균적으로 사람보다 우수
  - 어두운 면
    - 모델 크기 문제
    - 컴퓨팅 리소스 문제
    - All you need is money
    - 계속해서 커질 것으로 예상되는 모델
- 모델 경량화(Model Compression) - 추론/서비스 영역에서 필수
  - GPU와 궁합이 좋은 기법
    - Low-rank adaption(LoRA), quantized low-rank adaption(QLoRA)
    - Knowledge distillation
    - Costratined quantization

        - Structured pruning
    - GPU와 궁합이 좋지 않은 기법(전용 가속기로 효과 극대화 필요(
        - Aggressive quantization, mixed-precision quantization
        - Unstructured/partially-structured pruning

Quantization/Pruning/Distillation

양자화(Quantization)

- 연산/메모리 부하를 줄이는 가장 직관적인 방법: 정밀도 축소
    - Clipping range 에 따른 scaling factor 정하기
- QAT vs PTQ
    - **QAT(Quantization-aware training)**

      학습 단계에서 forward pass quantization을 함께 수행(학습 데이터 필요)

      동일 accuracy 기준 압축률 높일 수 있음, 학습 복잡도가 엄청나게 상승 (retrain/fine-tuning접근)

      상대적으로 CNN등 오리지널 학습 복잡도/데이터가 크지 않은 경우 쉽게 적용 가능
    - **PTQ(Post-training quantization)**

      모델, 학습데이터 모두 너무 큰 경우에는 QAT 사용 불가

      사전 학습된 모델 바탕으로 양자화 바로 수행

      적절한 calibration 데이터를 사용해 경량화 효율 높임
- Weight-activation vs Weight-only: 모델 사이즈에 얼마나 집중? De-quantization 얼마나 허락?
- Symmetric vs Asymmetric: activation 및 weight 분포 어디까지 반영? 하드웨어 연산기 수준 지원?
- Integer vs Floating-point: 정수 양자화 방식? (예: BCQ 같은 방식, FP 표현, IEEE 754 표준 활용, 구글 BF16 - IEEE FP32의 truncation 버전, TensorFloat-32 표현, FP8 등)
- Mixed-precision quantization: 레이어별로 최적화된 양자화 적용

가지치기(Pruning)

- **불필요한 기여도가 적은 weight(가중치)를 제거 → 모델 크기 줄어듦**

  살아남은 가중치의 위치를 기록하기 위한 추가정보 필요(indexing data, meta data 등)

- 가지치기 규칙?
  - 가지치기 대상의 순서를 어떻게 정할 것인가?

    → magnitude-based pruning

    → pruning ratio(압축률) 대비 높은 accuracy를 유지하는 다양한 기법들 존재 (예: momentum-based pruning)

- Unstructured vs Structured
  - 추론(inference) 효율과 accuracy 방어 사이의 선택
  - channel/layer 수준의 극단적인 structured pruning 도 활용 가능
- Indexing data format
  - 일반적으로 CSR(compressed sparse row) → pruning ratio 가 극단적으로 높아야 GPU가속
  - 그 외 on-off encoding, XOR-gate compression 등
- Fine-tuning/retraining 가능 여부
  - 양자화 경우와 비슷한 양상

    → 작은 모델과 학습데이터에서는 training 단계부터 도입 가능(CNN 등)

    - CNN 기반 이미지 처리는 80% 이상도 pruning 가능(1~2% accuracy drop)

    → LLM 같이 큰 모델의 경우 post-training pruning이 현실적 옵션

    - 언어 처리를 위한 LLM 계열은 50% 내외 수준
- GPU 호환 여부
  - 양자화와 비슷하게 GPU가 감당할 수 없는 pruning 패턴이 형성되는 경우 존재
  - AI/ML 모델 특성상 0이 많이 포함됨
  - 전용 가속기 및 NPU 설계단계에서 sparsity-aware 구조를 많이 고민

지식 증류(Knowledge Distillation)

- 작은 모델을 만드는 효과적 방법

    - Teacher: 똑똑하지만 매우 큰 모델
    - Student: 동일한 문제를 푸는 작은 모델, Edge에 적용 가능한 매우 쓸만한 모델
    - 훈련 방법: 두 가지 loss를 정의 Total loss = student loss + distillation loss

      → Student loss: Ground truth 의 student 추론의 차이

      → Distillation loss: 동일 데이터에 대해서 teacher 와 student의 결과 차이
- 보다 효과적인 지식 전달/증류(Knowledge transfer/distillation)
    - 조교 활용 Assistant model
        - NOKD: 학생을 바로 학습 No Knowledge Distillation
        - BLKD: 선생님 혼자 학생을 교육 Baseline Knowledge Distillation
        - TAKD: 조교의 개입 Teacher Assistant Knowledge Distillation
- LLM distillation
    - 상대적으로 작은 LLM 학습에서 기존 엄청나게 큰 LLM 활용
    - 학습에 사용해야 하는 데이터가 너무 크면 여전히 문제가 됨
        - 극복 방안 예: 합성 데이터 활용 - 프롬프트 데이터로 선생님 모델의 응답을 뽑고 학생을 fine tuning

경량화 요약

- 모델 경량화의 필요성과 기법
    - 대형 모델의 크기, 비용, 자원의 문제로 경량화는 필수
    1. Quantization: 정밀도 축소
    2. Pruning: 불필요한 가중치 제거
    3. Distillation: 큰 모델 지식 → 작은 모델로 전수

파인 튜닝 Fine Tuning

1. **Full fine tuning: 데이터가 많아야 함**
2. **Partial fine tuning: 여러가지 기술**
3. **PEFT(Parameter Efficient Fine Tuning) - LoRA**

이미 학습된 원본 모델을 가져와 특정 목적이나 데이터에 맞게 조정하는 과정

- 모델 경량화 관점에서 파인 튜닝
  - 원본 모델의 크기를 줄이는 과정에서 떨어진 성능을 복원하는 데 사용됨
  - 비슷한 예: 양자화 인지훈련(QAT)처럼, 학습 초기부터 경량화 개념을 적용해 성능 손실을 최소화 하는 방식
- 모델 서비스 관점(LLM)
  - 일반적으로 능력이 뛰어난 원본 모델을 특정 영역/도메인에 특화시키는 역할
- 튜닝 방법론
  - In-context 학습: 원본 모델을 두고 원하는 방향으로 동작을 튜닝하기 위한 프롬프트 고민
  - **Full fine tuning**: 특정 도메인의 데이터를 사용해 원본 모델의 모든 weight를 fine tuning
    - 장점

      → 처음부터 새로 학습하는 것에 비해 적은 데이터만 활용해도 도메인에 특화 가능

      → 도메인 특화 관점에서 accuracy 이득을 기대할 수 있음
    - 단점

      → 원본 모델이 일반적으로 크기에 모든 weight 를 갱신하는 학습(tuning) 비용이 부담

      → 메모리 사용량 관점에서 사실상 처음부터 학습하는 수준이 요구될 수 있음

      → 원본 모델의 weight 자체가 변화하는 과정에서 기존 학습 정보의 망각이 일어날 수 있음
- **파라미터 효율적 파인 튜닝(Parameter Efficient Fine Tuning, PEFT)**
  - LLM을 도메인 특화 시키는 과정의 복잡도 개선
    - 원본 모델의 전체 가중치를 전부 갱신하지 않고서도 튜닝 작업을 지원
    - 기본적으로 모델을 특화 시키기 위한 추가 가중치를 어디에/어떻게 추가할 것인가의 문제
  - Adapter layer
    - 입력원본 모델 내부에 새로운 layer 추가 보통 MLP 형태
    - 목표로 하는 특화 데이터를 활용해 adapter layer만 학습 원본 가중치는 고정
    - full tuning 보다는 적지만, 상당한 추가 가중치 필요 → 높은 표현력 제공

        - 원본 모델의 구조가 바뀌고 adapter layer 의 메모리/연산 오버헤드가 크다는 단점
    - Prompt tuning
        - 입력 임베딩 앞/중간에 가능한 가짜 토큰(수도 토큰) 추가
        - 원본 모델의 구조를 전혀 건들지 않고 아주 적은 추가 가중치로 입력을 조정해 특정 동작 유도
        - 매우 가볍고 배포 쉬움
        - 표현력의 한계로 성능 제한
    - **Low-Rank Adaptation(LoRA)**
        - 특정 가중치 매트릭스(base weight)에 low-rank update를 추가
        - 원본 가중치를 고정하고 추가되는 low-rank weight만 학습
        - 높은 표현력에 adapter layer 보다 훨씬 적은 추가 가중치
    - **Quantized Low-Rank Adaptation(QLoRA)**
        - 경량화 + LoRA

          원본 모델을 강하게 양자화 → 범용 성능의 하락 존재 (예: 4bit normal format, NF4)

          높은 해상도 유지하면서 학습 (예: BF16 등)

          전체 모델의 복잡도 대폭 축소, 목표로 하는 task에 대해서 높은 성능 달성
- LoRA/QLoRA 활용하기
    - 최신 파인튜닝 라이브러리 활용: Unsloth, Axolotl 등
    - 구현에 요구되는 파라미터 이해하기 rank, alpha, target_modules 등
    - 주요 실습 포인트
        - Base 모델 및 토크나이저 로딩
        - LoRA 설정 및 적용
        - 데이터셋 준비 및 전처리
        - 모델 학습 및 저장
        - 파인튜닝된 모델로 추론