4-1 LangChain 서비스 개발

목차

1. Post-Training (Instruction-tuning, RLHF)

학습 목표

- 거대 언어 모델의 학습 패러다임에 대한 이해를 한다.
- Instruction-tuning에 대해 설명할 수 있다.
- RLHF의 개념을 이해하고 Instruction-tuning 과의 차이를 이해한다.
- RLHF 이후의 post-training 접근들을 이해한다.

학습시작(overview)

- 거대 언어 모델의 학습 패러다임이 어떻게 되는가?
    - Pre-training(사전 학습)과 Post-traning(사후 학습)의 개념과 차이
- 언어 모델이 어떻게 사용자의 질문에 응답을 할 수 있게 되었는가
    - Instruction-tuning에 대한 개념 이해
- 언어 모델이 어떻게 사용자가 선호하는 답변을 내도록 학습이 되었는가
    - Instruction-tuning의 한계와 RLHF의 개념에 대한 이해
- RLHF 이후에 post-training 접근들은 어떤 게 있는가?
    - RLHF의 한계점과 이후 접근들에 대한 소개(DPO, RLVR)

---

1. Pre-training vs Post-training

Pre-training (사전 학습) : 방대한 인터넷 텍스트를 통해 언어와 지식을 배우는 단계

- 방대한 인터넷 텍스트 데이터를 이용한 Self-supervised learning 을 통해 언어 패턴, 지식 등을 배운다.

- 학습 목표 : 다음 단어 예측(Next Token Prediction)
  - ex) 내일은 비가 ___ → 온다의 확률을 가장 높게 만들도록 파라미터 업데이트
  - Decoder : Transformer, LSTM ... 등등
- 특징
  - 다음 단어를 예측하는 데에 강점을 보이지만 질문에 대한 대답을 하지 않음

> 💡 사전 학습 후 거대 언어 모델은 유저의 의도와 일치하지 않았다.
>
> → Fine-Tuning을 진행하게 됨 (사후 학습의 등장)

Post-training (사후 학습): 사람이 원하는 방식으로 대화하고, 안전하고 유용하게 만드는 단계

- 유저의 의도를 파악하고 원하는 답변을 모델이 응답하도록 사후 학습을 진행한다.
- 대표 기법 : Instruction-tuning, RLHF(Reinforcement Learning from Human Feedback), DPO(Direct Preference Optimization), RLVR(Reinforcement Learning with Verifiable Reward)

2. Instruction-tuning

Instruction-tuning

- 언어 모델이 사람이 내린 지시문을 따르도록 학습하는 단계
- 요구사항 : 정답 레이블
- 목적 : 다양한 태스크를 풀 수 있도록 적응하는것
- 구현 방법
  - 다양한 태스크에서 (지시문, 응답) 쌍을 모아서 언어 모델을 파인튜닝한다.
  - 새로운 (모델이 학습하지 않은) 태스크에서 평가를 진행한다.
- 특징
  - 더 많은 태스크를 가진 데이터 학습
  - 데이터와 모델의 크기가 핵심이다.

> 💡 질문 : 이러한 데이터로 학습한 거대 언어 모델은 어떻게 평가해야 하는가?
>
> → MMLU의 등장
>
> MMLU
>
> - Massive Multitask Language Understanding
> - 57개의 지식을 요구하는 태스크에서 언어모델의 성능을 평가하는 벤치마크
> - K-MMLU 도 존재

- 발전 사례 : Alpaca 모델
  - Meta의 LLaMA 7B 모델을 기반으로 52K개의 instruction-following 데이터를 이용해 파인튜닝한 모델의 등장
  - Instruction-following 데이터는 GPT-3을 이용해 자동생성한 데이터를 사용했다.
  - → 결과적으로 많은 데이터보다 좋은 데이터가 더 중요하다는 사실을 알게 됨.

> 💡 Instruction-tuning의 한계
>
> - 정답 데이터를 수집하는 것이 비싸다
> - 언어모델의 목표(LM objective)와 인간의 선호를 만족시키는 것 사이에 여전히 불일치가 존재한다.
>
> → RLHF의 등장

3. RLHF

인간의 선호를 반영한 최적화(Optimizing for human preference)

- 언어모델을 학습할 때, 어떤 지시문 x와 언어 모델의 응답 y가 있을 때, 그 응답이 인간의 선호를 만족시키도록 학습을 유도함
- 응답에 대해서 인간의 선호를 만족할 때마다 보상을 주게 하여, 더 큰 보상을 얻을 수 있도록 학습시킨다.
- RLHF의 학습 과정
  1. Instruction-tuning 을 한다.
  2. 같은 질문에 대해 모델이 낸 여러 개의 답변을 사람이 평가한다.
     - 좋은 답변일 수록 높은 점수를 주는 보상 모델(Reward Model)을 만든다.
  3. 언어 모델이 답변을 생성하면, Reward Model이 품질 점수를 매긴다. (강화학습)
     - 이제 모델은 보상을 최대화하도록 학습한다.
- 특징
  - RLHF 의 성능 > Instruction-tuning, pre-training의 성능
  - Reward Model이 잘 작동하는 지 잘 확인해야 한다.
  - ChatGPT의 등장 : Instruction-tuning+ RLHF

> 💡 RLHF의 한계
>
> - 인간의 판단은 일관성이 부족하고 기준이 어긋난다. (Reward Hacking)
>   - 챗봇이 정답의 여부와 관계없이 생산적이고 도움이 되어 보이는 정답 생성 (환각 Hallucination)
> - 인간의 판단을 학습한 Reward 모델의 일관성도 어긋난다.
>
> → RL을 빼버리는 게 어때? DPO의 등장

4. DPO (Direct Preference Optimization)

RLHF 의 강화학습(RL) 을 거치지 않고, 사람이 선호하는 데이터를 직접 활용해 모델을 바로 최적화 하는 방법

- RLHF의 단점을 보완하기 위해 등장
- 특징
  - 보상 모델, 강화학습 알고리즘이 필요 없음. → 학습 파이프라인이 단순해짐
  - RLHF보다 학습 효율이 높고 안정적임
  - 기존 Instruction-tuning된 모델에 바로 적용 가능
- 단점
  - 선호 데이터 품질에 크게 의존
  - 사람의 세밀한 평가 기준을 충분히 반영하기 어렵다고 함

5. RLVR

RLHF에서 수학문제와 같이 답이 분명한 문제들은 정답여부로 보상을 준다.

- 대표적인 사례 : DeepSeek-R1

---

2. Retrieval-augmented Language Models (Information Retrieval, RAG)

학습 목표

- 검색증강 언어모델(Retrieval-augmented LM)에서 사용하는 용어들에 대해 이해한다.
- 정보 검색(Information Retrieval)에서 사용되는 검색 메서드들을 이해한다.
- 검색증강 언어모델(Retrieval-augmented LM)을 이해한다.

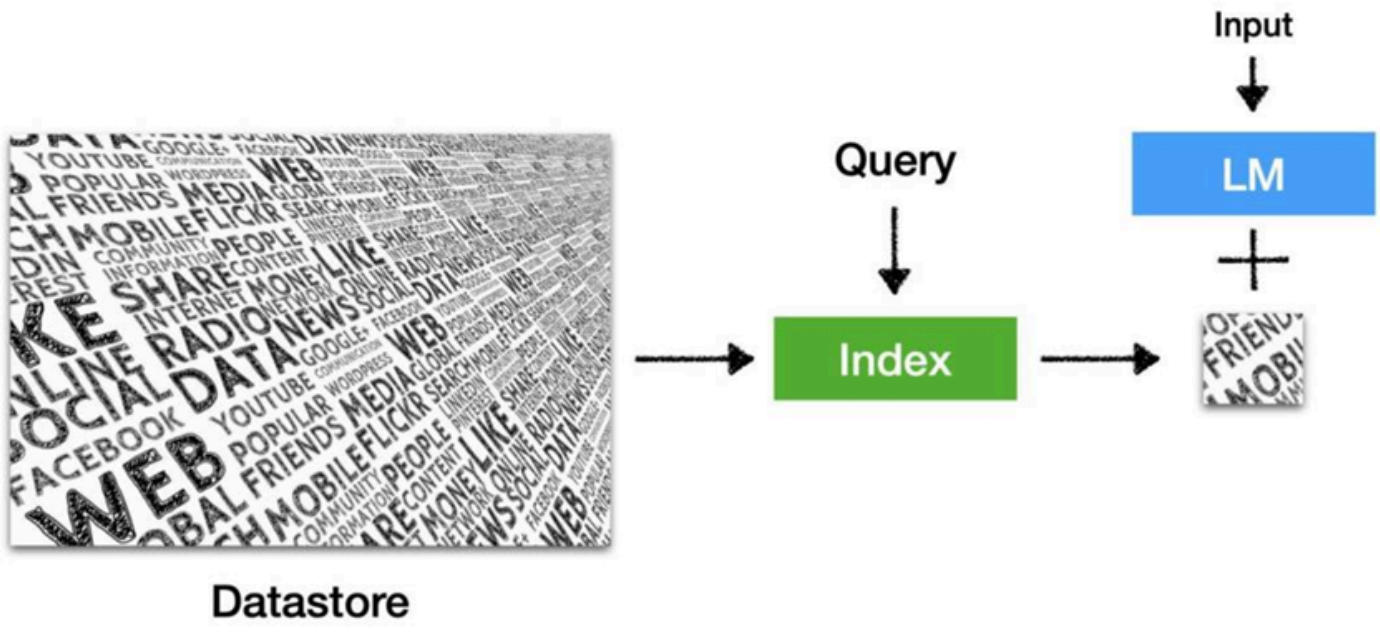
학습 시작(Overview)

- 검색증강 언어모델(Retrieval-augmented LM)이 무엇일까?
  - Retrieval-augmented LM에서 사용되는 용어들에 대한 이해
- Information Retrieval 이 무엇일까?
  - 정보 검색의 활용에 대한 이해와 검색기 (Retrieval) 이해
- Retrieval-augmented LM은 언제 사용해야 할까?
  - Retrieval-augmented LM이 필요한 이유와 도전과제들 이해

1. Basic Concepts of Retrieval-augmented LM

검색증강 언어모델(Retrieval-augmented LM)

- 추론 시 외부 데이터를 불러와 활용하는 언어 모델
- 구성 요소 : Datastore, Query , Index, Language Model



- Datastore
  - 가공되지 않은 대규모 텍스트 코퍼스
    - 최소 수십억에서 수조 단위의 토큰으로 구성
    - 라벨링된 데이터셋이 아님
    - 지식베이스(Knowledge base)와 같은 구조화된 데이터가 아님

- Query(쿼리)
  - 검색 질의 / Retrieval input
    - 언어 모델의 질의(input)와 같아야 하는 것은 아니다
- Index(인덱스)
  - 문서나 단락과 같은 검색 가능한 항목들을 체계적으로 정리하여 더 쉽게 찾을 수 있도록 하는 것
  - 각 정보 검색(Information Retrieval) 메서드는 인덱싱 과정에서 구축된 인덱스를 활용해, 쿼리와 관련 있는 정보를 식별한다.
- Retrieval (검색)
  - Datastore에 있는 수 많은 정보 중에서, 주어진 쿼리(Query)와 가장 관련성이 높은 정보를 찾아내는 과정
- Retrieval-augmented Generation(RAG)
  - 사용자의 질문에 답하기 위해, Datastore에서 관련 정보를 검색(Retrieval) 해서, 이를 언어모델이 생성(Generation) 단계에 함께 활용하는 방법

2. Information Retrieval

Information Retrieval (정보 검색)

- 정보 검색은 사용자의 질의에 기반하여 데이터 베이스나 인터넷과 같은 대규모 저장소에서 관련 있는 정보를 찾아내는 과정
- 목표 : 사용자의 검색 질의(Query)에 가장 관련성이 높은 정보를 제공하는 것
- 활용
  - 웹 서치 & 아이템 서치
    - 서치 엔진 (Search Engines) : 구글, 네이버
    - 이커머스 (E-Commerce) : 아마존, 쿠팡
  - 추천 시스템
    - OTT 서비스
    - 이커머스
  - 검색 증강 생성 (RAG)

Retriever(검색기) 종류

1. Sparse Retriever

- 전통적인 정보 검색 기법, 쿼리와 문서 간의 정확한 용어 일치 (어휘적 유사도) 에 기반
  - ex) TF-IDF, BM-25
- 장점
  - 단순성(Simplicity) : 구현과 이해가 쉽다
  - 효율성(Efficiency) : Inverted Index 구조 덕분에 빠른 검색과 효율적인 질의 처리가 가능
  - 투명성(Transparency) : 검색 결과가 보통 해석 가능하며, 용어 매칭에 기반하기 때문에 설명이 명확
- 단점
  - 제한된 의미 이해 (Limited semantic understanding), 정확히 일치해야 하는 단점

TF-IDF(Term Frequency-Inverse Document Frequency)

- 문서 내 특정 단어의 중요도를 나타내는 가중치 방식
  - TF (Term Frequency) : 단어가 문서에 얼마나 자주 등장하는 지
  - IDF (Inverse Document Frequency) : 단어가 전체 코퍼스에서 얼마나 드물게 등장하는 지
- 직관
  - 문서 안에서 자주 등장하는 단어는 중요하다 (TF)

    - 하지만 너무 많은 문서에 등장하는 단어는 덜 중요하다 (IDF)

2. Dense Retriever

- 쿼리와 문서를 표현하기 위해 Dense Vector 을 활용해 의미적 유사도에 기반
    - ex) DPT, Contriever, Openai-embeddings 등
- 장점
    - 의미적 이해 (Semantic understanding) : Dense retriever는 동의어나 다양한 표현을 더 효과적으로처리할 수 있어, 글의 문맥과 의미를 포착한다.
    - 풍부한 쿼리 표현 (Rich query representation) : 복잡한 쿼리와 긴 검색 쿼리의 의미를 더 잘 포착할 수 있다.
- 단점
    - 높은 연산 비용 (High computational cost) : 모델 학습에는 상당한 계산 자원과 시간이 필요, 추론과정에서 많은 자원을 소모할 수 있다.
    - 제한된 투명성 (Limited transparency) : Dense Retriever 는 블랙박스처럼 동작할 수 있어, 왜 특정 문서가 검색되었는 지 해석하기 어렵다.
    - 데이터 및 모델 의존성 (Dependency on data and models) : Dense retriever의 성능은 학습 데이터의 품질이나 모델 변경에 크게 영향을 받는다.

Embedding Models(임베딩 모델)

임베딩 모델은 단어/문장의 의미를 표현할 수 있다.

- ex) BERT (Only-Encoder)

Bi-Encoder Retreiver

- 대조 학습을 통해 학습되며, 이는 쿼리가 긍정적인 문서와 가깝게 유지되도록 하고 부정적인 문서에는 멀어지도록 유도한다
- Dense Retreiver의 종류

Cross-encoder Retreiver

- Cross-encoder 아키텍쳐는 두 개의 텍스트를 하나의 시퀀스로 결합
- Dense Retreiver의 종류
- 장점
    - Self-Attention을 통해 모든 쿼리와 문서 토큰이 완전히 상호작용할 수 있어, bi-encoder보다 더 높은 정확도를 얻을 수 있다.
- 단점
    - 모든 쿼리-문서 쌍을 개별적으로 모델에 입력해야 하므로, 계산 비용이 크고, 처리 속도가 느리다는 단점이 있다.

Bi-encoder VS Cross-encoder

Bi-encoder는 두 문장을 따로 인코딩 한다.

- 매우 빠르고 대규모 데이터베이스 검색에 적합하지만 정확도는 낮다

Cross-encoder 는 두 문장을 함께 처리한다.

- 매우 느리지만, 세밀한 상호작용을 포착하기 때문에 정확도는 높다

3. Retrieval-augmented LM

Retrieval-augmented LM (검색기반 LM)

- 추론 시 외부 데이터 저장소를 불러와 활용하는 언어모델
- RAG(Retrival-augmented Generation) : 정보 검색부터 답변 생성까지의 프레임워크를 RAG라 한다.

거대 언어 모델의 한계

- 거대 언어 모델은 모든 지식을 다 자신의 파라미터에 저장하지 못한다.
  - 자주 등장하는 쉬운 정보는 잘 기억하지만, 드물게 등장하는 정보는 잘 기억하지 못함
- RAG는 모델이 잘 모르는 드문 정보를 외부 지식에서 검색해서 활용 가능
  - 희귀한 사실이나 전문적인 정보에서 큰 효과
  - 모델이 이미 잘 아는 정보에 대해서는 오히려 부정적이다.
- 거대 언어모델이 보유한 지식은 금새 시대에 뒤쳐지며 갱신이 어렵다.
  - 현재의 지식 편집(knowledge editin) 메서드들은 확장성이 부족
  - RAG의 저장소(Database)는 쉽게 업데이트 가능
- 거대 언어모델의 답변은 해석과 검증이 어려움
- 기업 내부 정보와 같은 보안 정보는 언어모델 학습에 활용되지 않음
  - 사내 챗봇/기업 내부 시스템에 언어모델을 사용하는 경우 내부 데이터를 학습 시 정보 유출의 위험성이 있음

Retrieval-augmented LM 파이프라인

- 검색 증강 언어모델 : 언어 모델에 질문과 더불어 검색 엔진 결과를 함께 이용

1. 질의 추출
2. 문서 검색
3. 언어 모델 추론

Retrieval-augmented LM 의 도전 과제

1. Context 구성의 어려움
   - 어떤 길이로 잘라서 넣어야 할까?
   - 복잡한 질문의 경우 더 많고 긴 문서가 필요한 경우 존재

   → 언어 모델의 컨텍스트 길이를 늘리자 (Position embedding, Efficient inference, Fine-tuning on long context 등)
2. RAG의 결과는 검색 모델 성능에 의존

   → 검색 노이즈에 취약
   - 정확하지 않은 유사 정보가 언어 모델이 답하는 것을 방해할 수 있음
   - 컨텍스트 안의 정보를 이용하려는 LLM의 경향성 → 검색에서의 노이즈가 환각을 증가시킴
   - 정보 검색을 사용하는 것만으로는 사실의 정확성 보장이 어렵다.

   → Noise 학습
   - Noise Robustness (잡음이 있어도 올바른 답을 찾는 강건함 갖기)
   - Negative Rejection (모든 결과가 정답과 무관하면, 모른다고 하기)
3. LLM의 사전지식과 컨텍스트 간의 충돌(Conflict) 발생

   → 1) Context 위에서 Grounding 학습 강화

   → 2) Context가 없을 때, 답변 회피/거절 학습
4. 복잡한 추론 필요 & 문서가 명확한 사실에 대한 오류를 포함할 때

   → 정보 통합 문제(Information Integration) : 여러 문서에 답이 나뉘어져 있을 경우 ,두 문서의 정보를 통합해서 최종 답변을 해야함

   → 반사실적 강인성문제(Counterfactual Robustness) : 검색된 문서가 사실과 다른 잘못된 정보를 포함할 경우, 문서에 휘둘리지 않고, 사실적 오류를 인지해야 함.

→ 아직도 갈길이 멀다!

3. LLMs with Tool Usage(LLM Agent, Tool Use, MCP)

학습 목표

- LLM Agent에 대한 기본 개념을 이해한다
- LLM의 외부 툴 사용에 대한 개념을 이해한다
- MCP 개념을 이해한다

학습 시작(Overview)

- LLM Agent이란 무엇일까?
  - Agent에 대한 개념과 LLM Agent를 구성하는 요소들을 이해한다
- LLM이 어떻게 외부 툴을 사용하게 되었을까?
  - 언어모델이 어떻게 툴을 사용하는 방법을 배우는 지 이해한다
- MCP란 무엇일까?
  - MCP가 등장하게 된 배경과 개념을 이해한다

---

1. Basic Concepts of LLM Agents

Agent 란?

- 센서를 통해 환경(environment)을 인지하고, 액츄에이터(Actuator)를 통해 환경에 대해 액션(action)을 하는 것으로 영향을 미칠 수 있는 모든 것

강화학습 (Reinforcement Learning) - 의사결정의 과학

LLM Agent란?

- 거대 언어모델(LLM)을 핵심 구조(backbone)로 삼아 환경을 이해하고 행동을 수행하는 에이전트
- LLM-first view : 기존 LLM을 활용한 시스템을 에이전트로 만든다
  - 서치(search) 에이전트 , 심리상담 에이전트, 코드(code) 에이전트
- Agent-first view : LLM을 AI 에이전트에 통합하여, 언어를 활용한 추론과 의사소통을 가능하게 한다
  - 로봇, 임바디드(embodied) 에이전트

Agent 예시

- 웹을 탐색하는 LLM 시스템
- OS의 파일을 검색하고 코드를 사용해 처리하는 LLM 시스템
- 정보를 검색하고 생성하는 LLM 시스템 (X)
- 복잡한 추론을 하는 O1 같은 LLM (X)

Agent의 요건들

- 도구 사용(Tool Use)
- 추론과 계획(Reasoning and Planning)
- 환경 표현(Environment Representation)
- 환경 이해(Environment understanding)

- 상호작용/의사소통(Interaction/Communication)

LLM Agent 프레임워크

- Controller : 사용자의 요청을 이해하고 실행 가능한 계획을 세움 (Foundation Model 기반)
- Tool Set : 다양한 기능을 가진 도구 모음 (검색, 계산, API 호출 등)
- Perceiver : 환경에서 얻은 피드백을 요약해 Controller 에게 전달
- Environment : 도구가 실제로 동작하는 플랫폼
- Human : 지시를 내리고, 필요할 경우 피드백 제공

> 💡 LLM Agent의 Tool Set에 대해 먼저 알아본다.

2. Tool Usage in LLMs

Tool 이란?

- 언어 모델 외부(external) 에서 실행되는 프로그램에 연결되는 함수(function) 인터페이스를 의미한다.
  - LLM은 함수 호출과 입력 인자를 생성함으로써 이 도구를 활용할 수 있다.

Tool의 세 가지 유형

1. Physical Interaction-based Tools (물리적 상호작용 도구)
   - 실제 세상과 연결되는 도구들 (로봇, 자율주행, IoT 기기 등).
   - Agent가 물리적 세계를 관찰하고 행동으로 반영.
2. GUI-based Tools (그래픽 인터페이스 도구)
   - GUI를 통해 사용자가 다루는 소프트웨어 (웹, Photoshop, MS Office 등).
   - LLM은 GUI 조작을 통해 가상 세계에서 작업을 수행.
3. Program-based Tools (프로그래밍 인터페이스 도구)
   - API, SDK, 데이터베이스, 지식 그래프, GitHub Copilot 같은 개발 환경.
   - 코드 실행이나 데이터 질의 등 프로그래밍 수준의 조작 가능.

도구 사용 패러다임(Tool Use Paradigms)

- 도구 사용(Tool Use) : 두 모드 간의 전환
  - 텍스트 생성 모드 (text-generation mode)
  - 도구 실행 모드 (tool-execution mode)
- 도구 사용을 유도하는 방법
  - 추론 시 프롬프트 (inference-time prompting)
  - 학습 (Training; Tool Learning)

툴 러닝(Tool Learning) 방식

- 모방 학습(Imitation Learning) : 인간의 도구 사용 행동 데이터를 기록, 언어 모델이 인간의 행동을 모방
- ex) OpenAI : WebGPT(2022)
  - 검색 엔진을 사용하기 위해 인간의 행동 모방

    - 지도학습 + 강화학습
    - 장문의 질의 응답에서 뛰어난 성능을 보이며, 인간보다 나음
- Meta: Transformer(2023)
    - 모델 스스로 학습 데이터 생성(Self-supervised)
        1. API 호출 샘플링 : LLM이 기존 데이터셋에서 문맥을 보고 API 호출 후보를 생성
        2. API 실행 : 생성된 API 호출을 실제로 실행하여 응답을 얻음
        3. API 호출 필터링 : API 호출이 문맥에 유용한지 평가 및 필터링 진행
    - 지도 학습(Supervised Fine-tuning)
    - 검색 엔진 뿐만 아니라 달력, 계산기와 같은 여러 API를 활용
- ToolLLM(2023)
    - 기존 연구의 한계점인 툴의 다양성과 범용성을 타겟팅 한 연구
    - 방법
        - API를 수집
        - 수집한 API를 기반으로 명령문 작성
        - 정답 어노테이션 & 필터링
- 멀티 모달 툴러닝 (최근 방식)
    - 멀티모달 대규모 언어 모델(MLLM)을 기반으로 도구를 정의하고 활용하는 연구
        - ex) GUI 에이전트, Embodied 에이전트
- 강화학습
    - 지도학습을 넘어 에이전트에서의 강화학습을 도입하는 연구

3. Model Context Protocol (MCP)

MCP의 도입

- 외부 툴을 사용하는 연구가 급증하면서 회사/모델마다 각기 다른 툴 호출 방식 및 스키마 개발
    - 호환성 부족(모델마다 다름)
    - 재사용 어려움(같은 툴도 다른 모델에서는 다시 정의해야 함)

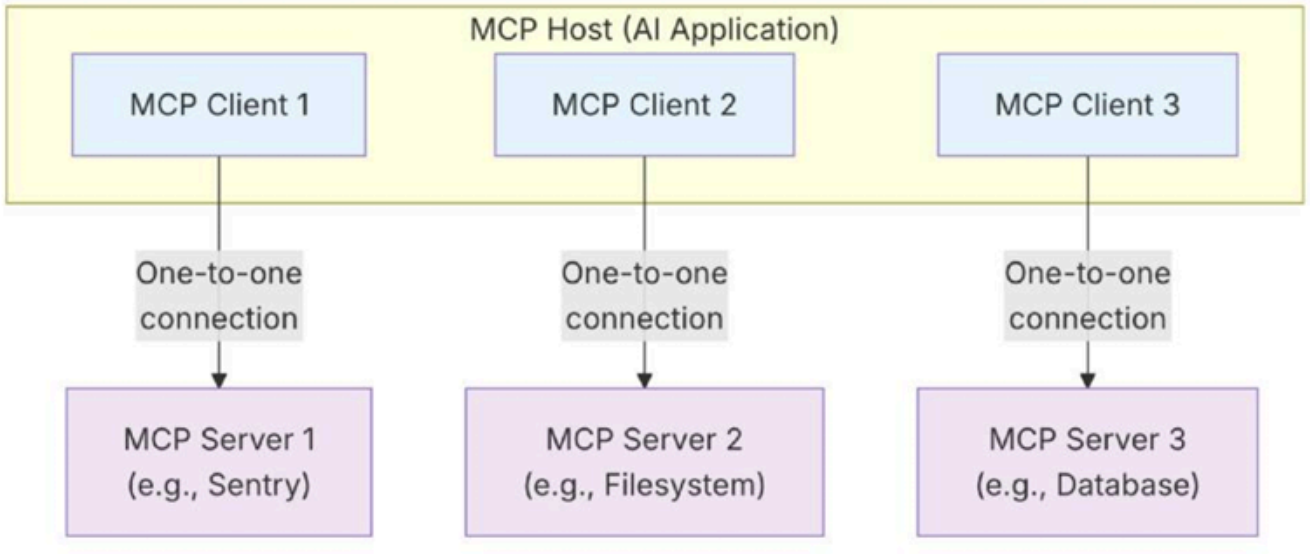
MCP란?

Model Context Protocol

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

MCP 아키텍쳐

- MCP Host : 하나 또는 여러 개의 MCP 클라이언트를 조정하고 관리하는 AI 애플리케이션
- MCP Client : MCP 서버와의 연결을 유지하며 MCP 호스트가 사용할 수 있도록 MCP 서버로부터 컨택스트를 가져오는 구성요소
- MCP Server : MCP 클라이언트에게 컨택스트를 제공하는 프로그램



MCP 계층(Layer)

- MCP는 두 개의 계층으로 구성된다.
  - 데이터 계층(Data Layer) : 클라이언트-서버 통신을 위한 JSON-RPC 기반 프로토콜로 정의
    - 내부 계층(Inner Layer)
    - 라이프 사이클 관리, 툴, 리소스, 프롬프트와 같은 핵심 요소들이 포함된다.
  - 전송 계층 (Transport Layer) : 클라이언트 서버 간 데이터 교환을 가능하게 하는 통신 매커니즘과 채널을 정의
    - 외부 계층 (Outer Layer)
    - 전송 방식에 특화된 연결 수립, 메시지 프레이밍, 인증이 포함

MCP로 통합된 툴 호 방식 (JSON-RPC)

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/call",
  "params": {
    "name": "get_weather",
    "arguments": {
      "location": "Seoul",
      "units": "celsius"
    }
  }
}
```

MCP의 장점

- 표준화 (Standardization) : 모든 모델/툴이 동일한 호출 규격 사용
- 확장성 (Extensibility) : 새로운 툴 쉽게 추가 가능
- 호환성 (Interoperability) : 모델/플랫폼에 상관없이 같은 툴 호출 가능
- 재사용성 (Reusability) : 한 번 정의한 툴을 여러 모델에서 활용 가능
- 투명성 (Transparency) : 호출 과정이 명확히 기록/검증됨

MCP 서버 예시코드

```
#calc_server.py
from tastmcp import FastMCP
```

```
#1. 서버 인스턴스
mcp = FastMCP("calc")

#2. MCP 툴 선언
# 타입 힌트-> JSON-Schema 자동 변환
@mcp.tool()
def add(a: int, b: int) → int:
    """Add two number"""
    return a + b

#stdio, HTTP, WS 등 자동 선택
if __name__ == "__main__":
    mcp.run()
```

MCP 현재

- MCP는 현재 학계/업계에서도 모두 표준으로 확립
- Web과 AI 개발을 구분하게 되어 양쪽 생태계 모두에서 표준적이고 호환가능한 개발이 가능해짐

---

4. AI Agents & LangChain(AI Agents, LangChain)

학습 목표

- 에이전트가 상호 작용을 하는 환경은 어떻게 표현되는지와 LLM은 이를 어떻게 이해하는 지 알아본다
- 에이전트가 태스크를 수행하기 위해 추론/계획을 어떻게 수립하는 지 이해한다
- LangChain이 무엇인지 이해한다

학습시작(OverView)

- 에이전트가 상호 작용하는 환경이란 무엇일까?
  - 태스크 환경에 대한 이해와 언어 모델이 어떻게 환경을 이해하는 지에 대해 사례 중심으로 이해한다
- 에이전트가 추론/계획은 어떻게 할까?
  - 사용자의 요청을 수행하기 위해 에이전트는 어떻게 추론하고 계획을 수립하는 지 사례중심으로 이해한다
- LangChain이 무엇일까?
  - LangChain이 무엇인지 이해하고, 어디서 시작하면 되는 지 안다

1. Environment Representation & Understanding

LLM 에이전트의 요건들 중 환경 표현과 환경 이해에 대해 알아본다.

💡 Agent의 요건들

- 도구 사용(Tool Use)
- 추론과 계획(Reasoning and Planning)
- 환경 표현(Environment Representation)
- 환경 이해(Environment understanding)
- 상호작용/의사소통(Interaction/Communication)

환경과 에이전트 활용 예시들

- 챗봇
- 로보틱스
- 임바디드(Embodied) 에이전트
- 게임
- 소프트웨어 개발

에이전트가 환경을 이해하기 위해 필요한 것

- Tool : 환경에 접근
- Representation : 환경의 표현
- 환경을 이해/탐색하기 위한 방법론들

환경의 표현 방법

1. Text (텍스트)
    - ALFWorld(2021)
        - 텍스트 기반 임바디드 에이전트 시뮬레이션 엔진
        - 물리적 세계에 대한 정보를 언어 모델이 텍스트 기반으로 이해하고 명령을 수행할 수 있도록 환경을 표현함
2. Image(이미지)
    - Touchdown(2018)
        - Google Street View 기반 내비게이션 및 지시 따르기 데이터셋
        - 사용자가 제공한 자연어 지시문에 따라 실제 도시 환경 이미지 내에서 경로 탐색을 수행
        - 시각적 환경을 텍스트와 연결하여 지시 수행 및 환경 이해를 가능하게 함
    - 이미지 기반 표현의 단점
        - 에이전트로서 좋은 성능을 내려면 세부적인 이해가 중요
            - ex) OCR, 복잡한 레이아웃에서의 그라운딩
        - 많은 모델들이 이런 태스크에서 실패하지만 일정 수준의 학습을 통해 개선이 가능하다
3. 텍스트 기반 웹 (Textual Web Represantations)
    - WebArena(2024)
        - 웹 환경을 텍스트(HTML, DOM, Accessibility Tree)로 표현하여 에이전트가 상호 작용 가능
        - 단순한 스크린샷 이미지 대신 구조화된 웹 표현을 제공
        - 버튼 클릭, 항목 선택, 같은 작업을 더 정확히 수행 가능

복잡한 환경은 어떻게 이해할 수 있을까?

- 모델은 자신이 상호작용하는 환경에 대해 모든 것을 알고 있지 않음
- 일부 지식은 LLM 파라미터 안에 포함
- 다른 지식은 실시간 환경과의 상호작용을 통해 발견해야 함

복잡한 환경에 대한 이해

1. 환경 특화 프롬프트(Environment-specific Prompts)
    - 환경에 맞게 수동으로 프롬프트를 제작하여 에이전트가 지시를 따르도록 유도
        - ex) SteP : 웹 탐색을 위한 프롬프트 탬플릿
    - 문제점 : 일반화(Generalization)

    - 특정 환경에는 잘 작동하지만, 새로운 환경에서 성능이 떨어짐

2. 비지도 프롬프트 유도 (Unsupervised Induction of Prompts)
    - Agent Workflow Memory(2024)
        - 성공적으로 수행된 워크플로우를 기억하고, 이를 기반으로 새로운 프롬프트를 생성하여 에이전트에 제공
        - 환경에서의 상호 작용을 메모리에 통합

      → 프롬프트를 사람이 수동으로 설계하지 않고 에이전트가 경험을 통해 자동 생성 및 일반화
3. 환경 탐색(Environment Exploration)
    - 모델이 환경을 탐색할 때 보상(reward)을 부여하여 학습을 유도 → 호기심 기반 보상
        - ex) 강화학습에서 예측 불가능한 상태 공간에 진입할 경우 보상을 증가
4. 탐색 기반 궤적 기억 (Exploration-based Trajectory Memorization)
    - BAGEL(2024)
        - 명령어를 샘플링하여 실행한 뒤, 더 정확한 새로운 명령어로 궤적을 재라벨링
        - LM-agent가 환경을 탐색하고, LM-labeler가 이를 바탕으로 지시를 다시 정제

      → 기존 데이터에 의존하지 않고, 탐색과 자기 교정(self-correction)을 통해 데이터 생성

      → 에이전트의 일반화 성능과 환경 적응력을 향상

2. Reasoning & Planning

LLM Agent의 추론과 계획에 대해 알아본다.

> 💡 Agent의 요건들
>
> - 도구 사용(Tool Use)
> - 추론과 계획(Reasoning and Planning)
> - 환경 표현(Environment Representation)
> - 환경 이해(Environment understanding)
> - 상호작용/의사소통(Interaction/Communication)

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
- 전역적 계획 (Global Planning)
    - 실행 가능한 전체 계획 경로(planning path)를 한번에 생성
    - 여러 개의 툴을 조합하여 시퀀스 형태로 결정
    - 효율적이나, 복잡한 환경에서는 실패가능

Local Planning 예시

- ReACT(2023)
    - 추론(reasoning)과 액션(action)을 결합하여 에이전트가 환경과 상호작용하도록 하는 방식
    - 단순한 추론/행동 분리를 넘어, 두 과정을 경합해 상황 적응형(local) 계획 가능
    - 다양한 도구 사용 및 웹 탐색, 추론 기반 Q&A 등에서 높은 성능

Global Planning 예시

- Plan-and-Solve Prompting (2023)
  - 전체 계획을 세운 뒤, 그 계획에 따라 순차적으로 문제를 해결
  - 단순한 step-by-step 방식보다 체계적으로 안정적인 추론 가능
  - 복잡한 문제 해결에서 오류 축적 감소

오류 식별과 회복(Error identification and Recovery)

- 에이전트는 에러/실수에서 회복할 방법 필요
- ex) Reflextion(2023)
  - 수행한 궤적(Trajectory)을 평가 후 잘못된 부분을 Reflection 단계에서 분석하여 재 시도하는 방식
  - 단순히 오류를 탐지하는 것에 그치지 않고, 스스로 반성(Reflection) 후 개선된 행동을 수행
  - 의사결정, 프로그래밍, 추론 등 다양한 태스크에서 성능 회복 및 향상을 보여줌

계획 재검토(Revisiting Plans)

- CoAct(2024)
  - 에이전트가 실행 도중 계획을 재검토(Revisit)하고 수정 가능
  - 두 개의 에이전트가 서로 협력하여 오류가 발생 시 재계획과 피드백을 수행한다.

→ 초기 계획이 불안정하더라도 실행과정에서 보완 가능

→ 장기 작업 (Long-horizon tasks) 에서 안전성과 성공률 향상

3. LangChain

LangChain 이란?

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

LangChain 특징

- 다양한 LLM Providers(OpenAI, Anthropic, Google)과 통합하여 모델/회사별 API 차이를 공통 인터페이스로 관리 가능
- Prompt, Memory, Tools와 같은 컴포넌트들이 모듈화 되어 있어 재사용성과 확장성 확보
- LangGraph 기반으로 복잡한 워크플로우를 시각적으로 설계 및 관리 가능

LangChain 주요 컴포넌트

- Prompt Template : 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
- 등등

LangChain 은 튜토리얼이 잘 되어 있다.

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

- 이번 강의에서 배운 Retreival-augmented LM, Agent 등이 다 포함

RAG w/ LanChain (https://python.langchain.com/docs/tutorials/rag/)

- LLM 설정부터 인덱싱과 RAG 다 존재

Agent w/ LangChain (https://python.langchain.com/docs/tutorials/agents/)

- ReACT 에이전트 셋업부터 실행까지 전반적인 흐름을 알려주는 튜토리얼
- 사용할 수 있는 툴 리스트 : https://python.langchain.com/docs/integrations/tools/

MCP w/ LangChain (https://github.com/langchain-ai/langchain-mcp-adapters?tab=readme-ov-file)

- MCP를 LangChain을 활용할 수 있도록 최근 LangChain에서 개발한 라이브러리
- 나만의 MCP 서버 구축 가능