

VIP Cheatsheet: 트랜스포머와 대형 언어 모델

아프신 아미디, 셰르빈 아미디
옮긴이 YJ (Yongjin) Kim - 김용진

2025 년 4 월 14 일

이 VIP Cheatsheet 은 250 페이지에 걸쳐 약 600 장의 일러스트로 구성된 「Super Study Guide: Transformers & Large Language Models」 책에 담긴 내용을 간단히 요약해 보여줍니다.
더 자세한 내용은 <https://superstudy.guide> 에서 확인하실 수 있습니다.

1 기초

1.1 토큰

□ **정의** – 토큰 (token) 은 단어, 단어 조각 (또는 부분 단어), 또는 문자와 같이 더 이상 나눌 수 없는 텍스트 단
위이며, AI 가 미리 알고 있는 단어 목록 (어휘집) 에 정의된 것들 중 하나입니다.

비고: [UNK] 토큰은 모델이 알 수 없는 텍스트 조각을 나타내며, [PAD] 토큰은 입력 시퀀스의 길이를 맞추기
위해 빈 자리를 채울 때 사용합니다.

□ **토큰라이저** – 토큰라이저 (tokenizer) T 는 텍스트를 임의의 세분화 수준 (단어 단위, 문자 단위 등) 에 따라
토큰으로 나누어 처리합니다.

이 곰 인형은 너어어어무 귀여워 \rightarrow T \rightarrow [이][곰][인형]은[UNK][귀여워][PAD]...[PAD]

다음은 대표적인 토큰라이저 유형들입니다:

유형	장점	단점	일러스트레이션
단어	<ul style="list-style-type: none"> 해석하기 직관적임 시퀀스 길이가 짧음 	<ul style="list-style-type: none"> 어휘 사전이 매우 커짐 단어 변형 (어미·활용형 등) 을 처리하기 어려움 	[곰] [인형]
하위 단어	<ul style="list-style-type: none"> 단어의 뿌리 (기본 형태) 를 잘 활용 가능 임베딩이 직관적 (원형 정보를 유지) 	<ul style="list-style-type: none"> 시퀀스 길이가 늘어날 수 있음 토큰화 과정이 더 복잡 	[곰] [인] [#형]
문자 바이트	<ul style="list-style-type: none"> 미등록 단어 문제 (out-of-vocabulary) 가 없음 어휘 사전이 작음 	<ul style="list-style-type: none"> 시퀀스가 훨씬 길어짐 너무 낮은 레벨 (문자·바이트) 이라 패턴 파악이 어려움 	[가][로][미][오][리] [홍][기][오]

비고: Byte-Pair Encoding (BPE) 와 Unigram 은 하위 단어 (subword) 수준에서 많이 사용되는 토큰
이저입니다.

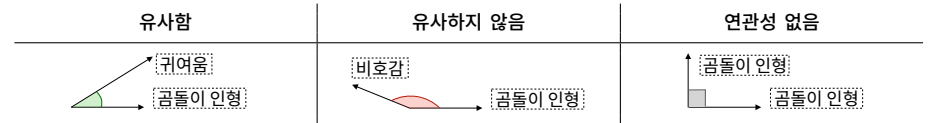
1.2 임베딩

□ **정의** – 임베딩 (embedding) 이란 토큰이나 문장과 같은 언어 정보를 나타내는 수치화된 표현이며, n 차원
실수 공간 (\mathbb{R}^n) 에 속하는 벡터 x 로 표현됩니다.

□ **유사도** – 두 토큰 t_1, t_2 사이의 코사인 유사도는 다음과 같이 계산됩니다:

$$\text{유사도}(t_1, t_2) = \frac{t_1 \cdot t_2}{\|t_1\| \|t_2\|} = \cos(\theta) \in [-1, 1]$$

두 토큰 사이의 각도 (θ) 는 이 두 토큰 사이의 유사도를 나타냅니다:

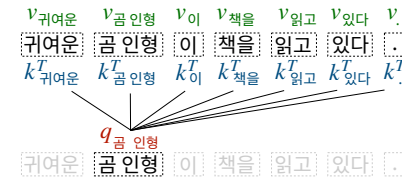


비고: Approximate Nearest Neighbors (ANN) 와 Locality Sensitive Hashing (LSH) 는 대규모 테
이터베이스에서 유사도를 효율적으로 추정하기 위한 기법입니다.

2 트랜스포머

2.1 어텐션

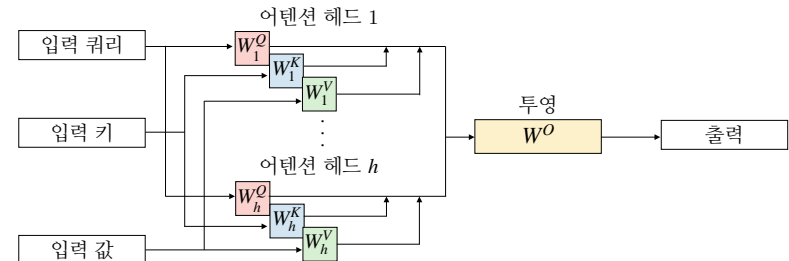
□ **공식** – 주어진 쿼리 q 에 대해, 우리는 연결된 값 v 와 관련하여 여러 키 (힌트) 들 중 어느 키 k 에 쿼리가 「어
텐션」 ("attention") 을 집중해야 하는지를 결정하고자 합니다.



어텐션 (attention) 은 각각 쿼리 (q), 키 (k), 값 (v) 를 담은 행렬 Q, K, V 와 키의 차원 d_k 를 사용하여 효
율적으로 계산할 수 있습니다:

$$\text{어텐션} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

□ **MHA** – 멀티헤드 어텐션 (Multi-Head Attention, MHA) 층은 여러 「헤드」 ("head") 에서 동시에 어텐
션 계산을 수행하고, 이렇게 얻은 결과를 합쳐서 모델이 다음 단계에서 활용할 수 있는 형태로 바꿉니다. 이 과정
을 통해 텍스트 속 다양한 관계와 패턴을 한 번에 파악할 수 있어, 더 풍부한 정보를 얻을 수 있게 됩니다.

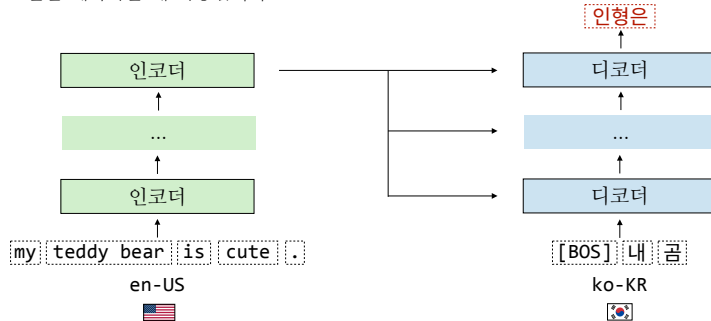


MHA 는 h 개의 어텐션 헤드와 W^Q, W^K, W^V 행렬로 구성됩니다. 이 행렬들은 입력을 쿼리 Q , 키 K , 값 V 형태로 투영 (프로젝션) 하는 역할을 하며, 최종 투영은 W^O 행렬로 이루어집니다.

비고: *Grouped-Query Attention (GQA)* 와 *Multi-Query Attention (MQA)* 는 *MHA* 의 변형으로, 여러 헤드가 키와 값을 공유하게 함으로써 계산 비용을 줄여줍니다.

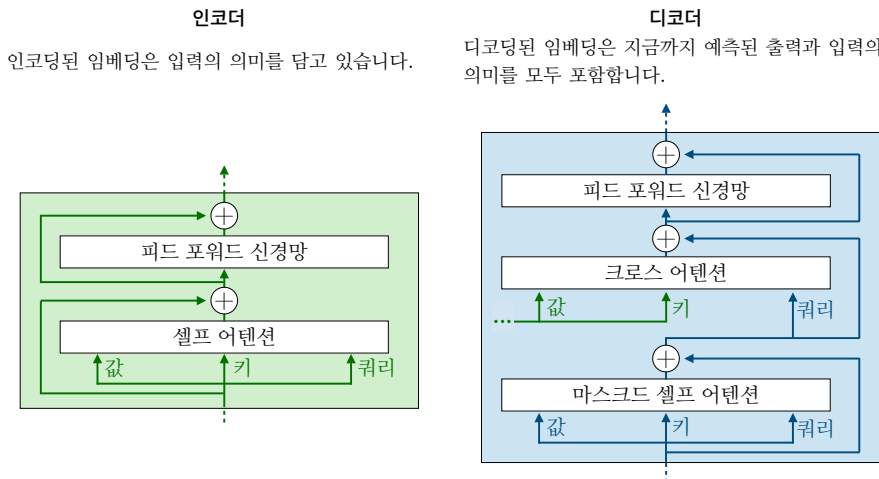
2.2 아키텍처

□ **개요** – 트랜스포머 (Transformer) 는 셀프-어텐션 (self-attention) 메커니즘에 의존하는 모델로, 인코더와 디코더로 구성되어 있습니다. 인코더는 입력의 의미 있는 임베딩을 계산하고, 이 임베딩은 디코더에 의해 시퀀스에서 다음 토큰을 예측하는 데 사용됩니다.



비고: 트랜스포머는 초기에는 주로 번역 작업에 사용되는 모델이었지만, 최근들어 더욱 더 많은 종류의 작업에 사용되어지기 시작했습니다.

□ **구성 요소** – 인코더와 디코더는 트랜스포머의 핵심적인 두 가지 구성 요소이며, 각각 다른 역할을 담당합니다:

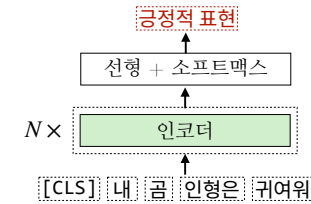


□ **포지션 임베딩** – 포지션 임베딩 (position embeddings) 은 문장 안에서 특정 토큰이 어떤 위치에 있는지 알려주는 정보를 담으며, 토큰 임베딩과 동일한 차원을 사용합니다. 이 임베딩은 직접 정의할 수도 있고 (예: 사인·코사인 함수를 사용해 위치를 표시), 학습 과정에서 데이터로부터 자동으로 학습할 수도 있습니다.

비고: 로터리 포지션 임베딩 (*Rotary Position Embeddings, RoPE*) 은 쿼리 (*query*) 와 키 (*key*) 벡터를 회전시키는 방식으로 상대적 위치 정보를 반영하는 방법입니다. 계산 효율이 높고 다양한 모델에서 효과가 좋아 인기 있는 변형 (*variation*) 으로 꼽힙니다.

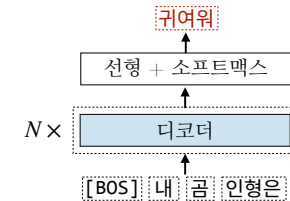
2.3 변형

□ **인코더-온리** – *Bidirectional Encoder Representations from Transformers (BERT)* 는 트랜스포머 구조 중 인코더 (encoder) 를 여러 겹 쌓아 만든 모델입니다. 텍스트를 입력하면, 문장 앞뒤의 맥락을 동시에 살펴보고 이를 의미 있는 숫자 벡터 (임베딩) 로 바꿔줍니다. 이렇게 생성된 임베딩은 이후 감정 분석, 문장 분류 등 다양한 후속 작업에 활용할 수 있습니다.



[CLS] 토큰은 문장의 의미를 잡아내기 위해 시퀀스 맨 앞에 추가됩니다. 이 [CLS] 토큰이 만들어낸 임베딩 (인코딩 결과) 은 이후 감정 분석 등의 후속 작업에서 자주 활용됩니다.

□ **디코더-온리** – *Generative Pre-trained Transformer (GPT)* 는 자기회귀 (autoregressive) 방식을 사용하는 트랜스포머 모델로, 여러 개의 디코더를 겹쳐 쌓아 구성됩니다. BERT 계열 모델과 달리, GPT 는 모든 문제를 텍스트 텍스트 형태로 다룹니다.



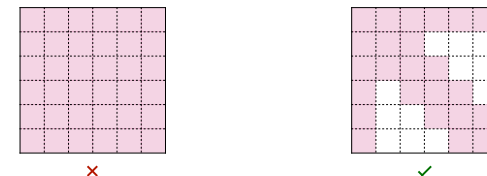
GPT 시리즈, LLaMA, Mistral, Gemma, DeepSeek 등과 같은 대부분의 최신 대형 언어 모델들은 디코더에만 의존하는 구조를 띄고 있습니다.

비고: 인코더-디코더 모델들은 *T5* 와 마찬가지로 자기회귀적이며 디코더-온리 모델과 많은 특징을 공유합니다.

2.4 최적화

□ **어텐션 근사** – 어텐션 연산은 시퀀스 길이 n 에 대해 $O(n^2)$ 의 복잡도를 가지는데, 시퀀스가 길어질수록 부담이 커집니다. 이를 완화 위한 주된 두 가지 방법은 다음과 같습니다:

- 회소성: 셀프 어텐션을 전체 시퀀스에 걸쳐 수행하는 대신, 더 중요한 토큰들 사이에서만 어텐션을 계산합니다.



- 로우 랭크: 어텐션 공식을 낮은 차원 (로우 랭크) 행렬의 곱으로 단순화해, 계산 부담을 줄입니다.

□ **플래시 어텐션** – 플래시 어텐션은 GPU 하드웨어를 영리하게 활용해 어텐션 계산을 최적화하는 방법으로, 빠른 정적 랜덤 액세스 메모리 (*Static Random-Access Memory*, SRAM) 를 우선 사용해 행렬 연산을 처리한 뒤, 결과를 느린 고대역폭 메모리 (*High Bandwidth Memory*, HBM) 에 기록합니다.

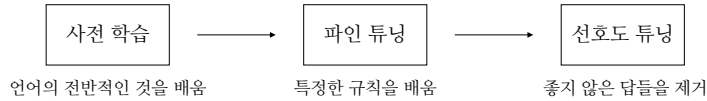
비고: 실제로 플래시 어텐션을 적용하면 메모리 사용량이 줄고 계산 속도도 빨라집니다.

3 대형 언어 모델

3.1 개요

□ **정의** – 대형 언어 모델 (*Large Language Model*, LLM) 은 강력한 자연어 처리 능력을 갖춘 트랜스포머 기반 모델입니다. 여기서 ‘대형’이라는 명칭이 붙은 이유는 일반적으로 수십억 개의 파라미터 (매개변수) 를 포함하기 때문입니다.

□ **LLM 라이프사이클** – LLM 학습 과정은 크게 사전 학습 (pretraining), 파인 튜닝 (finetuning), 그리고 선호도 튜닝 (preference tuning) 의 세 단계로 이루어집니다.

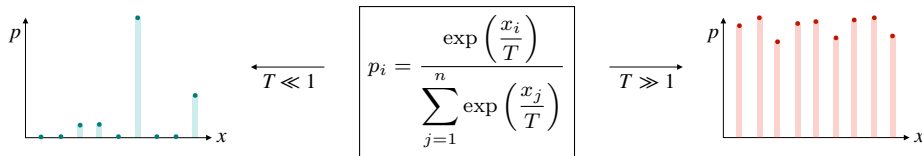


파인 튜닝과 선호도 튜닝은 모델의 기본 학습이 끝난 뒤, 특정 작업을 더 잘 수행하도록 모델을 조정하는 후처리 (post-training) 기법입니다.

3.2 프롬프팅

□ **컨텍스트 길이** – 모델의 컨텍스트 길이는 입력에 포함될 수 있는 토큰의 최대 개수를 의미합니다. 일반적으로 수만 개에서 수백만 개의 토큰까지 처리할 수 있습니다.

□ **디코딩 샘플링** – 토큰 예측은 예측 확률 분포 p_i 에서 뽑아내며, 이때 샘플링 결과는 하이퍼파라미터 온도 (temperature) T 에 따라 달라집니다.



비고: 온도가 높을수록 더욱 더 창의적인 결과를 도출하게 되고, 온도가 낮을수록 더욱 더 보수적인 결과를 도출하게 됩니다.

□ **Chain-of-thought** – *Chain-of-Thought* (CoT) 는 모델이 복잡한 문제를 해결할 때, 그 과정을 여러 단계로 쪼개서 논리적으로 추론하는 방식입니다. 이를 통해 모델이 최종 답변을 더 정확하게 이끌어낼 수 있습니다. *Tree of Thoughts* (ToT) 는 CoT 를 한층 발전시킨 방식으로, 여러 갈래의 추론 경로를 동시에 탐색하여 최적의 결론에 도달하도록 합니다.

비고: 자기 일관성 (*self-consistency*) 는 추론 경로에서 나온 여러 답변들을 모아, 가장 일관성 있는 결론을 도출하는 방법입니다.

3.3 파인 튜닝

□ **SFT** – *Supervised FineTuning* (SFT) 는 모델의 기본 학습이 끝난 후 모델의 행동을 원하는 최종 작업에 맞추는 방법입니다. 이 방법은 해당 작업과 밀접하게 관련된 고품질 입력-출력 쌍 (input-output pairs) 에 의존합니다.

비고: SFT 에 사용하는 데이터가 지시사항 (*instruction*) 위주라면, 이 과정을 “*instruction tuning*” 이라고 부릅니다.

□ **PEFT** – *Parameter-Efficient FineTuning* (PEFT) 는 SFT 과정을 보다 효율적으로 수행하기 위한 방법들을 묶어서 부르는 용어입니다. 예를 들어 *Low-Rank Adaptation* (LoRA) 은 학습 가능한 가중치 W 를 기존의 W_0 로 고정하고, 대신 낮은 차원의 행렬 A, B 를 학습하여 W 를 근사하는 방식입니다.

$$W \approx W_0 + B \times A$$

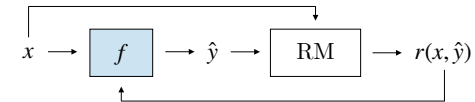
비고: 그 밖의 PEFT 기법으로는 프리픽스 튜닝 (*prefix tuning*) 과 어댑터 (*adapter*) 레이어 삽입이 있습니다.

3.4 선호 튜닝

□ **리워드 모델** – 리워드 모델 (*Reward Model*, RM) 은 입력 x 가 주어졌을 때, 모델이 만든 출력 \hat{y} 가 원하는 행동 (목표) 과 얼마나 잘 맞는지를 예측합니다. *Best-of-N* (BoN) 샘플링 또는 ‘거부 (rejection) 샘플링’ 은, 한 번에 N 개의 답변을 생성한 뒤 리워드 모델을 이용해 가장 좋은 답변을 골라내는 방법입니다.

$$x \rightarrow f \rightarrow \hat{y}_1, \hat{y}_2, \dots, \hat{y}_N \rightarrow \text{RM} \rightarrow k = \underset{i \in \{1, N\}}{\operatorname{argmax}} r(x, \hat{y}_i)$$

□ **강화 학습** – 강화 학습 (*Reinforcement Learning*, RL) 은 리워드 모델 (RM) 로부터 받은 보상 정보를 활용해 모델 f 를 업데이트하는 접근 방식입니다. 만약 보상 모델이 인간의 선호도에 기반한다면, 이를 ‘인간 피드백 기반 강화 학습 (*Reinforcement Learning from Human Feedback*, RLHF)’이라고 부릅니다.

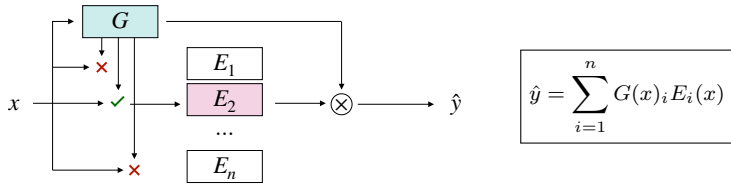


Proximal Policy Optimization (PPO) 는 강화 학습 알고리즘 중 하나로, 모델이 높은 보상을 받도록 유도하면서도 기본 모델과 크게 달라지지 않도록 (보상 해킹 방지) 제약을 거는 기법입니다.

비고: *Direct Preference Optimization* (DPO) 처럼 보상 모델과 강화 학습 과정을 하나의 지도 학습 과정으로 합치는 방법도 있습니다. 이 방식을 통해 RM 과 RL 을 동시에 처리할 수 있습니다.

3.5 최적화

□ **전문가 혼합** – 전문가 혼합 (*Mixture of Experts*, MoE) 이란 모델이 추론 시 일부 뉴런만 활성화되도록 하는 방식으로, 게이트 (gate) G 와 전문가 (expert) E_1, \dots, E_n 으로 구성됩니다.



전문가 혼합 기반의 LLM 은 이 게이팅 메커니즘을 FFNN (피드포워드 신경망) 내부에서 활용합니다.

비고: *MoE* 기반 LLM 의 학습 (*training*) 은 매우 까다로운 것으로 유명합니다. *LLaMA* 논문에서도 추론 효율성이 높음에도 불구하고 이 아키텍처를 채택하지 않았다고 밝히고 있습니다.

□ **지식 증류** – 지식 증류 (distillation) 는 큰 모델 (선생, teacher) T 가 만든 예측 결과를 가지고, 작은 모델 (학생, student) S 를 학습시키는 과정입니다. 학습 시, KL 발산 (KL divergence) 손실 함수를 사용합니다:

$$\text{KL}(\hat{y}_T || \hat{y}_S) = \sum_i \hat{y}_T^{(i)} \log \left(\frac{\hat{y}_T^{(i)}}{\hat{y}_S^{(i)}} \right)$$

비고: 이때 사용되는 학습 레이블은 클래스 확률 형태로 주어지므로 “소프트 (*soft*) 레이블”이라고 부릅니다.

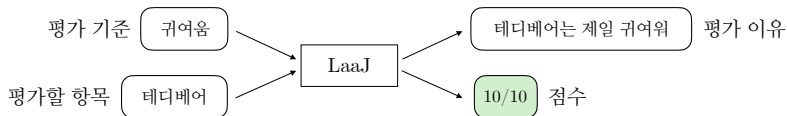
□ **양자화** – 모델 양자화 (model quantization) 는 모델 가중치의 정밀도를 낮춰 모델 용량을 줄이면서, 성능 저하를 최소화하려는 방법들을 통칭합니다. 이렇게 하면 모델 메모리 사용량을 줄이고 추론 속도를 높일 수 있습니다.

비고: *QLoRA* 는 *LoRA* (낮은 차원 적응) 기법에 양자화를 적용한 대표적인 변형 기법으로 자주 사용됩니다.

4 응용

4.1 LLM-as-a-Judge

□ **정의** – *LLM-as-a-Judge* (LaaJ) 는 대형 언어 모델 (LLM) 을 활용하여, 마치 사람이 평가하는 것처럼 주어진 결과물들을 특정 기준에 따라 점수화하는 기법입니다. 주목할 점은, 모델이 매긴 점수의 이유를 직접 설명해 주어 해석이 더 쉬워진다는 것입니다.



이전 LLM 시대의 대표적인 평가 지표인 *Recall-Oriented Understudy for Gisting Evaluation* (ROUGE) 처럼 참조 텍스트 (기준 답변) 이 필요한 방식과 달리, LaaJ 는 어떠한 참조 텍스트도 필요 없습니다. 그래서 거의 모든 유형의 과제에 손쉽게 평가를 적용할 수 있습니다. 특히, LaaJ 가 강력한 대형 모델 (예: GPT-4) 에 의존할 경우, 필요한 추론 능력을 갖추고 있어 인간 평가와도 높은 상관관계를 보이는 것으로 확인되었습니다.

비고: *LaaJ* 기법은 평가 작업을 보다 빠르고 효율적으로 수행할 수 있으나, 사람의 평가 결과와 일치하는지를 주의 깊게 살펴봐야 합니다. 이 둘이 크게 어긋나지 않도록 모니터링하는 과정이 중요합니다.

□ **자주 나타나는 편향 사례** – LaaJ 모델에서는 다음과 같은 편향들이 흔히 발생할 수 있습니다:

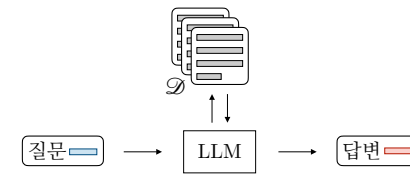
	위치 편향	상황성 편향	자기향상 편향
문제	쌍으로 비교할 때, 첫 번째 응답에 치우쳐서 선호함	길고 자세한 답변을 선호함	모델이 자신이 생성한 답변을 더 좋게 평가함
해결책	여러 위치로 무작위 배치 후 결과를 평균 냄	답변 길이에 패널티를 둠	다른 모델로 만든 judge 를 사용해 교차 검증

이러한 편향을 해결하는 한 가지 방법은 맞춤형 LaaJ 를 파인 튜닝하는 것이지만, 이는 많은 작업이 필요합니다.

비고: 위 표에 나열된 편향 사례가 전부는 아닙니다.

4.2 RAG

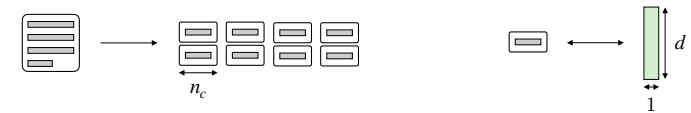
□ **정의** – 검색 증강 생성 (*Retrieval-Augmented Generation*, RAG) 이란 LLM 이 주어진 질문에 답변 하기 위해, 관련된 외부 지식에 접근할 수 있도록 해주는 기법입니다. 이는 특히 LLM 이 사전 학습 (프리트레이닝) 을 마친 시점 이후에 새로 생긴 정보를 반영해야 할 때 매우 유용합니다.



지식 베이스 \mathcal{D} 와 질문이 주어졌을 때, 리트리버 (retriever) 는 가장 관련성 높은 문서를 찾아옵니다. 그런 다음, 선택된 정보를 프롬프트에 추가하고 최종 답변을 생성합니다.

비고: 문서를 찾아오는 단계는 대개 인코더 전용 (*encoder-only*) 모델에서 얻은 임베딩을 활용합니다.

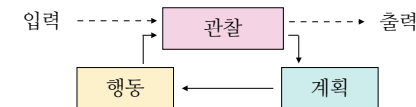
□ **하이퍼파라미터** – 지식 베이스 \mathcal{D} 는 문서를 사이즈가 n_c 인 덩어리로 쪼갬 뒤, 각 덩어리를 \mathbb{R}^d 차원의 벡터로 임베딩합니다.



4.3 에이전트

□ **정의** – 에이전트 (agent) 는 사용자를 대신해 자동으로 목표를 추구하고, 작업을 완료해주는 시스템입니다. 이를 위해 여러 단계의 LLM 호출을 활용하기도 합니다.

□ **ReAct** – *Reason + Act* (ReAct) 는 복잡한 작업을 완료하기 위해 여러 고리의 LLM 을 호출을 가능하게 하는 프레임워크 입니다.



이 프레임워크는 다음과 같은 구성 요소들이 있습니다:

- 관찰: 이전에 실행한 작업들을 정리하고, 현재 알고 있는 사실을 명시적으로 기술합니다.

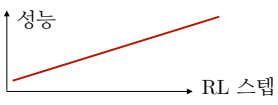
- 계획: 해결해야 할 문제가 무엇인지 구체화하고, 어떤 툴을 호출할지 결정합니다.
 - 실행: 필요한 API 를 호출하거나, 지식 베이스 (knowledge base) 에서 관련 정보를 찾아봅니다.
- 비고: 에이전트 시스템을 평가하는 일은 쉽지 않습니다. 그럼에도 불구하고, 구성 요소별 입출력 (로컬 수준) 과 여러 호출 과정을 거치는 전체 시스템 (체인 수준) 모두에서 평가가 가능합니다.

4.4 추론 모델

□ 정의 – 추론 모델은 CoT 방식의 추론 과정을 바탕으로 수학, 코딩, 논리와 같은 더 복잡한 문제를 해결하는 모델입니다. 예로는 OpenAI 의 o 시리즈, DeepSeek-R1, Google 의 Gemini Flash Thinking 등이 있습니다.

비고: DeepSeek-R1 모델은 <think> 태그 안에 추론 과정을 명시적으로 출력합니다.

□ 스케일링 – 추론 능력을 향상하기 위해 다음 두 가지 스케일링 방법을 사용합니다:

	설명	일러스트레이션
학습 단계 스케일링	강화 학습 (RL) 과정을 더 길게 진행하여, 모델이 답변을 내기 전에 CoT 스타일의 추론과정을 학습하도록 합니다.	
테스트 단계 스케일링	모델이 답을 내기 전에 좀 더 오래 생각하도록 "wait" 같은 키워드로 시간을 확보합니다.	