언어모델(Language Model)과 거대언어모델(Large Language Model)

Hyopil Shin
hpshin@snu.ac.kr
http://nlp.snu.ac.kr

목차

언어모델(Language Model)

Predict Next Word

- P(새빨간 거짓말) > P(새빨간 희망)
- P(이 강의는 참 재미있어요)
- P(재미있어요| 이 강의는)

Assign a Probability to a sentence

- Statistical Language Model: N-gram based -Unigram, Bigram, Trigram…
- Neural Network based: Word Embedding (Static Language Model)
- Transformer –based (Dynamic Contextual) : Autoencoding(BERT), Autoregressive(GPT)
- Large Language Model : GPT, PaLM, LLaMA, LLaMA2, HyperCloverX…

확률적 언어모델(Statistical Language Model): N-Gram

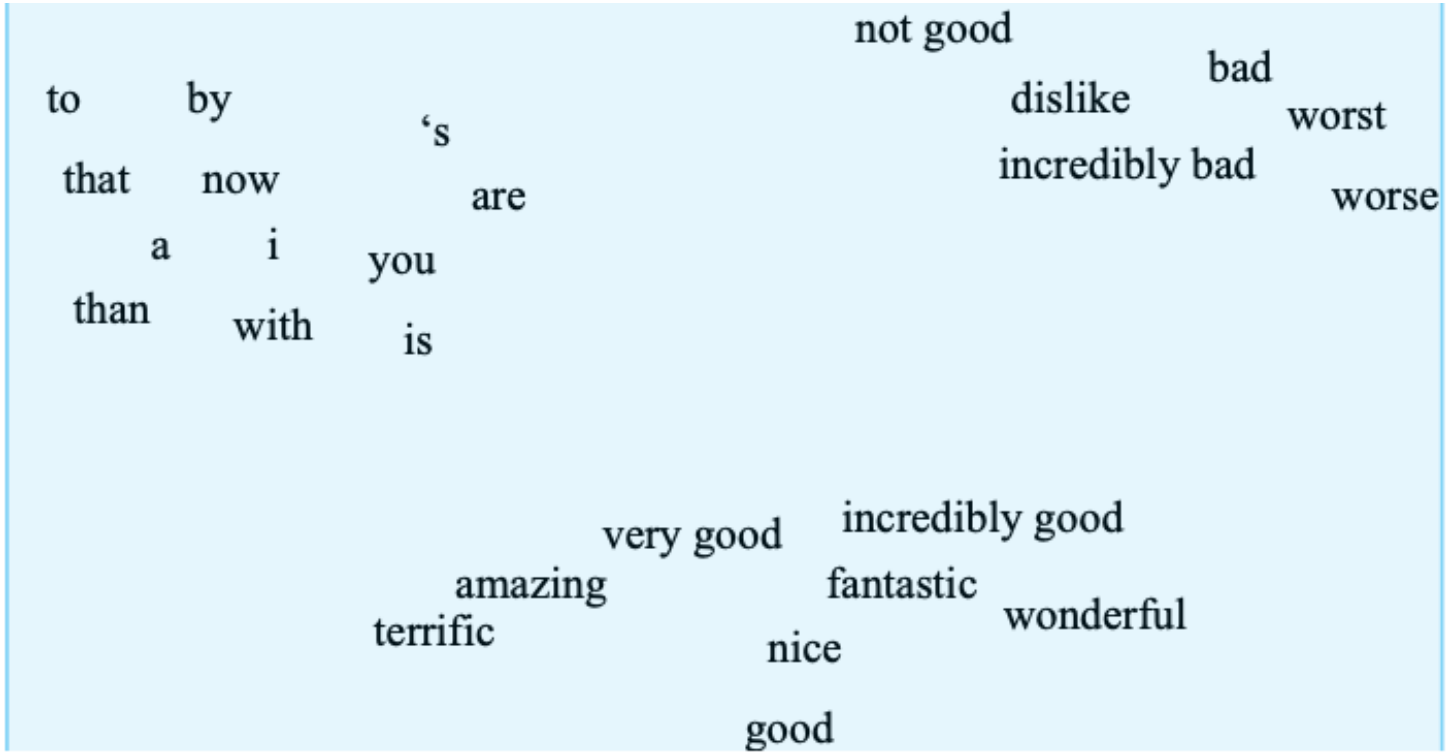
- 문장의 확률을 어떻게?
  - 조건부 확률: P(B|A) = P(A, B)/P(A) → P(A, B) = P(A)p(B|A)
    - P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)
  - Chain Rule:
    - $P(X_1, X_2, X_3, \ldots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2) \ldots P(X_n|X_1, \ldots X_{n-1})$
    - P("이 강의는 참 재미 있어요") = P(이) X P(강의는|이) X P(참|이 강의는) X P(재미 | 이 강의는 참) X P(있어요 | 이 강의는 참 재미)
    - P(있어요 | 이 강의는 참 재미) = Count(이 강의는 참 재미 있어요) / Count(이 강의는 참 재미)
- Markov Assumption
  - P(있어요| 이 강의는 참 재미) ≈ P(있어요 | 재미) 또는 P(있어요 | 참 재미)
  - P("이 강의는 참 재미 있어요") = P(이) X P(강의는|이) X P(참|강의는) X P(재미 | 참) X P(있어요 | 재미)

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

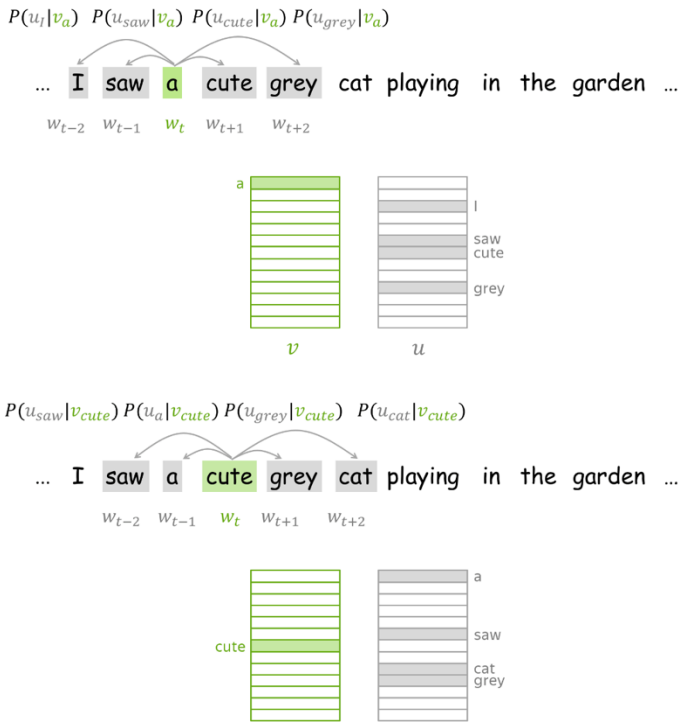
$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

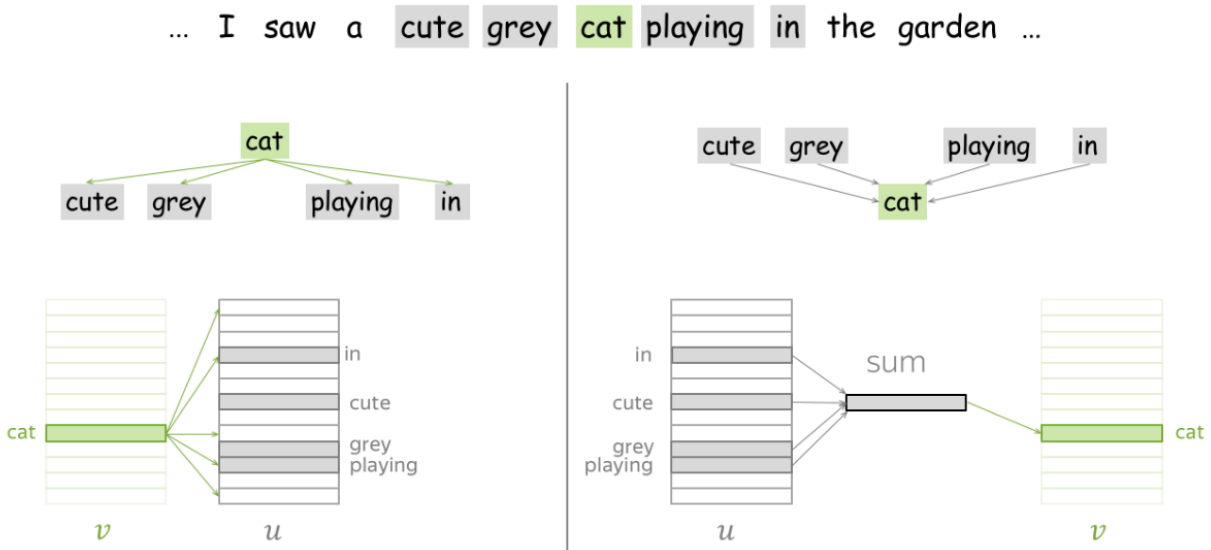
Neural Network based Language Model: 단어 임베딩(Word Embedding)

- Numericalization : 텍스트를 숫자로 바꾸는 방법
  - One-hot encoding, N-Gram 언어 모델, Bag of Words 언어 모델…
- Vector 의미론(Semantics)
  - 단어의 의미를 어떻게 표상할 수 있는가?
    - 동음이의어, 유사어, 다의어, 관련어…
- Word Embedding
  - Distributional Hypothesis
  - Word2Vec, Glove, FaxtText..
  - Static Word Embedding (다의어, 동의어 구별이 되지 않음)



Word2Vec Language Modeling: Skip-gram/Continuous bag of words

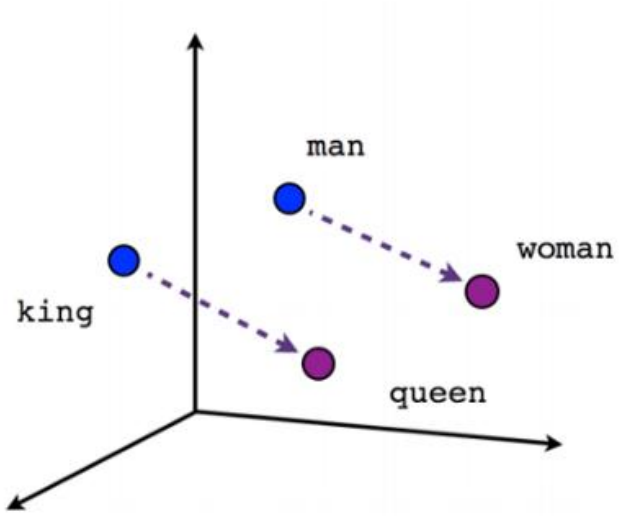




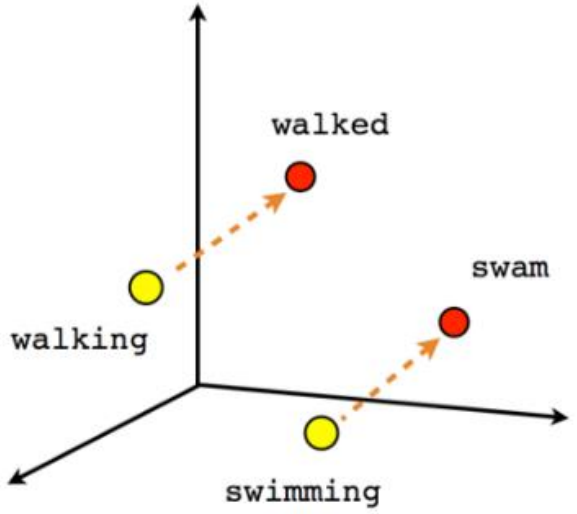
Skip-Gram: from central predict context (one at a time)

CBOW: from sum of context predict central

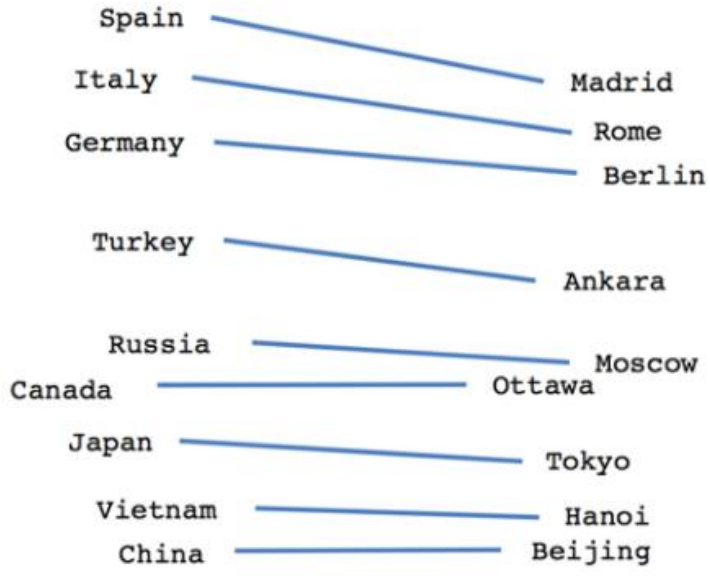
단어 임베딩



Male-Female



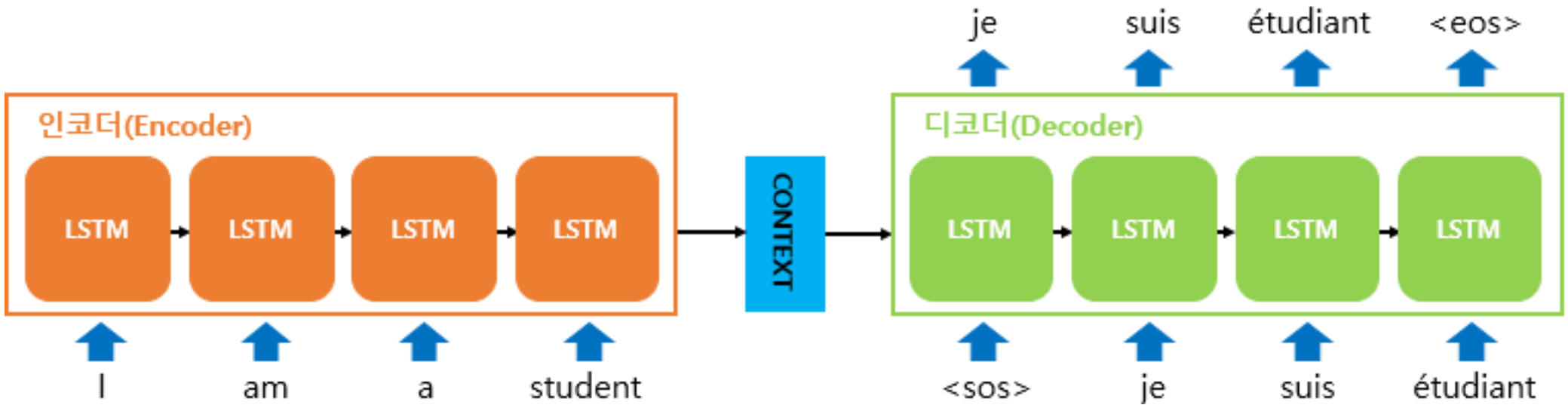
Verb tense



Country-Capital

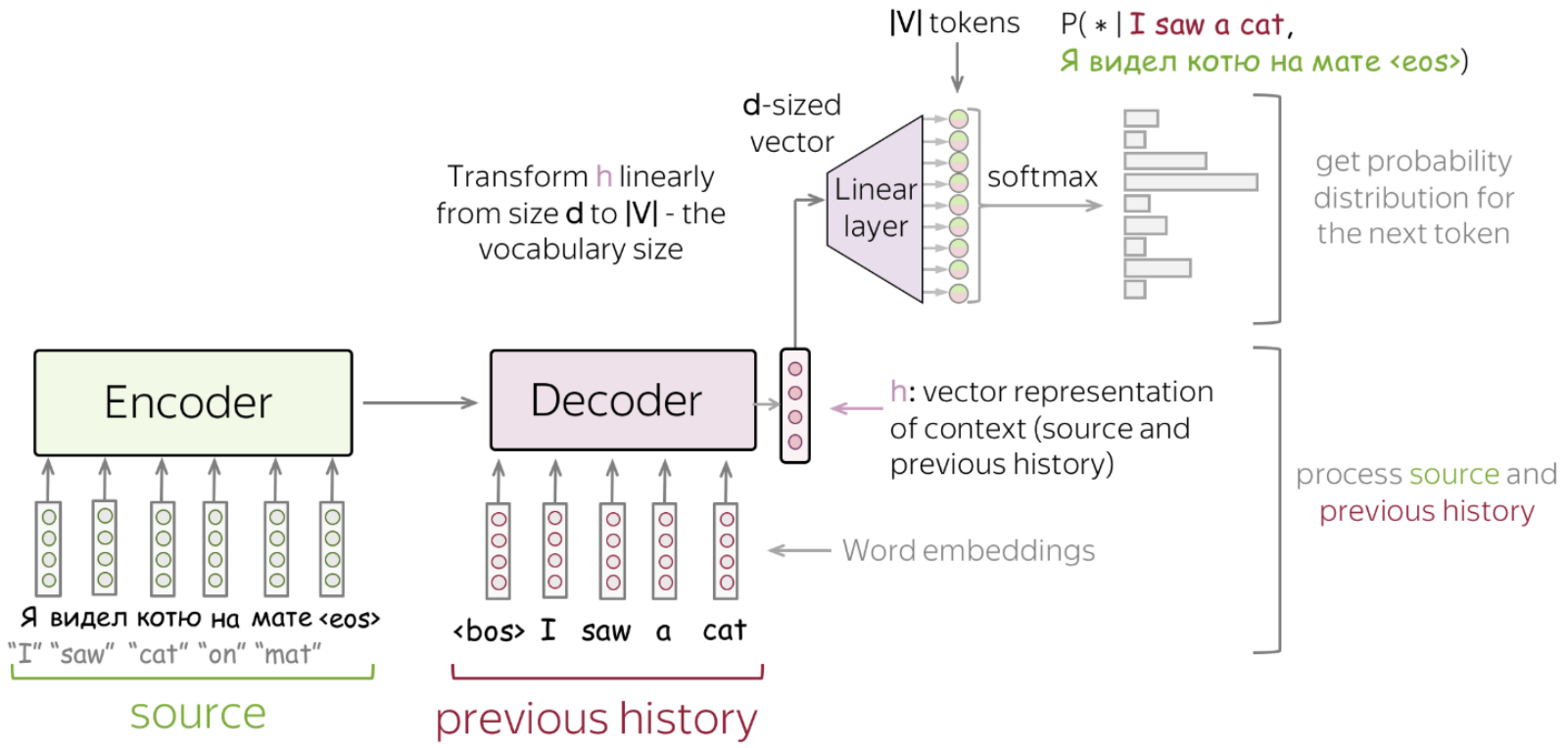
Transformer-based Language Model: Sequence To Sequence/Attention

- Sequence to sequence Model
- Encoder-Decoder



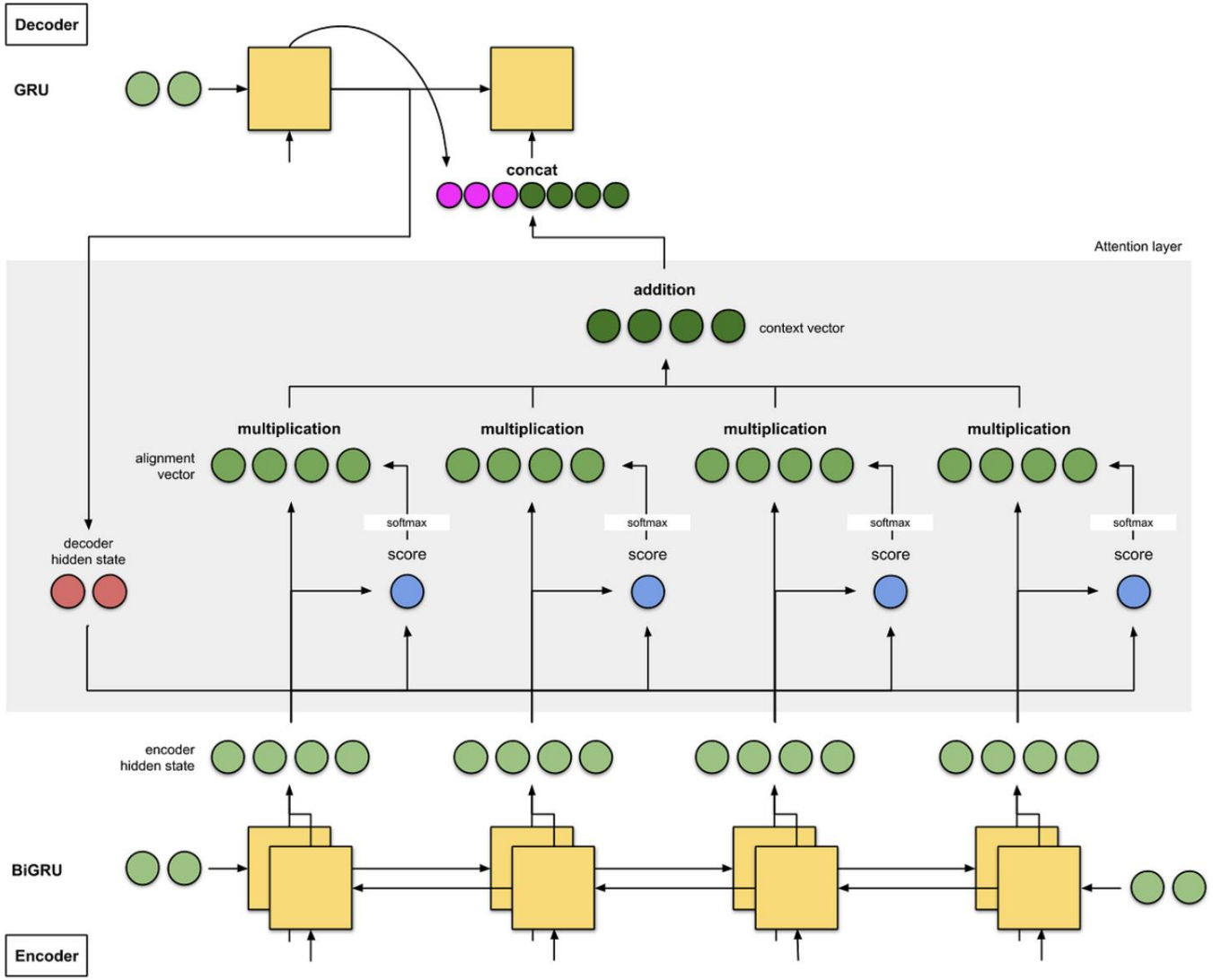
Transformer-based Language Model: Sequence To Sequence/Attention

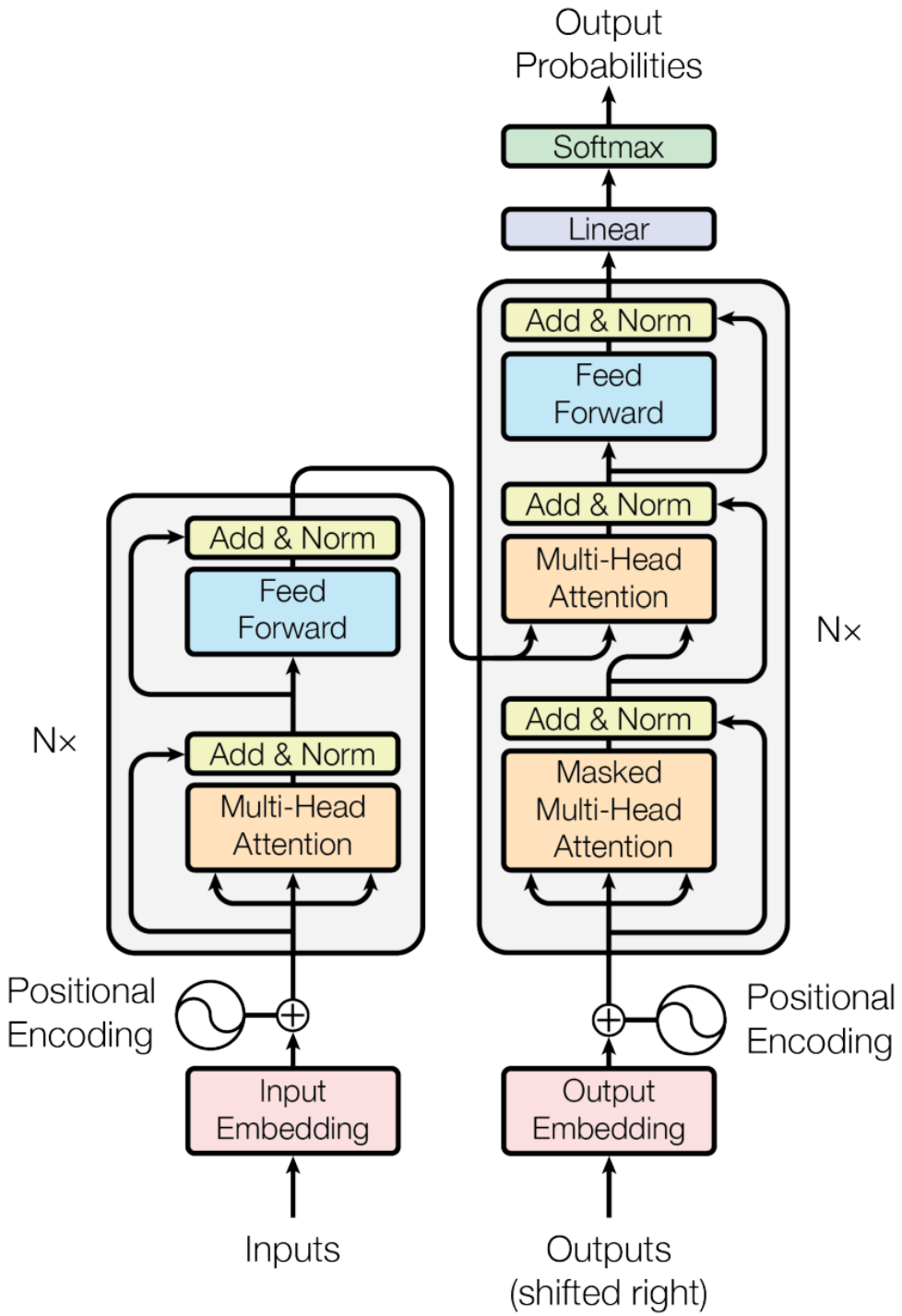
- Sequence to sequence Model
  - Encoder-Decoder



Transformer-based Language Model: Sequence To Sequence/Attention

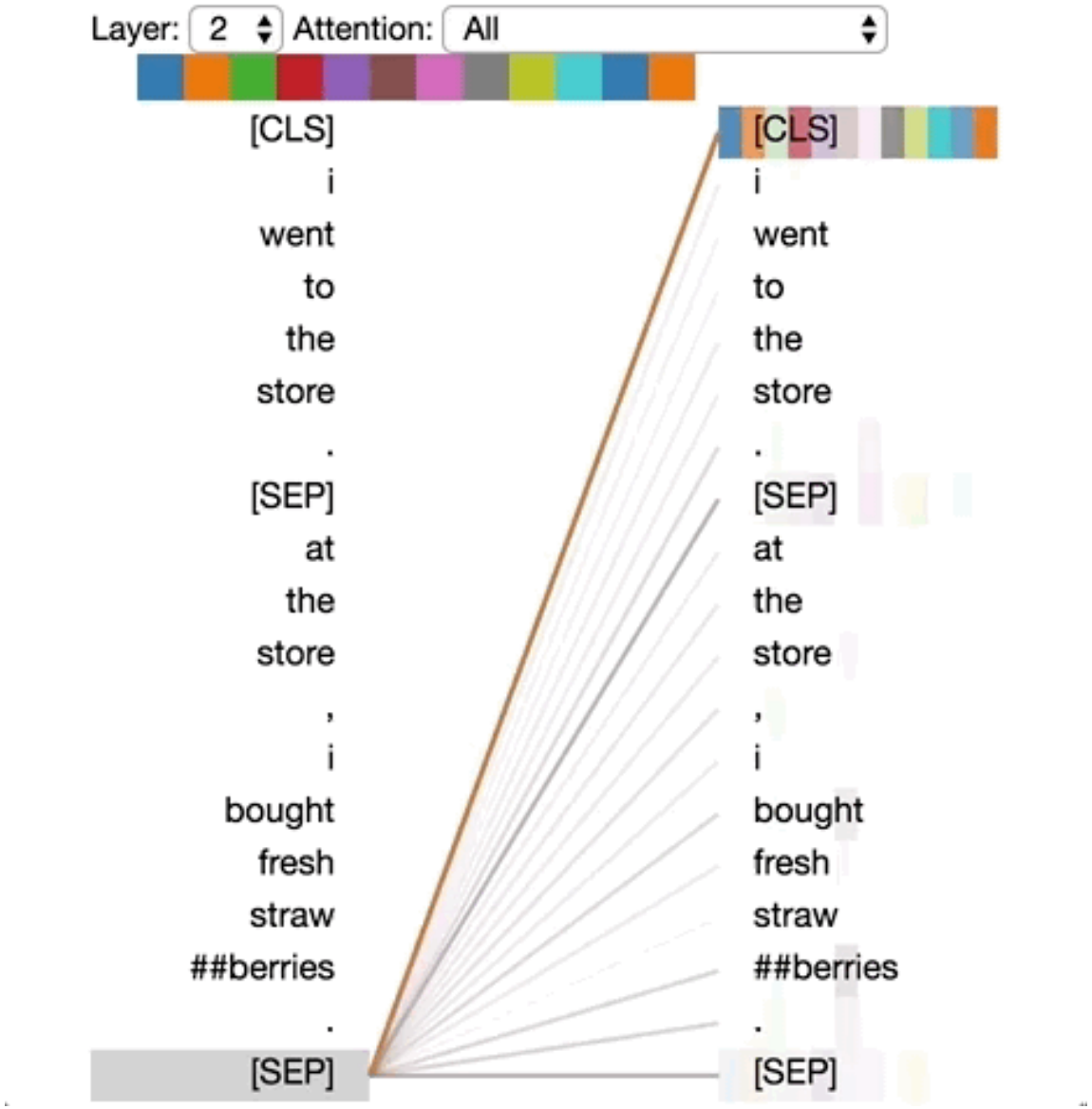
Attention





Transformer

- Parallel Processing
- Positional Embedding
- Three types of Attentions: Multi-head Self-Attention, Masked Multi-head Self-Attention, Cross Attention
- Layer Normalization
- Residual Connection



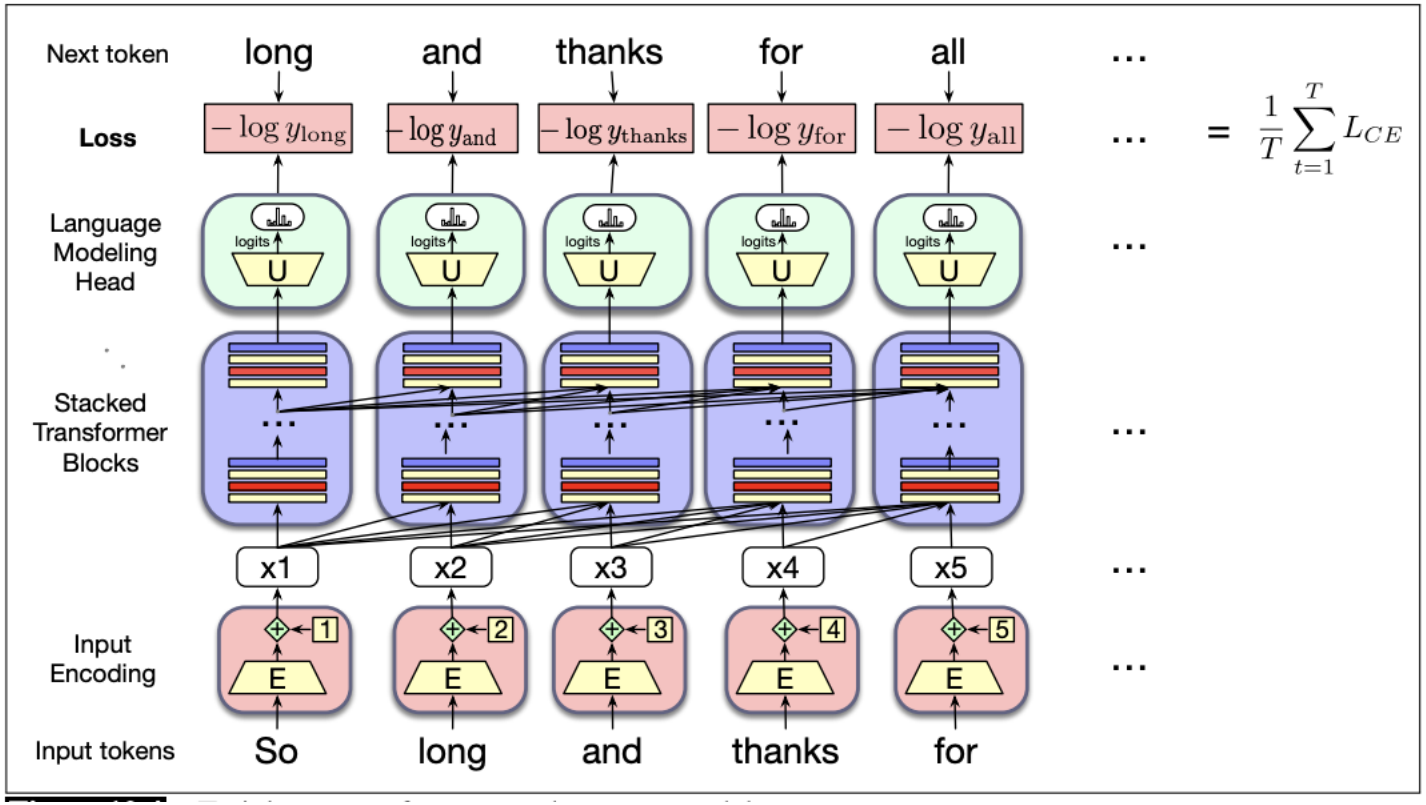
트랜스포머(Transformer)

Self Attention

https://towardsdatascience.com/deconstructing-bert-distilling-6-patterns-from-100-million-parameters-b49113672f77

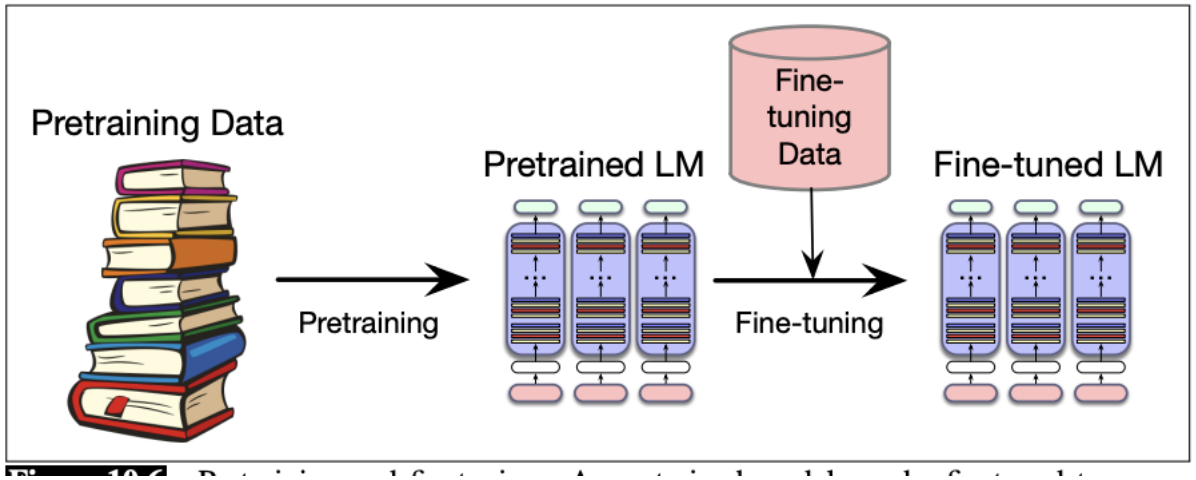
Transformer-based language model

- Self-supervised training algorithm



Finetuning

- Pretraining and Finetuning



Finetuning

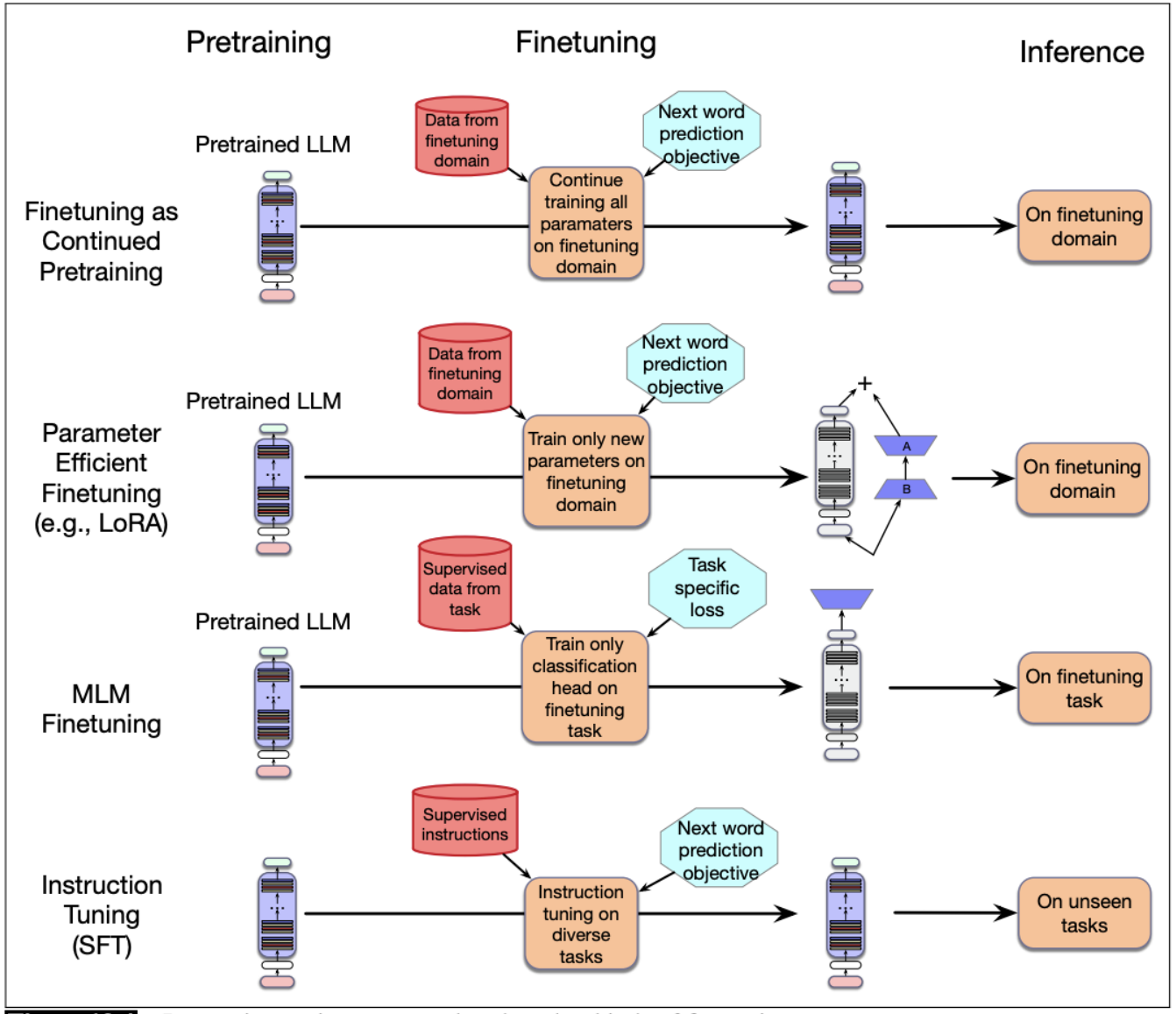
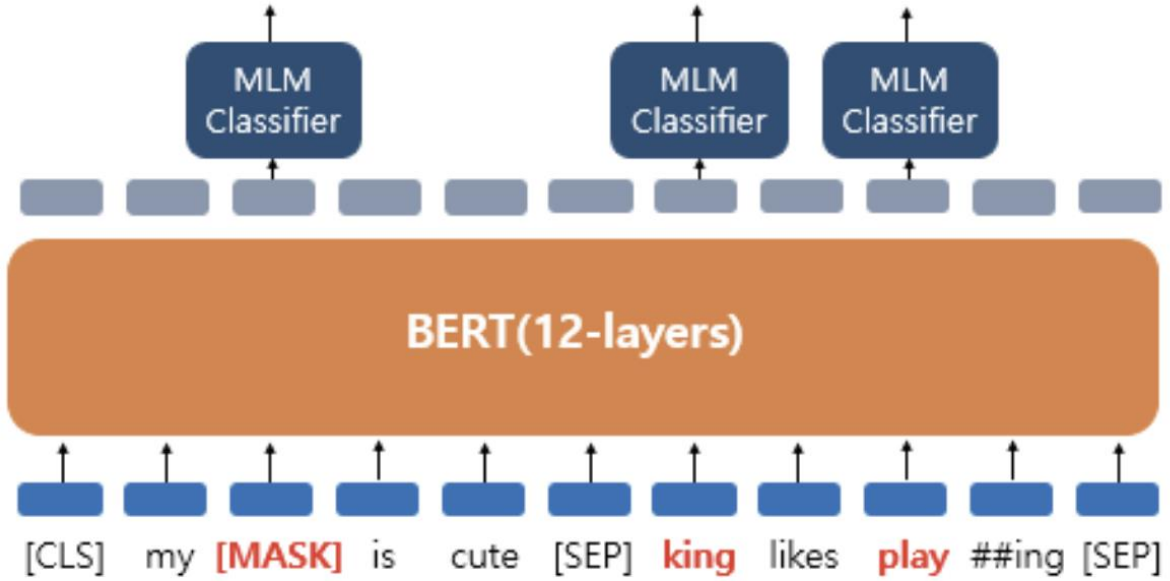


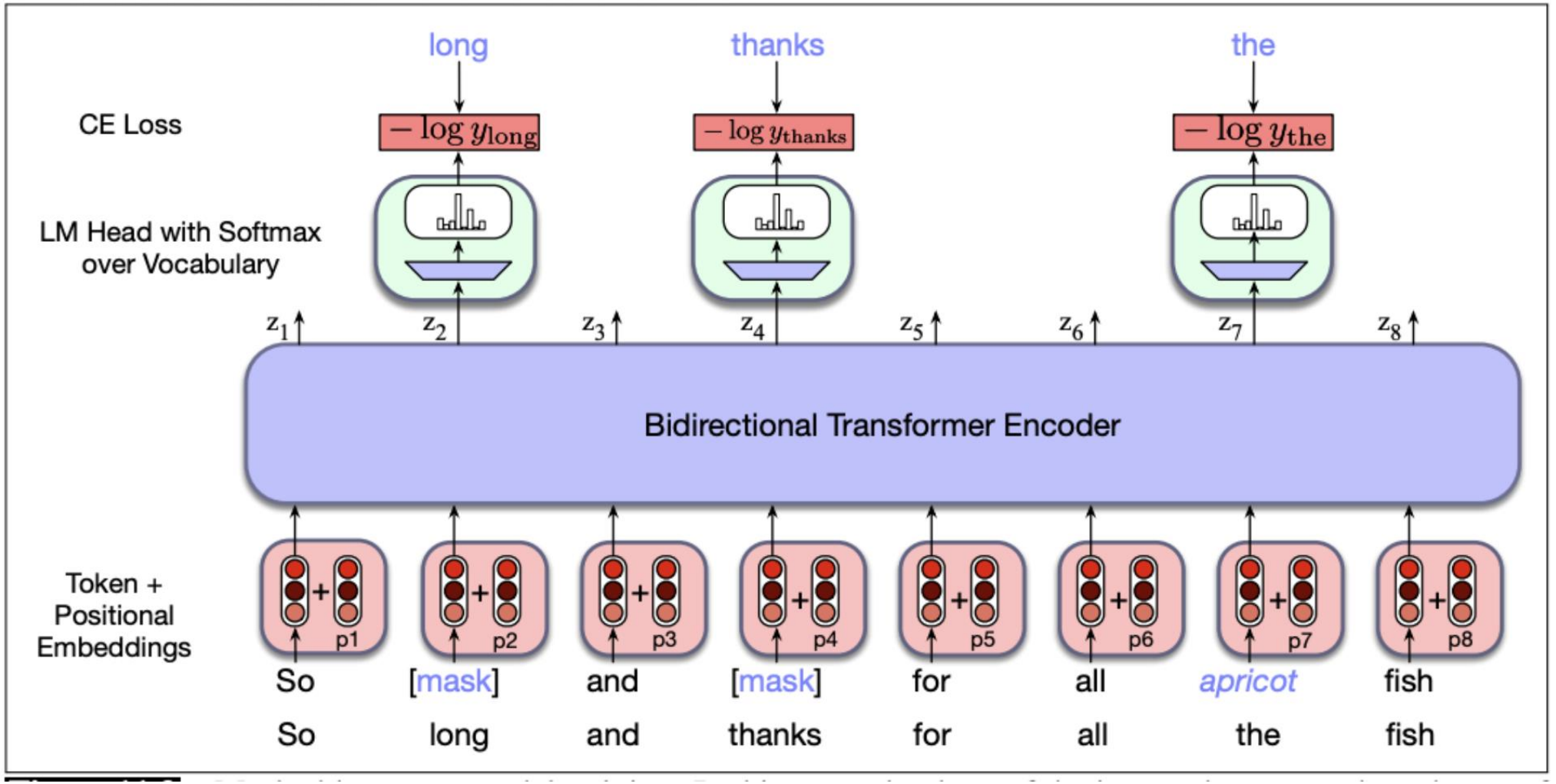
Figure 12.4 Instruction tuning compared to the other kinds of finetuning.

Masked Language Modeling(MLM)in BERT

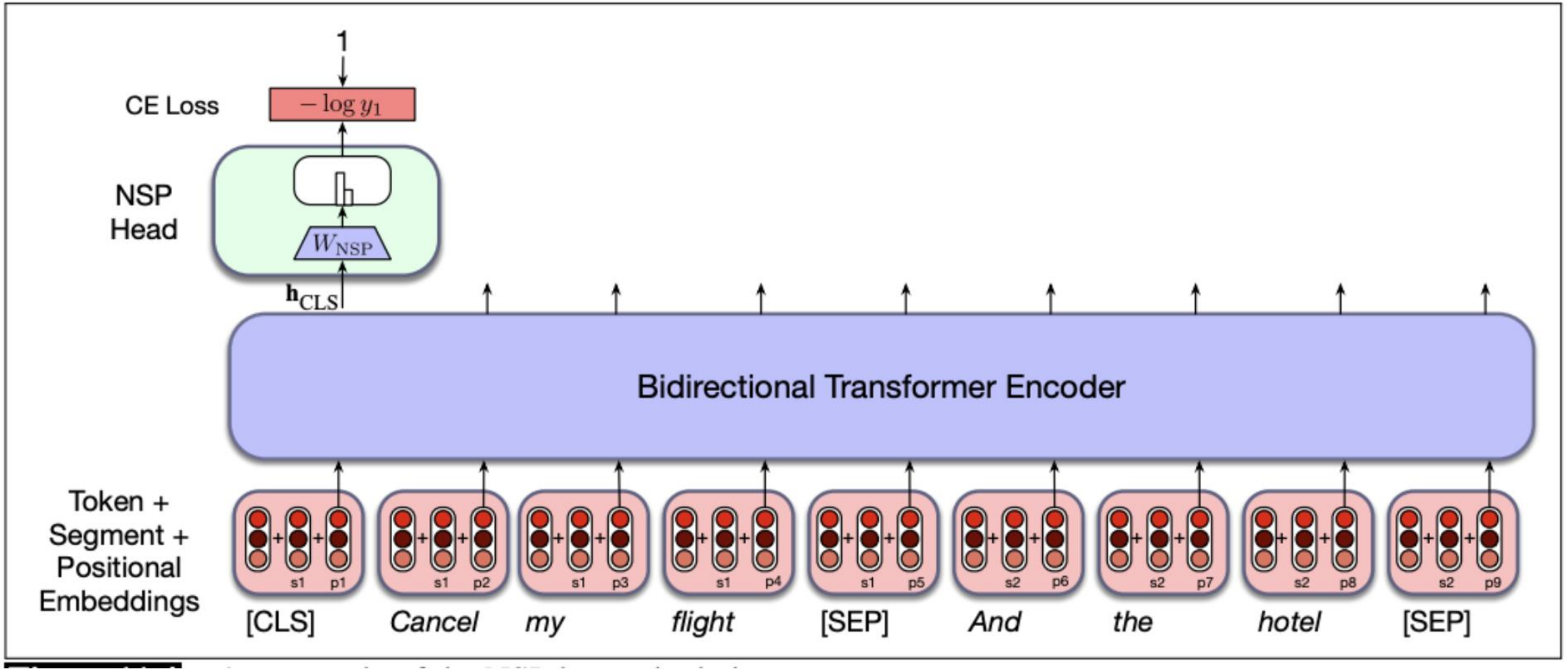
- 80% of the time the words were replaced with the masked token [MASK]
- 10% of the time the words were replaced with random words
- 10% of the time the words were left unchanged



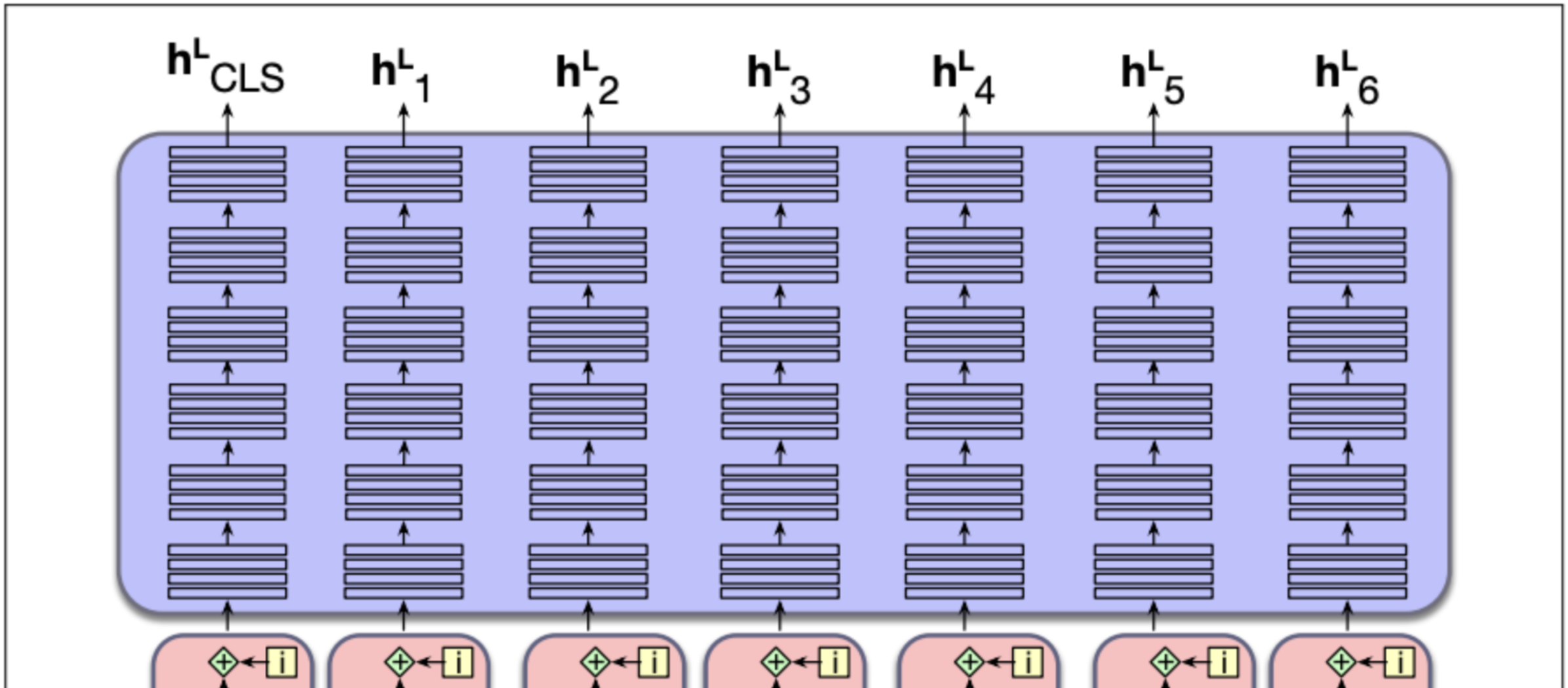
Masked Language Modeling(MLM)in BERT



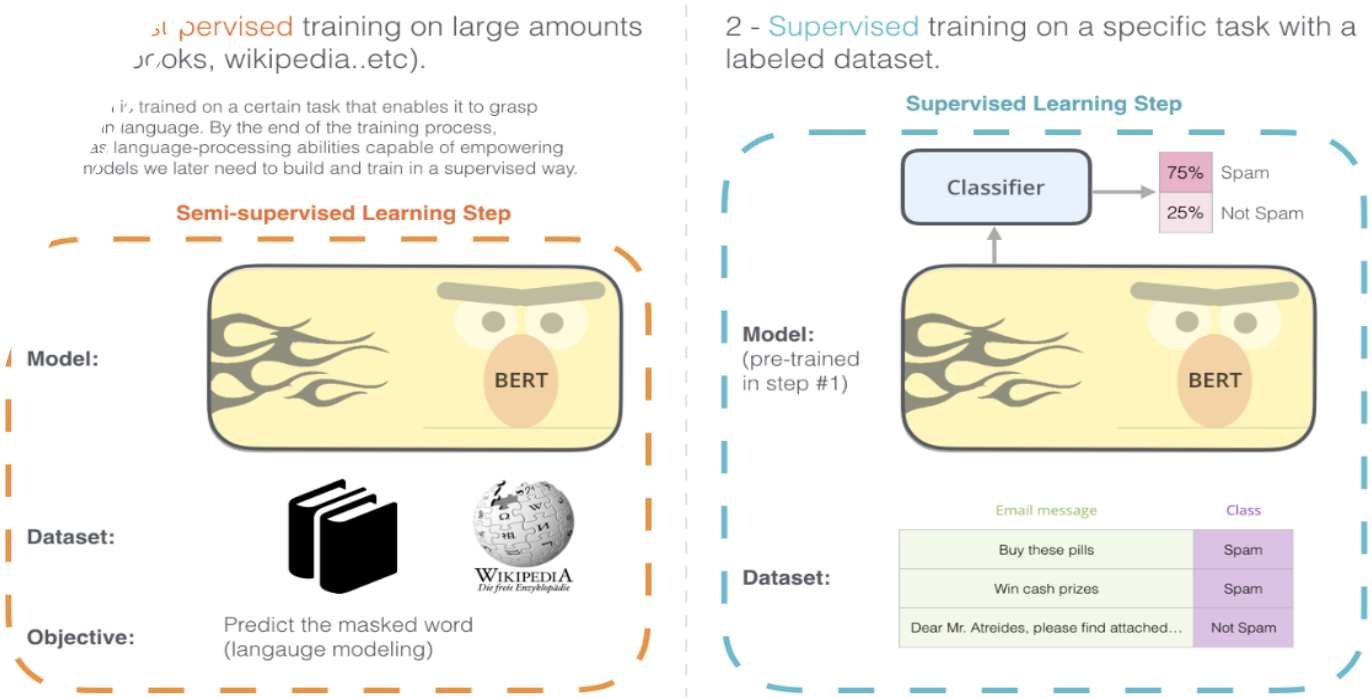
Masked Language Modeling(MLM)in BERT

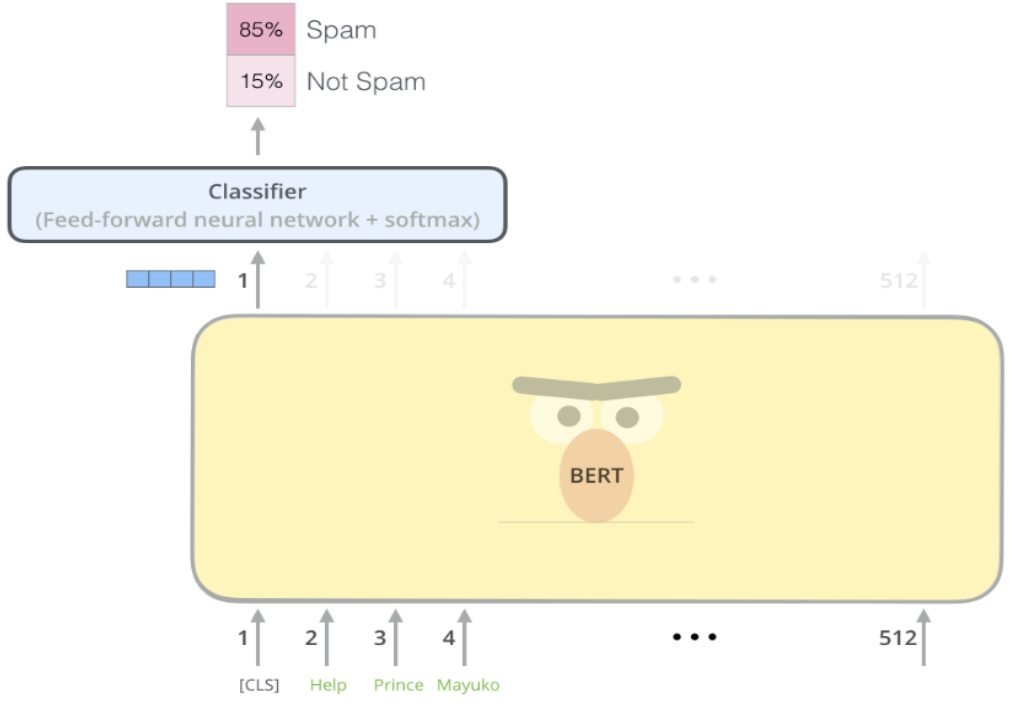


Contextual Embeddings



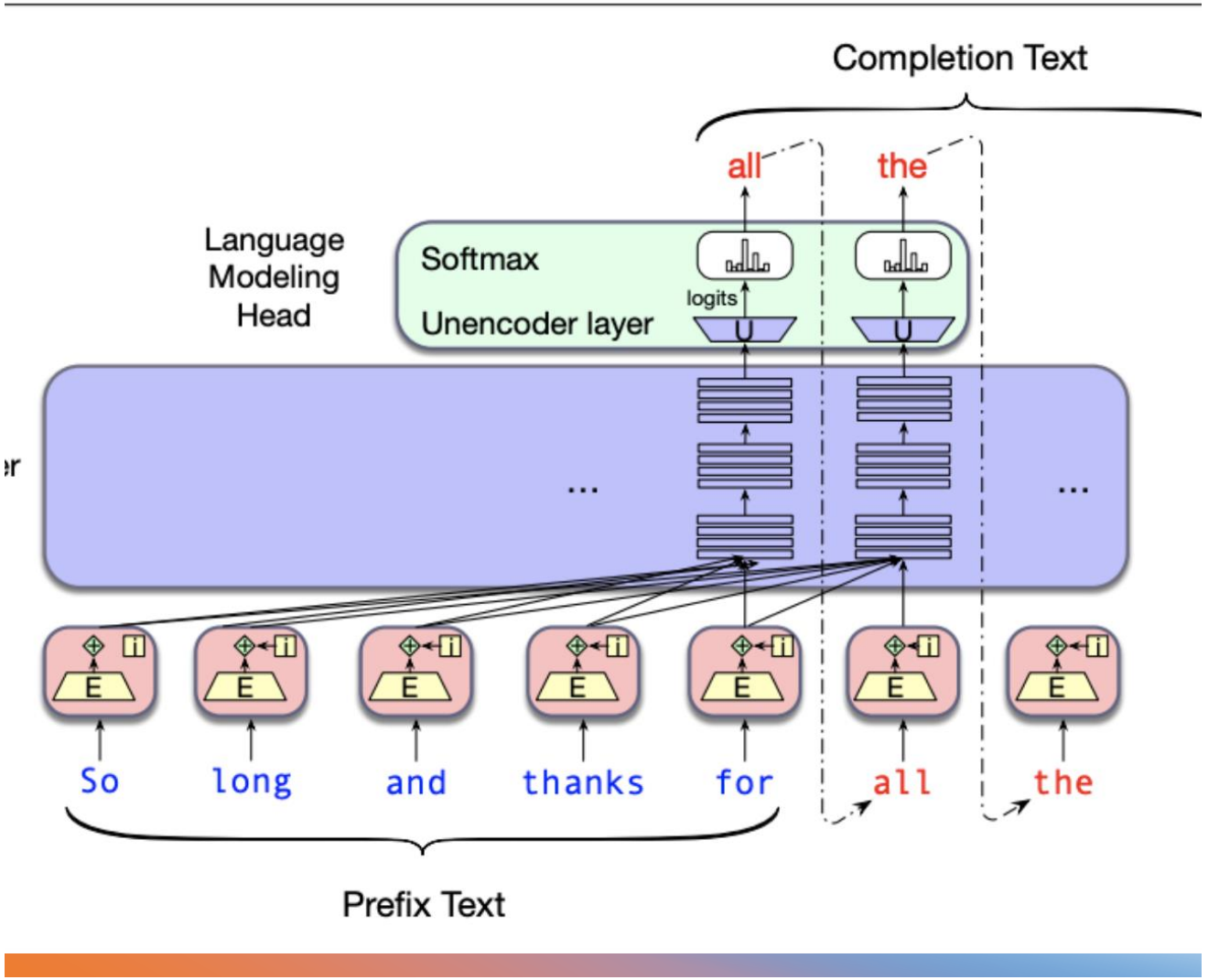
Sequence Classification in BERT





Autoregressive / Causal LM in GPT

- Conditional Generation
  - Conditional conditional generation is the task of generating text conditioned on an input piece of text.
  - That is, we give the LLM an input piece of text, generally called a **prompt**, and then have the LLM continue generating text token by token, conditioned on the prompt.
  - The fact that transformers have such long contexts (many thousands of tokens) makes them very powerful for conditional generation, because they can look back so far into the prompting text.



Why should we care about predicting upcoming words or tokens

- The insight of large language modeling is that many practical NLP tasks can be cast as **word prediction**, and that a powerful-enough language model can solve them with a high degree of accuracy

- Sentiment Analysis
  - The sentiment of the sentence ‘‘I like Jackie Chan" is:
  - P(positive|The sentiment of the sentence ‘‘I like Jackie Chan" is:) P(negative|The sentiment of the sentence ‘‘I like Jackie Chan" is:)
- Question-Answering
  - Q: Who wrote the book ‘‘The Origin of Species"? A:
  - P(w|Q: Who wrote the book ‘‘The Origin of Species"? A:) P(w|Q: Who wrote the book ‘‘The Origin of Species"? A: Charles)

Why should we care about predicting upcoming words or tokens

- Text Summarization

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

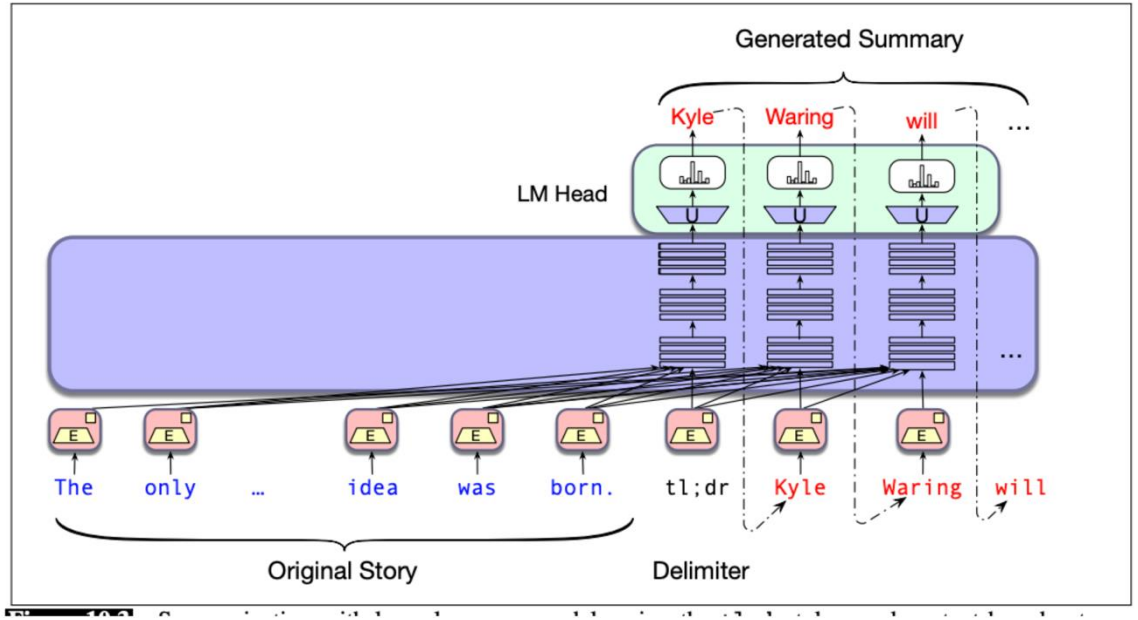
But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!"

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. [...]

Summary

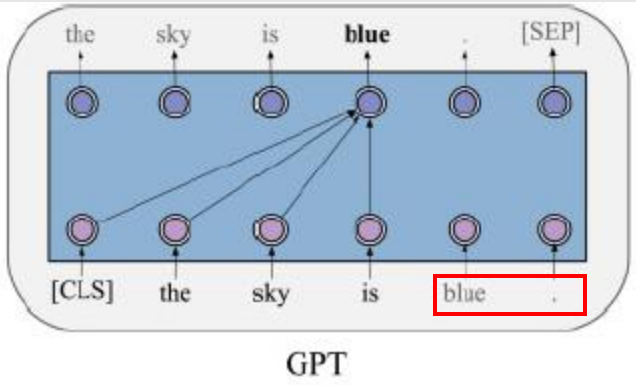
Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.



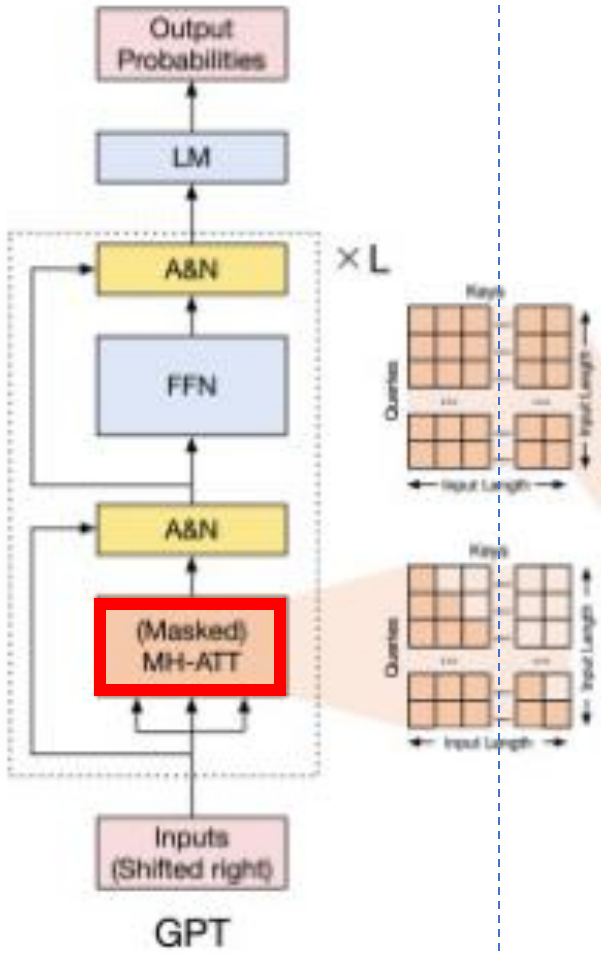
GPT

Generative Pre-trained Transformer

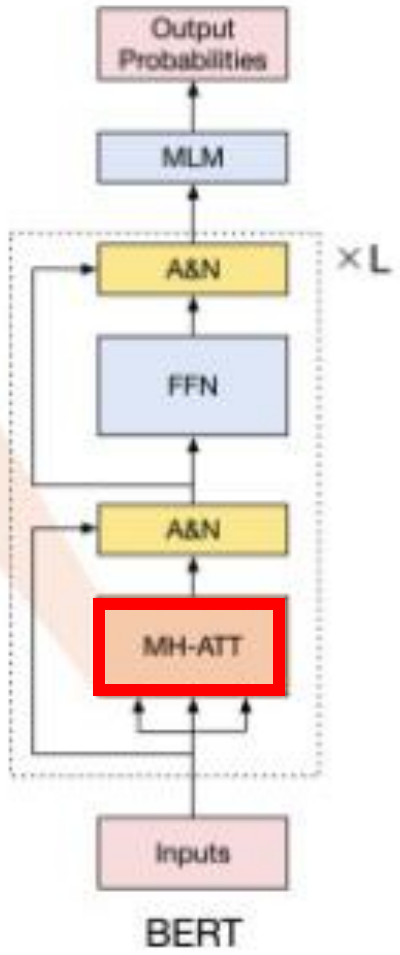
- Given large-scale corpora without labels, GPT optimizes a standard **autoregressive language modeling**, that is, maximizing the conditional probabilities of all the words by taking their previous words as contexts.
- The adaptation procedure of GPT to specific tasks is fine-tuning, by using the pre-trained parameters of GPT as a start point of downstream tasks.



GPT



GPT

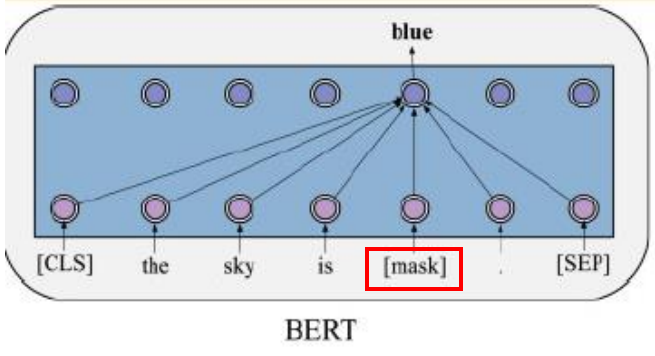


BERT

BERT

Bidirectional Encoder Representations from **Transformers**

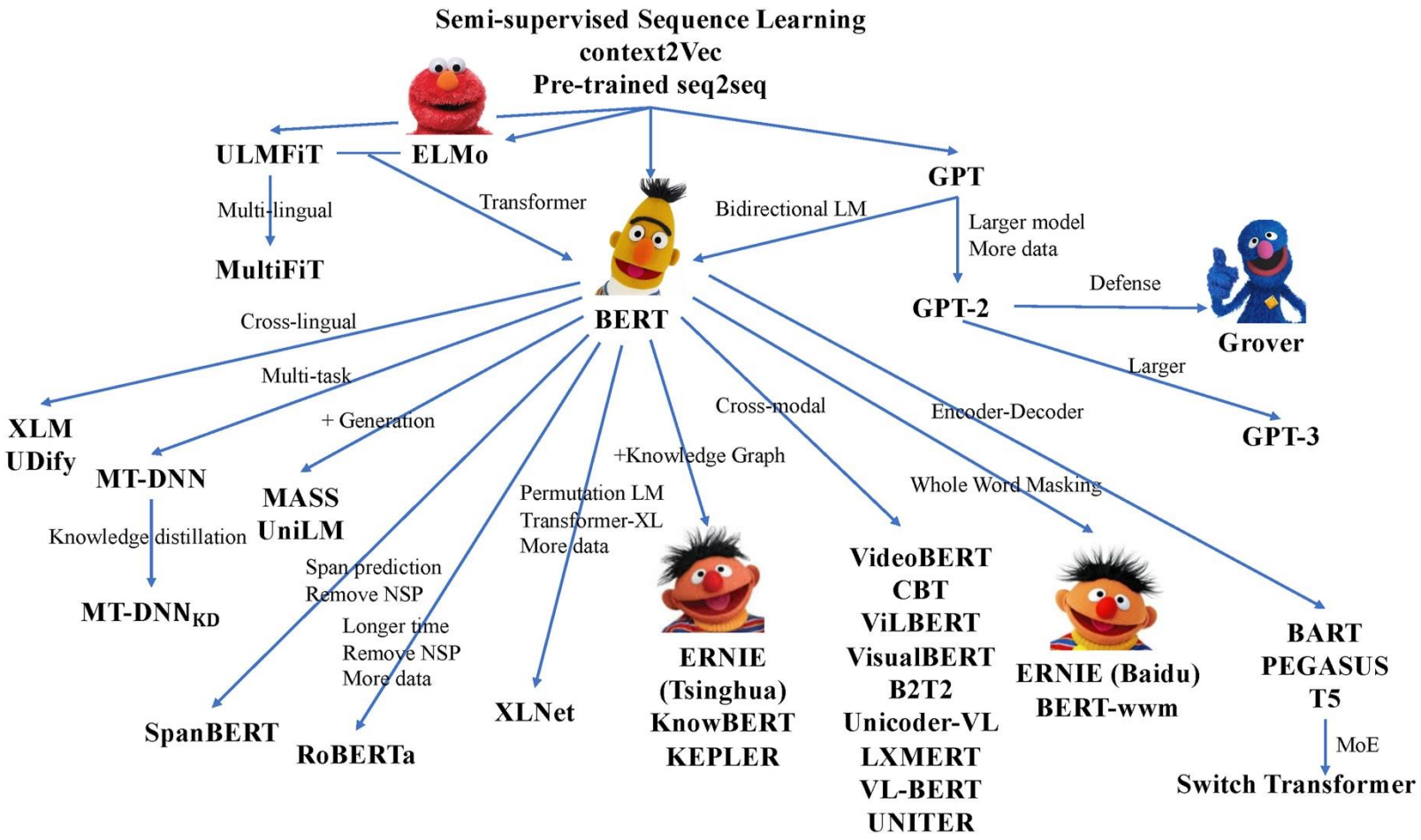
- In the pre-training phase, BERT applies **autoencoding language modeling** rather than autoregressive language modeling used in GPT. More specifically, inspired by cloze (Taylor, 1953), the objective masked language modeling (MLM) is designed.
- By modifying inputs and outputs with the data of downstream tasks, BERT could be fine-tuned for any NLP tasks.



BERT

Pre-Trained Language Model (사전학습언어모델)

https://ars.els-cdn.com/content/image/1-s2.0-S2666651021000231-gr8.jpg



거대언어모델(Large Language Model)

사전학습모델(Pre-trained Language Model)의 모델크기나 데이터 크기를 확장하면 downstream task에서 모델 용량이 향상됨

- LLM은 기존 언어모델에서 볼 수 없던 새로운 능력을 보여줌
- LLM은 인간이 AI를 개발하고 사용하는 방식에 변화를 가져옴
  - LLM에 접근하는 방식은 prompt를 통해서 주로 이루어짐
  - 따라서 LLM이 이해할 수 있는 방식으로 지시문(instruction)을 만들어야 함
- chatGPT와 GPT-4의 출현으로 Artificial General Intelligence의 가능성 모색됨

그럼에도 불구하고 LLM의 기본원리는 잘 밝혀지지 않음

- 작은 PLM이 아닌 LLM에서 새로운 기능이 발생하는지 의문
- LLM을 구축하기 위해서는 엄청난 양의 데이터와 고성능의 GPU가 대량으로 필요하기 때문에 훈련 비용이 많이 듬
  - 몇 거대 기업들만 훈련 가능한 상황
  - 구축된 모델 및 사용된 데이터셋의 비공개

LLM의 배경

- 일반적으로 거대언어모델은 트랜스포머(Transformer) 아키텍쳐를 기반으로 수천억 또는 그 이상의 parameter를 포함하는 언어모델을 지칭
  - Scaling: 거대언어모델의 능력향상은 모델의 크기에 따라 성능이 대략적으로 증가하는 scaling 법칙으로 부분적으로 설명가능.
  - Emergent Ability: “작은 모델에는 존재하지 않지만 큰 모델에서 발생하는 능력”. 규모가 일정수준에 도달하면 성능이 무작위보다 크게 상승함
    1. In-Context Learning(ICL): GPT-3에서 처음으로 도입. 언어모델에 자연어 지시 또는 여러 작업의 데모가 제공되었다면, 추가 학습이나 gradient 업데이트 없이 입력 테스트의 순서를 완성하여 예상 출력을 생성함
    2. Instruction-following: 자연어 설명을 통해 포맷된 다중 작업 데이터셋을 혼합하여 미세조정(instruction tuning)하면, LLM은 명령어 형태로 설명되는 보이지 않는 작업에서도 잘 수행됨
    3. Step-by-step Reasoning: 수학 연산, 논리적 추론 등을 위해 사고의 연쇄(Chain-of-Thoughts)를 사용하여, 최종 답을 도출하기 위한 중간 추론 단계가 포함된prompt를 활용하여 적용

LLM의 주요 기술(Key Technics)

- Scaling:
  - 더 큰 모델/데이터/학습이 모델의 성능을 향상. GPT-3 175B, PaLM 540B Parameter 사용
- Training:
  - 모델크기가 엄청나기 때문에 성공적인 LLM을 잘 학습하기가 쉽지 않음.
  - 분산훈련, 병렬 훈련(DeepSpeed, Megatron-LM) 등의 프레임 개발.
  - 최적화(optimization) 방법론 개발
- Ability Eliciting: 범용 작업능력을 끌어내기 위한 방법.
  - CoT(Chain of Thoughts), Instruction tuning...
- Alignment Tuning : 학습된 데이터에 따라 유해하거나 편향된 내용을 생성할 수 있음
  - InstructGPT: 인간의 피드백을 통한 강화학습(RLHF)을 적용하여 LLM에 예상되는 지침을 따르도록 함
- Tools manipulation: LLM은 방대한 일반 텍스트 말뭉치에 대해서 텍스트 생성기를 위해 훈련되었기 때문에 텍스트 형태로 잘 표현되지 않는 작업(숫자 계산 등)에서는 성능이 떨어짐
  - 이를 해결하기 위해 외부도구를 활용해 LLM의 결함을 보완
  - 외부 계산기나, 검색 엔진을 사용, 외부 플러그인 결합
  - 이외에도 많은 다른 요인 (하드웨어 업그레이드)

List of Large Langauge Models (https://en.wikipedia.org/wiki/Large_language_model)

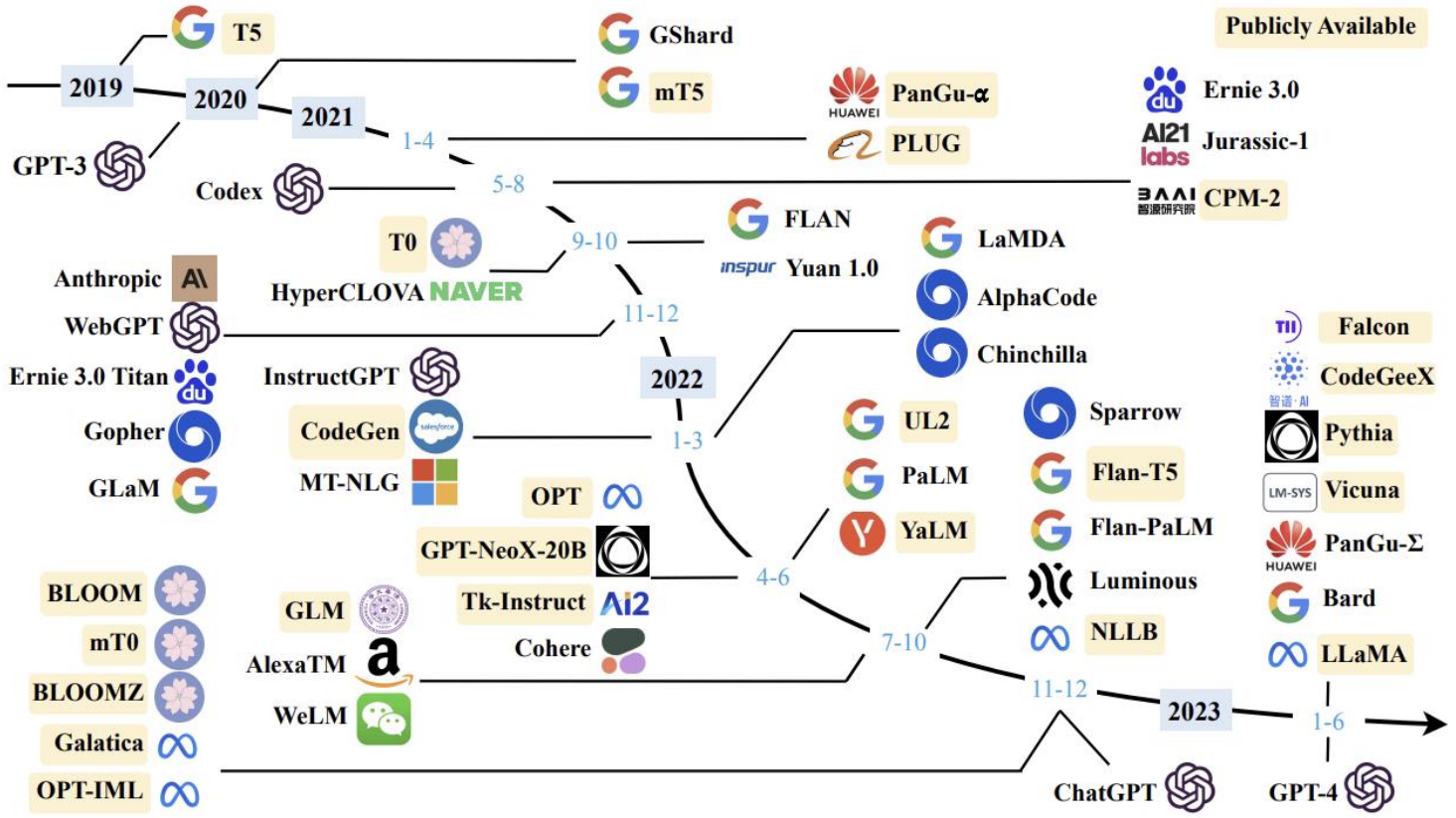
Name	Release date[a]	Developer	Number of parameters[b]	Corpus size	License[c]	Notes
BERT	2018	Google	340 million[55]	3.3 billion words[55]	Apache 2.0[56]	An early and influential language model,[2] but encoder-only and thus not built to be prompted or generative[57]
XLNet	2019	Google	~340 million[58]	33 billion words		An alternative to BERT; designed as encoder-only[59][60]
GPT-2	2019	OpenAI	1.5 billion[61]	40GB[62] (~10 billion tokens)[63]	MIT[64]	general-purpose model based on transformer architecture
GPT-3	2020	OpenAI	175 billion[24]	300 billion tokens[63]	public web API	A fine-tuned variant of GPT-3, termed GPT-3.5, was made available to the public through a web interface called ChatGPT in 2022.[65]
GPT-Neo	March 2021	EleutherAI	2.7 billion[66]	825 GiB[67]	MIT[68]	The first of a series of free GPT-3 alternatives released by EleutherAI. GPT-Neo outperformed an equivalent-size GPT-3 model on some benchmarks, but was significantly worse than the largest GPT-3.[68]
GPT-J	June 2021	EleutherAI	6 billion[69]	825 GiB[67]	Apache 2.0	GPT-3-style language model
Megatron-Turing NLG	October 2021[70]	Microsoft and Nvidia	530 billion[71]	338.6 billion tokens[71]	Restricted web access	Standard architecture but trained on a supercomputing cluster.
Ernie 3.0 Titan	December 2021	Baidu	260 billion[72]	4 Tb	Proprietary	Chinese-language LLM. Ernie Bot is based on this model.
Claude[73]	December 2021	Anthropic	52 billion[74]	400 billion tokens[74]	Closed beta	Fine-tuned for desirable behavior in conversations.[75]
GLaM (Generalist Language Model)	December 2021	Google	1.2 trillion[76]	1.6 trillion tokens[76]	Proprietary	Sparse mixture-of-experts model, making it more expensive to train but cheaper to run inference compared to GPT-3.
Gopher	December 2021	DeepMind	280 billion[77]	300 billion tokens[78]	Proprietary	
LaMDA (Language Models for Dialog Applications)	January 2022	Google	137 billion[79]	1.56T words,[79] 168 billion tokens[78]	Proprietary	Specialized for response generation in conversations.
GPT-NeoX	February 2022	EleutherAI	20 billion[80]	825 GiB[67]	Apache 2.0	based on the Megatron architecture
Chinchilla	March 2022	DeepMind	70 billion[81]	1.4 trillion tokens[81][78]	Proprietary	Reduced-parameter model trained on more data. Used in the Sparrow bot.
PaLM (Pathways Language Model)	April 2022	Google	540 billion[82]	768 billion tokens[81]	Proprietary	aimed to reach the practical limits of model scale
OPT (Open Pretrained Transformer)	May 2022	Meta	175 billion[83]	180 billion tokens[84]	Non-commercial research[d]	GPT-3 architecture with some adaptations from Megatron
YaLM 100B	June 2022	Yandex	100 billion[85]	1.7TB[85]	Apache 2.0	English-Russian model based on Microsoft's Megatron-LM.

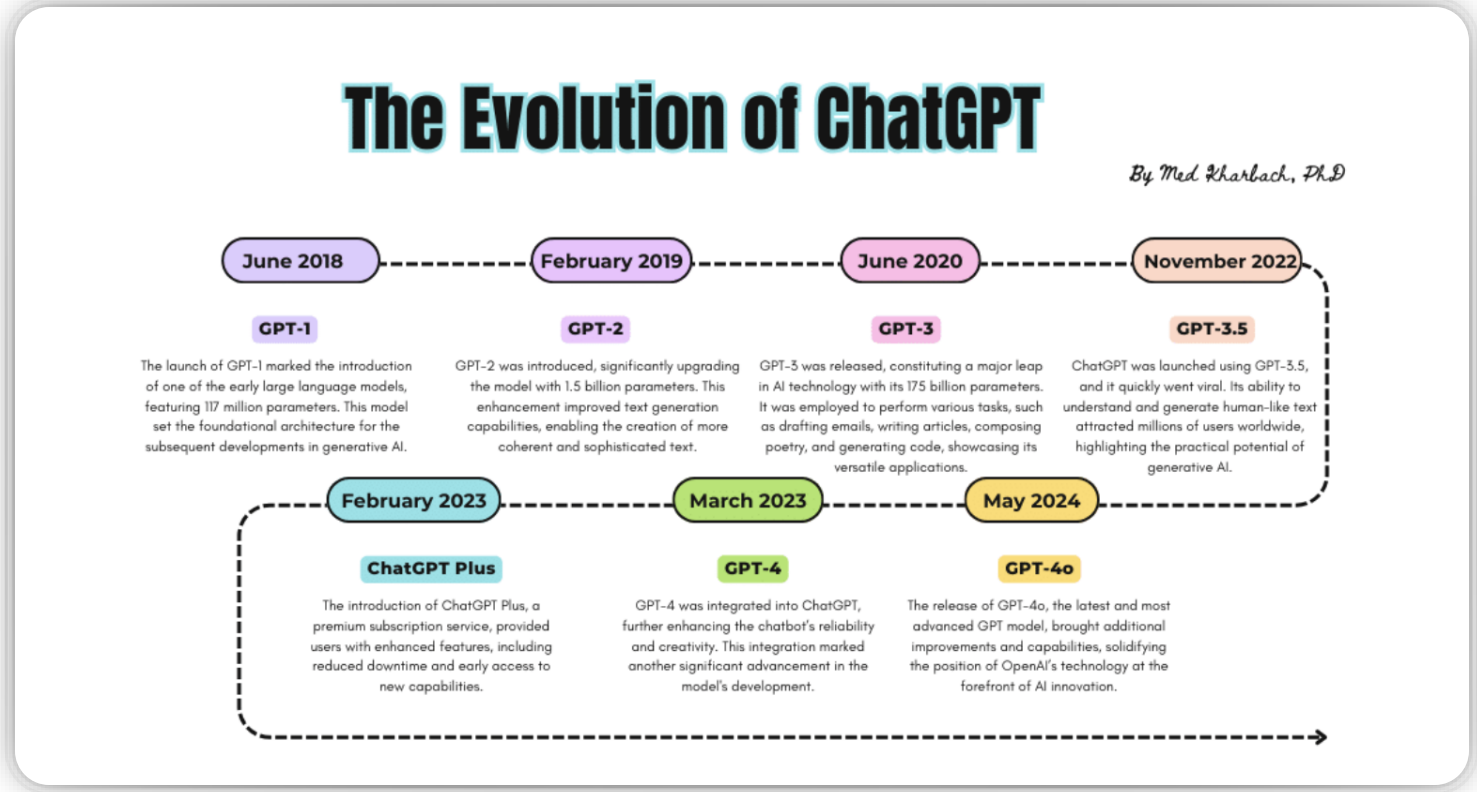
List of Large Language Models

Minerva	June 2022	Google	540 billion[86]	38.5B tokens from webpages filtered for mathematical content and from papers submitted to the arXiv preprint server[86]	Proprietary	LLM trained for solving "mathematical and scientific questions using step-by-step reasoning".[87] Minerva is based on PaLM model, further trained on mathematical and scientific data.
BLOOM	July 2022	Large collaboration led by Hugging Face	175 billion[88]	350 billion tokens (1.6TB)[89]	Responsible AI	Essentially GPT-3 but trained on a multi-lingual corpus (30% English excluding programming languages)
Galactica	November 2022	Meta	120 billion	106 billion tokens[90]	CC-BY-NC-4.0	Trained on scientific text and modalities.
AlexaTM (Teacher Models)	November 2022	Amazon	20 billion[91]	1.3 trillion[92]	public web API[93]	bidirectional sequence-to-sequence architecture
LLaMA (Large Language Model Meta AI)	February 2023	Meta	65 billion[94]	1.4 trillion[94]	Non-commercial research[e]	Trained on a large 20-language corpus to aim for better performance with fewer parameters.[94] Researchers from Stanford University trained a fine-tuned model based on LLaMA weights, called Alpaca.[95]
GPT-4	March 2023	OpenAI	Exact number unknown, approximately 1 trillion [f]	Unknown	public web API	Available for ChatGPT Plus users and used in several products.
Cerebras-GPT	March 2023	Cerebras	13 billion[97]		Apache 2.0	Trained with Chinchilla formula.
Falcon	March 2023	Technology Innovation Institute	40 billion[98]	1 Trillion tokens (1TB)[98]	Apache 2.0[99]	The model is claimed to use only 75% of GPT-3's training compute, 40% of Chinchilla's, and 80% of PaLM-62B's.
BloombergGPT	March 2023	Bloomberg L.P.	50 billion	363 billion token dataset based on Bloomberg's data sources, plus 345 billion tokens from general purpose datasets[100]	Proprietary	LLM trained on financial data from proprietary sources, that "outperforms existing models on financial tasks by significant margins without sacrificing performance on general LLM benchmarks"
PanGu-Σ	March 2023	Huawei	1.085 trillion	329 billion tokens[101]	Proprietary	
OpenAssistant[102]	March 2023	LAION	17 billion	1.5 trillion tokens	Apache 2.0	Trained on crowdsourced open data
PaLM 2 (Pathways Language Model 2)	May 2023	Google	340 billion[103]	3.6 trillion tokens[103]	Proprietary	Used in Bard chatbot.[104]

Technical Evolution of GPT-series Models

A timeline of Existing Large Language Models(Zhao et al. (2023))



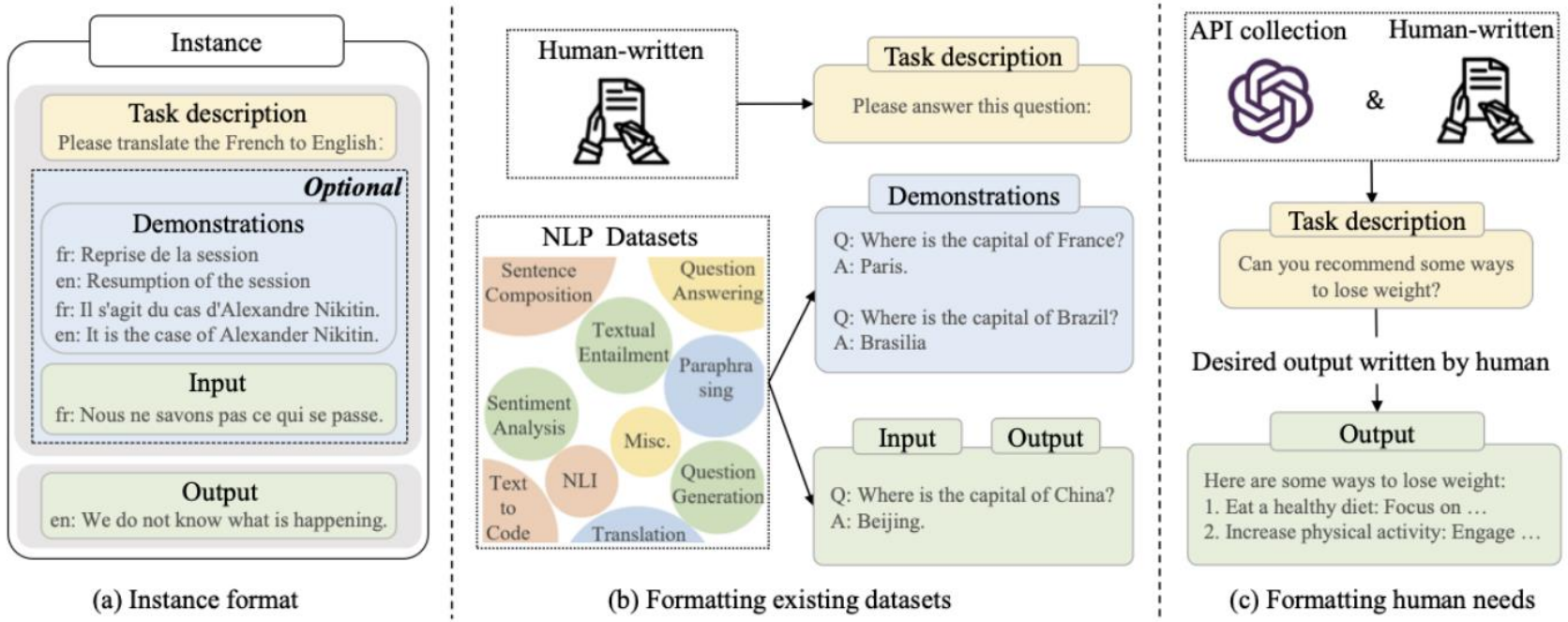


Technical Evolution of GPT-series Models

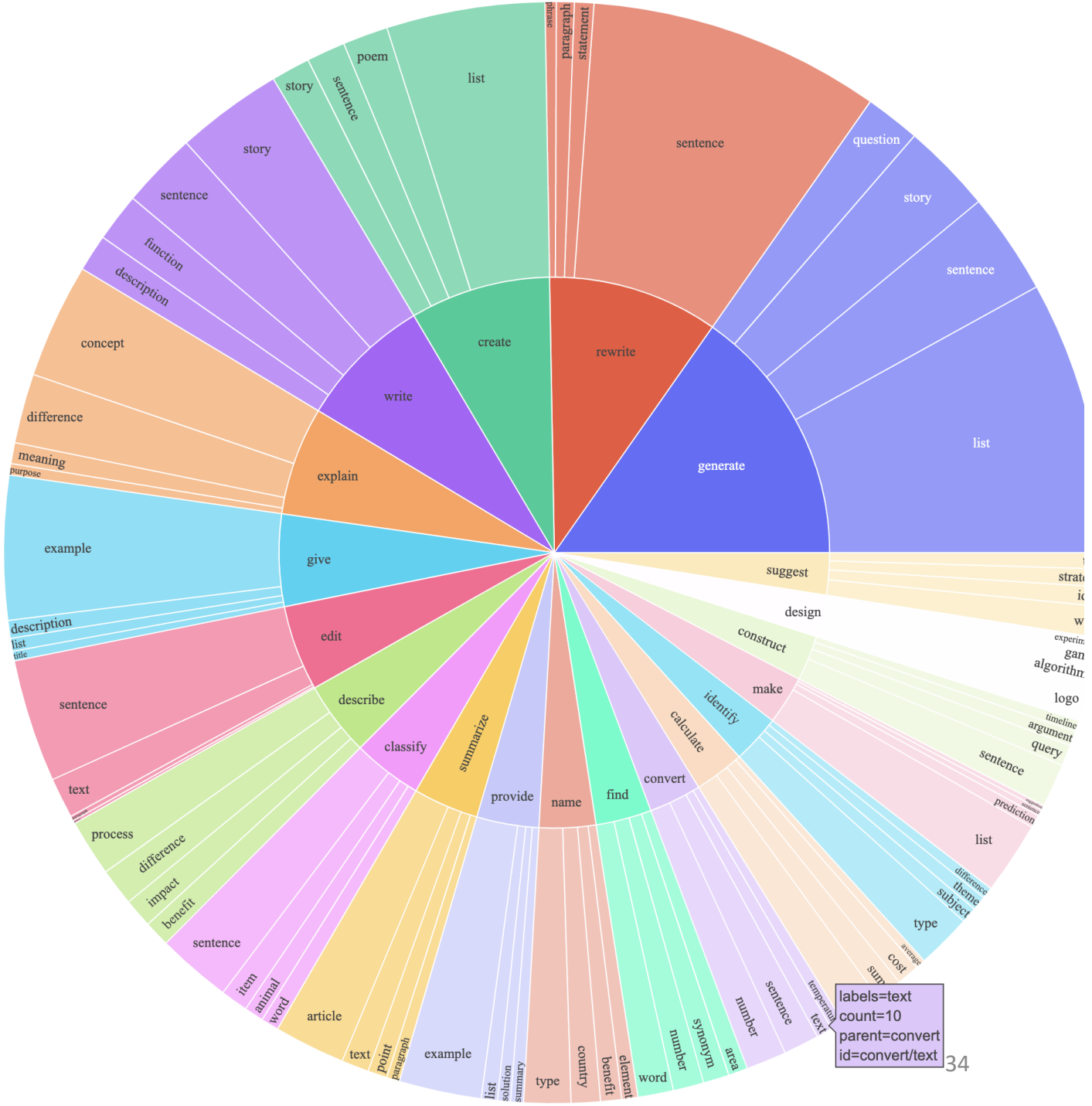
HTTPS://WWW.EDUCATORSTECHNOLOGY.COM/2024/06/THE-EVOLUTION-OF-CHATGPT.HTML

Adaptations of LLMs: Instruction Tuning

- 사전 훈련 후 LLM은 다양한 작업을 위해 일반적인 능력을 학습
  - Instruction Tuning: LLM의 능력을 향상시키는 것을 목표
  - Alignment Tuning: LLM의 행동을 인간의 가치 또는 선호도에 맞추는 것



Instruction-Following Model by Alpaca



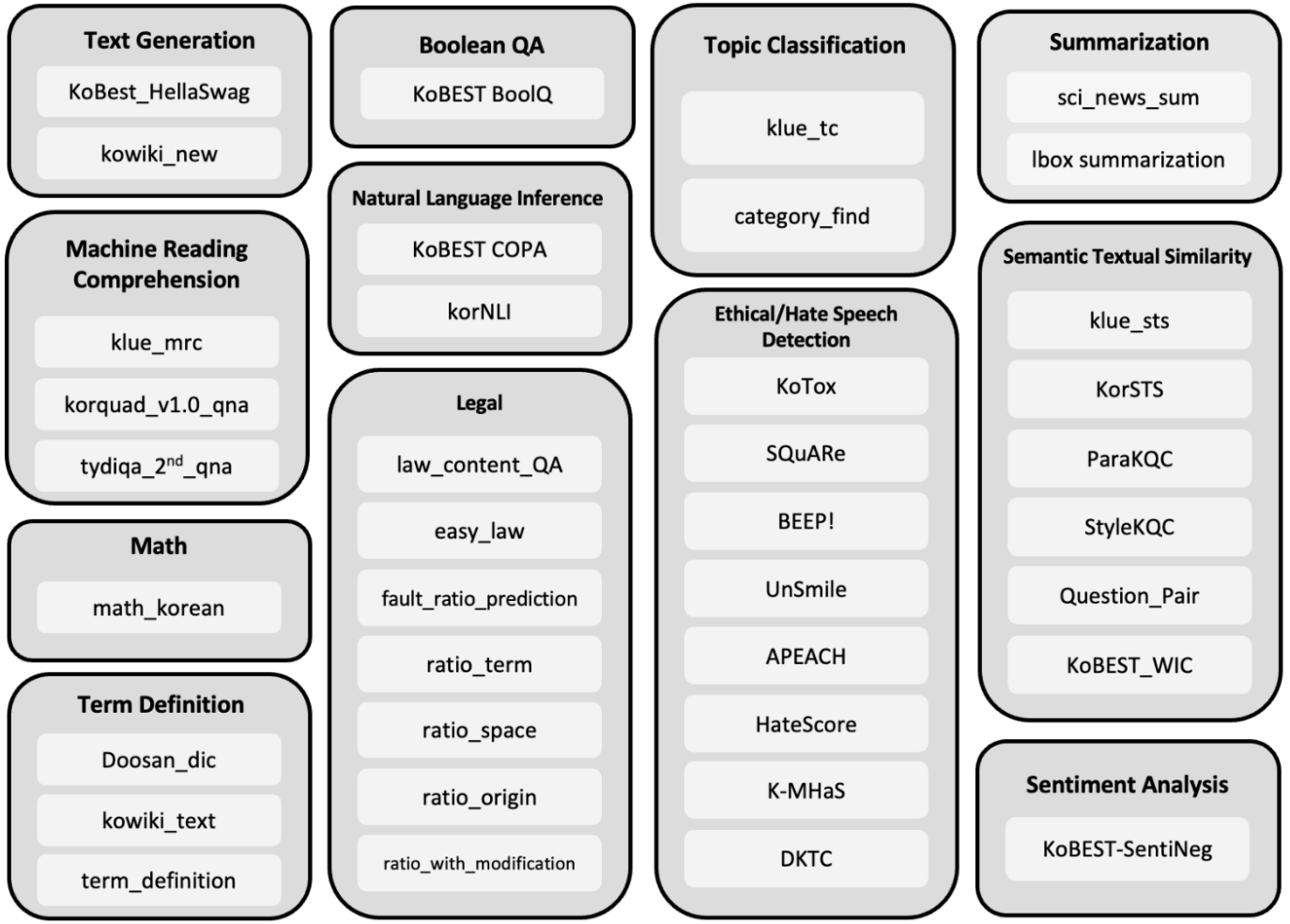
Instruction-Following (Response) Model Dataset (Alpaca)

```
...
{
    "instruction": "Describe a time when you had to make a difficult decision.",
    "input": "",
    "output": "I had to make a difficult decision when I was working as a project manager at a const
},
{
    "instruction": "Identify the odd one out.",
    "input": "Twitter, Instagram, Telegram",
    "output": "Telegram"
},
...
```

Instruction-Following (Response) Model Dataset (GPT4ALL)


```
{
    "id": "user_oriented_task_235",
    "motivation_app": "Yelp",
    "instruction": "전문 분야에 따라 레스토랑, 홈 서비스, 자동차 서비스, 기타 중 하나로 비즈니스를 분류합니다.",
    "instances": [
        {
            "input": "견적을 받으려면 650-636-4884로 전화하거나 웹사이트를 방문하세요. 이 매장은 신품 타이어 및 일반 자동
            "output": "Auto Services"
        }
    ]
},
```

Instruction DataSets in DaG



Key Factors for Instance Construction

Scaling the instructions

- 일반적으로 LLM의 task 수를 확장하면 일반화 능력을 크게 확장할 수 있다고 알려져 있으나 과제를 더 추가해도 추가적인 이득을 얻지 못할 수도 있음
- 길이, 구조, 창의성 등 여러 면에서 task 설명의 다양성을 높이는 것이 중요
- Task 당 instance의 수는 일반적으로 적은 수의 instance가 모델의 일반화 성능을 충족시키는 것으로 밝혀짐

Formatting design

- 입력문에 task description과 optional demonstration을 추가할 수 있는데, task description은 거대언어모델이 그 task를 이해하는데 가장 중요한 부분
- 적절한 수의 예제를 데모로 사용하면 성능의 개선이 이루어질 수 있음

Instance의 수보다는 Instruction의 다양성이 더 중요

특정 데이터에 한정된 task 보다는 인간이 필요로 하는 task를 레이블하는 것이 더 유용함

Improvement Strategies

Enhancing the instruction complexity

- 더 많은 taks 요구나 더 많은 추론 단계로 이루어진 복잡한 instruction을 구성하여 LLM의 모델의 성능을 향상.

Increasing the topic diversity

- 다양한 주제로 된 instance 데이터셋 구축이 모델의 성능을 향상
- 그러나 다양한 주제를 self-instruct 방법으로 구축하기는 쉽지 않음.

Scaling the instruction number

- Instruction의 수가 모델의 성능에 영향을 미치고 더 많은 instruction을 사용하면 task의 지식을 확장하게 되고 거대언어모델의 instruction의 능력을 증대시키게 됨

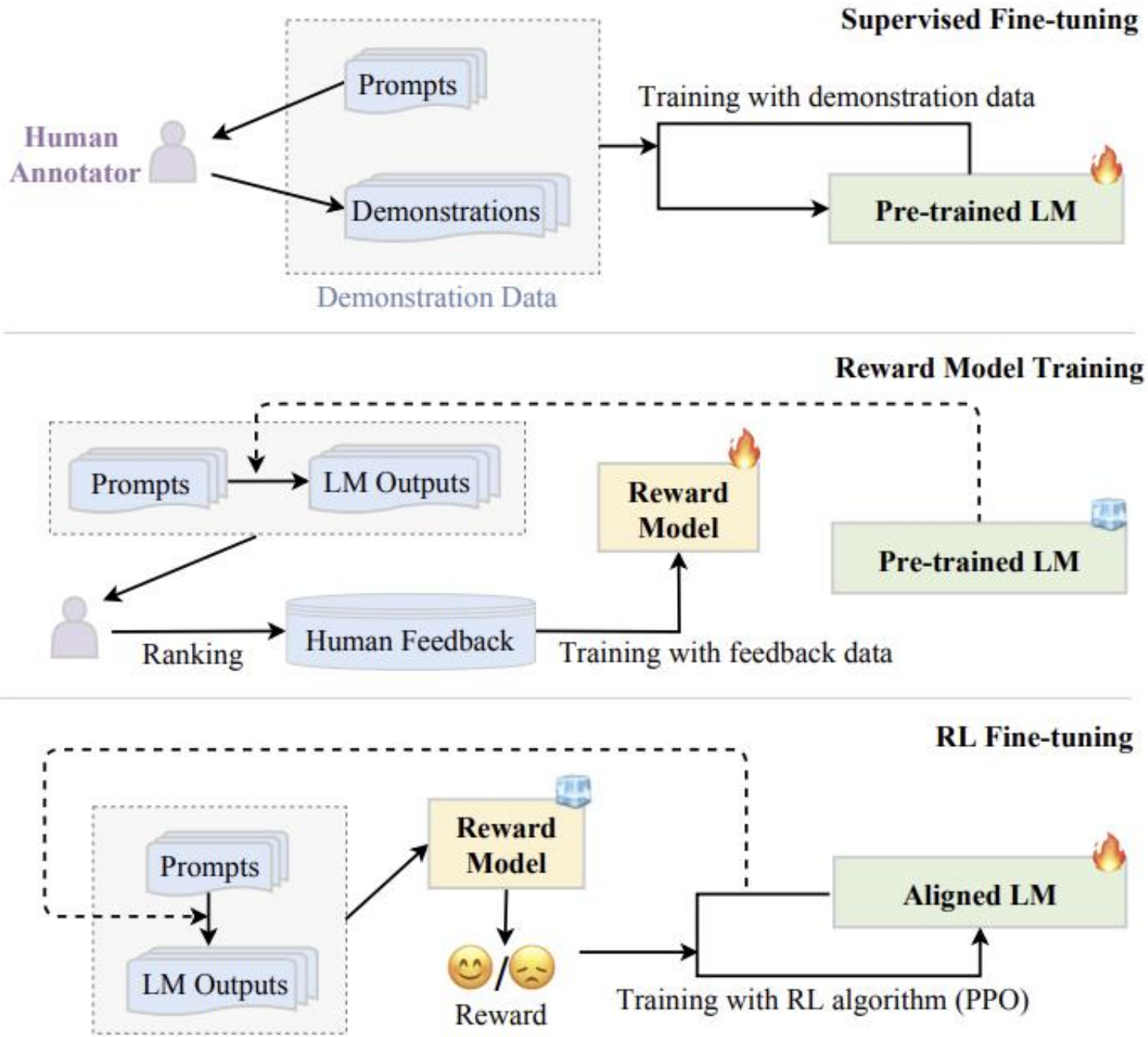
Balancing the instruction Difficulty

- Synthetic하게 instruction을 만들 경우 너무 쉽거나 어려운 instruction이 생성될 수 있고 이 경우 학습이 불안정하거나 LLM에 과적합이 생기기 쉬움
- LLM의 perplexity 점수를 사용하여 너무 쉽거나 어려운 instruction을 제거

Results of Instruction-tuning experiment

TABLE 8: Results of instruction-tuning experiments (all in a single-turn conversation) based on the LLaMA (7B) and LLaMA (13B) model under the chat and QA setting. We employ four instruction improvement strategies on the Self-Instruct-52K dataset, *i.e.*, enhancing the complexity (*w/ complexity*), increasing the diversity (*w/ diversity*), balancing the difficulty (*w/ difficulty*), and scaling the instruction number (*w/ scaling*). *Since we select the LLaMA (7B)/(13B) model fine-tuned on Self-Instruct-52K as the baseline, we omit the win rate of the fine-tuned model with Self-Instruct-52K against itself.

Models	Dataset Mixtures	Instruction Numbers	Lexical Diversity	Chat	QA	
				AlpacaFarm	MMLU	BBH3k
LLaMA (7B)	① FLAN-T5	80,000	48.48	23.77	38.58	32.79
	② ShareGPT	63,184	77.31	81.30	38.11	27.71
	③ Self-Instruct-52K	82,439	25.92	/*	37.52	29.81
	② + ③	145,623	48.22	71.36	41.26	28.36
	① + ② + ③	225,623	48.28	70.00	43.69	29.69
	③ Self-Instruct-52K	82,439	25.92	/*	37.52	29.81
	w/ complexity	70,000	70.43	76.96	39.73	33.25
	w/ diversity	70,000	75.59	81.55	38.01	30.03
	w/ difficulty	70,000	73.48	79.15	32.55	31.25
	w/ scaling	220,000	57.78	51.13	33.81	26.63
LLaMA (13B)	① FLAN-T5	80,000	48.48	22.12	34.12	34.05
	② ShareGPT	63,184	77.31	77.13	47.49	33.82
	③ Self-Instruct-52K	82,439	25.92	/*	36.73	25.43
	② + ③	145,623	48.22	72.85	41.16	29.49
	① + ② + ③	225,623	48.28	69.49	43.50	31.16
	③ Self-Instruct-52K	82,439	25.92	/*	36.73	25.43
	w/ complexity	70,000	70.43	77.94	46.89	35.75
	w/ diversity	70,000	75.59	78.92	44.97	36.40
	w/ difficulty	70,000	73.48	80.45	43.15	34.59
	w/ scaling	220,000	57.78	58.12	38.07	27.28

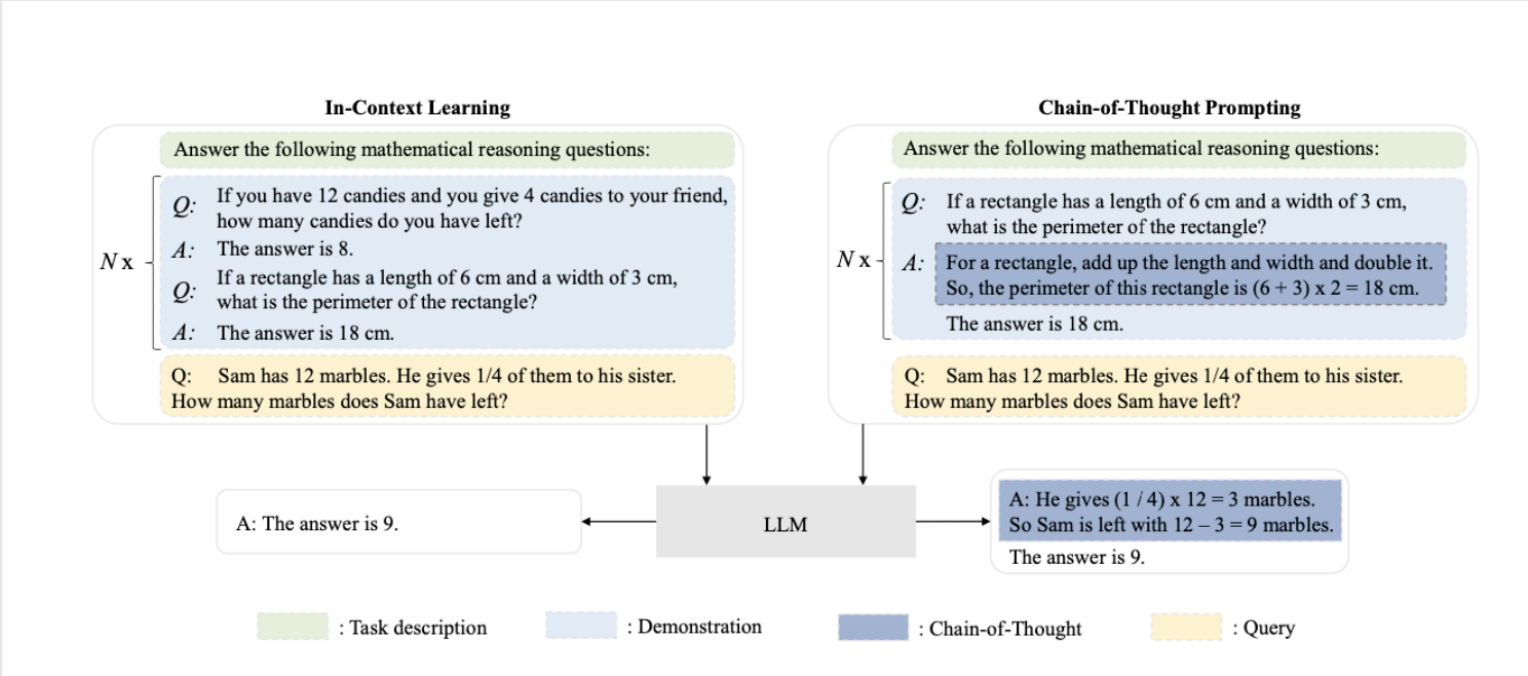


Adaptations of LLMs: Alignment Tuning

- LLM이 인간의 기대에 따라 행동하도록 tuning
  - 유용성: LLM은 가능한 간결하고 효율적인 방식으로 사용자의 과제 해결이나 질문에 대한 답변을 지원하려는 시도를 명확히 보여줘야 함
  - 정직성: LLM은 조작된 정보 대신 정확한 콘텐츠를 사용자에게 제공해야 함
  - 무해성: 모델이 생성하는 페이지가 불쾌감을 주거나 차별적이지 않아야 함
- 사람의 피드백을 통한 강화학습(Reinforcement Larningn from Human Feedback)
- DPO (Direct Preference Optimization)
- PPO (Proximal Policy Optimization)

Utilization

- Prompting :사전 훈련이나 적응 튜닝 후 LLM을 사용하기 위한 중요한 접근 방법
  - Task에 따른 prompt는 직접 만들거나 또는 자동으로 최적화할 수 있음
  - 상황내학습(in-context learning): 자연어 텍스트 형태로 과제 설명 및 또는 데모를 공식화하는 prompting
  - 연쇄적 사고 prompt(chain-of-thought Prompt): 일련의 중간 추론 단계를 prompt에 포함시켜 상황내 학습을 강화하기 위함



Prompting

- A prompt is a text string that a user issues to a language model to get the model to do something useful.
- In prompting, the user's prompt string is passed to the language model, which iteratively generates tokens conditioned on the prompt.
- Thus the prompt creates a context that guides LLMs to generate useful outputs to achieve some user goal.
- The process of finding effective prompts for a task is known as **prompt engineering**

A prompt consisting of a review plus an incomplete statement

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax. In short, our stay was

Hotel Review Completions

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relaxed. In short our stay was

```
...  not a pleasant one.  The staff at the front desk
were not welcoming or friendly, and seemed disinterested
in providing good customer service.

...  uncomfortable and not worth the price we paid.  We
will not be returning to this hotel.
```

Sample Hotel Review

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax.

Prompting

Basic Prompt Templates

Summarization	{input} ; `tldr;`
Translation	{input} ; `translate to French:`
Sentiment	{input}; `Overall, it was`
Fine-Grained-Sentiment	{input}; `What aspects were important in this review?`

A prompt consisting of a review plus an incomplete statement

Human: Do you think that "input" has negative or positive sentiment?
Choices:
(P) Positive
(N) Negative

Assistant: I believe the best answer is: (

LLM Outputs for Basic Prompts

Original Review ($INPUT)	`Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax and away from the city life.`
Sentiment	**Prompt**: `$INPUT + In short, our stay was` **Output**: `not enjoyable`
Fine-grained Sentiment	**Prompt**: `$INPUT + These aspects were important to the reviewer:` **Output**: `1. Poor service 2. Unpleasant location 3. Noisy and busy area`
Summarization	**Prompt**: `$INPUT + tl;dr` **Output**: `I had a bad experience with the hotel's service and the location was loud and busy.`
Translation	**Prompt**: `$INPUT + Translate this to French` **Output**: `Je n'ai pas aimé le service qui m'a été offert lorsque je suis entré dans l'hôtel. Je n'ai également pas aimé la zone dans laquelle se trouvait l'hôtel. Trop de bruit et d'événements pour que je me sente détendu et loin de la vie citadine.`

Learning from Demonstrations: Few-Shot Prompting

- It's often possible to improve a prompt by including some labeled examples in the prompt template.
- We call such examples demonstrations.
- The task of prompting with examples is sometimes called few-shot prompting, as contrasted with zero-shot prompting which means instructions that don't include labeled examples.

Definition: This task is about writing a correct answer for the reading comprehension task. Based on the information provided in a given passage, you should identify the shortest continuous text span from the passage that serves as an answer to the given question. Avoid answers that are incorrect or provides incomplete justification for the question.

Passage: Beyoncé Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".

Examples:
Q: In what city and state did Beyoncé grow up?
A: Houston, Texas

Q: What areas did Beyoncé compete in when she was growing up?
A: singing and dancing

Q: When did Beyoncé release Dangerously in Love?
A: 2003

Q: When did Beyoncé start becoming popular?
A:

Figure 12.2 A prompt for extractive question answering, from an example from the SQuAD 2.0 dataset (Rajpurkar et al., 2018). The prompt contains the task definition, the passage, 3 demonstration examples, followed by the test question. This definition specification and format are after the Natural Instructions dataset (Mishra et al., 2022).

Learning from Demonstrations: Few-Shot Prompting

How Many Demonstrations?

- The number of demonstrations doesn't have to be large.
- A small number of randomly selected labeled examples used as demonstrations can be sufficient to improve performance over the zero-shot setting.

Why isn't it useful to have more demonstrations?

- The reason is that the primary benefit in examples is to demonstrate the task to be performed to the LLM and the format of the sequence, not to provide relevant information as to the right answer.

How to Select Demonstrations?

- Demonstrations are generally created by formatting examples drawn from a labeled training set
- For example, using demonstrations that are similar to the current input seems to improve performance.
- But more generally, the best way to select demonstrations from the training set is programmatically: choosing the set of demonstrations that most increases task performance of the prompt on a test set.
- Task performance for sentiment analysis or multiple-choice question answering can be measured in accuracy; for machine translation with chrF, and for summarization via Rouge

In-Context Learning and Induction Heads

- Learning via pretraining means updating the model's parameters by using gradient descent according to some loss function.
- But prompting with demonstrations can teach a model to do a new task.
- The term in-context learning was first proposed by Brown et al. (2020) in their introduction of the GPT3 system, to refer to either of these kinds of learning that language models do from their prompts.
- In-context learning means language models learning to do new tasks, better predict tokens, or generally reduce their loss during the forward-pass at inference-time, without any gradient-based updates to the model's parameters's learning something as it processes the prompt
- How does in-context learning work?
  - While we don't know for sure, there are some intriguing ideas.
  - One hypothesis is based on the idea of induction heads (Elhage et al., 2021; Olsson et al., 2022)
  - Induction heads are the name for a circuit, which is a kind of abstract component of a network.
  - The induction head circuit is part of the attention computation in transformers, discovered by looking at mini language models with only 1-2 attention heads.

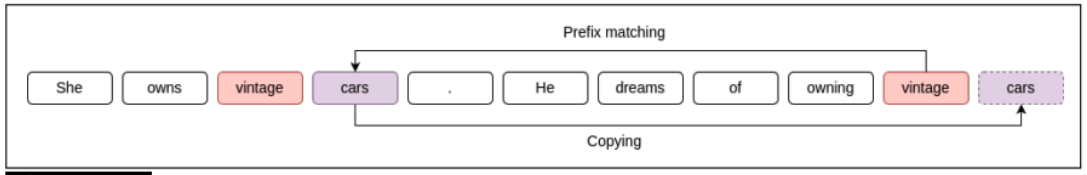


Figure 12.3 An induction head looking at `vintage` uses the *prefix matching* mechanism to find a prior instance of `vintage`, and the *copying* mechanism to predict that `cars` will occur again. Figure from Crosbie and Shutova (2022).

Chain-of-thought

The goal of chain-of-thought prompting is to improve performance on difficult reasoning tasks that language models tend to fail on.

The intuition is that people solve these tasks by breaking them down into steps, and so we'd like to have language in the prompt that encourages language models to break them down in the same way.

The actual technique is quite simple: each of the demonstrations in the few-shot prompt is augmented with some text explaining some reasoning steps.

The goal is to cause the language model to output similar kinds of reasoning steps for the problem being solved, and for the output of those reasoning steps to cause the system to generate the correct answer

Chain-of-thought

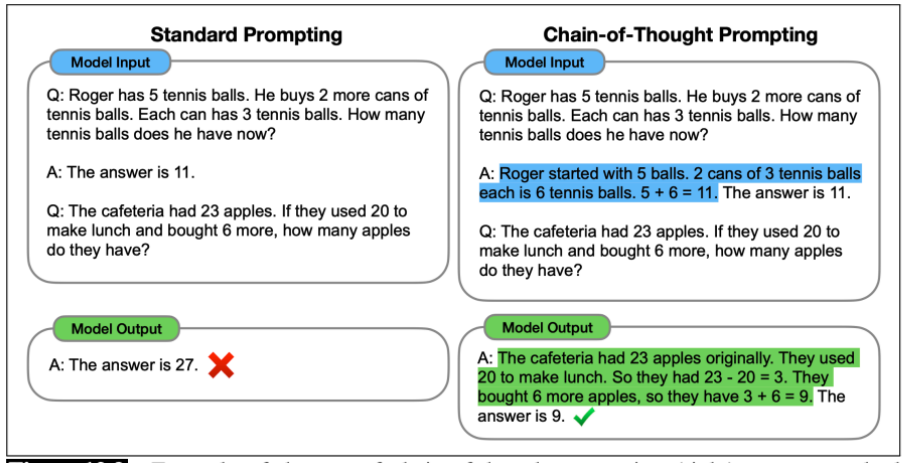


Figure 12.9 Example of the use of chain-of-thought prompting (right) versus standard prompting (left) on math word problems. Figure from Wei et al. (2022).

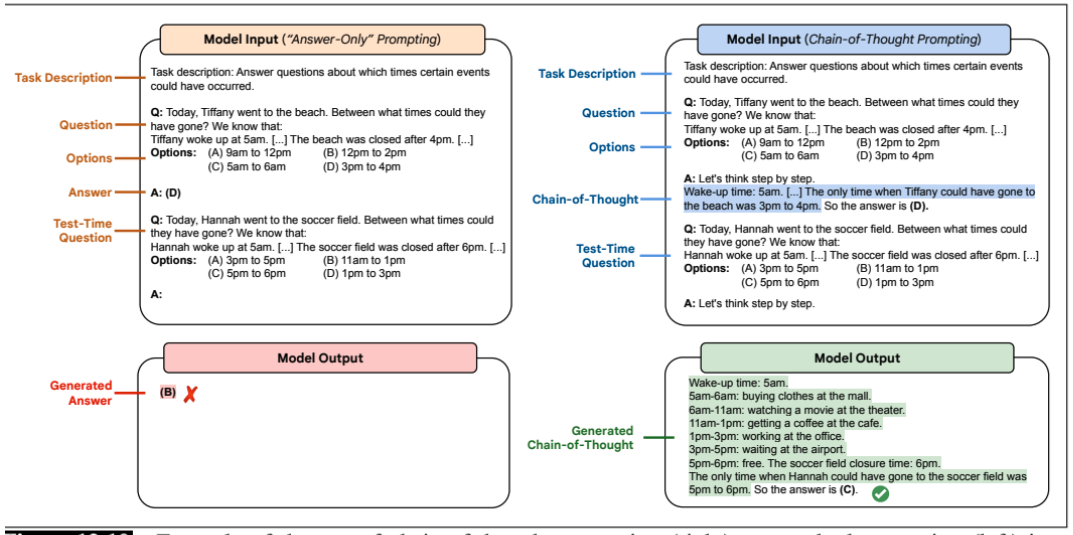


Figure 12.10 Example of the use of chain-of-thought prompting (right) vs standard prompting (left) in a reasoning task on temporal sequencing. Figure from Suzgun et al. (2023).

LLM의 성능평가 : Benchmark

- LLM의 성능을 비교하기 위해 여러 BENCHMARK 데이터셋이 존재
- LLM의 성능을 비교하기 어려운 이유
  - 정답이 없거나 다양할 수 있음: 인간이 특정 질문에 대해 정답을 도출하는 방식은 맥락과 해석 그리고 그 정도에 따라 달라질 수 있음
  - LLM 자체적인 변수: 모델마다 다른 학습 데이터, 파라미터, 아키텍쳐에 따라 성능이 다름
  - 모델별로 프롬프트의 내부구조에 따른 결과가 상이할 수 있음
  - RAG(Retrieval Augmentation Generation)과 여러 다른 외부 모듈, 사용자 인터페이스 등에 따라 성능이 차이가 나기 때문에 단순한 벤치마크만으로 평가하기 어려움

	Claude 3.5 Sonnet	Claude 3 Opus	GPT-4o	Gemini 1.5 Pro	Llama-400b (early snapshot)
Graduate level reasoning *GPQA, Diamond*	59.4%* 0-shot CoT	50.4% 0-shot CoT	53.6% 0-shot CoT	—	—
Undergraduate level knowledge *MMLU*	88.7%** 5-shot 88.3% 0-shot CoT	86.8% 5-shot 85.7% 0-shot CoT	— 88.7% 0-shot CoT	85.9% 5-shot —	86.1% 5-shot —
Code *HumanEval*	92.0% 0-shot	84.9% 0-shot	90.2% 0-shot	84.1% 0-shot	84.1% 0-shot
Multilingual math *MGSM*	91.6% 0-shot CoT	90.7% 0-shot CoT	90.5% 0-shot CoT	87.5% 8-shot	—
Reasoning over text *DROP, F1 score*	87.1 3-shot	83.1 3-shot	83.4 3-shot	74.9 Variable shots	83.5 3-shot Pre-trained model
Mixed evaluations *BIG-Bench-Hard*	93.1% 3-shot CoT	86.8% 3-shot CoT	—	89.2% 3-shot CoT	85.3% 3-shot CoT Pre-trained model
Math problem-solving *MATH*	71.1% 0-shot CoT	60.1% 0-shot CoT	76.6% 0-shot CoT	67.7% 4-shot	57.8% 4-shot CoT
Grade school math *GSM8K*	96.4% 0-shot CoT	95.0% 0-shot CoT	—	90.8% 11-shot	94.1% 8-shot CoT

주요벤치마크 와 언어현상별 벤치마크

LLM의 성능평가 : Benchmark (해외 주요 벤치마크)

벤치마크명	평가 항목
GLUE (Wang et al., 2018)	• 자연어 이해 • 어법, 패러프레이징, 텍스트 유사도, 추론, 함의 (entailment), 대명사 참조 해결 (Resolving Pronoun References)
SuperGLUE (Wang et al., 2019)	• 자연어 이해 • GLUE의 확장 버전 (난이도, 태스크 다양성 측면) • 자연어 이해, 추론, 생성 결과의 응집성, 상식 추론, 정보 검색, 독해 등
BigBench (Srivastava et al., 2022)	• 일반화(Generalization) 능력
MMLU (Hendrycks et al., 2020)	• 다양한 태스크 및 도메인에서의 언어 이해
EleutherAI LM Eval	• 다양한 태스크에 대한 few-shot 평가 지원 프레임워크

태스크	설명
COPA (Choice of Plausible Alternatives, Roemmele et al., 2011)	• 분류: causal reasoning • 구성: 전제(premise) 문장, 선지 2개 • 예측: 원인/결과가 되는 선지 예측
MultiRC (Multi-Sentence Reading Comprehension, Khashabi et al., 2018)	• 분류: 기계독해 • 구성: 맥락, 질문, 후보 답안들 • 예측: 후보 답안들 각각의 T/F 여부
ReCoRD (Reading Comprehension with Commonsense Reasoning Dataset, Zhang et al., 2018)	• 분류: 다지선다 질의응답 (multiple choice QA)
RTE (Recognizing Textual Entailment)	• 분류: 함의 관계 파악
WiC (Word-in-Context, Pilehvar and Camacho-Collados, 2019)	• 분류: 단어 중의성 해소(word sense disambiguation) • 구성: 동일한 다의어 포함하는 문장쌍 • 예측: 이진분류 (다의어의 의미 일관성 여부)
WSC (Winograd Schema Challenge, Levesque et al., 2012)	• 분류: 상호참조 해결 • 구성: 대명사(pronoun) 및 명사구를 포함하는 문장 • 예측: 대명사가 가리키는 대상 (Winograd schema 기반)

층위별 종합 평가

구분	측정 항목	기반 데이터	수량	평가 방법
BLIMP	언어 능력 평가 • 형태, 통사, 의미	• 템플릿 기반 자동 생성 후 검수 및 문법성 여부 태깅 • 12개 대분류, 67개 소분류	67,000건	• 정답/오답 선지의 생성 확률 비교
CoLA	언어 능력 평가 • 형태, 통사, 의미	• 언어학 출판물에서 수집된 예문에 문법성 여부 태깅	10,657문장	• 문법성 여부 분류 정확도 • Matthews Correlation Coefficient (MCC) 계산
WIDET	모델의 문법규칙 학습 가능성 평가 • 어휘, 통사 층위에서 추론	• BLiMP에서 현상 7개 선정 -> GPT4로 자동 생성 후 검수	1,200건	• 간접증거 기반으로 문법성 예측
Curriculum	언어 능력 평가 • 어휘, 통사, 의미, 논리 / 문맥추론 등	• NLI, NLU 등 기존 데이터셋 중 필터링 • 언어학적 현상에 관한 영어 데이터	1M 이상 (언어 능력 평가 관련 현상별 상이)	• 정문 예측 정확도 (토큰 개수로 정규화)
Holmes (Waldis et al. 2024)	언어 능력 평가 • 형태, 통사, 의미, 추론, 담화	• 기존 언어자원 활용: OntoNotes, English Web Treebank, BLiMP • 66개의 언어 현상 포괄	208개 데이터셋	• Classifier-based Probing 적용 -> 태스크 타입별로 macro F1 계산

LLM의 성능평가 : Benchmark (한국어 벤치마크)

- 한국어의 경우 영어 중심 벤치마크 구성 체계 / 데이터셋의 영향을 강하게 받고 있음
- 영어 벤치마크를 번역기를 돌려 그대로 활용하는 경우도 많음

현황

이슈:

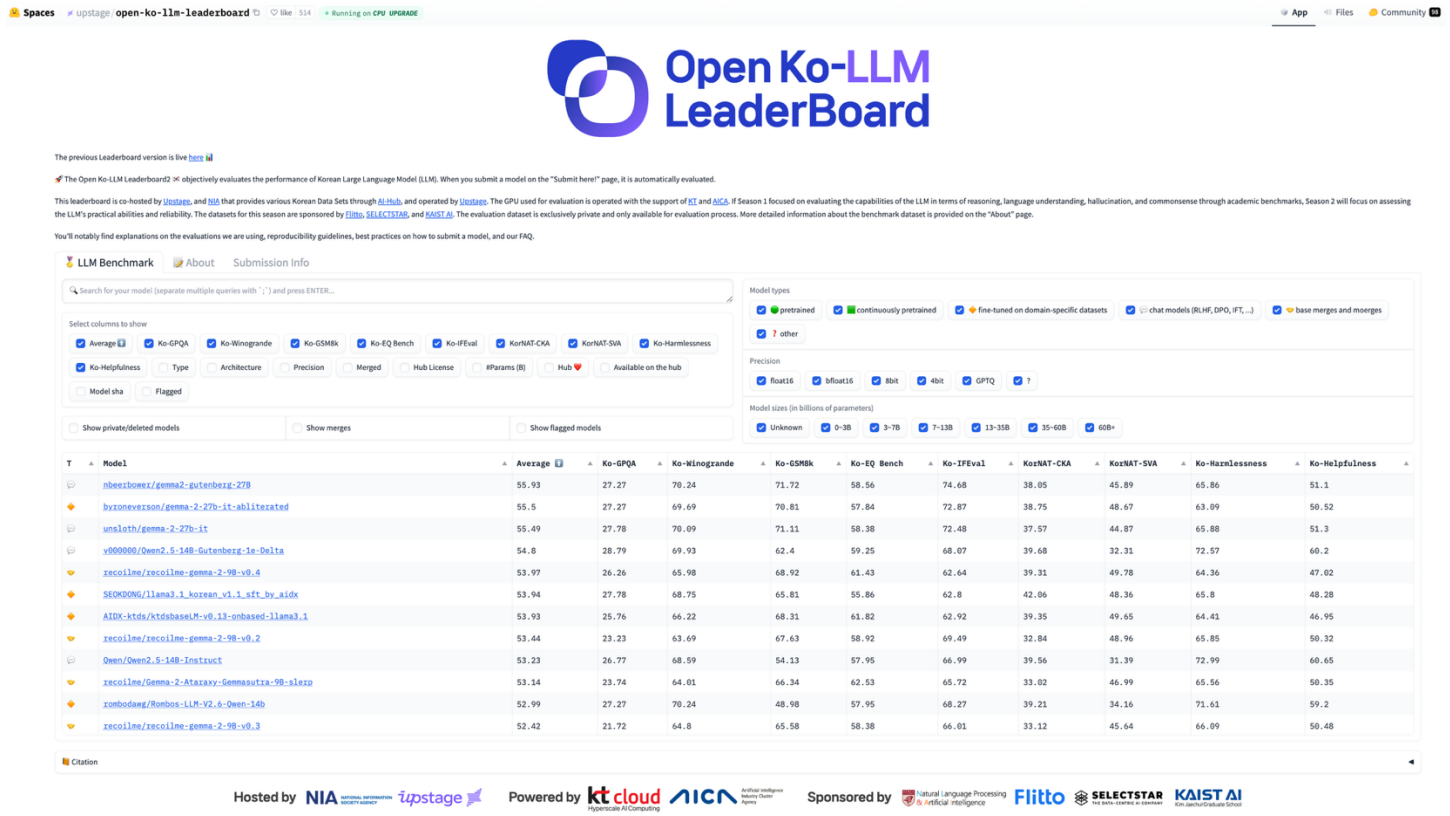
1) 대체로 영어 벤치마크의 구성 체계를 수용하여 구축되는 경향성 존재
2) 평가 시 영어 벤치마크 데이터셋을 번역하는 경우 다수 -> 평가의 정확도 저해 가능성
3) 영어와 구별되는 한국어의 고유한 특성을 반영하여 한국어에 특화된 태스크 및 구성 체계 설계
4) 데이터셋 규모와 다양성 측면의 향상 필요

벤치마크 태스크 범주화

비립 외 2022:

	소분류	설명	예시
자연어 이해 (NLU)	기본형 (fundamental)	구체적인 언어적 자질과 특성 판별	언어학적 층위별 태스크 • 음성/음운(음성인식, 합성), 형태(pos tagging) • 통사(청킹, 파싱), 의미(어휘의미분석, 의미역주석) 화용(무형 대용어 복원, 상호 참조 해결) 정보 추출 태스크 (Information extraction)
	고급형 (advanced)	복합적인 언어 자질을 이용해 의미적 차이 판별	감성 분석, 자연어 추론, 패러프레이즈 탐지
자연어 생성 (NLG)	일반형 (general)	구체적으로 정확한 언어 정보를 일정한 언어 단위에 적용하고 생성	기계번역, 요약, 질의응답, 대화시스템
	특수형 (characteristic)	특수한 조건에 맞춰 필요한 언어 정보를 일정한 형태로 전환	특정 생성 환경 및 입력 반영(도메인 특화) 인간의 창조적인 능력 모사(문체, 은유 등) 동일언어의 변이형(방언)

LLM의 성능평가 : Benchmark (open ko-LLM Leaderboard)



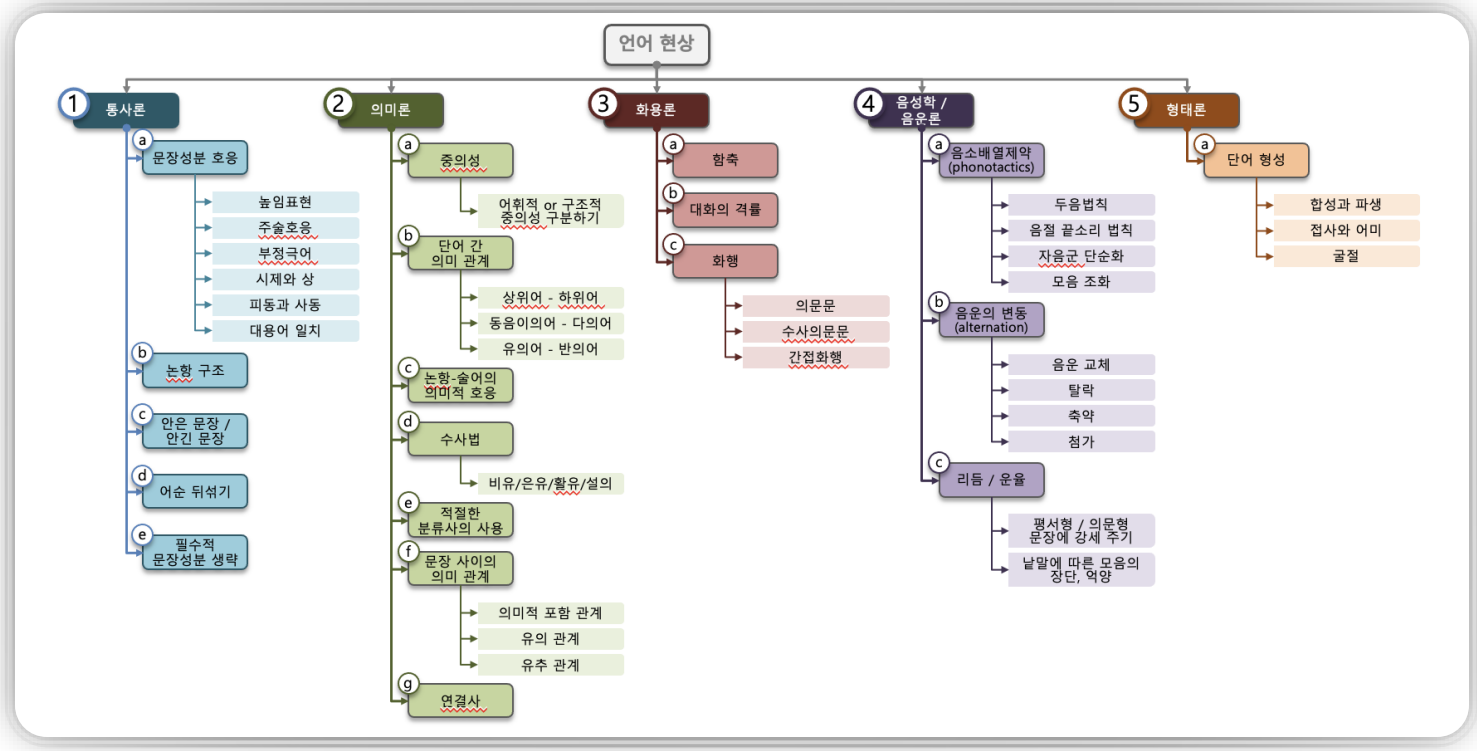
- https://huggingface.co/spaces/upstage/open-ko-llm-leaderboard
- 벤치마크의 성능이 우수해도 실제 한국어 생성능력이나 시스템 적용에 있어 성능이 좋은 것은 아님!
- 모델과 Benchmark와의 괴리

LLM의 성능평가 : Benchmark (평가지표, Metric)

분류	타겟 태스크	예시	상세
OBM (Overlap-based metrics)	Similarity	BLEU, ROUGE, METEOR	Literal n-gram matching 기반
SBM (Semantic similarity-based metrics)	Similarity	BERTScore, MoverScore	contextualized embedding 기반 벡터로 의미적 유사성 측정
SSM (Summary-specialized metrics)	Similarity	SUPERT, BLANC	Pretrained embedding 기반 입력 문서와 요약 간 의미적 유사성 측정
	QA Pair	QAEval, QAFactEval, QuestEval	질의응답 시스템 사용, 원본 문서와 요약 생성 결과 비교
Etc.	Similarity	Perplexity, Levenshtein Distance	-
	NER	InterpretEval	텍스트에서 개체 식별 및 분류

LLM 한국어 벤치마크 데이터셋 개발 및 성능 평가 방법 구축(SNU_CL_NLP lab and LG AI Lab)

- 한국어의 어휘, 문장, 그리고 문단 단위 등에서 나타나는 언어적 특성을, LLM 모델을 통해 생성되는 문장의 한국어 구사 능력 및 한국어 언어 지식 수준을 평가할 수 있는 데이터셋 및 평가방법을 구축
- 기존의 벤치마크 데이터셋의 상당 수가 타언어로부터의 번역을 통해 구축되어 한국어의 특성이 제대로 반영되어 있지 않을 뿐더러 데이터 품질이 낮다는 점과 벤치마크상의 점수가 실제 사용자가 체감하는 성능으로 직결되지 못한다는 문제의식에서 출발



LLM 한국어 벤치마크 데이터셋 개발 및 성능 평가 방법 구축(SNU_CL_NLP lab and LG AI Lab)

- KoBALT (Korean Benchmark for Advanced Linguistic Tasks)
- https://huggingface.co/datasets/snunlp/KoBALT-700

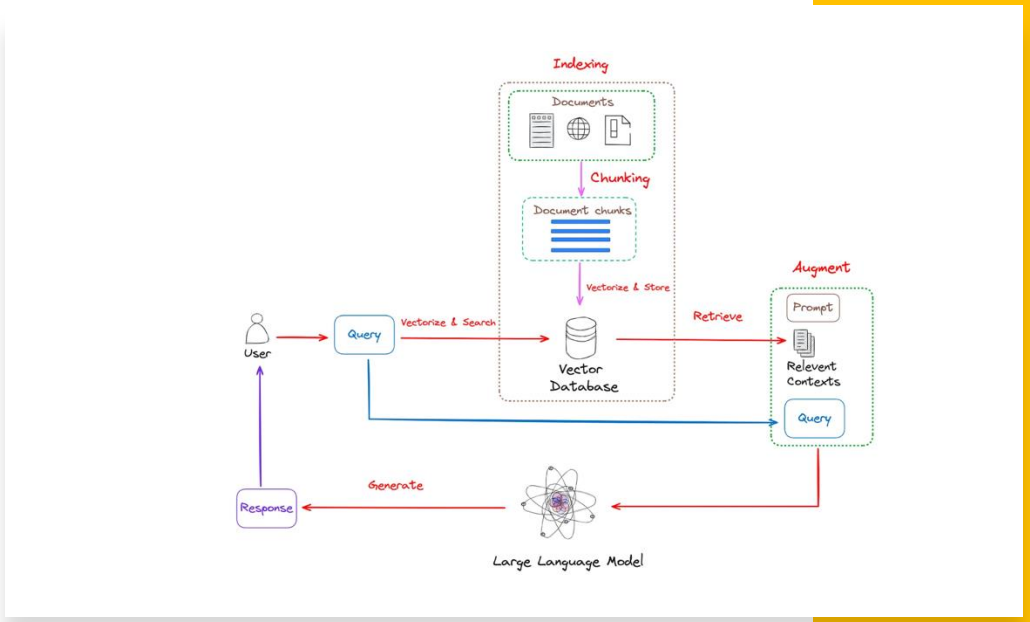
Model	Avg	Syntax	Semantics	Pragmatics	Morphology	Phonetics
Claude-3-7-sonnet	0.61	0.66	0.66	0.64	0.36	0.31
Claude-3-5-sonnet	0.52	0.52	0.65	0.51	0.36	0.24
DeepSeek-V3-XL	0.47	0.49	0.56	0.42	0.24	0.29
GPT-4o	0.44	0.45	0.55	0.40	0.17	0.26
DeepSeek-V3	0.43	0.41	0.57	0.42	0.26	0.23
C4ai-command-a-03	0.36	0.30	0.52	0.36	0.24	0.18
Gemma-3-27b	0.35	0.30	0.53	0.27	0.24	0.11
Qwen2.5-72B	0.37	0.33	0.51	0.37	0.24	0.18
Mistral-Small-24B	0.32	0.27	0.49	0.30	0.21	0.11
Llama-3.3-70B	0.32	0.25	0.50	0.35	0.17	0.15
Qwen2.5-32B	0.30	0.23	0.49	0.28	0.21	0.11
Gemma-2-9b	0.21	0.17	0.34	0.15	0.12	0.11
Aya-expanse-32b	0.25	0.21	0.40	0.12	0.10	0.16
Aya-expanse-8b	0.19	0.15	0.33	0.11	0.12	0.06
Qwen2.5-7B	0.19	0.14	0.33	0.11	0.19	0.06
Llama-3.1-8B	0.17	0.13	0.26	0.12	0.10	0.11
Ministral-8B	0.17	0.11	0.29	0.15	0.10	0.11
Mistral-7B-v0.3	0.12	0.11	0.16	0.11	0.14	0.06

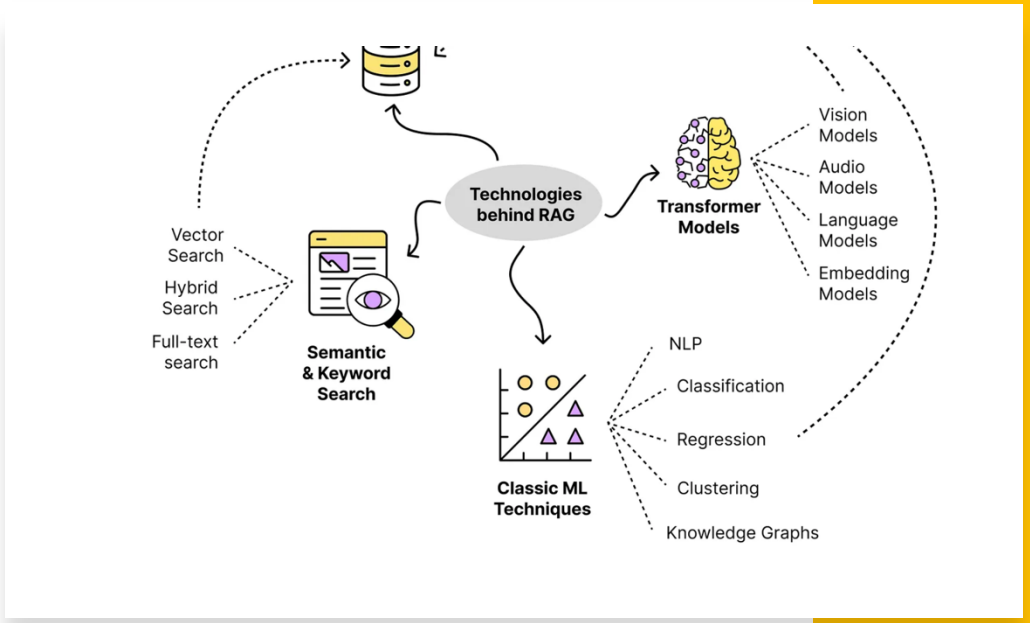
RAG(Retrieval Augmented Generation)

- **General**
  - Retrieval Augmented Generation(RAG) was initially proposed in 2020 as an end-to-end approach that combined a pre-trained retriever and a pre-trained generator. At that time, its main goal was to **improve performance through model fine-tuning**.
  - The release of ChatGPT in December 2022 marked a significant turning point for RAG. Since then, RAG has focused more on **leveraging the reasoning capabilities** of large language models (LLM) to achieve better generation results **by incorporating external knowledge.**
  - **RAG technology eliminates the need for developers to retrain the entire large-scale model for every specific task.** Instead, they can simply connect relevant knowledge bases to provide additional input to the model, enhancing the accuracy of the answers.

Retrieval Augmented Generation

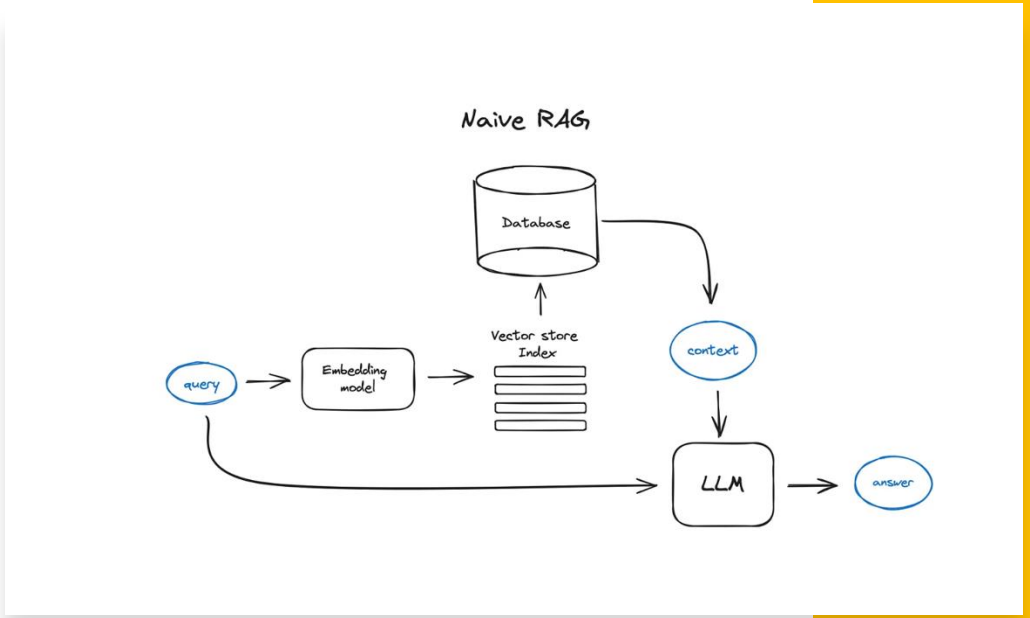
- **What is Retrieval Augmented Generation (RAG)?**
  - **Retrieval Augmented Generation (RAG)** is the process of enhancing large language models (LLMs) by **incorporating additional information from external knowledge sources**. This enables the LLMs to generate more **accurate and context-aware answers, while also reducing hallucinations**.
  - **When answering questions or generating text, relevant information is initially retrieved from existing knowledge bases or a large number of documents. The answer is then generated using LLM, which improves the quality of the response by incorporating this retrieved information, rather than relying solely on LLM to generate it.**

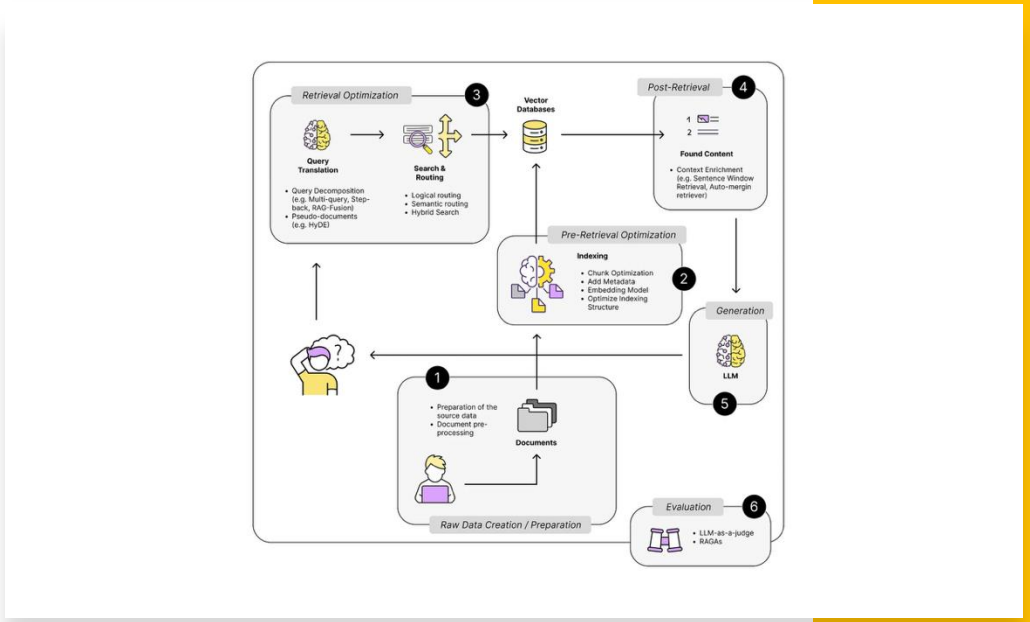




Retrieval Augmented Generation

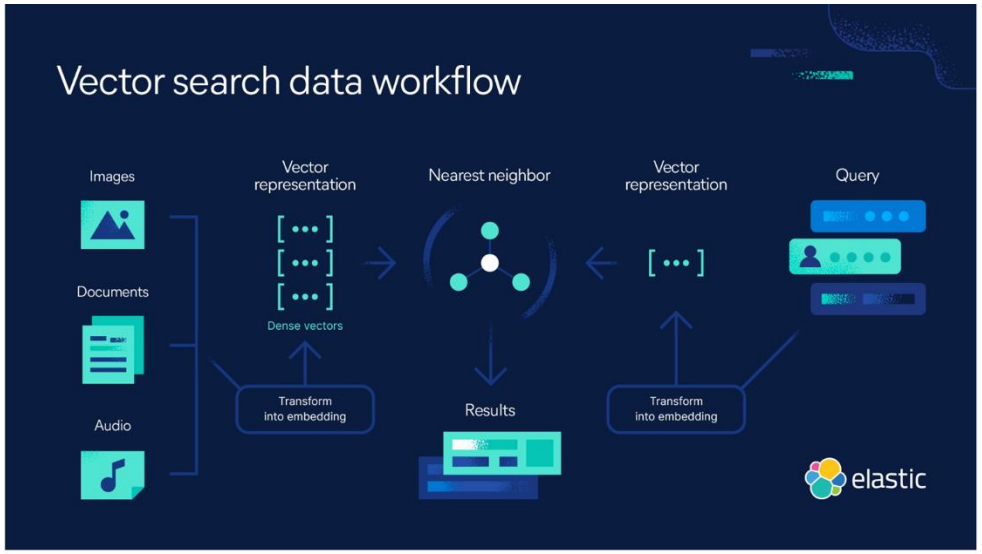
- **Naïve vs Advanced RAG**





Retrieval Augmented Generation (Semantic search through vector)

- **Semantic search is a search engine technology that interprets the meaning of words and phrases.**
- The results of a semantic search will return content matching the *meaning* of a query, as opposed to content that literally matches words in the query.
- Semantic search is powered by **vector search**, which enables semantic search to deliver and rank content based on context relevance and intent relevance. Vector search encodes details of searchable information into fields of related terms or items, or vectors, and then compares vectors to determine which are most similar



Retrieval Augmented Generation (chunking and vectorization)

- Transformer models have fixed input sequence length and even if the input context window is large, the vector of a sentence or a few better represents their semantic meaning than a vector averaged over a few pages of text (depends on the model too, but true in general), so chunk your data — **split the initial documents in chunks of some size without loosing their meaning (splitting your text in sentences or in paragraphs, not cutting a single sentence in two parts)**
    1. Recursive Structure Aware Splitting
    2. Structure Aware Splitting (by Sentence, Paragraph)
    3. Content-Aware Splitting (Markdown, LaTeX, HTML)
    4. Chunking using NLP: Tracking Topic Changes
- **Vectorization**
    - The next step is to choose a **model to embed our chunks**
    - **MMTEB: Massive Multilingual Text Embedding Benchmark**

Mean (TaskType)	Bitext Mining	Classification	Cl
56.00	73.92	61.55	53
55.17	**80.13**	**64.94**	51
54.21	70.34	62.24	51
54.00	70.00	60.02	52
53.83	70.53	61.83	50
53.18	70.58	60.31	51
53.00	73.55	62.77	47
52.75	62.51	58.32	52
52.72	60.80	58.24	52
52.91	68.84	59.01	**54**

(참고) 한국어 retrieval 벤치마크

MTEB-ko-retrieval Leaderboard

MTEB에 등록된 모든 Korean Retrieval Benchmark에 대한 평가를 진행하였습니다.

Korean Retrieval Benchmark

- Ko-StrategyQA: 한국어 ODQA multi-hop 검색 데이터셋 (StrategyQA 번역)
- AutoRAGRetrieval: 금융, 공공, 의료, 법률, 커머스 5개 분야에 대해, pdf를 파싱하여 구성한 한국어 문서 검색 데이터셋
- MIRACLRetrieval: Wikipedia 기반의 한국어 문서 검색 데이터셋
- PublicHealthQA: 의료 및 공중보건 도메인에 대한 한국어 문서 검색 데이터셋
- BelebeleRetrieval: FLORES-200 기반의 한국어 문서 검색 데이터셋
- MrTidyRetrieval: Wikipedia 기반의 한국어 문서 검색 데이터셋
- MultiLongDocRetrieval: 다양한 도메인의 한국어 장문 검색 데이터셋
- XPQARetrieval: 다양한 도메인의 한국어 문서 검색 데이터셋

Top-k 1

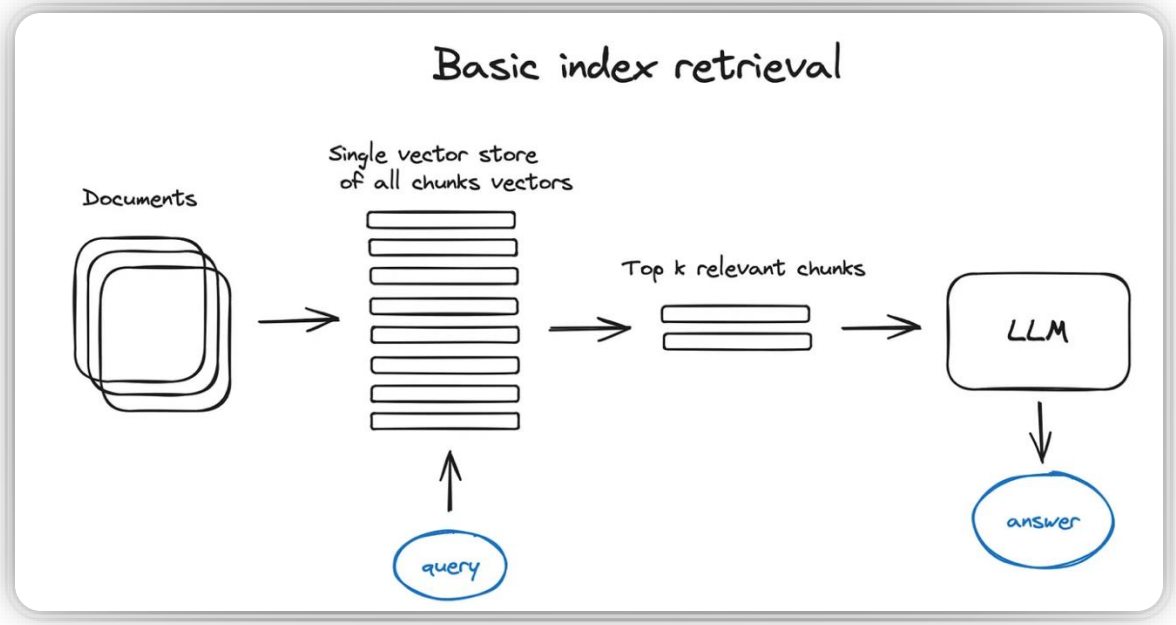
Model	Average Recall	Average Precision	Average NDCG	Average F1
nlpai-lab/KURE-v1	**0.52640**	**0.60551**	**0.60551**	**0.55784**
dragonkue/BGE-m3-ko	0.52361	0.60394	0.60394	0.55535
BAAI/bge-m3	0.51778	0.59846	0.59846	0.54998
Snowflake/snowflake-arctic-embed-l-v2.0	0.51246	0.59384	0.59384	0.54489
nlpai-lab/KoE5	0.50157	0.57790	0.57790	0.53178
intfloat/multilingual-e5-large	0.50052	0.57727	0.57727	0.53122
jinaai/jina-embeddings-v3	0.48287	0.56068	0.56068	0.51361
BAAI/bge-multilingual-gemma2	0.47904	0.55472	0.55472	0.50916
intfloat/multilingual-e5-large-instruct	0.47842	0.55435	0.55435	0.50826
intfloat/multilingual-e5-base	0.46950	0.54490	0.54490	0.49947
intfloat/e5-mistral-7b-instruct	0.46772	0.54394	0.54394	0.49781
Alibaba-NLP/gte-multilingual-base	0.46469	0.53744	0.53744	0.49353

Retrieval Augmented Generation (text Embedding models)

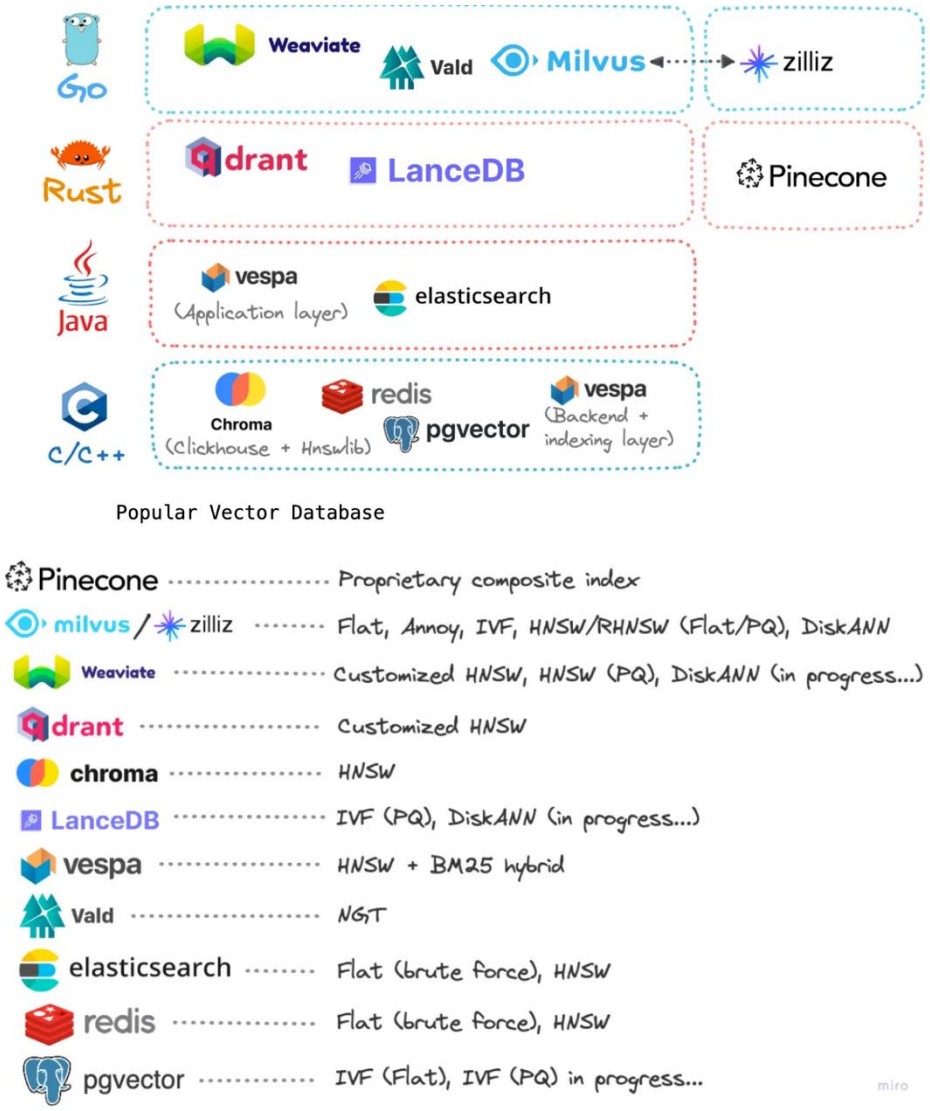
- 한국어 retrieval benchmark 성능

Retrieval Augmented Generation (search index)

VECTOR STORE INDEX.

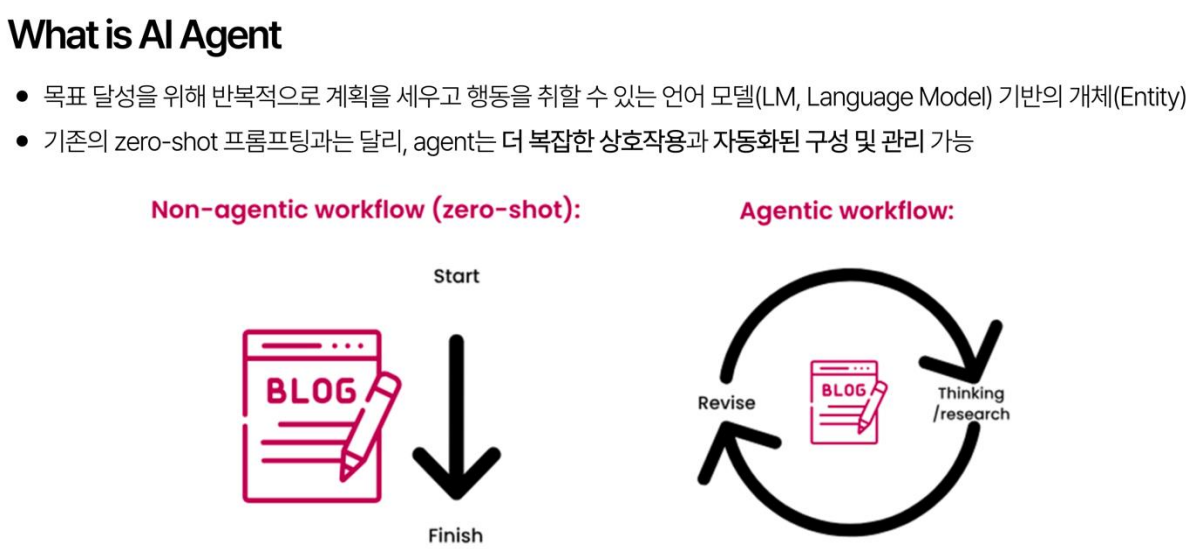


Retrieval Augmented Generation (Various databased with indexing methods)



AI Agent?

- An Agent is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks.



AI Agent

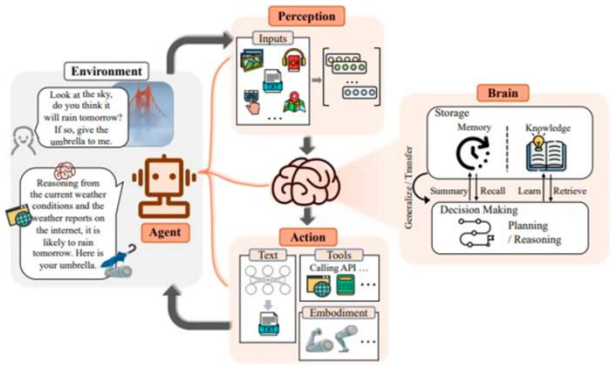
- **The Brain (AI Model)**
- This is where all the thinking happens. The AI model **handles reasoning and planning**. It decides **which Actions to take based on the situation**.
- **The Body (Capabilities and Tools)**
- This part represents **everything the Agent is equipped to do**.
- The **scope of possible actions** depends on what the agent **has been equipped with**. For example, because humans lack wings, they can't perform the "fly" **Action**, but they can execute **Actions** like "walk", "run" ,"jump", "grab", and so on.

AI Agent Architecture

1. Brain-Perception-action으로 구성

2. Agent Persona
   - Agent에게 주어지는 role
   - Agent에 대한 설명, 접근 가능한 tools 정의
   - System prompt를 제공함으로써 persona 실현

```
System: You are an experienced financial advisor with expertise in
personal finance, investment strategies, and retirement planning.
Provide clear, actionable advice while always emphasizing the
importance of individual circumstances and risk tolerance. Never
recommend specific stocks or make promises about returns. Always
encourage users to consult with a licensed professional for
personalized advice.
```



3. Tools
   - Agent가 호출할 수 있는 모든 종류의 함수나 API
   - Agent는 도구 사용을 통해 **외부 데이터 소스와 상호 작용**하여 더 많은 정보를 수집하고 활용

The spectrum of “Agency”

- Agents exist on a continuous spectrum of increasing agency:

Agency Level	Description	What that’s called	Example pattern
☆☆☆	Agent output has no impact on program flow	Simple processor	`process_llm_output(llm_response)`
★☆☆	Agent output determines basic control flow	Router	`if llm_decision(): path_a() else: path_b()`
★★☆	Agent output determines function execution	Tool caller	`run_function(llm_chosen_tool, llm_chosen_args)`
★★★	Agent output controls iteration and program continuation	Multi-step Agent	`while llm_should_continue(): execute_next_step()`
★★★	One agentic workflow can start another agentic workflow	Multi-Agent	`if llm_trigger(): execute_agent()`

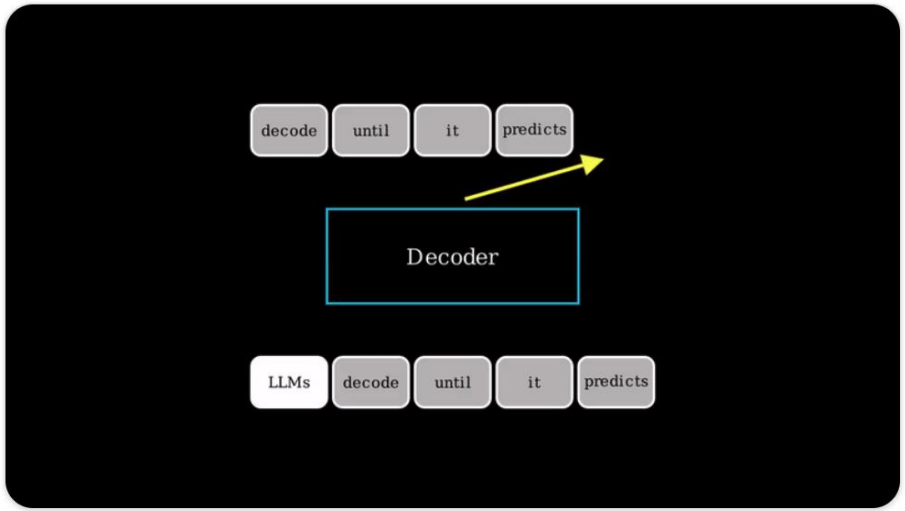
Agent Components: llm

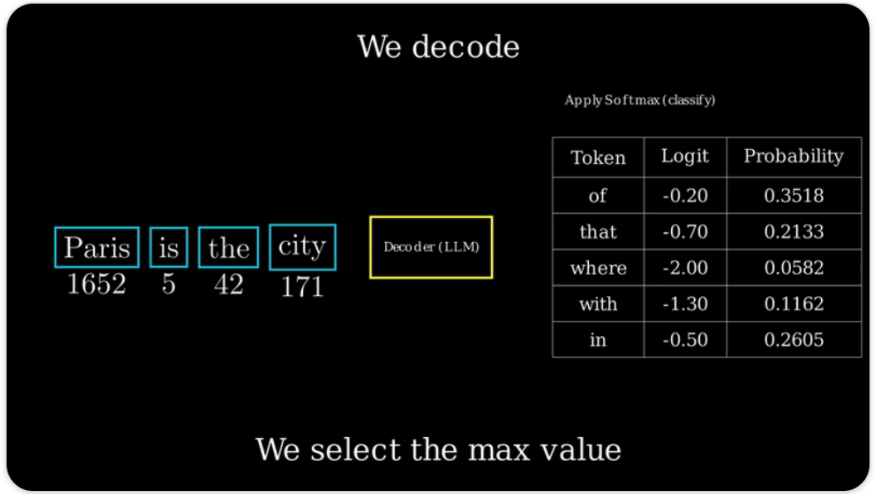
- Each Agent needs **an AI Model at its core**, and that LLMs are the most common type of AI models for this purpose.
- The underlying principle of an LLM is simple yet highly effective: **its objective is to predict the next token, given a sequence of previous tokens**.
- Each LLM has some **special tokens** specific to the model. The LLM uses these tokens to open and close the structured components of its generation.

Model	Provider	EOS Token	Functionality
GPT4	OpenAI	`<|endoftext|>`	End of message text
Llama 3	Meta (Facebook AI Research)	`<|eot_id|>`	End of sequence
Deepseek-R1	DeepSeek	`<|end_of_sentence|>`	End of message text
SmolLM2	Hugging Face	`<|im_end|>`	End of instruction or message
Gemma	Google	`<end_of_turn>`	End of conversation turn

Agent Components: llm
Understanding next token prediction.

- LLMs are said to be **autoregressive**, meaning that **the output from one pass becomes the input for the next one**.
- https://huggingface.co/datasets/agents-course/course-images/resolve/main/en/unit1/AutoregressionSchema.gif
- https://huggingface.co/datasets/agents-course/course-images/resolve/main/en/unit1/DecodingFinal.gif





Agent Components: **Messages and Special Tokens**

- **Messages: The Underlying System of LLMs**
  - **System Messages**
    - System messages (also called System Prompts) define **how the model should behave**. They serve as **persistent instructions**, guiding every subsequent interaction.
    - When using Agents, the System Message also **gives information about the available tools, provides instructions to the model on how to format the actions to take, and includes guidelines on how the thought process should be segmented.**
- **Conversations: User and Assistant Messages**
  - A conversation consists of alternating messages between a Human (user) and an LLM (assistant).
  - Chat templates help maintain context by preserving conversation history, storing previous exchanges between the user and the assistant. This leads to more coherent multi-turn conversations.

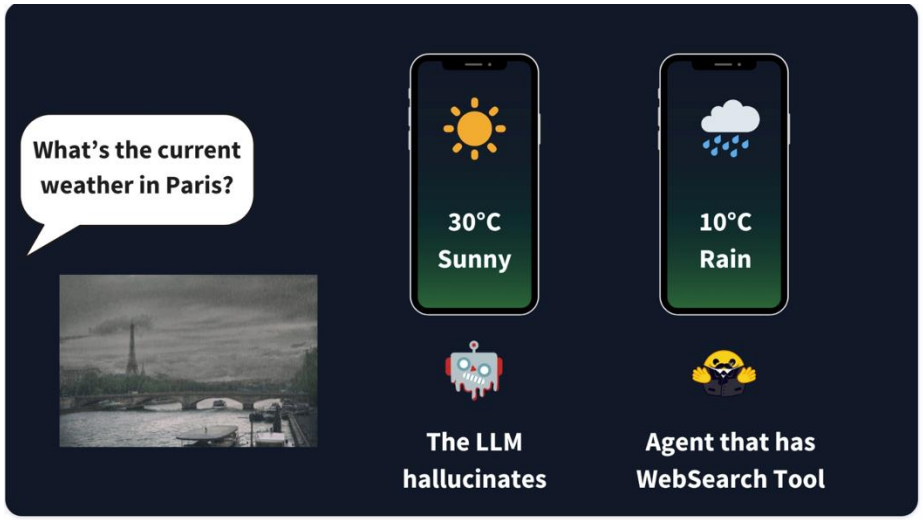
```
system_message = {
    "role": "system",
    "content": "You are a professional customer service agent. Always be polite, clear, and hel
}
```

```
conversation = [
    {"role": "user", "content": "I need help with my order"},
    {"role": "assistant", "content": "I'd be happy to help. Could you provide your order number
    {"role": "user", "content": "It's ORDER-123"},
]
```

Agent Components: **Tools**

- One crucial aspect of AI Agents is their ability to take **actions**.
- This happens through the use of **Tools**.
- A **Tool is a function given to the LLM**. This function should fulfill a **clear objective**.
- A good tool should be something that **complements the power of an LLM**.
  - For instance, if you need to perform arithmetic, giving a **calculator tool** to your LLM will provide better results than relying on the native capabilities of the model.
  - For instance, if you ask an LLM directly (without a search tool) for today's weather, the LLM will potentially hallucinate random weather.

Tool	Description
Web Search	Allows the agent to fetch up-to-date information from the internet.
Image Generation	Creates images based on text descriptions.
Retrieval	Retrieves information from an external source.
API Interface	Interacts with an external API (GitHub, YouTube, Spotify, etc.).



Agent Components: **Tools**

How do tools work?

- LLMs can only receive text inputs and generate text outputs. They have no way to call tools on their own. When we talk about providing tools to an Agent, we mean teaching the LLM about the existence of these tools and instructing it to generate text-based invocations when needed.
- For example, if we provide a tool to check the weather at a location from the internet and then ask the LLM about the weather in Paris, the LLM will recognize that this is an opportunity to use the “weather” tool. Instead of retrieving the weather data itself, the LLM will generate text that represents a tool call, such as call weather_tool(‘Paris’).

The **Agent** then reads this response, identifies that a tool call is required, executes the tool on the LLM’s behalf, and retrieves the actual weather data.

Agent Components: **Tools**

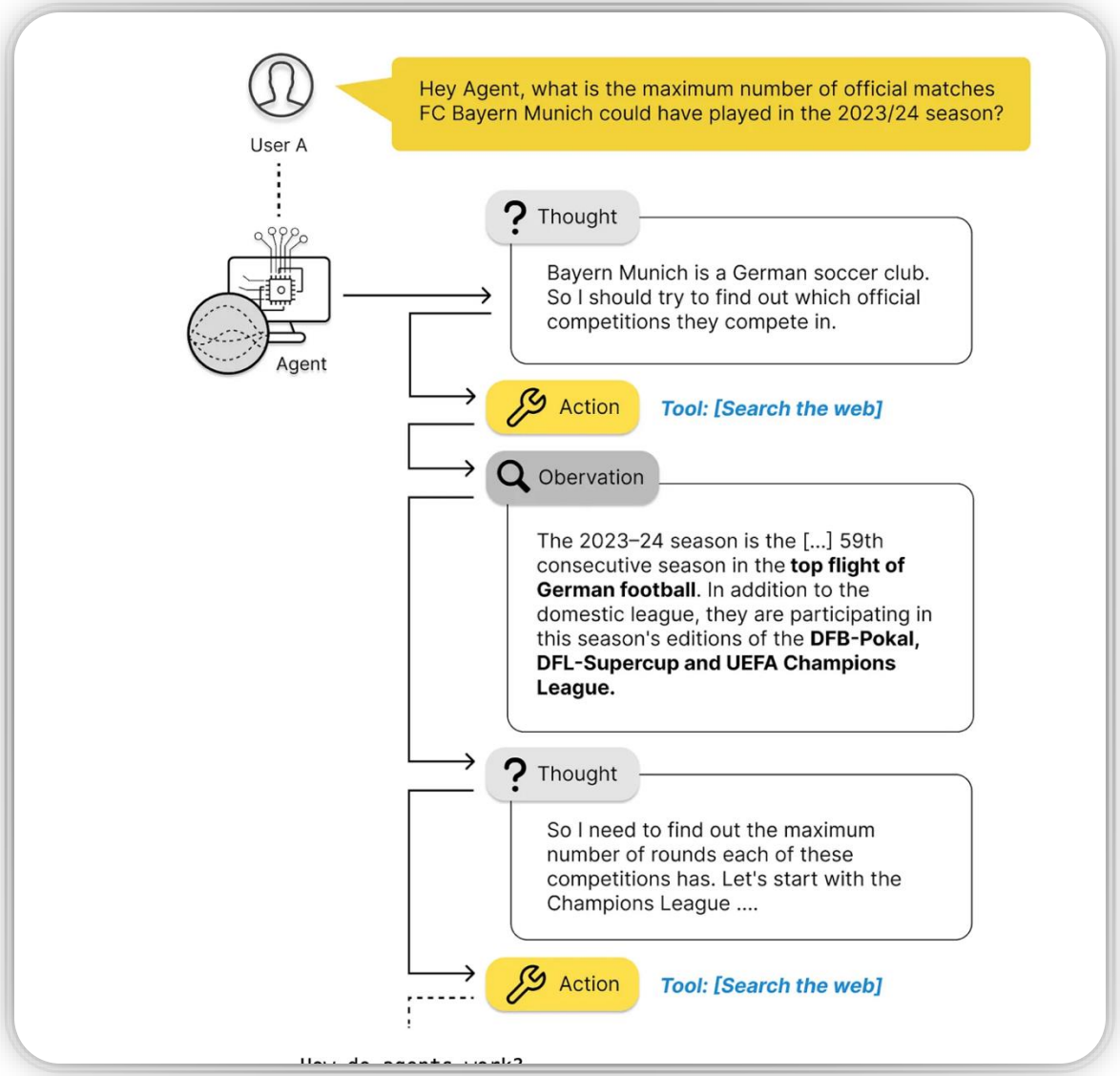
- **How do we give tools to an LLM?**
  - we essentially use the system prompt to provide textual descriptions of available tools to the model:
- **Generic Tool implementation Example**
- **Model Context Protocol (MCP): a unified tool interface**
  - Model Context Protocol (MCP) is an **open protocol** that standardizes how applications **provide tools to LLMs**. MCP provides:
    - A growing list of pre-built integrations that your LLM can directly plug into
    - The flexibility to switch between LLM providers and vendors
    - Best practices for securing your data within your infrastructure
- This means that **any framework implementing MCP can leverage tools defined within the protocol**, eliminating the need to reimplement the same tool interface for each framework.

```python
from typing import Callable


class Tool:
    """
    A class representing a reusable piece of code (Tool).

    Attributes:
        name (str): Name of the tool.
        description (str): A textual description of what the tool does.
        func (callable): The function this tool wraps.
        arguments (list): A list of argument.
        outputs (str or list): The return type(s) of the wrapped function.
    """
    def __init__(self,
                 name: str,
                 description: str,
                 func: Callable,
                 arguments: list,
                 outputs: str):
        self.name = name
        self.description = description
        self.func = func
        self.arguments = arguments
        self.outputs = outputs

    def to_string(self) -> str:
        """
        Return a string representation of the tool,
        including its name, description, arguments, and outputs.
        """
        args_str = ", ".join([
            f"{arg_name}: {arg_type}" for arg_name, arg_type in self.arguments
        ])

        return (
            f"Tool Name: {self.name},"
            f" Description: {self.description},"
            f" Arguments: {args_str},"
            f" Outputs: {self.outputs}"
        )

    def __call__(self, *args, **kwargs):
        """
        Invoke the underlying function (callable) with provided arguments.
        """
        return self.func(*args, **kwargs)
```

```python
calculator_tool = Tool(
    "calculator",                   # name
    "Multiply two integers.",       # description
    calculator,                     # function to call
    [("a", "int"), ("b", "int")],   # inputs (names and types)
    "int",                          # output
)
```

AI AGENT in RAG

- Agents (supported both by Langchain and LlamaIndex) have been around almost since the first LLM API has been released — **the idea was to provide an LLM, capable of reasoning, with a set of tools and a task to be completed.** The tools might include some deterministic functions like any code function or an external API or even other agents



AI AGENT in RAG

- The drawback of such a complex scheme can be guessed from the picture — **it's a bit slow due to multiple back and forth iterations with the LLMs inside our agents**.
- Just in case, an LLM call is always the longest operation in a RAG pipeline — search is optimised for **speed** by design.