

LLM Trend (feat. Llama)

LLM, SLM, RAG, Agent, ... 뭐가 너무 많은데?

Contents

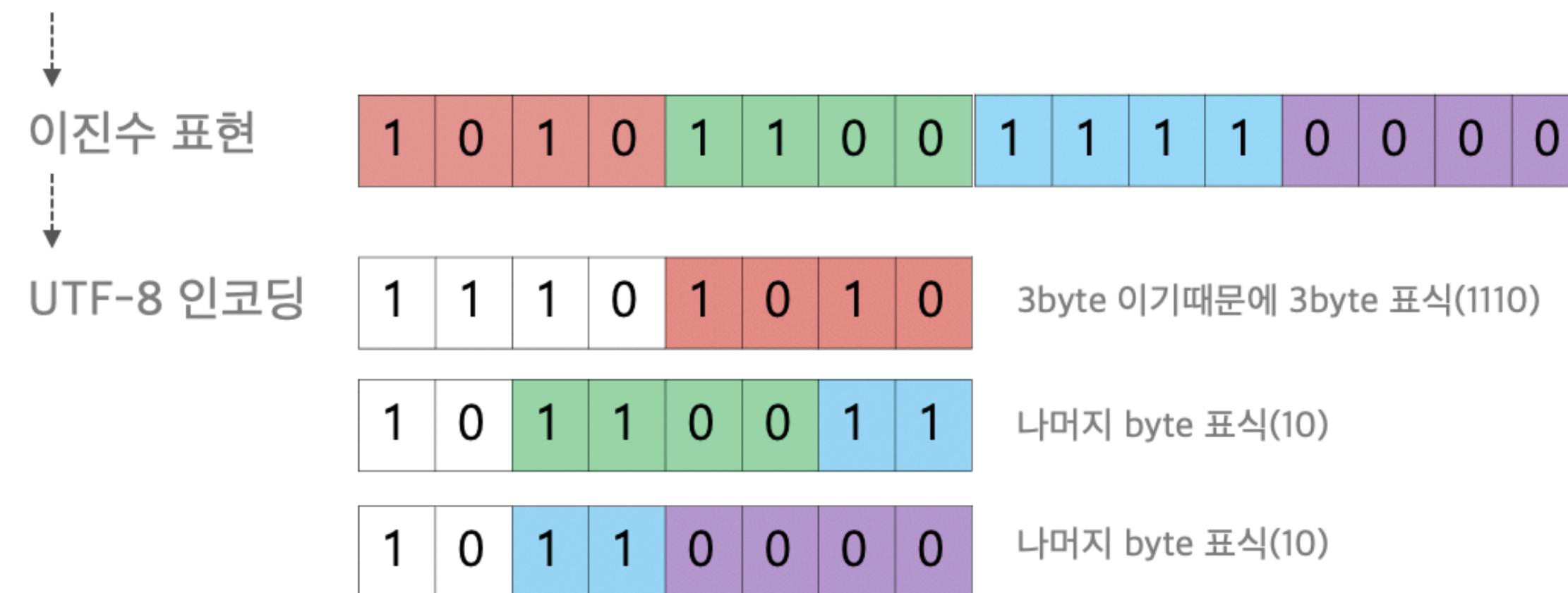
1. Llama 3.1을 중심으로 살펴보는 LLM Trend
2. Llama 3.2를 통해 보는 Large Multi-Model의 이해
3. RAG에 대한 이해
4. LLM App 구축시 고려할 사항들에 대한 이해

1. Llama 3.1을 중심으로 살펴보는 LLM Trend

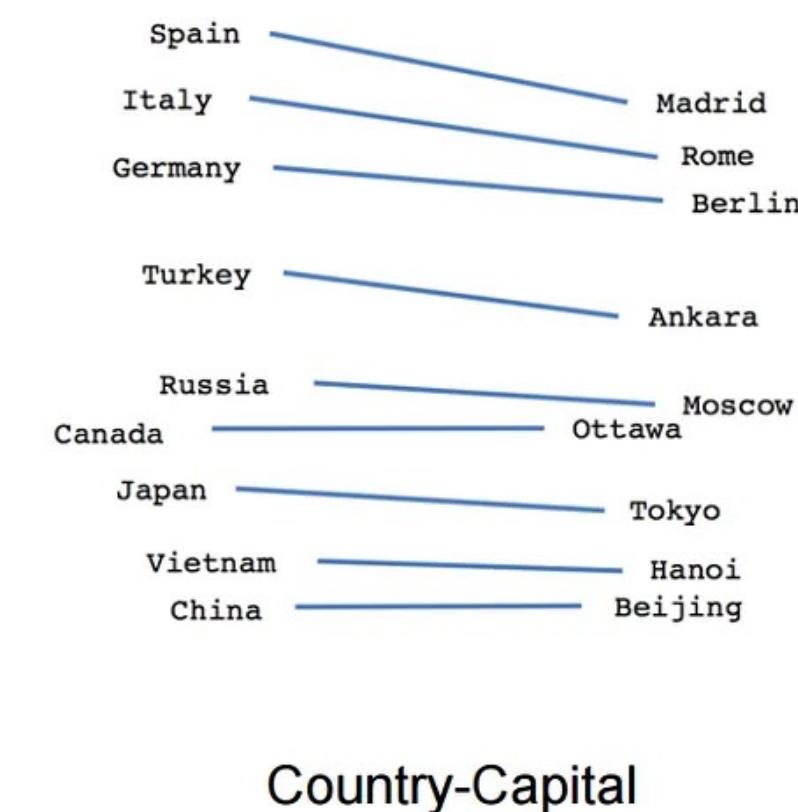
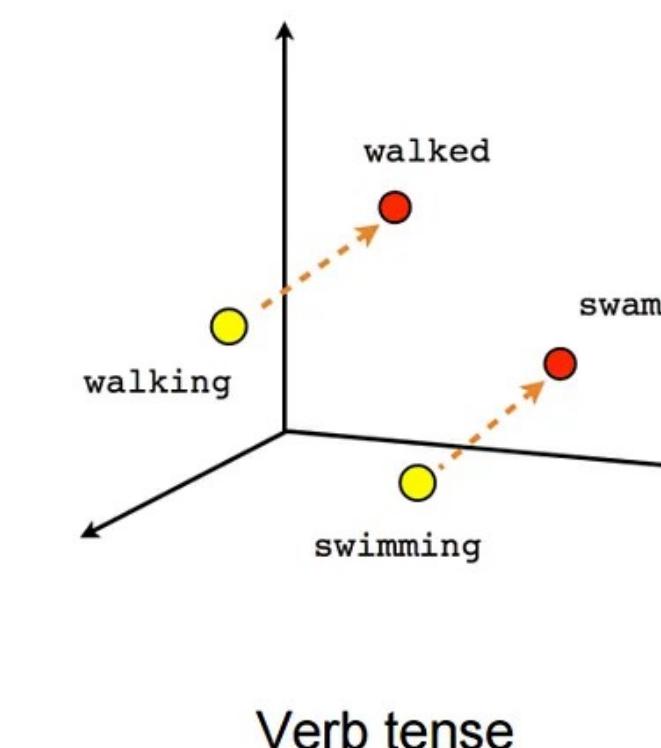
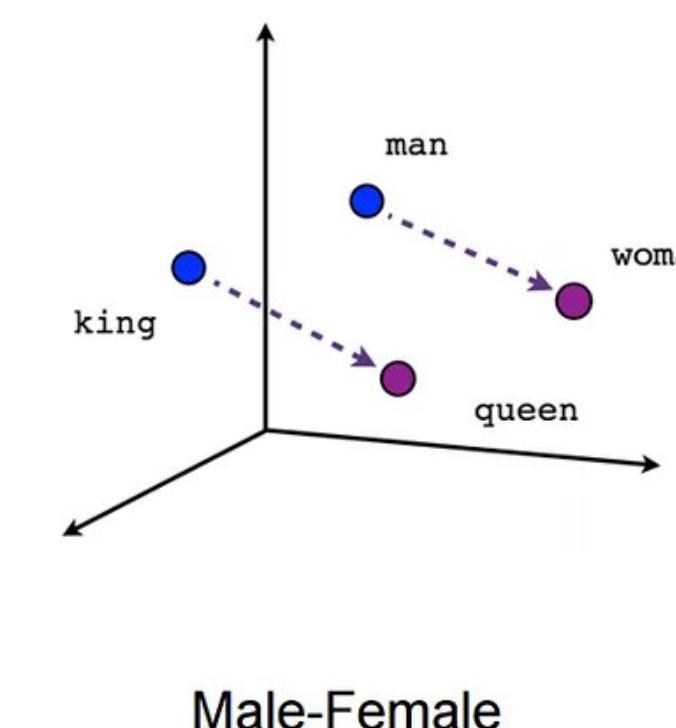
Language Model이란 무엇인가?

컴퓨터가 이해하는 텍스트란?

Unicode "곰" : U+**ACFO**



$$\mathbf{v} = (1, 3, 4, \dots, \dots,)$$

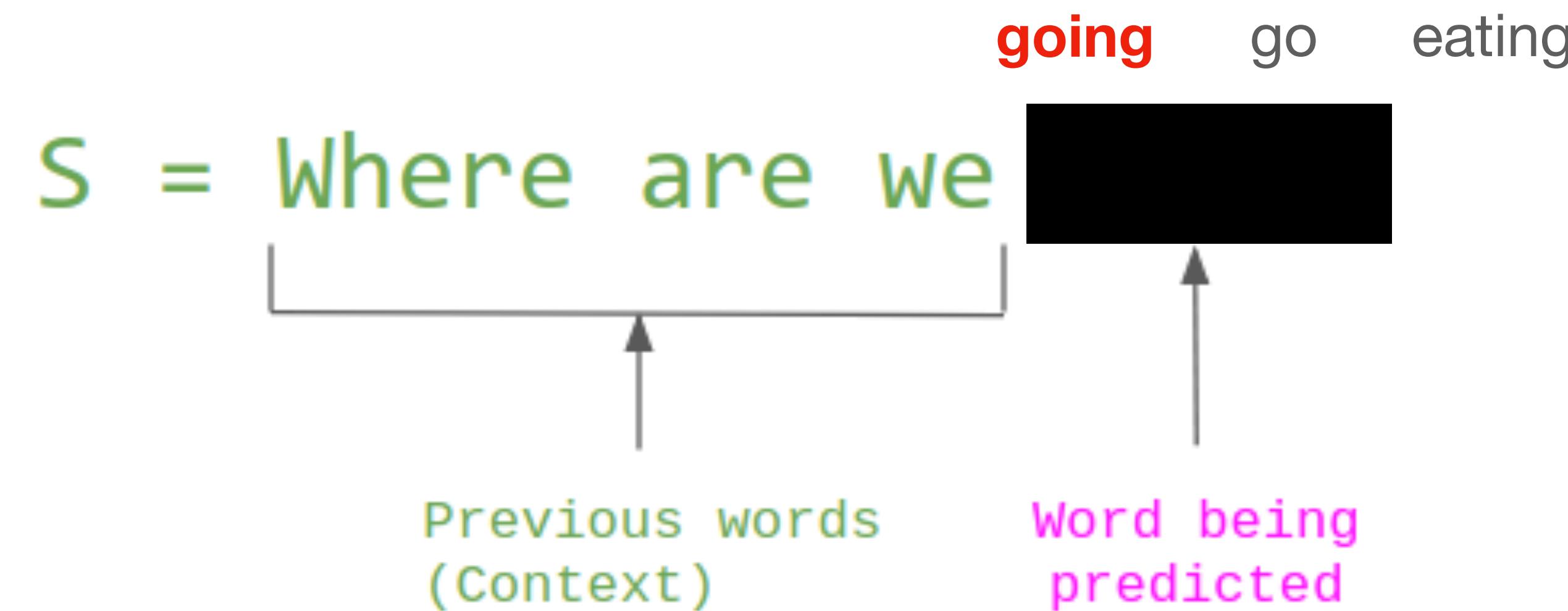


character encoding

(word) embedding

Language Model이란 무엇인가?

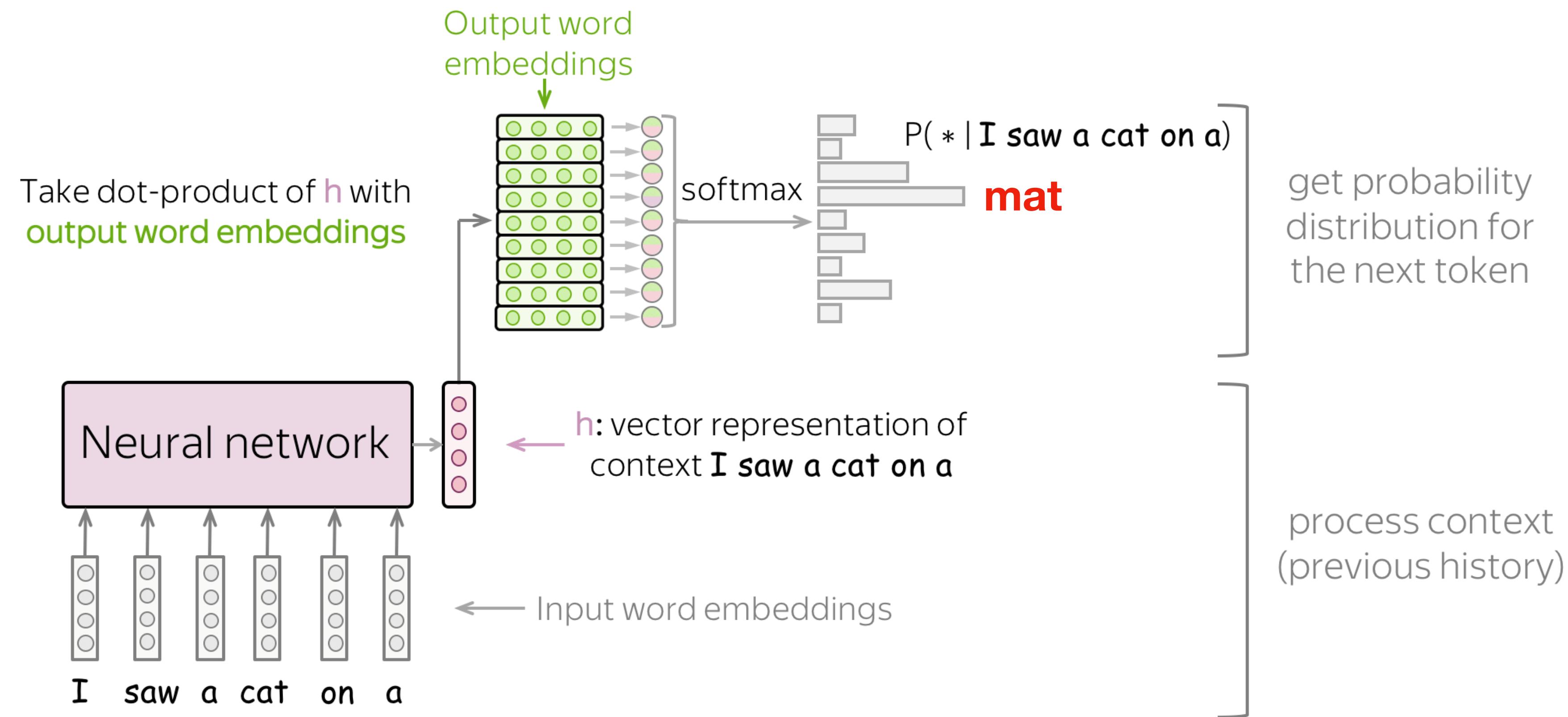
맥락의 이해 = Next Token Prediction



$$P(S) = P(\text{Where}) \times P(\text{are} \mid \text{Where}) \times P(\text{we} \mid \text{Where are}) \times P(\text{going} \mid \text{Where are we})$$

Language Model이란 무엇인가?

Neural Language Modeling

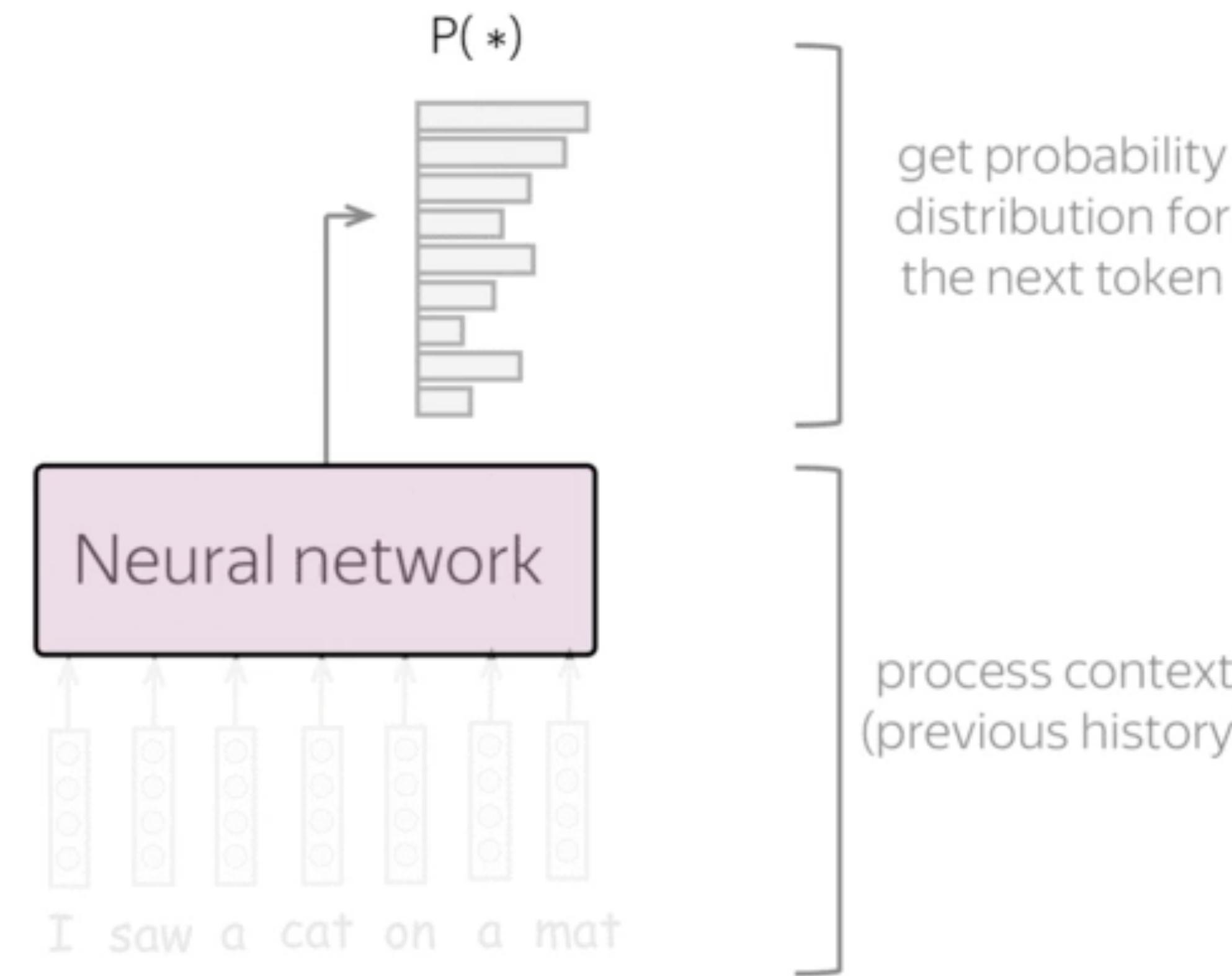


Source : https://lena-voita.github.io/nlp_course/language_modeling.html

(c) 2025. codingiscoffee Co. all rights reserved.

Language Model이란 무엇인가?

Neural Language Modeling **in Auto-Regressive manner**

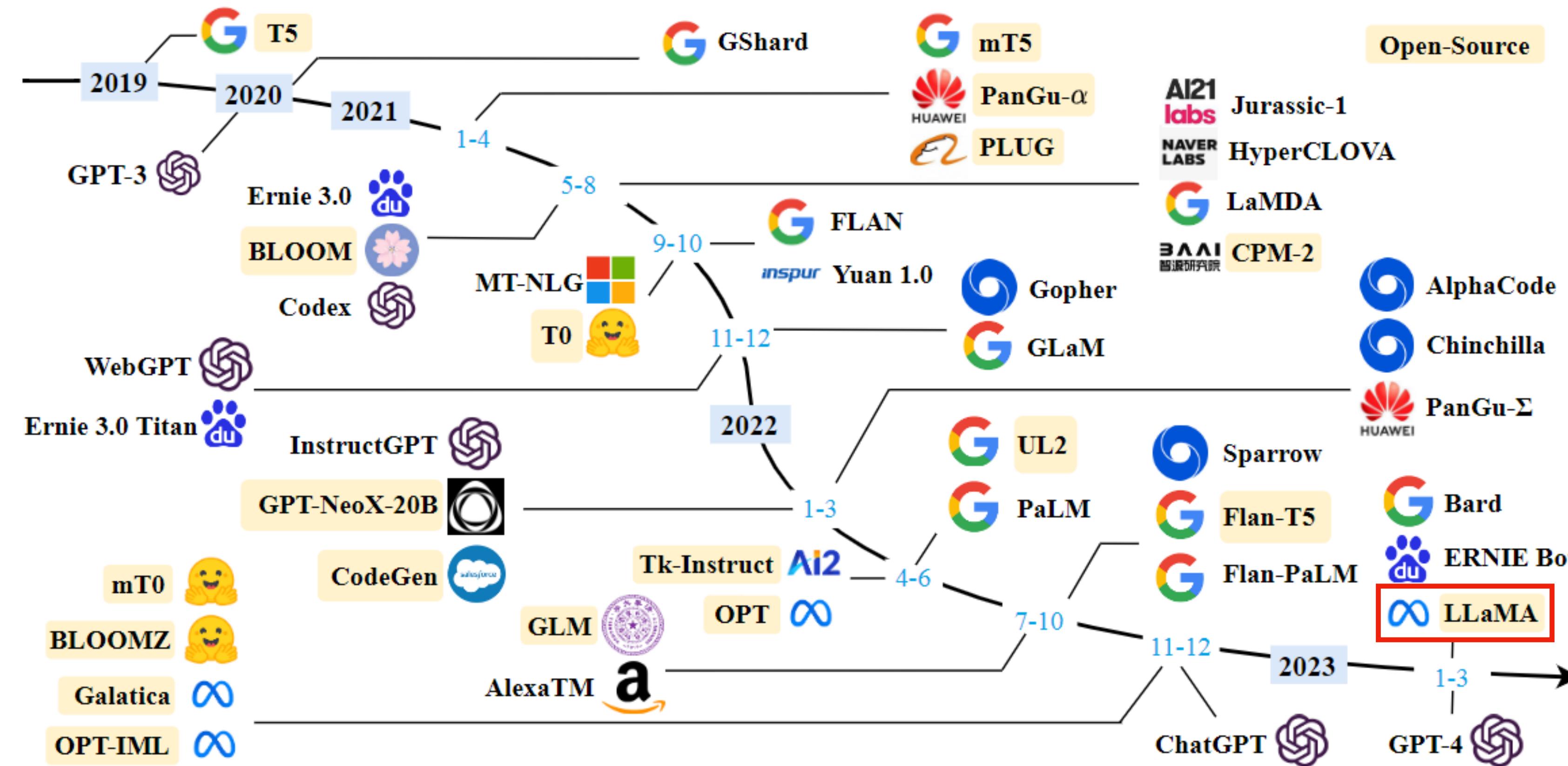


Source : https://lena-voita.github.io/nlp_course/language_modeling.html

(c) 2025. codingiscoffee Co. all rights reserved.

이제는 대LLM의 시대

모든 문제를 LLM으로 해결해보자!

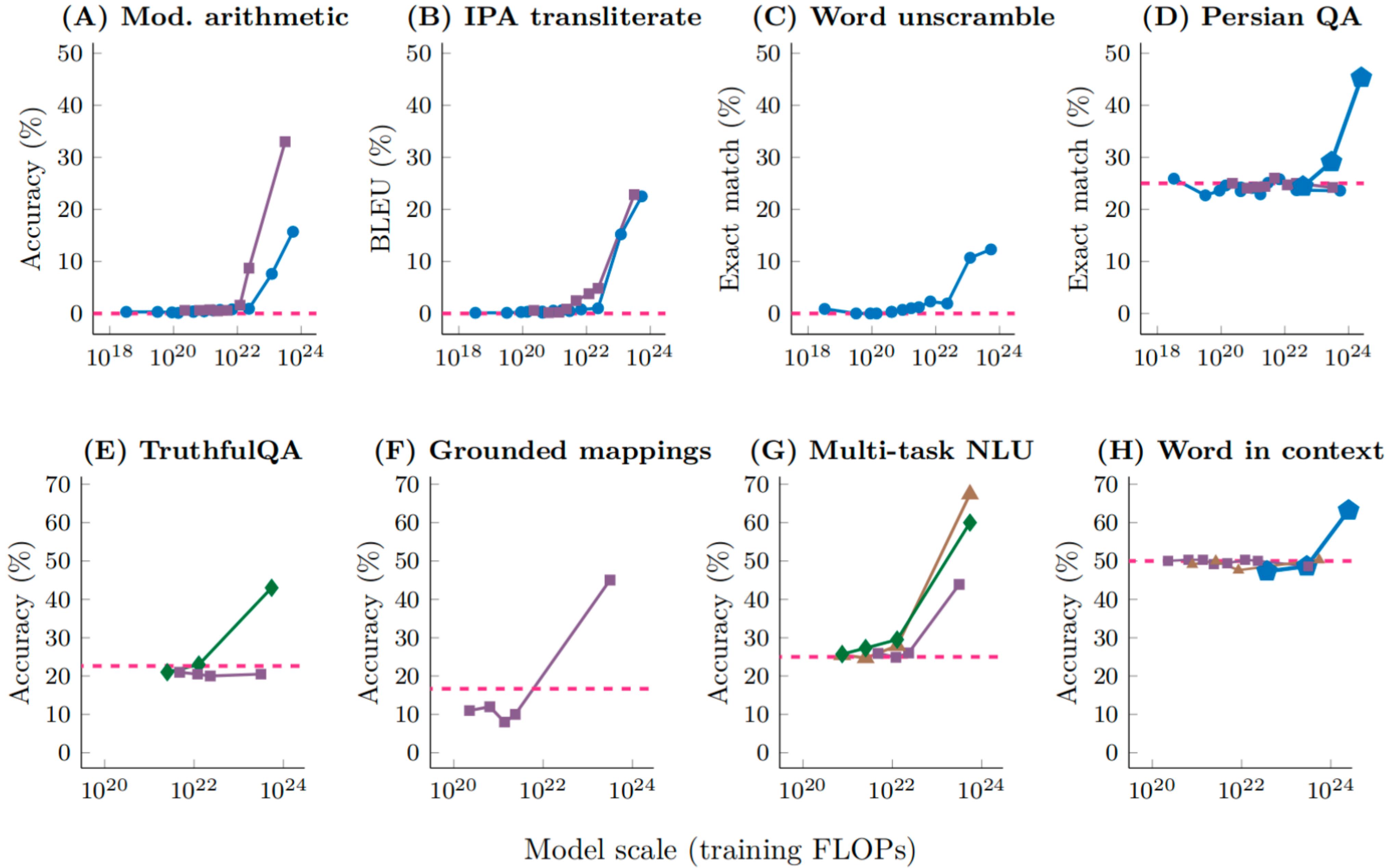


Large Language Model

Neural Scaling Law

- GPT-2 모델 이후부터 학습 규모를 키우면 성능이 올라갈거라는 믿음이 생김
 - Size of the model : 모델의 파라미터 수 (Transformer를 기반으로)
 - Size of the training dataset : 학습하는 데이터의 수
- 단, 모델과 데이터의 크기가 커지면 학습에 필요한 비용도 증가함.
 - GPT-4가 약 \$63M 정도 들었다고 알려짐 (unofficial)
 - A100 x 2500 x 100 days

● LaMDA ■ GPT-3 ◆ Gopher ▲ Chinchilla ▲ PaLM - - Random



Large Language Model Emergent Ability

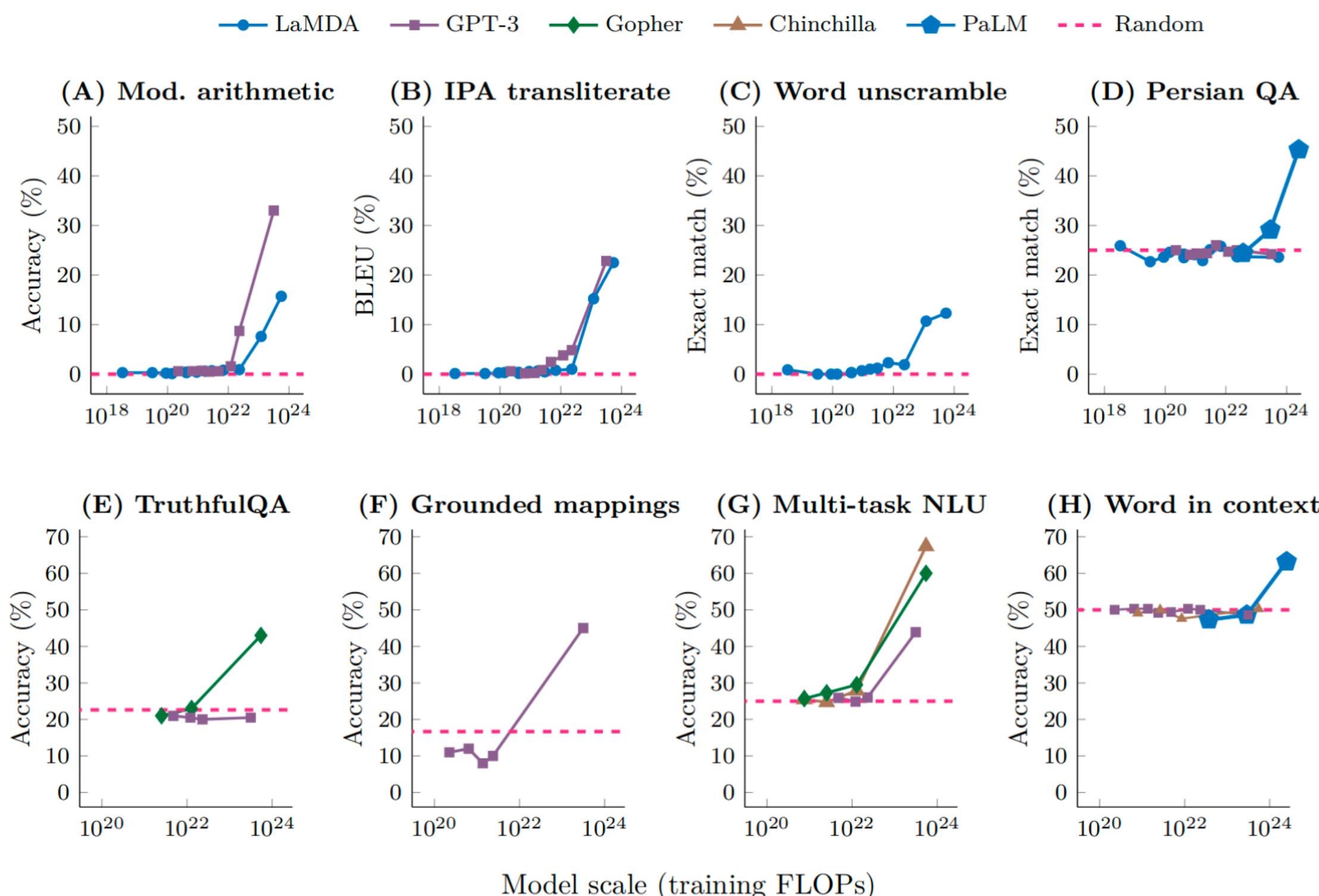


Figure 1: **Emergent abilities of large language models.** Model families display *sharp* and *unpredictable* increases in performance at specific tasks as scale increases. Source: Fig. 2 from [33].

- 파라미터 수가 특정 개수를 넘어가면 그 때 성능이 확 증가하는 구간이 존재함
- 이것을 LLM의 "Emergent ability" 라고 정의함
- 더 많은 경험을 할 수록 기존에 하지 못하던 새로운 인사이트를 이끌어 내는 느낌

Large Language Model as Foundation Model

- CV, NLP, RS 등의 모든 분야에서 큰 LLM 하나를 가지고 여러 task를 모두 수행하기 시작함
- 대표적인 예로 ChatGPT-4o-with-canvas 같은 서비스들이 있음
- LLM + Multi-modal approach로 LMM(Large Multi-Model)로 불리우며 지금은 대부분의 모델이 Vision & Language 만을 담당하고 있음
- 대부분의 데이터가 이미지 또는 텍스트로 변환되며 이를 처리할 수 있으면 대다수의 정보 처리가 가능

Large Language Model as Foundation Model

- But, LLM은 학습하는데 비용이 너무 많이 든다는 문제가 있음. (데이터 + 인프라)
- Also, Open-domain의 상황에서 모든 응답에 높은 퀄리티의 답변을 생성하는 것에는 무리가 있음.
- 그리고, Google Deepmind의 Chinchilla 등장 이후, 더 적은 파라미터로 높은 성능을 내는 모델들이 등장하기 시작함.

Small Language Model

LLM은 너무 비용이 많이 듈다

Model	필요 GPU 갯수	A100 한장 시간 당 가격	시간 당 가격	10시간 학습
LLaMA 7B 모델 Full Finetuning	A100 × 4	\$1.10 / hr	\$4.40 / hr	\$44.0
LLaMA 7B 모델 peft + deepspeed	A100	\$1.10 / hr	\$1.10 / hr	\$11.0
LLaMA 65B 모델 Full Finetuning	A100 × 64	\$1.10 / hr	\$70.4 / hr	\$704
LLaMA 65B 모델 peft + deepspeed	A100 × 16	\$1.10 / hr	\$17.6 / hr	\$176

Small Language Model

LLM은 너무 비용이 많이 듈다

- Emergent ability의 등장으로 인해 큰 사이즈 모델이 각광받기 시작했으나 엄청난 비용이 발생하기 때문에 모든 회사들이 LLM을 계속 쓰기는 힘듬
- 이전에 Pretrained LM을 용도에 맞게 사용했듯, 가성비 있는 LM이 필요해짐
- 현재는 많은 실험을 통해서 < 13B 정도의 모델들을 주로 사용함

Small Language Model

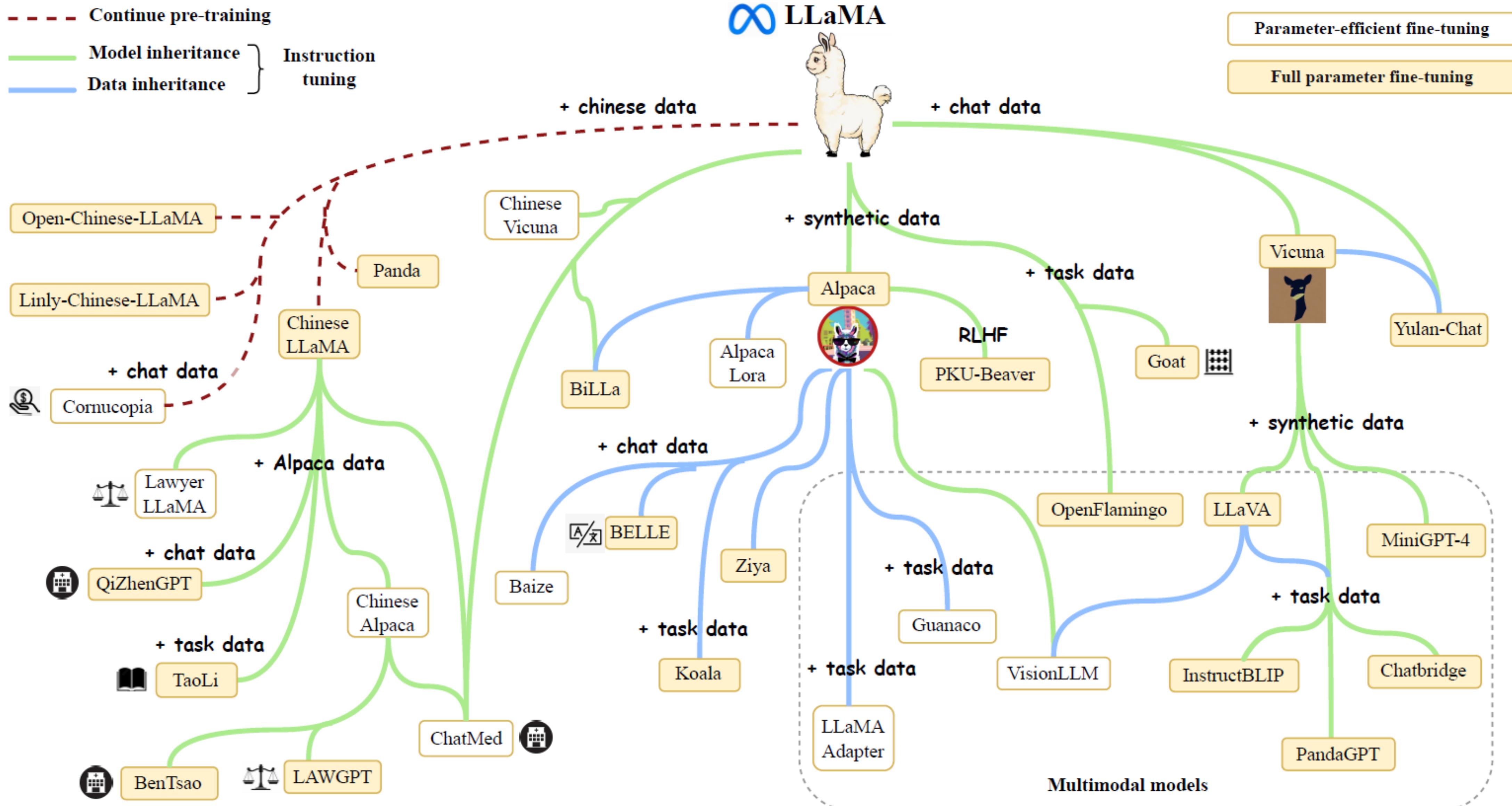
open-source Language Model의 등장

- Meta(구 Facebook)는 원래 AI Research의 강자였으나, Metaverse에 올인하느라 AI Follow-up이 좀 늦어짐.
- GPT-3부터 모델이 공개가 되지 않음에 따라 영업비밀들이 점점 늘어나기 시작.
- openAI vs Google의 구도로 경쟁
- Meta는 새로운 돌파구로 open-source LM을 공개함
 - OPT(Open Pretrained Transformer)를 무료로 올림
 - 125M ~ 175B까지 다양한 사이즈의 모델을 공개

Small Language Model

Llama(Large Language Model Meta AI)

- 2023년 2월 Meta가 아주 뛰어난 open-source LM Llama를 발표
- 7B, 13B, 33B, 65B 4개의 모델을 공개
- 7B, 13B는 1T token을 사용했고, 33B, 65B는 1.4T token을 학습에 사용함
- 이전까지 연구되어온 최신 기법들을 모아서 엄청난 연구비를 대신 써줌(?)



Llama의 시대

Llama2

- 2023년 7월 Llama의 대哄행에 힘입어 이어서 Llama2를 발표
- 7B, 13B, 70B 3개의 사이즈로 개발했으며 Chat-model도 추가로 개발 (Llama2-7B, Llama2-Chat-7B, ...)
- 모든 모델 2T token을 사용했고, 입력길이는 4096 tokens
- Llama 대비 성능도 모두 증가했고, 학습에 사용한 기법과 데이터가 바뀜

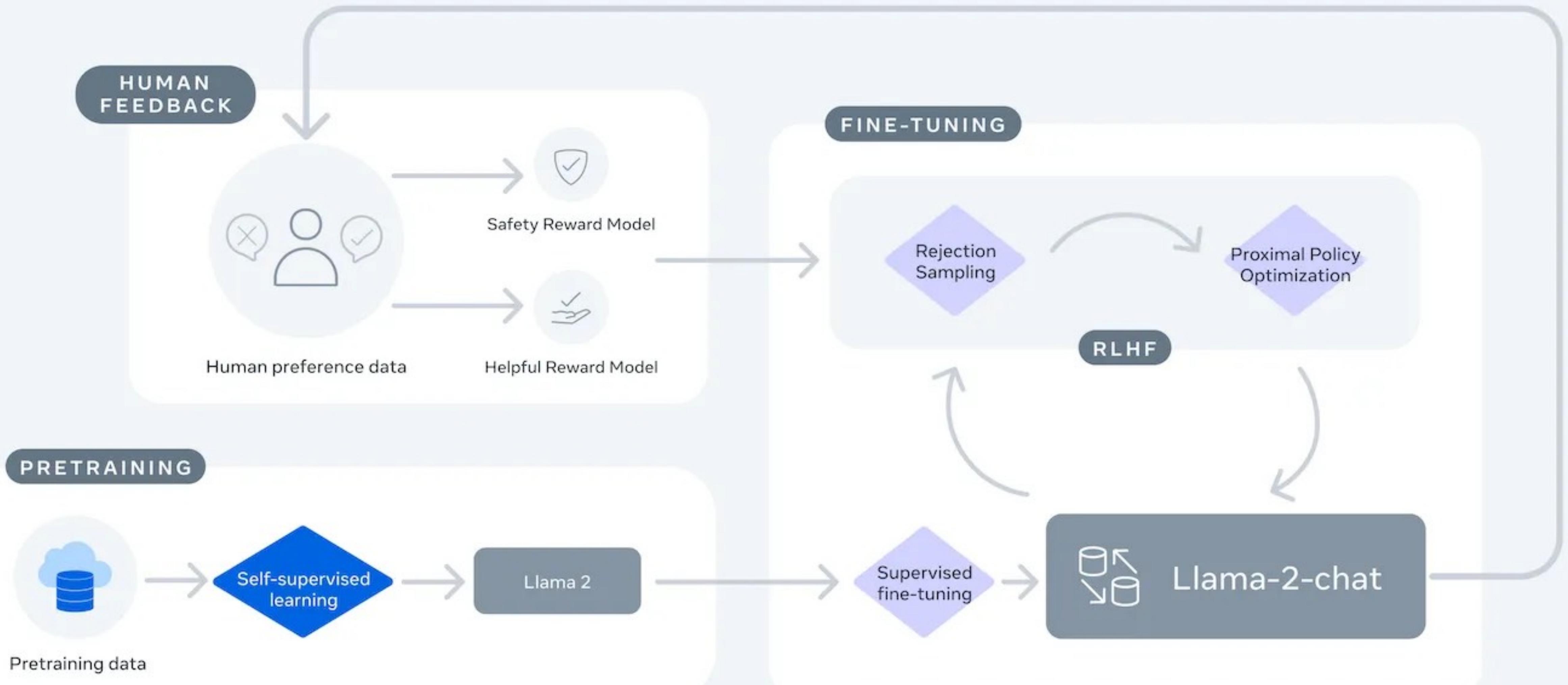
Llama의 시대

Llama vs Llama2

Model	Size	Code	Commonsense Reasoning	World Knowledge	Reading Comprehension	Math	MMLU	BBH	AGI Eval
Llama 1	7B	14.1	60.8	46.2	58.5	6.95	35.1	30.3	23.9
Llama 1	13B	18.9	66.1	52.6	62.3	10.9	46.9	37.0	33.9
Llama 1	33B	26.0	70.0	58.4	67.6	21.4	57.8	39.8	41.7
Llama 1	65B	30.7	70.7	60.5	68.6	30.8	63.4	43.5	47.6
Llama 2	7B	16.8	63.9	48.9	61.3	14.6	45.3	32.6	29.3
Llama 2	13B	24.5	66.9	55.4	65.8	28.7	54.8	39.4	39.1
Llama 2	70B	37.5	71.9	63.6	69.4	35.2	68.9	51.2	54.2

Source : <https://Llama-2.ai/Llama-2-model-details/>

(c) 2025. codingiscoffee Co. all rights reserved.



Llama의 시대

Llama3

- Llama, Llama2가 완전히 자리를 잡자 2024년 4월 Llama3도 발표
- 8B, 70B 2가지 모델을 발표
- 모든 모델 15T+ token을 학습했고, 입력길이는 8192 tokens
- Llama2에서 학습한 데이터의 크기를 증가시키고, 대세에 따라 더 많은 입력을 받을 수 있게 변경

Meta Llama 3 Instruct model performance

	Meta Llama 3 8B	Gemma 7B - It Measured	Mistral 7B Instruct Measured
MMLU 5-shot	68.4	53.3	58.4
GPQA 0-shot	34.2	21.4	26.3
HumanEval 0-shot	62.2	30.5	36.6
GSM-8K 8-shot, CoT	79.6	30.6	39.9
MATH 4-shot, CoT	30.0	12.2	11.0

	Meta Llama 3 70B	Gemini Pro 1.5 Published	Claude 3 Sonnet Published
MMLU 5-shot	82.0	81.9	79.0
GPQA 0-shot	39.5	41.5 CoT	38.5 CoT
HumanEval 0-shot	81.7	71.9	73.0
GSM-8K 8-shot, CoT	93.0	91.7 11-shot	92.3 0-shot
MATH 4-shot, CoT	50.4	58.5 Minerva prompt	40.5

Llama의 시대

Llama3.1

- Llama3가 엄청나게 큰 영향을 주면서 자리를 잡자, 이어 24년 7월에 3.1을 발표
- 8B, 70B, 405B 3가지 모델을 발표함. (405B는 오픈소스 중 가장 큰 크기의 모델)
- 모든 모델 15T+ token을 학습했고, 입력길이는 128K tokens
- Llama3보다 더 많은 언어에 대한 공식지원과 더 다양한 task에 대한 성능을 향상시킴

Llama의 시대

Llama3.1

- Llama 3.1까지 적용된 기술들은 다음과 같다.
 - RoPE(Rotary Positional Embedding)를 사용하여 긴 컨텍스트도 잘 이해할 수 있게 학습
 - 한번에 최대 128K tokens 입력 가능
 - trained over 15B tokens
 - Grouped-Query Attention 적용
 - RLHF로 모델의 응답이 safety guide를 준수하도록 학습함
 - 16,000개의 H100 cluster를 사용함

Category	Llama 3.1 405B	Nemotron 4 340B Instruct	GPT-4 (0125)	GPT-4 Omni	Claude 3.5 Sonnet
Benchmark					
General					
MMLU (0-shot, CoT)	88.6	78.7 (non-CoT)	85.4	88.7	88.3
MMLU PRO (5-shot, CoT)	73.3	62.7	64.8	74.0	77.0
IFEval	88.6	85.1	84.3	85.6	88.0
Code					
HumanEval (0-shot)	89.0	73.2	86.6	90.2	92.0
MBPP EvalPlus (base) (0-shot)	88.6	72.8	83.6	87.8	90.5
Math					
GSM8K (8-shot, CoT)	96.8	92.3 (0-shot)	94.2	96.1	96.4 (0-shot)
MATH (0-shot, CoT)	73.8	41.1	64.5	76.6	71.1
Reasoning					
ARC Challenge (0-shot)	96.9	94.6	96.4	96.7	96.7
GPQA (0-shot, CoT)	51.1	-	41.4	53.6	59.4
Tool use					
BFCL	88.5	86.5	88.3	80.5	90.2
Nexus	58.7	-	50.3	56.1	45.7
Long context					
ZeroSCROLLS/QuALITY	95.2	-	95.2	90.5	90.5
InfiniteBench/En.MC	83.4	-	72.1	82.5	-
NIH/Multi-needle	98.1	-	100.0	100.0	90.8
Multilingual					
Multilingual MGSM (0-shot)	91.6	-	85.9	90.5	91.6

Copyright reserved.

2. Llama 3.2를 통해 보는 Large Multi-Model(LMM)의 이해

Llama의 시대

Llama3.2

- 24년 7월에 3.1을 발표한지 얼마안되어 갑자기(?) 24년 9월에 3.2를 발표
- 2가지의 큰 키워드로 **Lightweight**과 **Multimodal**을 잡고 모델을 발표
- Lightweight용 1B, 3B를 Multimodal용 11B, 90B 사이즈를 출시
- 3.1과 동일하게 모든 모델 15T+ token을 학습했고, 입력길이는 128K tokens
- 3.1때보다 좀 더 서비스 지향적으로 바뀌었으며 AWS, Nvidia등 다양한 회사들과 협업

Category Benchmark	Llama 3.2 1B	Llama 3.2 3B	Gemma 2 2B IT (5-shot)	Phi-3.5 - Mini IT (5-shot)
General				
MMLU (5-shot)	49.3	63.4	57.8	69.0
Open-rewrite eval (0-shot, rougeL)	41.6	40.1	31.2	34.5
TLDR9+ (test, 1-shot, rougeL)	16.8	19.0	13.9	12.8
IFEval	59.5	77.4	61.9	59.2
Math				
GSM8K (0-shot, CoT)	44.4	77.7	62.5	86.2
MATH (0-shot, CoT)	30.6	48.0	23.8	44.2
Reasoning				
ARC Challenge (0-shot)	59.4	78.6	76.7	87.4
GPQA (0-shot)	27.2	32.8	27.5	31.9
Hellaswag (0-shot)	41.2	69.8	61.1	81.4
Tool use				
BFCL V2	25.7	67.0	27.4	58.4
Nexus	13.5	34.3	21.0	26.1

rved.

Modality	Category Benchmark	Llama 3.2 11B	Llama 3.2 90B	Claude3 - Haiku	GPT-4o-mini
Image	College-level Problems and Mathematical Reasoning	50.7	60.3	50.2	59.4
	MMMU (val, 0-shot CoT, micro avg accuracy)				
	MMMU-Pro, Standard (10 opts, test)	33.0	45.2	27.3	42.3
	MMMU-Pro, Vision (test)	27.3	33.8	20.1	36.5
	MathVista (testmini)	51.5	57.3	46.4	56.7
	Charts and Diagram Understanding				
	ChartQA (test, 0-shot CoT, relaxed accuracy)*	83.4	85.5	81.7	-
	AI2 Diagram (test)*	91.9	92.3	86.7	-
	DocVQA (test, ANLS)*	88.4	90.1	88.8	-
	General Visual Question Answering				
Text	VQAv2 (test)	75.2	78.1	-	-
	General				
	MMLU (0-shot, CoT)	73.0	86.0	75.2 (5-shot)	82.0
	Math				
Text	MATH (0-shot, CoT)	51.9	68.0	38.9	70.2
	Reasoning				
	GPQA (0-shot, CoT)	32.8	46.7	33.3	40.2
Text	Multilingual				
	MGSM (0-shot, CoT)	68.9	86.9	75.1	87.0

Llama의 시대

Llama3.2

- Llama 3.2가 발표되던 시점의 주요 키워드는 “on-device AI”와 “Large Multi-Model”이었음.
- 진짜 킬러 서비스가 등장하려면, 헤비하지 않게도 일상 생활에 도움이 되는 여러 디바이스들 (e.g. 스마트폰, 전기차, 스마트 가전 등)에서 작동해야 함
 > on-device AI
- 입력이 텍스트가 아닌 이미지, 오디오가 되면 훨씬 더 확장성이 커짐.
 > 이미지 입력을 받을 수 있는 LLM이 생기면 사용성이 증가함.

Llama의 시대

Llama3.2

- Lightweight model들은 Llama 3.1의 8B, 70B 모델을 Knowledge Distillation하여 학습시킴
- SpinQuant와 QLoRA를 적용하여 모델 크기는 52~60% 감소시키고, 추론 속도는 2.4~2.6배 향상시킴
- Llama 3.1보다 더 많은 안전성 테스트를 진행함
(adversarial prompting에 대한 red teaming도 수행)

Llama의 시대

Llama3.2 - Lightweight model

- Edge-device (스마트폰, 스마트카 등)에서 자체적으로 inference를 할 수 있으려면 적은 리소스로 도 충분한 성능을 내야함.
- 그러기 위해선 다양한 경량화 기법들을 적용하면서도 성능을 유지시키기 위한 노력들이 필요함.
 1. Model quantization
 2. PyTorch's ExecuTorch
 3. Quantization-Aware Training(QAT)
 4. Parameter Efficient Fine-Tuning(PEFT)
 5. Post Training

Llama의 시대

Llama3.2 - 1. Model Quantization

- Model Quantization은 같은 크기를 가지는 모델을 가볍게 만들기 위한 기법들로 파라미터의 수는 유지하되 사용하는 메모리량을 줄이는 것에 목적이 있음
- 가장 간단한 방법은 기존에 float32로 표현되던 실수형 데이터 타입을 bfloat16으로 변경하는 것. 말그대로 메모리 사용량이 절반으로 줄어듬.

bfloat16: Brain Floating Point Format



Range: $\sim 1e^{-38}$ to $\sim 3e^{38}$

fp32: Single-precision IEEE Floating Point Format

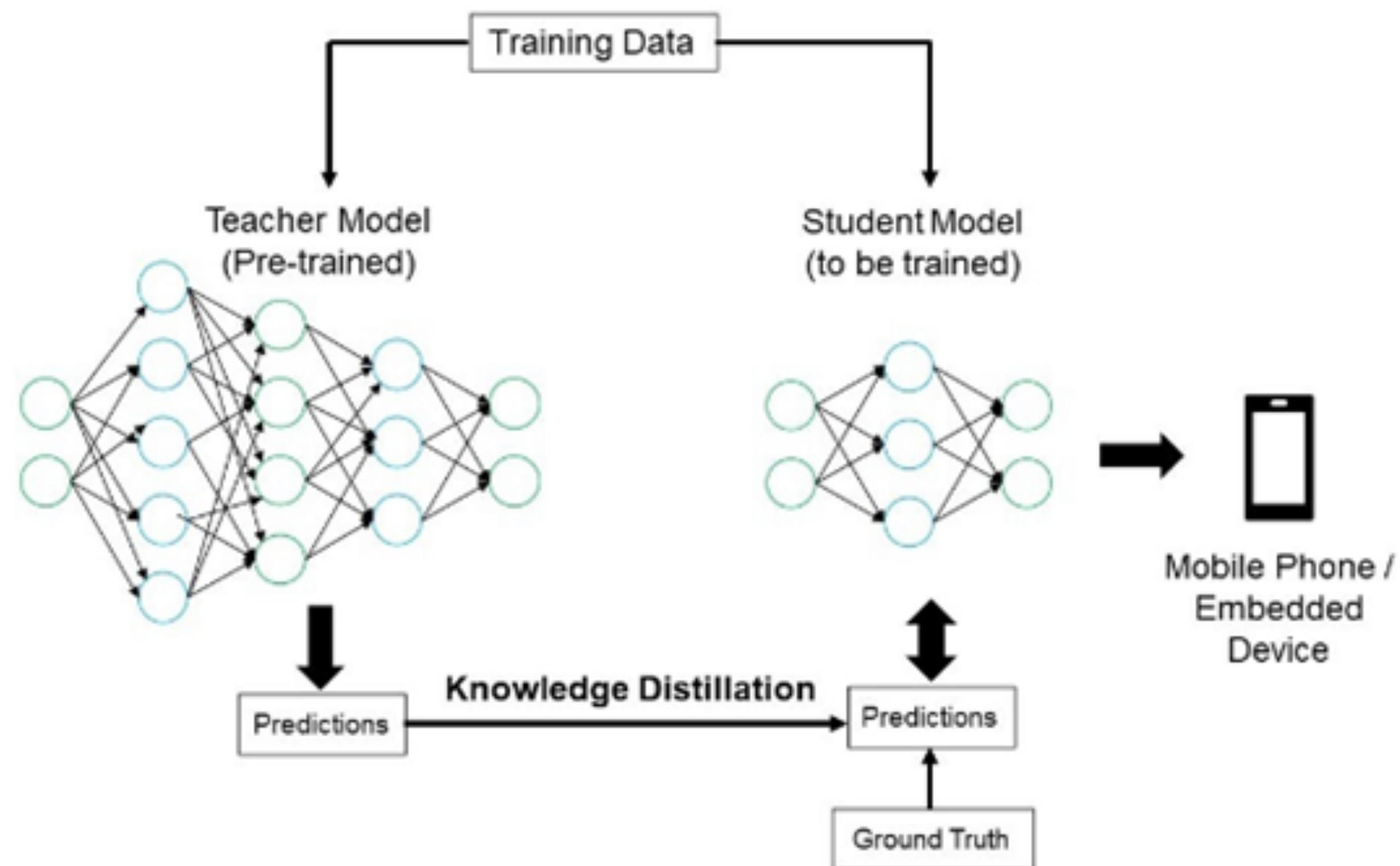


Range: $\sim 1e^{-38}$ to $\sim 3e^{38}$

Llama의 시대

Llama3.2 - 1. Model Quantization

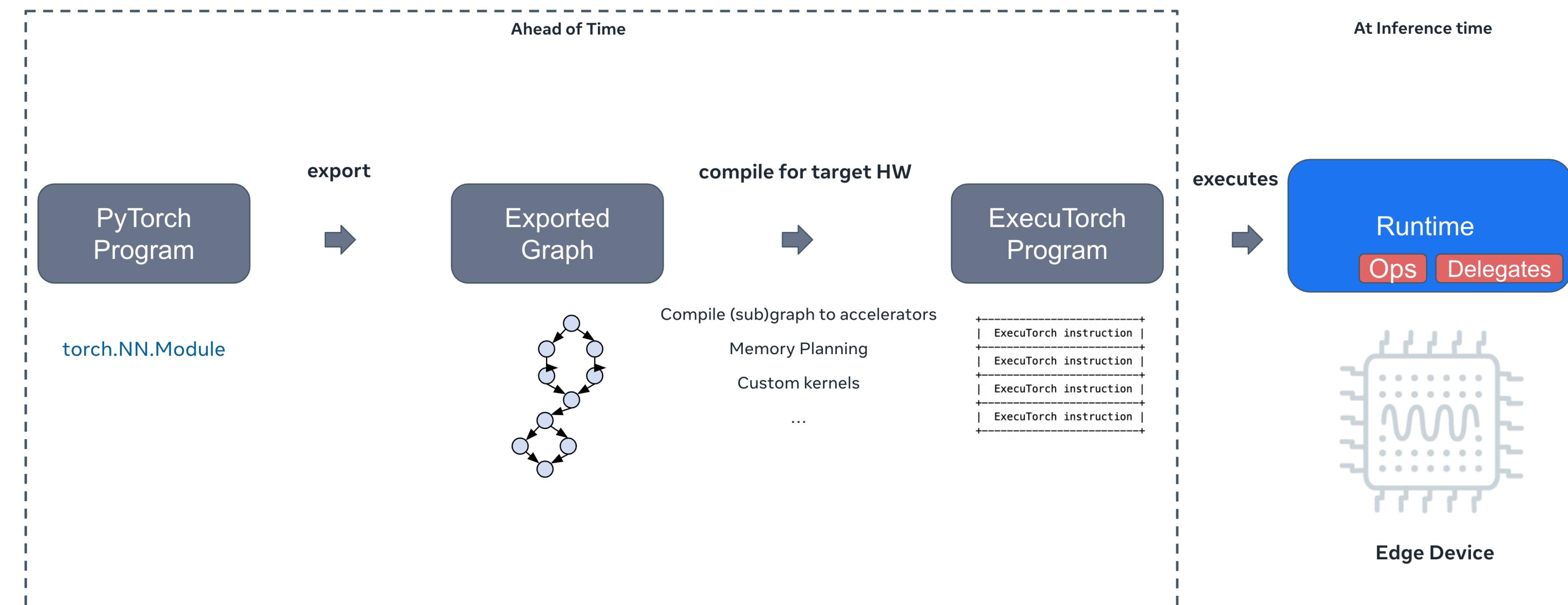
- 다른 방식으로는 큰 모델의 학습 결과를 모방하는 Knowledge Distillation이 있음. 이 방식은 작은 모델의 출력(logit)이 큰 모델의 출력(logit)과 같아지게 만드는 것에 있음.



Llama의 시대

Llama3.2 - 2. ExecuTorch

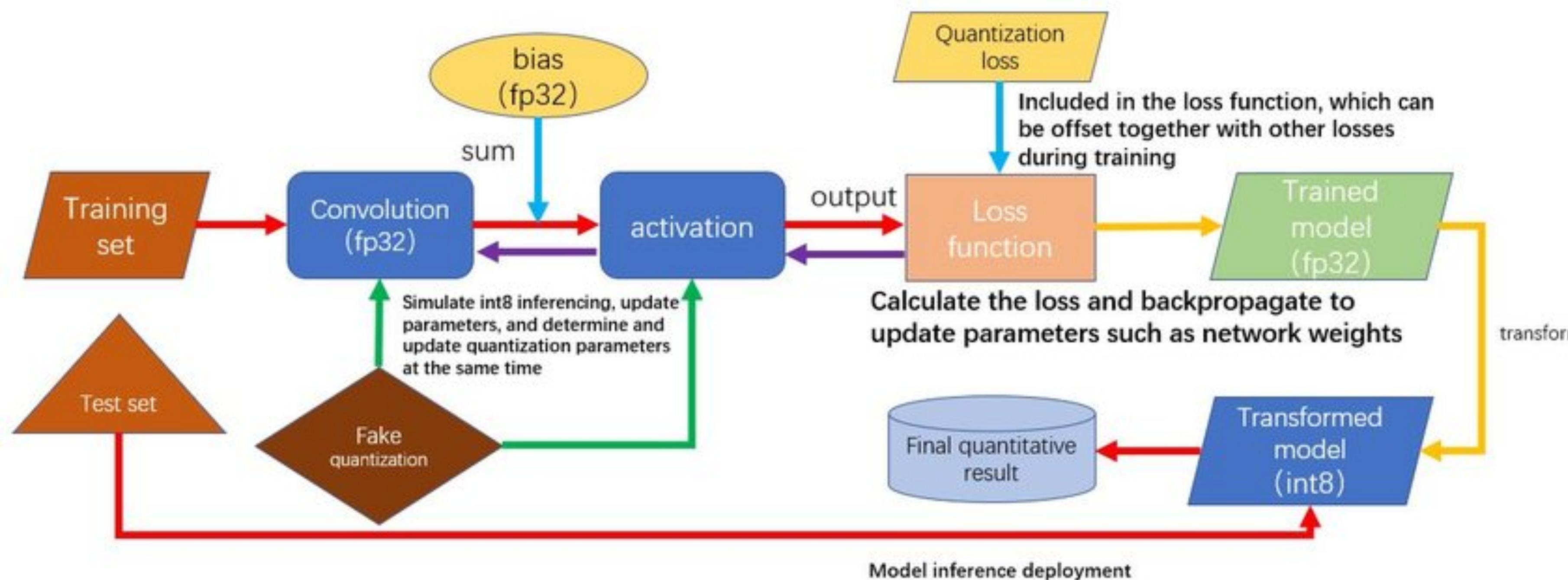
- 학습에 주로 사용하는 framework인 Pytorch에서 나온 on-device AI inference를 위한 ExecuTorch라는 framework를 사용하면 edge device의 CPU에서 빠르게 inference가 가능



Llama의 시대

Llama3.2 - 3. QAT

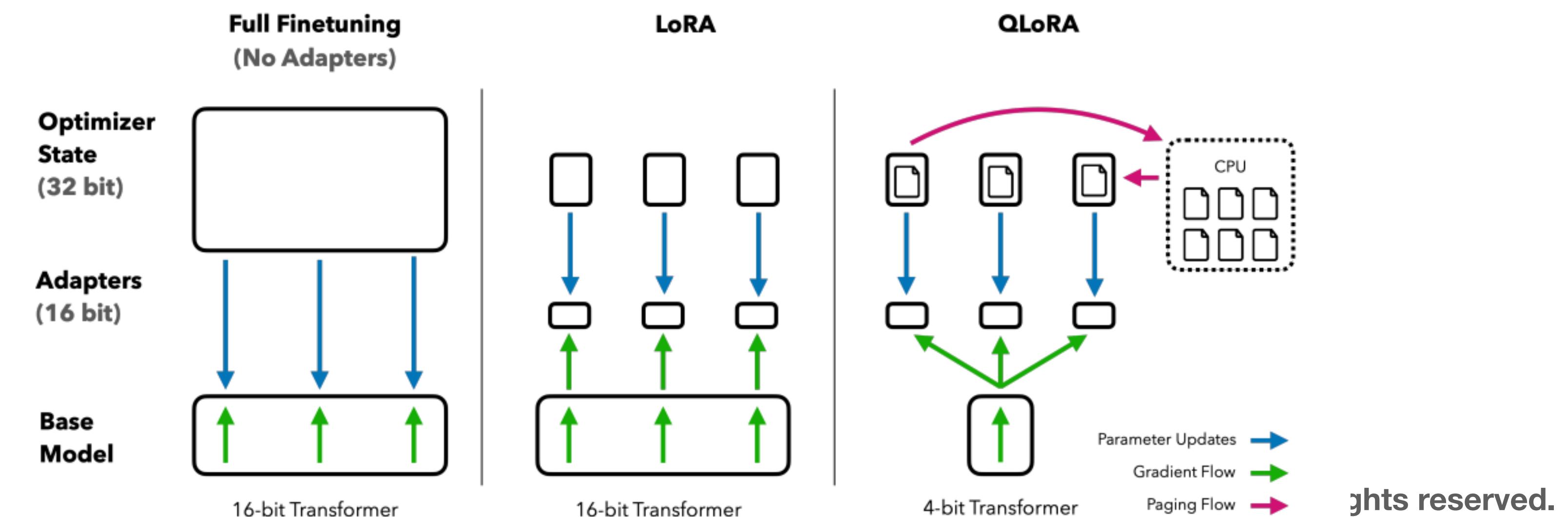
- 학습 과정에서 fake quantization을 하여 마치 quantization을 수행하는 것처럼 학습을 진행하면 INT8 데이터 타입을 사용하면서도 FP32에 가까운 성능을 낼 수 있는 기법



Llama의 시대

Llama3.2 - 4. PEFT

- Supervised Fine-Tuning을 수행할 때, 모든 weight를 업데이트하게 되면 많은 비용이 발생하므로 이를 효율적으로 하기 위해 업데이트되는 내용만 따로 추가적인 구조에 학습하는 것으로 적은 리소스로 학습한 효과를 내는 학습 방법
- 가장 유명한 기법엔 LoRA, QLoRA가 있는데 QLoRA가 quantization이 적용된 기법이라 더욱 효율적인 학습이 가능



Llama의 시대

Llama3.2 - 5. Post Training

- 일반적인 문맥과 정보를 학습하는 Pre-training 단계를 거친 Llama 3.2를 특정 task나 유저의 선호도에 맞는 답변을 생성하기 위해 수행하는 단계
- Supervised Fine-Tuning(SFT), Direct Preference Optimization(DPO), Rejection Sampling(RS)가 주로 사용됨.
- 이러한 방식을 사용하면 Llama 3.2를 Meta에서 원하는 방향대로 생성되는 답변의 퀄리티를 조절할 수 있음. (Alignment)

Llama의 시대

Llama3.2 - Vision

- LLM에 Vision Context를 추가하여, multi-modality를 추가함.
- Vision Reasoning이 가능해지면, 더욱 많은 서비스를 사용할 수 있음.
 - > 길을 물어볼 때, 말로 하는 것 vs 지도를 제공
 - > 코드 오류가 발생했을 때, 텍스트로 설명하는 것 vs 캡처 사진 제공
- 같은 맥락으로 Audio Context로 추가하는 식으로 계속해서 확장 가능함.

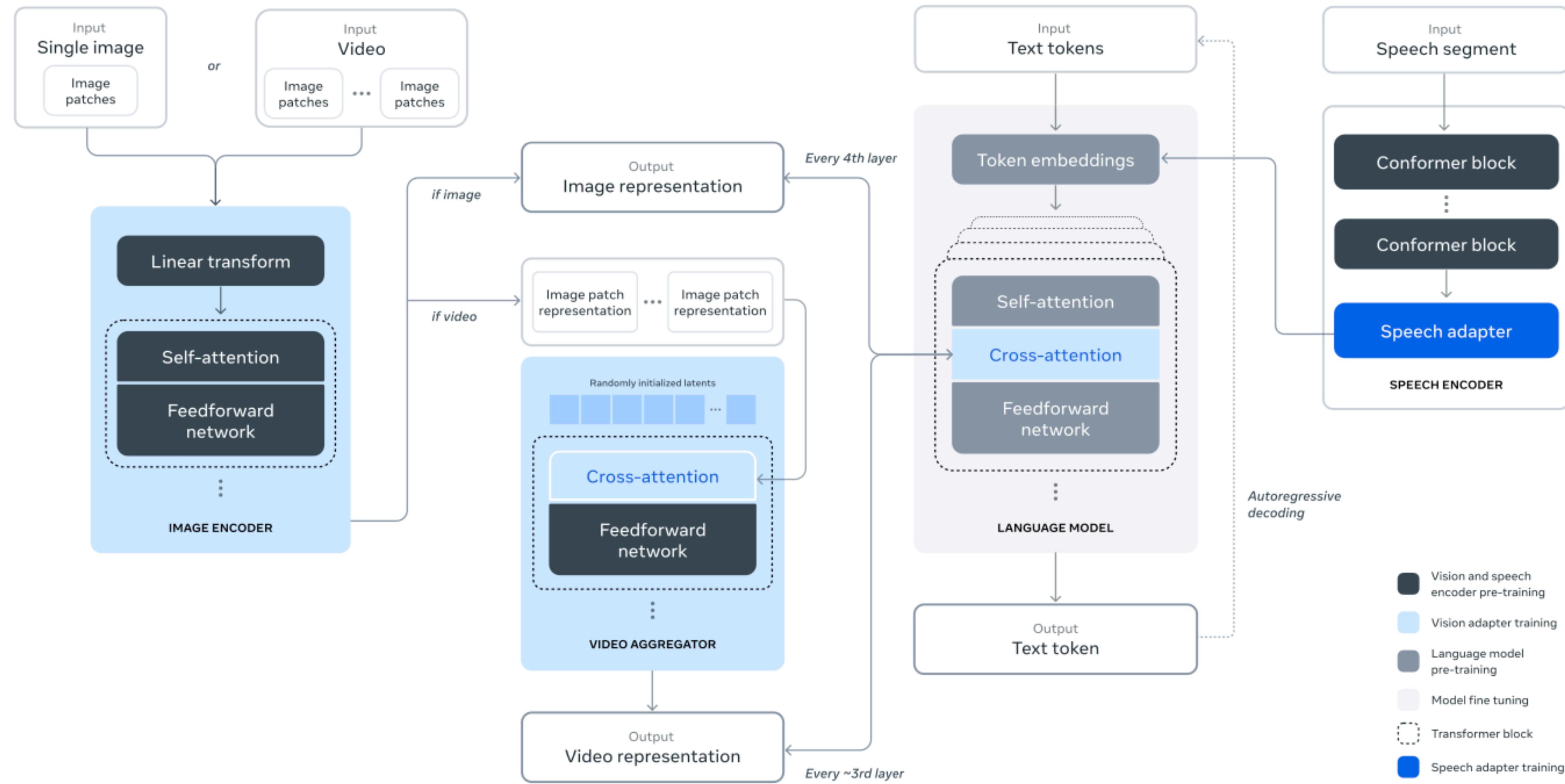
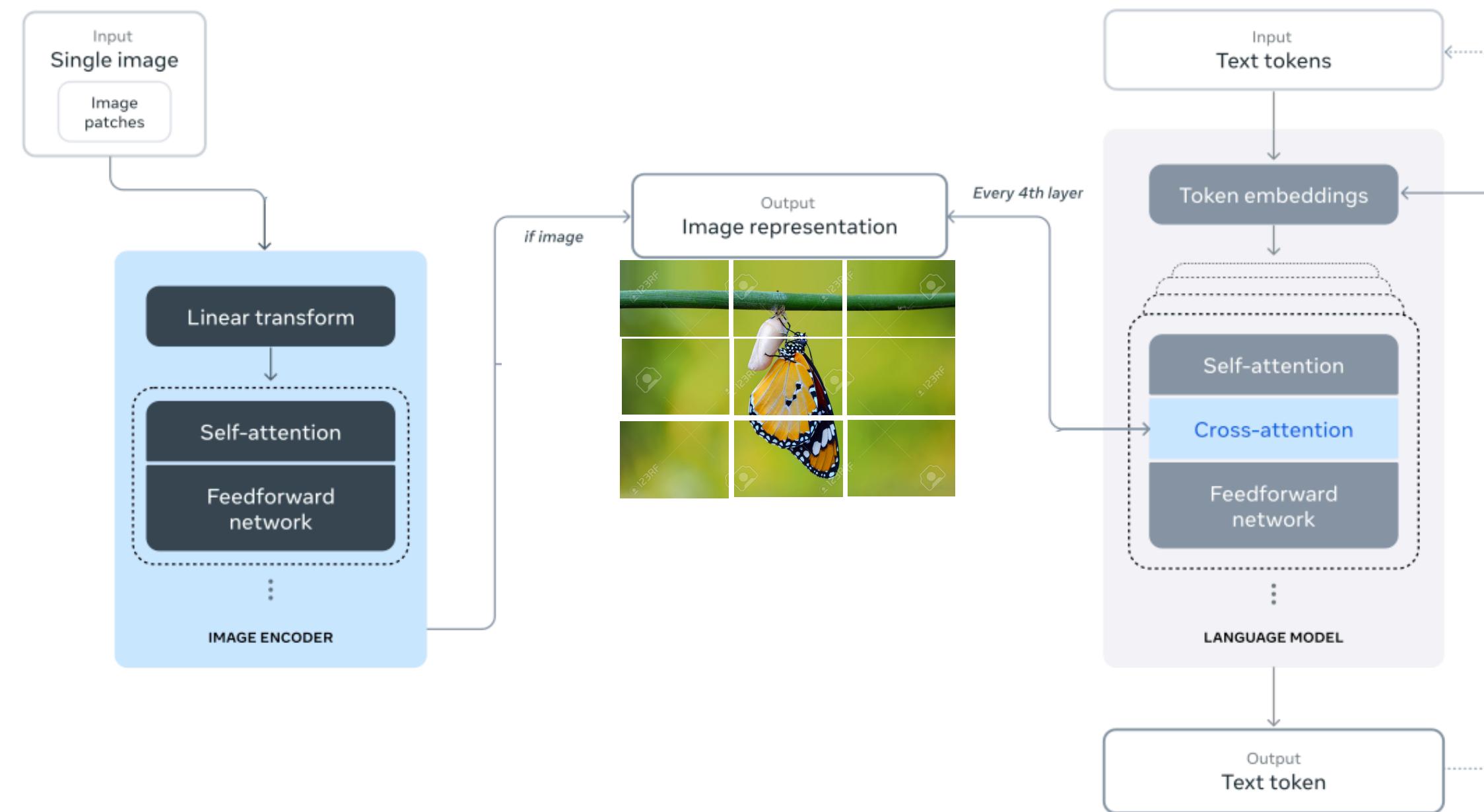


Figure 28 Illustration of the compositional approach to adding multimodal capabilities to Llama 3 that we study in this paper. This approach leads to a multimodal model that is trained in five stages: **(1)** language model pre-training, **(2)** multi-modal encoder pre-training, **(3)** vision adapter training, **(4)** model finetuning, and **(5)** speech adapter training.

Llama의 시대

Llama3.2 - Vision

- multi-modal의 기본 학습 원리는 input layer를 확장하는 것에 있음.
- 기존에 널리 사용되는 Image Encoder(ViT-H/14)를 사용하여 Image를 Language Model에 추가 학습함.
- LLM의 cross-attention layer에서 matching되어 들어오는 image token과 text token 사이의 관련성을 학습함.

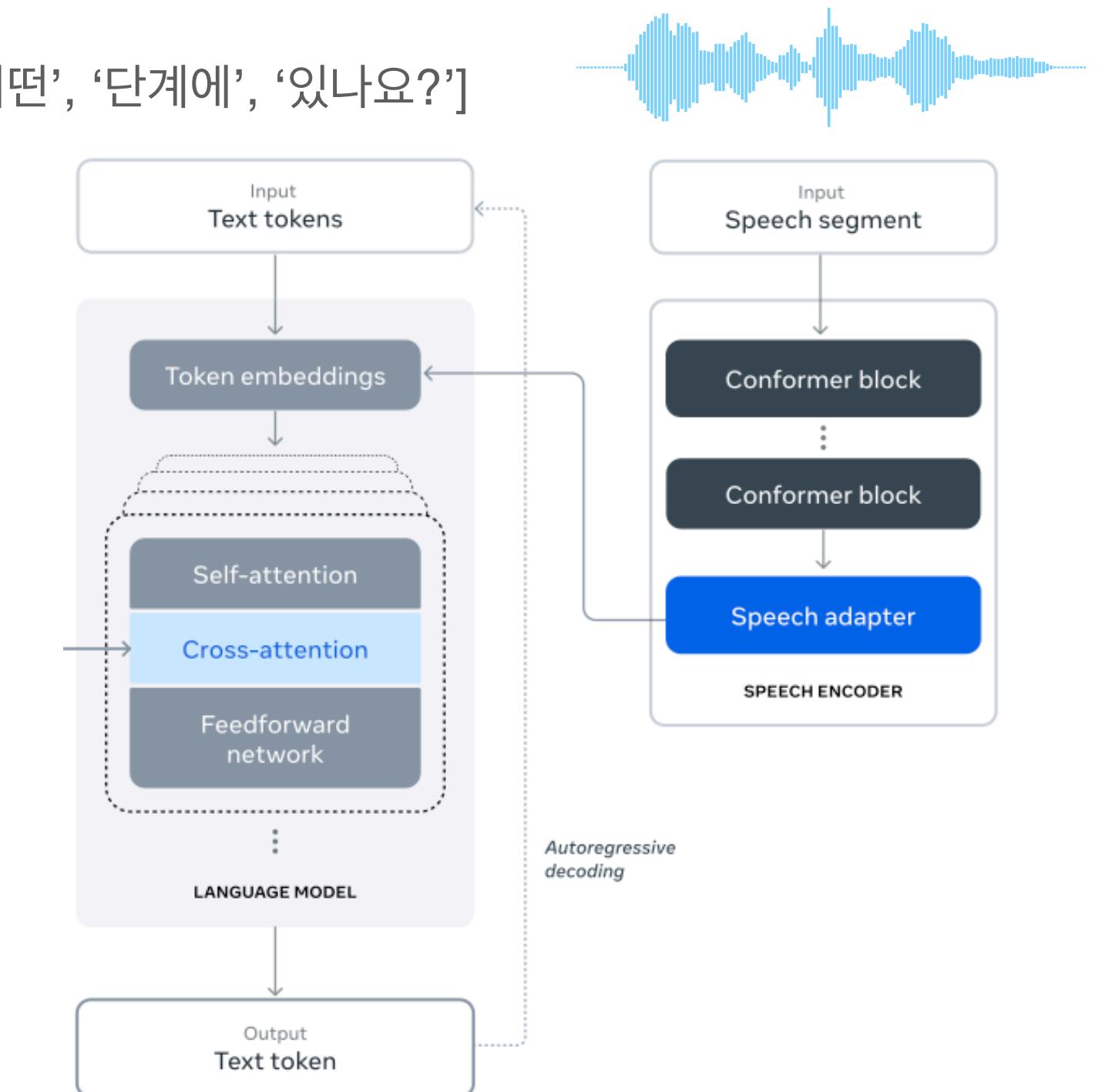


Llama의 시대

Llama3.2 - Audio

- multi-modal의 기본 학습 원리는 input layer를 확장하는 것에 있음.
- 기존에 널리 사용되는 Audio Encoder(Whisper-Large)를 사용하여 Audio Waveform을 Language Model에 추가 학습함.
- LLM의 cross-attention layer에서 matching되어 들어오는 Audio segment token과 text token 사이의 관련성을 학습함.

[‘이’, ‘나비는’, ‘어떤’, ‘단계에’, ‘있나요?’]



Llama의 시대

Llama3.3 (이젠 갑자기 나와도 놀랍지 않음...)

- 갑자기 24년 12월 6일에 Llama 3.3을 발표
- Multilingual의 키워드로 70B 모델 하나만 발표.
- 3.1의 70B의 성능을 향상 시킨 것으로 보이며, 실제 여러 benchmark 기준으로 어마어마한 성능 개선이 있는건 아님. (하지만 당연히 SOTA)
- 글로벌한 사용성을 기준으로 Multilingual 파트의 성능이 비약적으로 상승함.

Llama의 시대

Now available

Select your models •



Text

New



Llama 3.3: 70B

- State-of-the-art multilingual open source large language model
- Experience 405B performance and quality at a fraction of the cost

*Licensed under Llama 3.3 Community License Agreement



Lightweight



Llama 3.2: 1B & 3B

- Lightweight and most cost-efficient models you can run anywhere on mobile and on edge devices
- Llama Guard 3 1B is included
- Quantized models available

*Licensed under Llama 3.2 Community License Agreement



Text

Updated



Llama 3.1: 405B & 8B

- State-of-the-art multilingual open source large language model
- Llama Guard 3 8B and Prompt Guard are included

*Licensed under Llama 3.1 Community License Agreement



Multimodal



Llama 3.2: 11B & 90B

- Open multimodal models that are flexible and can reason on high resolution images and output text
- Llama Guard 3 11B Vision is included

*Licensed under Llama 3.2 Community License Agreement

그 외의 open-source LLMs

The screenshot shows the Microsoft Azure Phi open models landing page. At the top, it says "Phi open models" and describes them as "A family of powerful, small language models (SLMs) with groundbreaking performance at low cost and low latency". Below this, there's a callout for "Smaller, less compute-intensive models for generative AI solutions." and a button to "Experiment with Phi for free". A central feature is the "Phi-3 family" section, which includes a "vision" icon and three model variants: "Phi-3-medium", "Phi-3-small", and "Phi-3-mini". At the bottom, there's a navigation bar with links for Overview, Use cases, Security, Pricing, Related products, Resources, and FAQ. The "OVERVIEW" section contains the text "Redefining what's possible with SLMs".



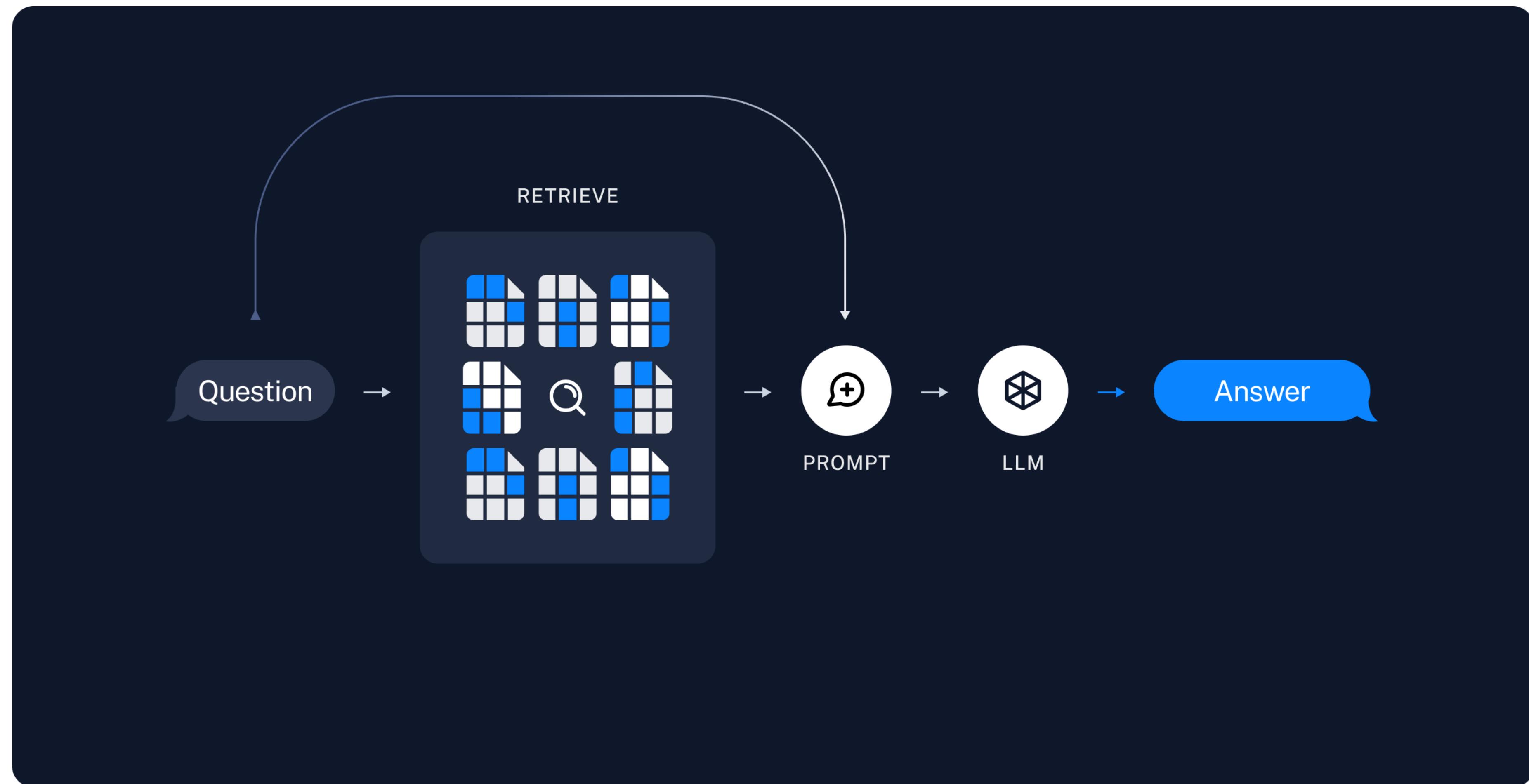
AI has the potential to address some of humanity's most pressing problems — but only if everyone has the tools to build with it. That's why earlier this year we introduced Gemma, a family of lightweight, state-of-the-art open models built from the same research and technology used to create the Gemini models. We've continued to grow the Gemma family with CodeGemma, RecurrentGemma and PaliGemma — each offering unique capabilities for different AI tasks and easily accessible through integrations with partners like Hugging Face, NVIDIA and Ollama.

Now we're officially releasing Gemma 2 to researchers and developers globally. Available in both 9 billion (9B) and 27 billion (27B) parameter sizes, Gemma 2 is higher-performing and more efficient at inference than the first generation, with significant safety advancements built in. In fact, at 27B, it offers competitive alternatives to models more than twice its size, delivering the kind of performance that was only possible with proprietary models as recently as December. And that's now achievable on a single NVIDIA H100 Tensor Core GPU or TPU host, significantly reducing deployment costs.

3. RAG에 대한 이해

3. RAG에 대한 이해

Retrieval Augmented Generation



3. RAG에 대한 이해

Retrieval System + LLM

- 사용자의 질문(Question)에 대한 답변을 미리 저장해둔 정보들(Knowledge)에서 찾고(Retrieve) 찾은 내용을 합쳐서(Augmented) LLM으로 답변을 생성(Generate)하는 방법.
- Question도 중요하고, Knowledge도 중요하고, Retrieve도 잘해야하고, Generation도 잘되어야 성능이 잘 나온다.
- LLM의 발전으로 Question, Generation은 어느정도 신경을 덜 써도 성능이 잘 나오는 편이지만, Knowledge 구축과 Retrieve를 잘하는 것은 큰 challenging이 있다.

3. RAG에 대한 이해

Retrieval System + LLM

- Knowledge는 사용자의 질문에 대해 정확한 답변을 제공하기 위한 용도로 구축된다.
- Knowledge를 구축할 때는 Database를 사용하며, 어떤 정보를 처리할 것이냐에 따라 RDBMS or Vector Store를 사용한다.
- RDBMS는 테이블 데이터를 다루는 용도로 사용되며, Text2SQL을 거쳐서 사용된다.
- Vector Store는 이미지/텍스트를 주로 다루는 용도로 사용되며, 이미지/텍스트를 embedding vector로 변환하여 저장하는 용도로 사용된다.

3. RAG에 대한 이해

Retrieval System + LLM

- 이렇게 Knowledge가 저장된 Database를 합쳐서 Knowledge Base(KB)라고 하며, KB를 잘 구축하여야 Retrieval의 성능이 올라감. 즉, 검색이 될 자료가 있어야 함.
- 사용자 질문에 도움이 되는 정보를 찾는 과정도 중요함. 질문과 유사성이 높은(=관련이 높은) 자료를 찾는 것이 Retrieval의 성능에 영향을 미침.
- 잘 찾는 것 = Ranking

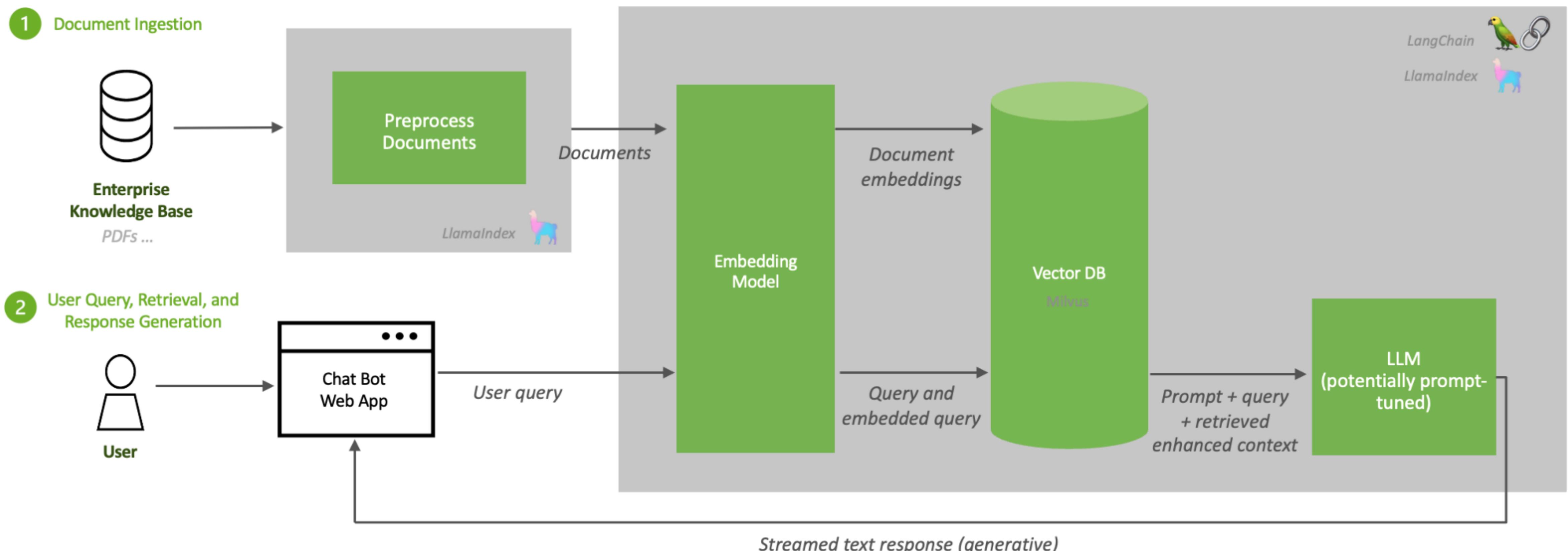
3. RAG에 대한 이해

Retrieval System + LLM

- LLM이 RAG에서 답변을 잘 생성하기 위해선 LLM의 pre-training 성능이 중요하다.
- 주로, 많은 token을 학습시킨 모델의 pre-training 성능이 높은 편이다. (SOTA)
- 하지만, LLM은 knowledge-cutoff 문제가 있어, 항상 최신 데이터를 학습할 수 없다. (대부분의 주요 LLM들이 23년 12월에 cutoff 되어 있음)
- 이러한 점을 극복할 수 있는게 Knowledge Base에 최신 데이터를 추가하는 방법이다.

3. RAG에 대한 이해

Langchain에서 RAG를 구축하는 순서



3. RAG에 대한 이해

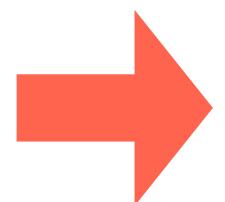
Langchain에서 RAG를 구축하는 순서

1. Document Loading
2. Text Splitting
3. Embedding
4. Vector Store (= Indexing)
5. Retrieving
6. Generating

3. RAG에 대한 이해

Langchain에서 RAG를 구축하는 순서

1. Document Loading
2. Text Splitting
3. Embedding
4. Vector Store (= Indexing)
5. Retrieving
6. Generating



- KB에서 데이터를 가져오는 작업.
- 파일의 종류에 따라 langchain_community.document_loaders에 존재하는 구현체가 나뉨.
e.g. PyPDFLoader, JsonOutputParser,
...

3. RAG에 대한 이해

Langchain에서 RAG를 구축하는 순서

1. Document Loading

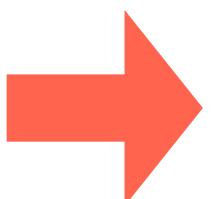
2. Text Splitting

3. Embedding

4. Vector Store (= Indexing)

5. Retrieving

6. Generating



- 가져온 텍스트를 chunk로 나누는 작업.
- 어떤 단위로 나누느냐에 따라 여러 구현체를 사용함.
e.g. RecursiveCharacterTextSplitter,
SemanticChunker, ...

3. RAG에 대한 이해

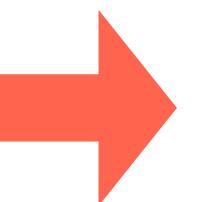
Langchain에서 RAG를 구축하는 순서

1. Document Loading
 2. Text Splitting
 3. Embedding
 4. Vector Store (= Indexing)
 5. Retrieving
 6. Generating
-
- 가져온 텍스트 데이터를 벡터 공간에 이식하는 작업.
 - 이 작업을 거쳐야 질문-지식 간의 유사성 계산이 가능함.
 - embedding은 주로 pre-trained embedding model을 사용함.
e.g. OpenAIEmbeddings,
HuggingFaceEmbeddings
...

3. RAG에 대한 이해

Langchain에서 RAG를 구축하는 순서

1. Document Loading
2. Text Splitting
3. Embedding
4. Vector Store (= Indexing)
5. Retrieving
6. Generating



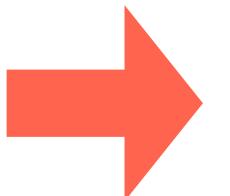
- 저장한 embedding vector들을 빠르게 계산하기 위해서 Index를 만드는 작업.
- Index를 만들어두면, 매번 검색할 때 전탐색을 수행할 필요가 없이 빠르게 필요한 영역만 탐색이 가능.
- 어떤 Indexing 방식을 쓰느냐에 따라 다양한 구현체가 존재함.

e.g. Chroma, FAISS, Milvus, ...

3. RAG에 대한 이해

Langchain에서 RAG를 구축하는 순서

1. Document Loading
2. Text Splitting
3. Embedding
4. Vector Store (= Indexing)
5. Retrieving
6. Generating

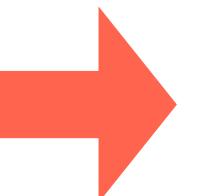


- 사용자의 입력과 연관있는 embedding vector를 찾는 과정.
- 요청한 K개의 유사성이 높은 vector를 찾고 해당 vector의 원본 텍스트를 반환.
- Embedding quality에 따라 성능이 큰 영향을 받음!

3. RAG에 대한 이해

Langchain에서 RAG를 구축하는 순서

1. Document Loading
2. Text Splitting
3. Embedding
4. Vector Store (= Indexing)
5. Retrieving
6. Generating



- 추가로 찾은 텍스트와 원본 질문을 함께 미리 디자인해둔 System Prompt의 형태로 LLM에 입력으로 제공.
 - Retrieval된 문서가 많을 수록, 문서의 길이가 길수록 input prompt가 길어지기 때문에 생성되는 답변에 큰 영향을 줌.
- + 사용하는 LLM의 token length도 고려해야 함!

3. RAG에 대한 이해

RAG Evaluation Metric

- RAG를 사용해서 LLM이 생성한 답변이 실제로 얼마나 좋은 답변인지를 평가하는 것은 매우 어려움.
(사람이 작성한 텍스트의 품질 평가도 쉽지 않음.)
- 주관적 해석이 많이 포함되고, 도메인에 따라 당연히 중요도가 달라지지만 대세를 결정하는 주요 지표들을 소개함.
- RAG를 잘 사용한 것과 LLM이 답변을 그냥 잘 만든 것은 조금 다름.
“얼마나 정보를 잘 사용했느냐”를 판단하는 것이 관건.

3. RAG에 대한 이해

RAG Evaluation Metric

- 기존에 정보 검색에서 많이 사용하는 Ranking metric들을 RAG의 특성에 맞게 변형하여 사용함.
- 기존에 NLP에서 텍스트 생성을 평가하던, BLEU, ROUGE-N 등의 점수는 잘 사용하지 않음.
(왜냐면, 생성된 token이 같은지 다른지만 체크하는게 전혀 의미가 없어서)
- RAG를 평가해야하기 때문에 ground truth의 영향을 많이 받을 수 밖에 없음.

3. RAG에 대한 이해

Evaluation Metric 1. Context Precision

- 수식의 의미는 검색한 문서 중에서 진짜로 관련된 문서가 차지하는 비율.

$$\text{Context Precision}@K = \frac{\sum_{k=1}^K (\text{Precision}@k \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

$$\text{Precision}@k = \frac{\text{true positives}@k}{(\text{true positives}@k + \text{false positives}@k)}$$

3. RAG에 대한 이해

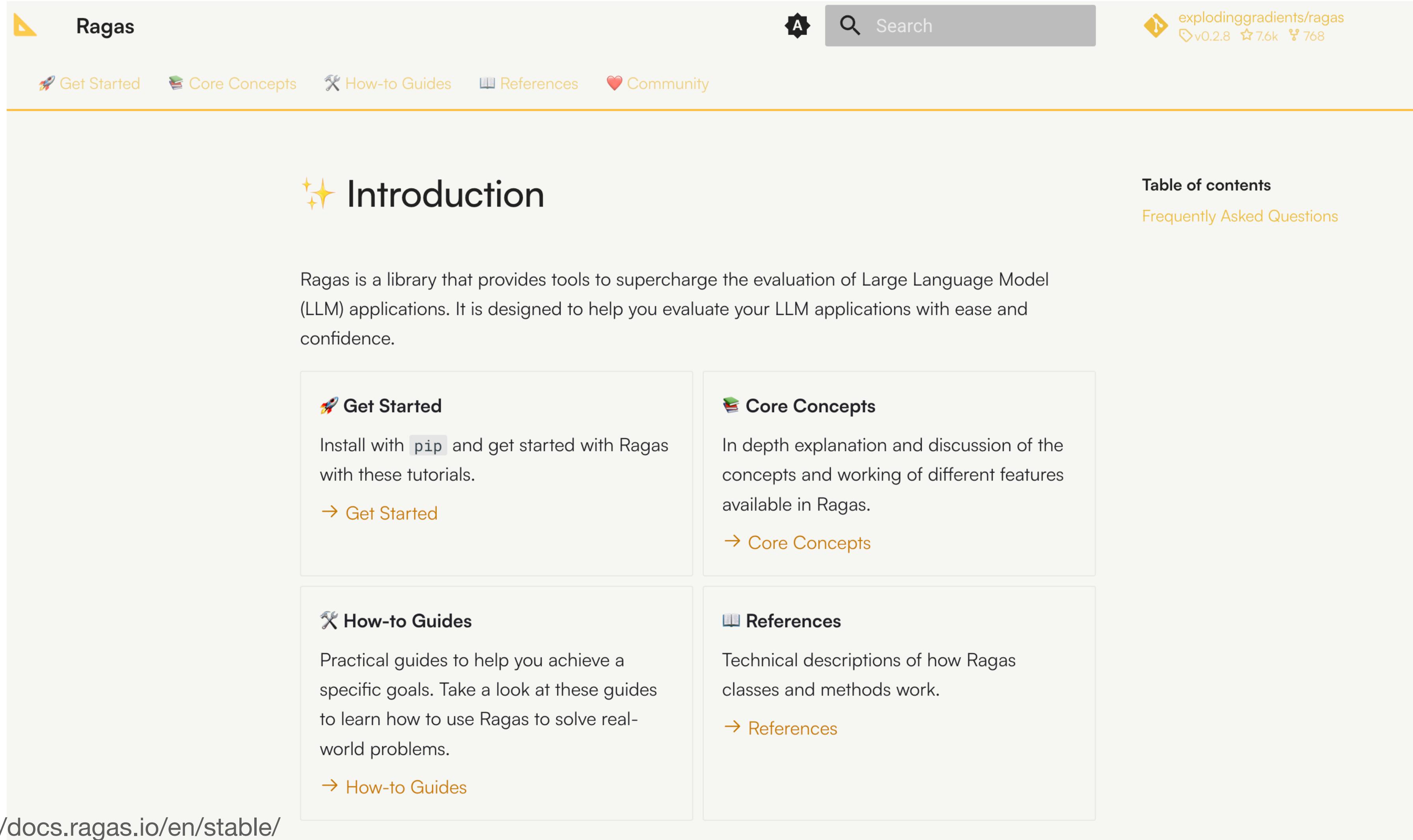
Evaluation Metric 2. Context Recall

- 수식의 의미는 실제로 관련된 문서 중에 얼마나 많이 Retrieval에 성공했는지.

$$\text{context recall} = \frac{|\text{GT claims that can be attributed to context}|}{|\text{Number of claims in GT}|}$$

3. RAG에 대한 이해

Evaluation Metric OS tool : Ragas



The screenshot shows the official documentation for Ragas, a library for evaluating Large Language Models. The top navigation bar includes links for "Get Started", "Core Concepts", "How-to Guides", "References", and "Community". A search bar and a GitHub repository link for "explodinggradients/ragas" are also present. The main content area features an "Introduction" section with a brief overview of what Ragas is and its purpose. Below the introduction are four main sections: "Get Started", "Core Concepts", "How-to Guides", and "References", each with a brief description and a "→ Get Started", "→ Core Concepts", "→ How-to Guides", or "→ References" link.

Ragas is a library that provides tools to supercharge the evaluation of Large Language Model (LLM) applications. It is designed to help you evaluate your LLM applications with ease and confidence.

Get Started
Install with `pip` and get started with Ragas with these tutorials.
[→ Get Started](#)

Core Concepts
In depth explanation and discussion of the concepts and working of different features available in Ragas.
[→ Core Concepts](#)

How-to Guides
Practical guides to help you achieve a specific goals. Take a look at these guides to learn how to use Ragas to solve real-world problems.
[→ How-to Guides](#)

References
Technical descriptions of how Ragas classes and methods work.
[→ References](#)

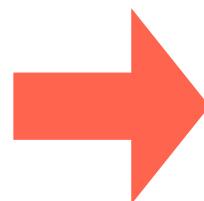
3. RAG에 대한 이해

Evaluation Metric tool : Ragas

- Ragas는 LLM의 성능 평가를 쉽게 구현해줄 수 있는 오픈소스 라이브러리이다.
- Langchain과 굉장히 쉽게 호환이 되며, 사용성이 편리하여 평가 지표 구현이 쉽다.

```
from ragas.llms import LangchainLLMWrapper
from ragas.embeddings import LangchainEmbeddingsWrapper
from langchain_openai import ChatOpenAI
from langchain_openai import OpenAIEMBEDDINGS
evaluator_llm = LangchainLLMWrapper(ChatOpenAI(model="gpt-4o"))
evaluator_embeddings = LangchainEmbeddingsWrapper(OpenAIEMBEDDINGS())

metrics = [
    LLMContextRecall(llm=evaluator_llm),
    FactualCorrectness(llm=evaluator_llm),
    Faithfulness(llm=evaluator_llm),
    SemanticSimilarity(embeddings=evaluator_embeddings)
]
results = evaluate(dataset=eval_dataset, metrics=metrics)
```



context_recall	factual_correctness	faithfulness
1.0	0.59	0.516129
1.0	0.11	0.137931
1.0	0.26	0.000000
1.0	0.25	0.600000
1.0	0.07	0.038462

Source : https://docs.ragas.io/en/stable/getstarted/rag_evaluation/

3. RAG에 대한 이해

Evaluation Metric tool : Ragas

- Ragas에는 RAG를 평가할 때 쓰이는 지표들뿐만 아니라, AI Agent, Natural Language Compression, SQL 등을 평가하는 지표들도 제공한다.
- 당연히 평가요소에서 Ground Truth가 핵심이다.
- 평가를 잘하기 위해서는 동의가 된 Ground Truth들이 필요하다.
- Ground Truth는 사람이 만드는게 정석이지만, 간접적인 여러가지 추가 방법론들도 존재한다.

e.g. LLM-as-a-Judge

4. LLM App 구축시 고려할 사항들에 대한 이해

1) LLM과 SLM 중 어떤 모델이 좋을까?

당연히 LLM이 더 좋을까?

- 성능은 당연히 LLM이 우수하지만, 두 가지의 큰 측면에서 한계점이 있다.
- 첫번째는 보안
- 두번째는 비용
- 세번째는 ~~우리끼가 갖고 싶음~~

1) LLM과 SLM 중 어떤 모델이 좋을까?

당연히 LLM이 더 좋을까?

- 보안의 측면에서 회사들은 자사의 데이터를 다른 회사에게 넘겨주고 싶지 않아함
(물론 아예 안되는 회사/기관들도 존재)
- LLM은 API라는 형태로 네트워크를 통해서 서버에서 동작하기 때문에 무조건 요청과 함께
데이터가 넘어가야 함
- Prompt Engineering 레벨에서 해결할 수 있지만 사용성에서 한계가 있어 어려움

1) LLM과 SLM 중 어떤 모델이 좋을까?

당연히 LLM이 더 좋을까?

- 비용에서는 여전히 갑론을박이 많지만, 대세에는 크게 2가지 의견이 존재함

1) LLM이 훨씬 비용이 저렴하다

- SLM 자체 개발에 드는 비용보다 API call당 비용을 지불하는게 더 합리적임
- 생각보다 LLM을 이용하여 서비스하는게 비용이 많이 들지 않음
- LLM API 비용이 점점 저렴해지고 있음

1) LLM과 SLM 중 어떤 모델이 좋을까?

당연히 LLM이 더 좋을까?

- 비용에서는 여전히 갑론을박이 많지만, 대세에는 크게 2가지 의견이 존재함

2) SLM이 훨씬 비용이 저렴하다

- 네이버, 카카오처럼 엄청나게 많은 사용량이 있는 회사에서 LLM API당 비용 지불은 너무 많은 지출이 필요함
- 생각보다 SLM을 학습시켜서 서비스를 하는게 크게 어렵지 않음
- 초기 투자 비용(인력 + 인프라)이 많이 들긴하지만, 서비스를 할수록 차이가 큼

1) LLM과 SLM 중 어떤 모델이 좋을까?

당연히 LLM이 더 좋을까?

- 대세는 Hybrid!
- 앞선 특징을 모두 보면 알 수있듯, 작은 서비스에는 LLM을
큰 서비스에는 SLM을 쓰는 것이
비용 측면에서 효율적이라고 볼 수 있다.
- 그럼 작은 서비스와.. 큰 서비스의 기준은..?
- LLM에서 보안 이슈는 어떻게 해결하는가?

2) SFT vs RAG

LM의 답변이 마음에 들지 않을때...?

- SFT(Supervised Fine Tuning)은 데이터를 모델에 학습시켜서, 내가 원하는 답변을 잘 만들 수 있게 하는 방식
- RAG(Retrieve Augmented Generation)은 데이터를 잘 정리해놓은 지식 창고 (Knowledge Base, 주로 VectorDB)에서 답변과 연관된 지식을 찾아서 답변을 생성하는 방식
- 두 가지를 모두 사용하면 제일 효과적이겠지만, 장단점이 존재함

2) SFT vs RAG

LM의 답변이 마음에 들지 않을때...?

- SFT를 하려면 우선 학습 데이터를 구축하는 것부터 시작해야 함.

- 명확한 task에 대한 정의와 그에 대한 데이터의 양이 충분해야 함.

- 이런 작업은 주로 Instruction Tuning이라는 이름으로 수행됨.

```
{} sft.json > ...
1  [
2  {
3      "instruction": "문장의 감정을 분류하세요. 가능한 분류는 '긍정', '부정', '중립'입니다.",
4      "input": "오늘 날씨가 정말 좋아서 기분이 좋아.",
5      "output": "긍정"
6  },
7  {
8      "instruction": "문장의 감정을 분류하세요. 가능한 분류는 '긍정', '부정', '중립'입니다.",
9      "input": "회의는 지루하고 무의미했어.",
10     "output": "부정"
11 },
12 {
13     "instruction": "문장의 감정을 분류하세요. 가능한 분류는 '긍정', '부정', '중립'입니다.",
14     "input": "그는 그냥 괜찮은 사람이야.",
15     "output": "중립"
16 },
17 {
18     "instruction": "문장의 감정을 분류하세요. 가능한 분류는 '긍정', '부정', '중립'입니다.",
19     "input": "새 프로젝트가 시작되어 너무 기대돼!",
20     "output": "긍정"
21 },
22 {
23     "instruction": "문장의 감정을 분류하세요. 가능한 분류는 '긍정', '부정', '중립'입니다.",
24     "input": "다시는 이런 실수를 하고 싶지 않다.",
25     "output": "부정"
26 }
27 ]
```

2) SFT vs RAG

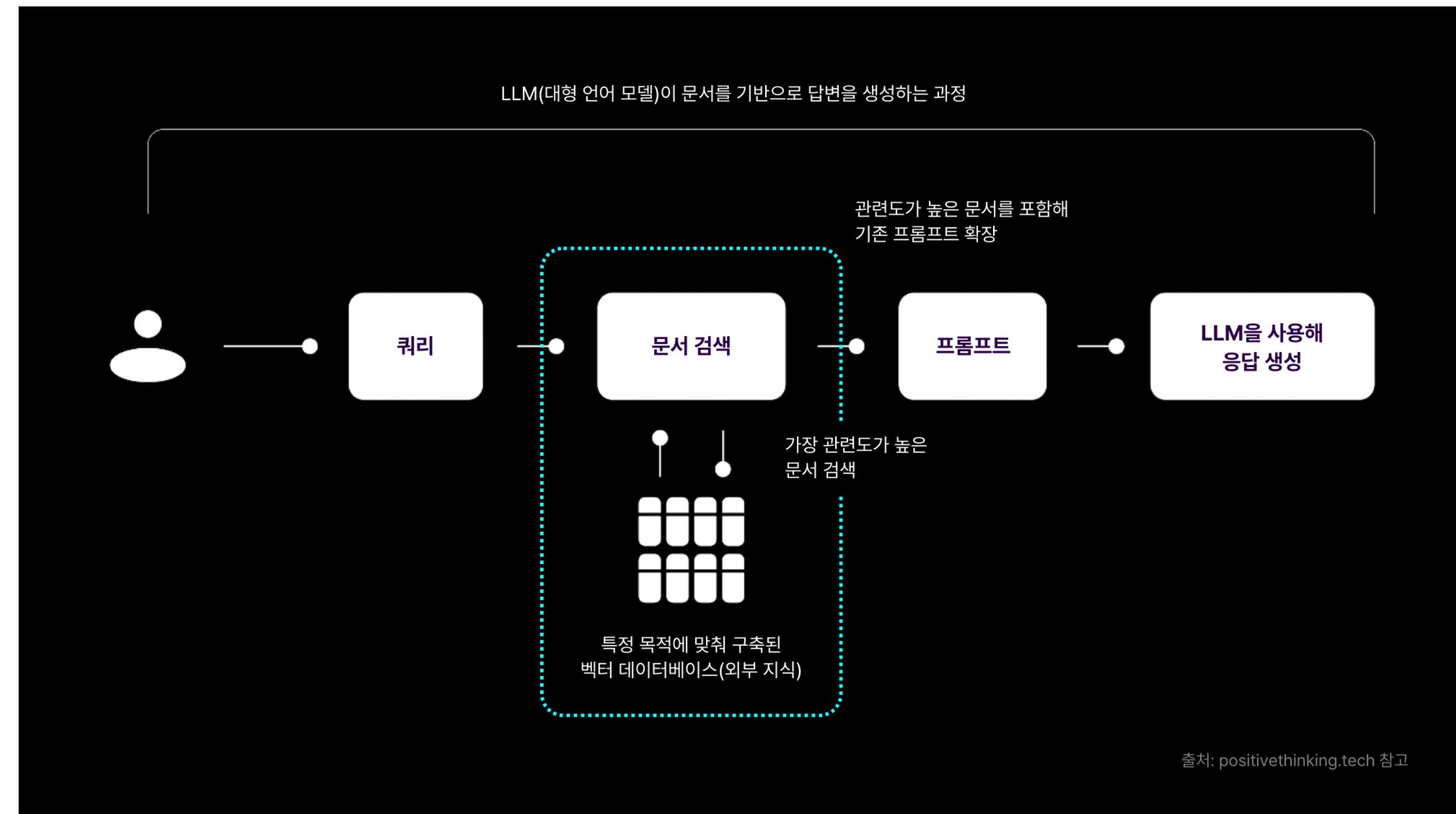
LM의 답변이 마음에 들지 않을때...?

- 하지만, LM의 크기가 클수록 SFT는 엄청나게 많은 비용이 발생함
(GPU가 곧 시간이자 돈..)
- SFT를 한다고 해서 무조건 성능이 증가하는게 아니라서 overfitting 이슈도 있고, 답변 생성의 성능 측정 자체가 굉장히 주관적임. (domain-dependent)
- 그래서 보통은 추가적으로 RLHF나 DPO 같은 강화학습 방법도 함께 사용하여 아예 성능을 끌어올리기 위해 최선을 다함.

2) SFT vs RAG

LM의 답변이 마음에 들지 않을때...?

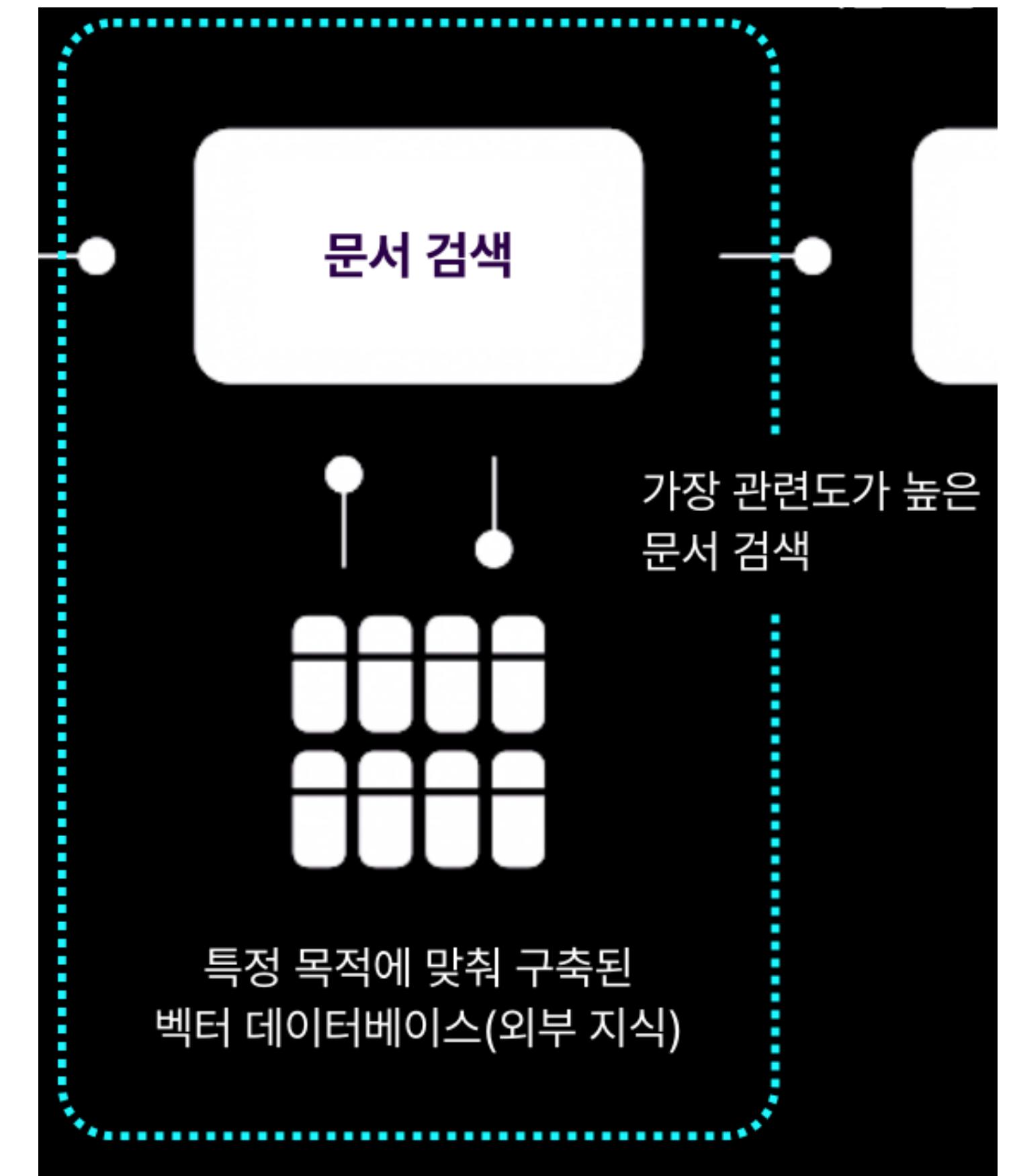
- RAG는 LLM의 입력 프롬프트에 사용할 수 있는 적절한 지식을 자동으로 찾아서 생성 답변의 퀄리티를 높이는 방식.
- 답변을 원하는 좋은 지식을 잘 정리해서 저장해두어야 함. (Knowledge Base 구축)
- 학습 대비 시간/비용 효율적이어서 요즘 많이 사용되고 있음



2) SFT vs RAG

LM의 답변이 마음에 들지 않을때...?

- RAG는 그냥 KB에 답변을 땠려넣는다고 해결이 되는게 아니라, 관련도가 높은 문서를 "잘" 찾아야 해결됨.
- 잘 찾는게 굉장히 어려움.
- SFT처럼 역시나 RAG를 위한 데이터셋 구축이 어렵고, 도메인마다 특성이 모두 달라서 도메인 전문가가 꼭 필요함.
- 대신에 실시간으로 데이터를 추가할 수 있고, 학습보단 필요한 데이터셋의 양이 적은 편



2) SFT vs RAG

당연히 둘 다 할 수 있는게 제일 좋음

- 대세는 Hybrid!
- 앞선 특징을 모두 보면 알 수있듯, 시작에는 RAG를
데이터가 쌓여감에 따라 SFT를 쓰는 것이
비용 측면에서 효율적이라고 볼 수 있다.
- 그럼 항상 시작할 땐 RAG부터...?
- RAG나 SFT를 하면 무조건 답변의 성능이 좋아지나?

3) Langchain vs Llamaindex

어떤 framework가 LLM service 개발에 적합할까?



VS



3) Langchain vs Llamaindex

어떤 framework가 LLM service 개발에 적합할까?

- 대세는 Langchain!
- 현재 가장 많은 app들이 Langchain 기반으로 작성되고 있으며, 사용자들이 쉽게 LLM을 구축할 수 있는 많은 기능들을 제공함.
- 쉽게 구현할 수 있으나, 다양한 데이터베이스를 고려하고 그에 맞는 쿼리나 메모리 관리를 효율적으로 하기엔 기능이 부족함.

3) Langchain vs Llamaindex

어떤 framework가 LLM service 개발에 적합할까?

- Llamaindex는 이러한 점들을 보완할 수 있는 다양한 데이터 소스와 고급 인덱싱 기법들을 지원함.
- 대신에 Langchain 대비 개발자 커뮤니티가 적으며, 아무래도 처음에 공부할 때 Langchain 대비 learning curve가 큰 편이라서 접근이 쉽지 않음.
- 대용량 트래픽을 고려해야하는 서비스의 경우 Llamaindex를 기반으로 구현하는 것이 장기적으로 봤을 때 서비스 구축에 도움이 됨.

4) Ollama

개인용 컴퓨터에서 LLM을 쉽게 실행할 수 있는 플랫폼

- 오픈소스이면서 LLM/SLM에 접근하기 쉽게 제작된 프로그램(이자 플랫폼)
- 다운로드 받아서 바로 모델들을 직접 돌려볼 수 있음.
- Rust로 구현되어 있어 굉장히 빠른 속도를 자랑함.
- 사용자들이 쉽게 쓸 수 있게 깔끔한 최소한의 기능들을 제공하며, 최신 모델들이 빠르게 업데이트 됨.

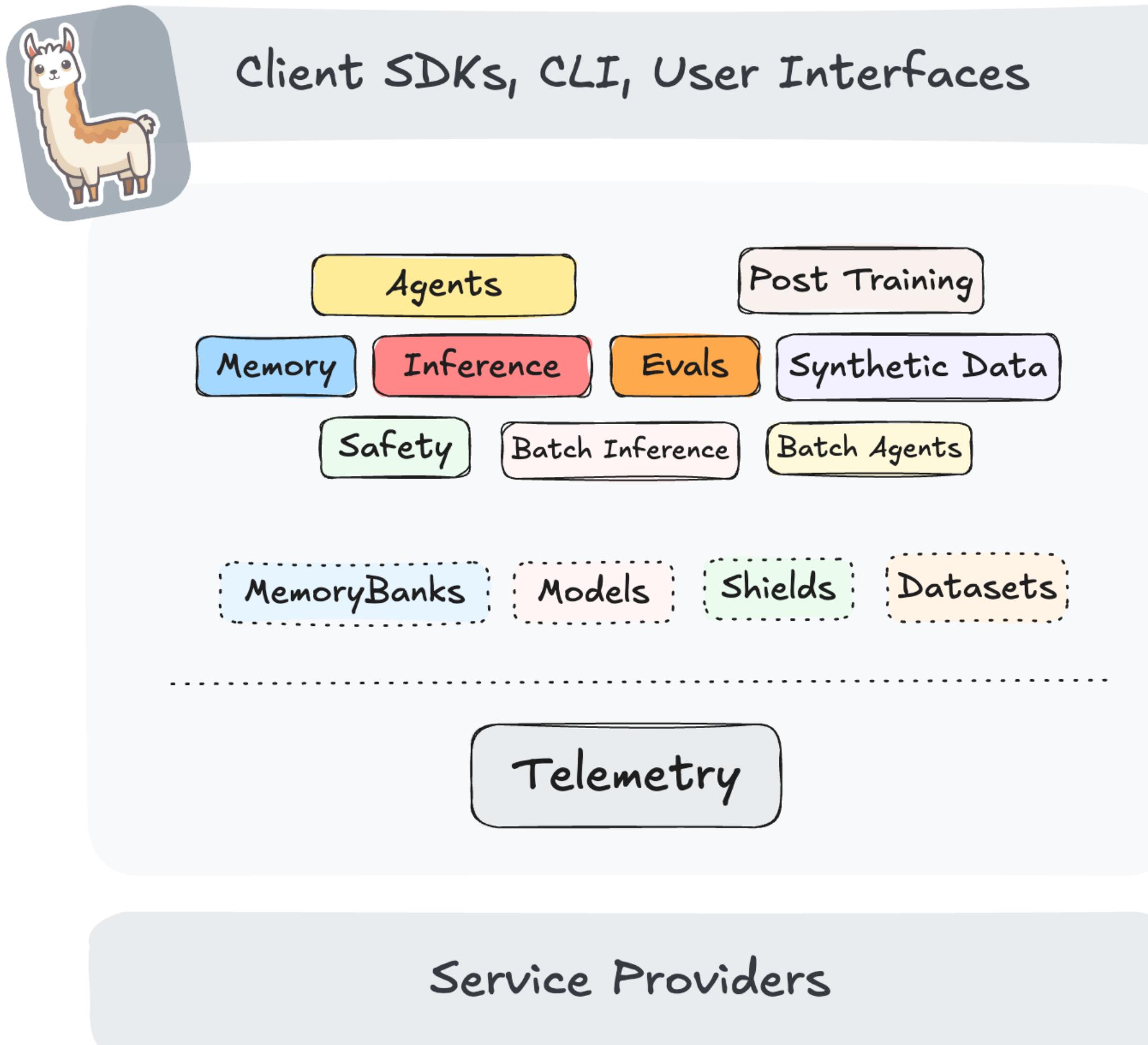
5) Llama Stack

Llama 모델을 이용하여 서비스를 빠르게 구현해볼 수 있는 플랫폼.

- Llama 3.2가 공개될 당시에 같이 공개됨.
- Llama가 본격적으로 하나의 상품처럼 인식되며 여러 클라우드 업체들과 협업하여 Llama를 이용해서 개발을 쉽게 할 수 있게 출시됨.
- 아직까지는 사용자가 거의 없어, 사용해볼 경우조차 많지 않음.
- 하지만, Llama 모델이 가지는 파워가 큰만큼 연동만 잘된다면 노코드 구현 서비스처럼 확장될 가능성이 있음.

5) Llama Stack

Llama 모델을 이용하여 서비스를 빠르게 구현해볼 수 있는 플랫폼.



End of Slides