Large Language Models: A Survey

(2024)

Shervin Minaee1, Tomas Mikolov2, Narjes Nikzad3, Meysam Chenaghlu4, Richard Socher5, Xavier Amatriain6, Jianfeng Gao7

1 Applied Scientist, Amazon Inc
2 Senior Researcher, CIIRC CTU
3 Cologne University of Applied Sciences
4 Staff Machine Learning Scientist, Ultimate.ai
5 CEO, You.com
6 VP of Product, AI and Compute Enablement, Google Inc
7 VP of Deep Learning Group, Microsoft Research

Paper : https://arxiv.org/abs/2402.06196

Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering & Artificial Intelligence

Introduction

- **LLMs** are **large-scale**, **pre-trained**, **statistical language models** based on neural networks. .

- **4 Waves for language models**
  - Statistical language models (SLM) : *Markov chain* models known as the n-gram models
  - Neural language models (NLM) : Early model
  - Pre-trained language models (PLM) : *pre-training* and *fine-tuning* paradigm
  - LLMs : Refer to transformer-based neural language models that contain 10B~100B of parameters, which are pre-trained on massive text data, such as PaLM, LLaMA, and GPT-4

- LLMs Emergent abilities

1) ***in-context learning*** ; LLMs learn a new task from a small set of examples presented in the prompt at inference time,
2) ***instruction following*** ; LLMs, after ***instruction tuning***, can follow the instructions for new types of tasks without using explicit examples
3) ***Multi-step reasoning*** ; LLMs can solve a complex task by breaking down that task into intermediate reasoning steps as demonstrated in the ***chain-of-thought*** prompt.

- LLMs can also be *augmented* by using external knowledge and tools and continually improve itself using feedback data collected through interactions (e.g. via reinforcement learning with human feedback (**RLHF**)).

- Through advanced usage and augmentation techniques, **LLMs can be deployed** as so-called ***AI agents***: artificial entities that *sense their* ***environment***, *make* ***decisions***, and *take* ***actions***.
- Possible to build **general-purpose AI agents** based on LLMs.

❖ LLMs are trained to produce responses in static settings, AI agents need to take actions to interact with dynamic environment.

Introduction

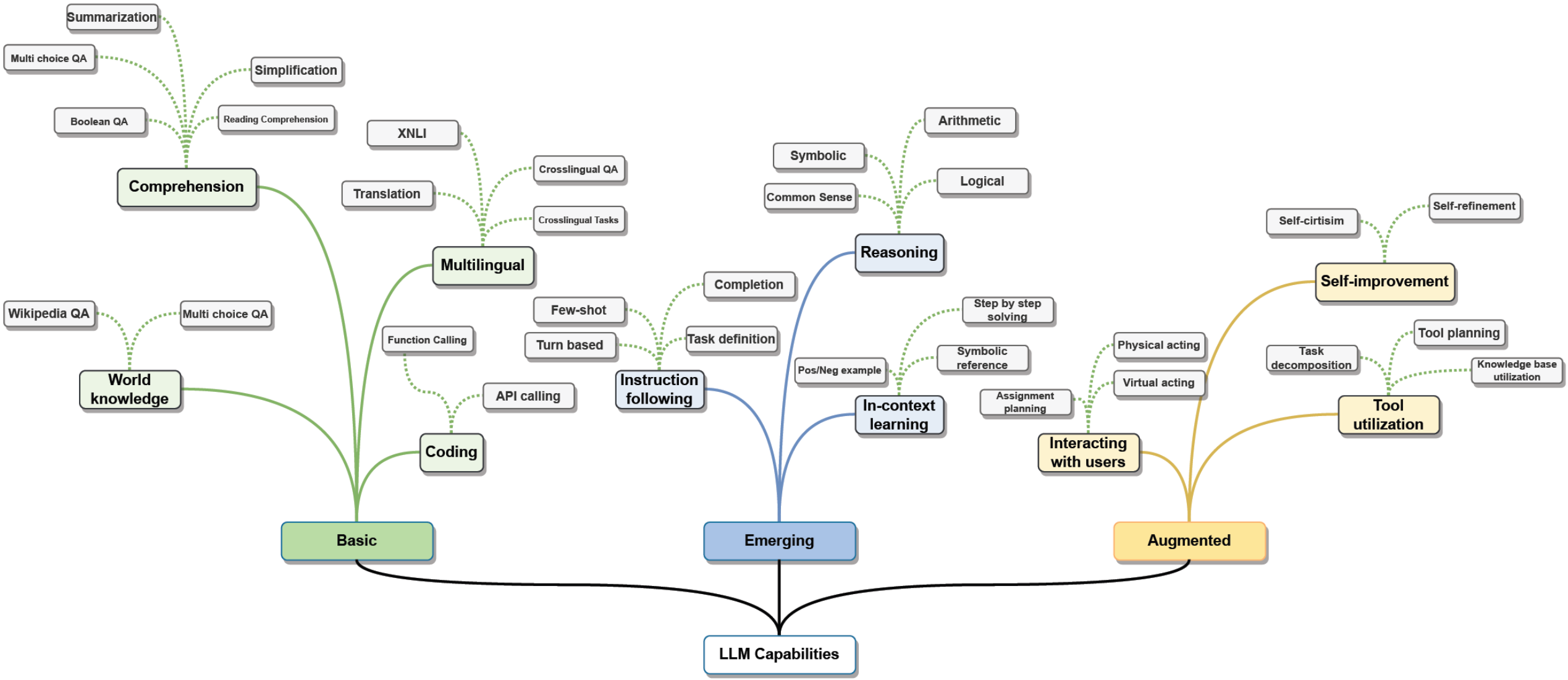


Fig. 1. LLMs Capabilities.

Introduction

II Large Language Models

- A Early Pre-trained Language Models
- B Large Language Model Families
- C Other Representative LLMs

I HOW LLMS ARE USED AND AUGMENTED

- A LLM limitations
- B Using LLMs: Prompt Design and Engineering
- C Augmenting LLMs through external knowledge - RAG
- D Using External Tools
- E LLM Agents

III HOW LLMS ARE BUILT

- A Dominant LLM Architectures
- B Data Cleaning
- C Tokenizations
- D Positional Encoding
- E Model Pre-training
- F Fine-tuning and Instruction Tuning
- G Alignment
- H Decoding Strategies
- I Cost-Effective Training/Inference, Adaptation & Compression

V POPULAR DATASETS FOR LLMS

- A Datasets for Basic Tasks: language modeling/understanding/generation
- B Datasets for Emergent: ICL, reasoning, instruction following
- C Datasets for Augmented: using external knowledge/tools

VI PROMINENT LLMS' PERFORMANCE ON BENCHMARKS

- A Popular Metrics for Evaluating LLMs
- B LLMs' Performance on Different Tasks

VII CHALLENGES AND FUTURE DIRECTIONS

- A Smaller and more efficient Language Models
- B New Post-attention Architectural Paradigms
- C Multi-modal Models
- D Improved LLM Usage and Augmentation techniques
- D Security and Ethical/Responsible AI

TABLE I: High-level Overview of Popular Language Models

Type	Model Name	#Parameters	Release	Base Models	Open Source	#Tokens	Training dataset
Encoder-Only	BERT	110M, 340M	2018	-	✓	137B	BooksCorpus, English Wikipedia
	RoBERTa	355M	2019	-	✓	2.2T	BooksCorpus, English Wikipedia, CC-NEWS, STORIES (a subset of Common Crawl), Reddit
	ALBERT	12M, 18M, 60M, 235M	2019	-	✓	137B	BooksCorpus, English Wikipedia
	DeBERTa	-	2020	-	✓	-	BooksCorpus, English Wikipedia, STORIES, Reddit content
	XLNet	110M, 340M	2019	-	✓	32.89B	BooksCorpus, English Wikipedia, Giga5, Common Crawl, ClueWeb 2012-B
Decoder-only	GPT-1	120M	2018	-	✓	1.3B	BooksCorpus
	GPT-2	1.5B	2019	-	✓	10B	Reddit outbound
Encoder-Decoder	T5 (Base)	223M	2019	-	✓	156B	Common Crawl
	MT5 (Base)	300M	2020	-	✓	-	New Common Crawl-based dataset in 101 languages (m Common Crawl)
	BART (Base)	139M	2019	-	✓	-	Corrupting text
GPT Family OpenAI	GPT-3	125M, 350M, 760M, 1.3B, 2.7B, 6.7B, 13B, 175B	2020		×	300B	Common Crawl (filtered), WebText2, Books1, Books2, Wikipedia
	CODEX	12B	2021	GPT	✓	-	Public GitHub software repositories
	WebGPT	760M, 13B, 175B	2021	GPT-3	×	-	ELI5
	GPT-4	1.76T	2023	-	×	13T	-
LLaMA Family Meta	LLaMA1	7B, 13B, 33B, 65B	2023	-	✓	1T, 1.4T	Online sources
	LLaMA2	7B, 13B, 34B, 70B	2023	-	✓	2T	Online sources
	Alpaca	7B	2023	LLaMA1	✓	-	GPT-3.5
	Vicuna-13B	13B	2023	LLaMA1	✓	-	GPT-3.5
	Koala	13B	2023	LLaMA	✓	-	Dialogue data
	Mistral-7B	7.3B	2023		✓	-	-
	Code Llama	34	2023	LLaMA2	✓	500B	Publicly available code
	LongLLaMA	3B, 7B	2023	OpenLLaMA	✓	1T	-
	LLaMA-Pro-8B	8.3B	2024	LLaMA2-7B	✓	80B	Code and math corpora
	TinyLlama-1.1B	1.1B	2024	LLaMA1.1B	✓	3T	SlimPajama, Starcoderdata

TABLE I: High-level Overview of Popular Language Models

Type	Model Name	#Parameters	Release	Base Models	Open Source	#Tokens	Training dataset
PaLM Family Google	PaLM	8B, 62B, 540B	2022	-	×	780B	Web documents, books, Wikipedia, conversations, GitHub code
	U-PaLM	8B, 62B, 540B	2022	-	×	1.3B	Web documents, books, Wikipedia, conversations, GitHub code
	PaLM-2	340B	2023	-	✓	3.6T	Web documents, books, code, mathematics, conversational data
	Med-PaLM	540B	2022	PaLM	×	780B	HealthSearchQA, MedicationQA, LiveQA
	Med-PaLM 2	-	2023	PaLM 2	×	-	MedQA, MedMCQA, HealthSearchQA, LiveQA, MedicationQA
Other Popular LLMs	FLAN	137B	2021	LaMDA-PT	✓	-	Web documents, code, dialog data, Wikipedia
	Gopher	280B	2021	-	×	300B	MassiveText
	ERNIE 4.0	10B	2023	-	×	4TB	Chinese text
	Retro	7.5B	2021	-	×	600B	MassiveText
	LaMDA	137B	2022	-	×	168B	public dialog data and web documents
	ChinChilla	70B	2022	-	×	1.4T	MassiveText
	Galactia-120B	120B	2022	-		450B	
	CodeGen	16.1B	2022	-	✓	-	THE PILE, BIGQUERY, BIGPYTHON
	BLOOM	176B	2022	-	✓	366B	ROOTS
	Zephyr	7.24B	2023	Mistral-7B	✓	800B	Synthetic data
	Grok-0	33B	2023	-	×	-	Online source
	ORCA-2	13B	2023	LLaMA2	-	2001B	-
	StartCoder	15.5B	2023	-	✓	35B	GitHub
	MPT	7B	2023	-	✓	1T	RedPajama, m Common Crawl, S2ORC, Common Crawl
	Mixtral-8x7B	46.7B	2023	-	✓	-	Instruction dataset
	Falcon 180B	180B	2023	-	✓	3.5T	RefinedWeb
	Gemini	1.8B, 3.25B	2023		✓	-	Web documents, books, and code, image data, audio data, video data
	DeepSeek-Coder	1.3B, 6.7B, 33B	2024	-	✓	2T	GitHub's Markdown and StackExchange
	DocLLM	1B,7B	2024	-	×	2T	IIT-CDIP Test Collection 1.0, DocBank

LARGE LANGUAGE MODELS

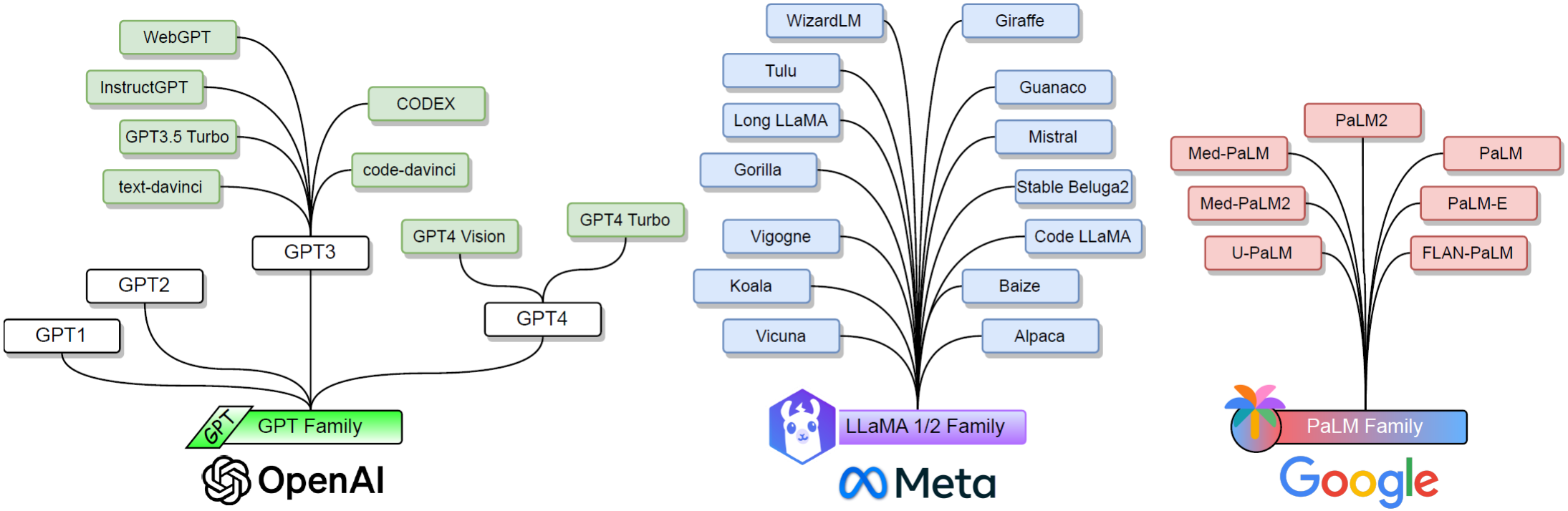


Fig. 8: Popular LLM Families

B. Large Language Model Families

1) GPT Family OpenAI

- Generative Pre-trained Transformers (GPT) are a family of *decoder-only* Transformer-based language models, developed by OpenAI.
- GPT-1, GPT-2, GPT-3, InstrucGPT, ChatGPT, GPT-4, CODEX, and WebGPT

➢ GPT3

- Pre-trained autoregressive language model with 175B parameters.
- **in-context learning ability** : Can be applied to any downstream tasks without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified via text interaction with the model.
- Achieved strong performance on many NLP tasks, including *translation*, *question-answering*, and the *cloze t*asks, as well as several ones that require on-the-fly *reasoning* or *domain adaptation*, such as unscrambling words, using a novel word in a sentence, 3-digit arithmetic.

➢ ***Codex*** (1세대, 2021~2023)

- A general-purpose programming model that can parse natural language and generate code in response.
- Descendant of GPT-3, fine-tuned for programming applications on code corpora collected from GitHub

✓ GPT-3 후속 코드 특화 LLM, GitHub Copilot·OpenAI API에 탑재 → 2023-03 API 폐지

❖ **Codex CLI** (2세대, 2025 ~)

- GPT-4 (o3/o4-mini 등)에 연결된 터미널 기반 코딩 에이전트
- 기본값 o4-mini → --model gpt-4.1 등 아무 GPT-4 계열과 연동 가능.
- TechCrunch자연어·스크린샷·스케치 입력을 받아 파일 생성·이동·빌드·테스트까지 자동화.
- Git integration, 커맨드 로그, 플러그인식 워크플로 구성.

B. Large Language Model Families

1) GPT Family OpenAI

➢ **InstructGPT**

- To enable LLMs to **follow expected human instructions**, InstructGPT aligns language models with user intent on a wide range of tasks by fine-tuning with **RLHF**.
- Improvements in *truthfulness* and *reductions in toxic output generation* while having minimal performance regressions on public NLP datasets.

❖ Model sizes & public API names

Parameters	Internal name	Public API alias (chronology)
1.3 B	InstructGPT-1.3B	text-davinci-001 (Jan 2022)
6.9 B	InstructGPT-6B	text-davinci-002 (Jun 2022)
175 B	InstructGPT-175B	text-davinci-003 (Dec 2022) → **merged into gpt-3.5-turbo-instruct (2023)**

► The original "**Davinci**" endpoints remain available but are being gradually deprecated in favour of **GPT-4o** and newer alignment methods.

[☞ 출처 Chatgpt o3; OpenAI InstructGPT]

- RLHF 3step pipeline

Step	Description	세부 수치
① Supervised Fine-Tuning (SFT)	Human Labeler가 작성한 시범 답변 13 k를 사용해 GPT-3를 미세조정	1차 데이터 ≈ 1 M Token
② Reward Model (RM)	동일 프롬프트에 대한 모델 출력쌍 33 k를 사람에게 선호도 순위로 라벨링 → 순위 학습	Pairwise Bradley-Terry loss
③ Reinforcement Learning (PPO)	RM을 보상 함수로 사용, PPO로 정책 최적화	Maximum KL Penalty Regularization

구분	InstructGPT-1.3B	InstructGPT-6.9B	GPT-3 175B(원본)
인간 선호도(%)	**75**	85	23
유해 발언 감소	-23 %	-30 %	기준

- 환각·편향 여전히 존재 → Constitutional AI, RLAIF(자체 피드백) 등 대안 연구
- 사람 의존 비용: 고품질 라벨 단가 상승 → 합성 데이터·강화 학습 혼합 시도.
- 지침 우회(jailbreak) 문제 → 자동 디텍터·방어적 디코딩 기법 연구 중.

B. Large Language Model Families

❖ RLHF

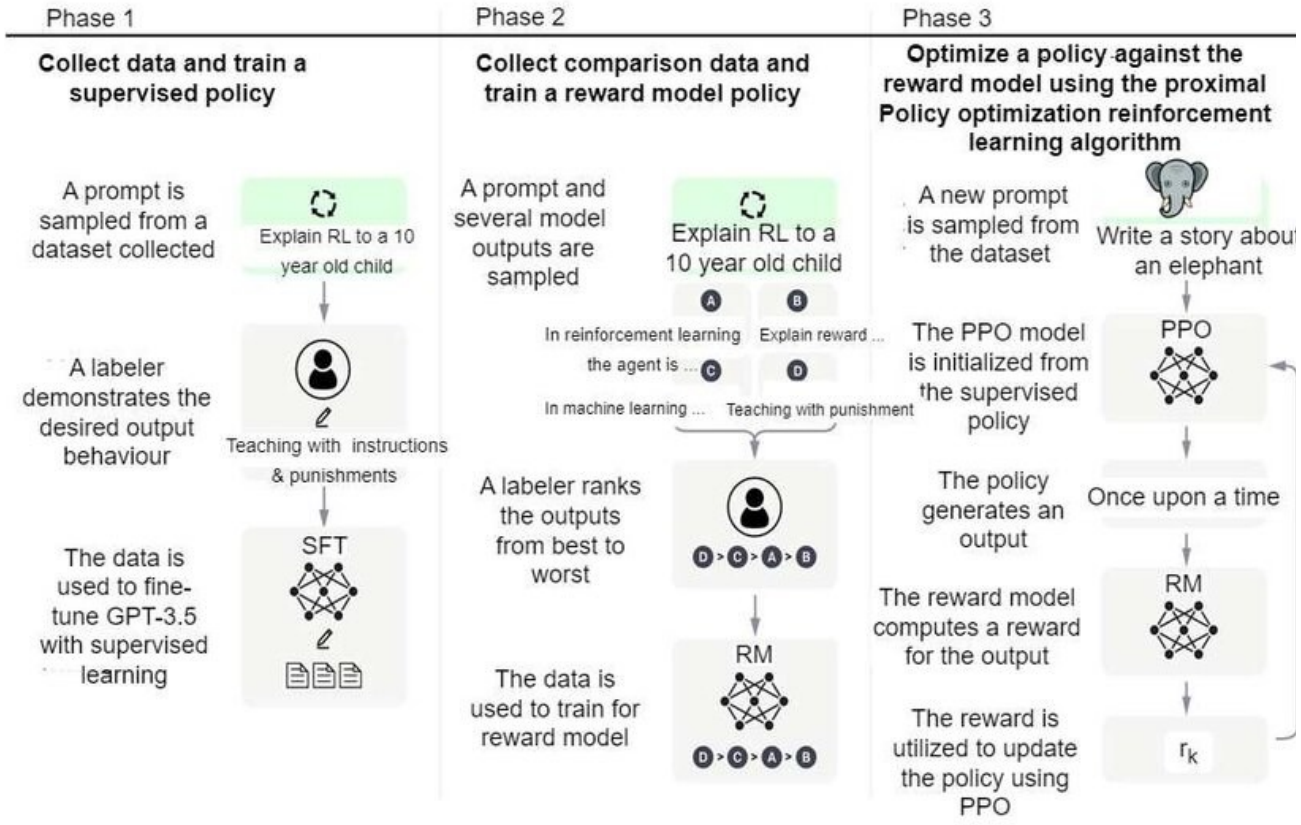


Fig. 10: The high-level overview of RLHF.

- **SFT Phase (Supervised Fine-Tuning):** This stage start with a pre-trained LM, and then fine-tuned with supervised learning (typically maximum likelihood loss) on a high quality dataset for the downstream task of interest. Outcome of this stage is denoted as π^{SFT}.
- **Reward Learning Phase.** In the second phase the SFT model is prompted with prompts $\boldsymbol{x}$ to produce pairs of answers $\boldsymbol{y}_1, \boldsymbol{y}_2 \sim \pi^{\text{SFT}}(\boldsymbol{y}|\boldsymbol{x})$. The responses pairs are then presented to human labelers who express preferences for one answer, denoted as $\boldsymbol{y}_w \succ \boldsymbol{y}_l|\boldsymbol{x}$, where $\boldsymbol{y}_w$ is the one favored by the labeler and $\boldsymbol{y}_l$ is the one less favored. Under these preferences is the inaccessible latent reward model $r^*(\boldsymbol{y}, \boldsymbol{x})$. According to the Bradley-Terry (Bradley & Terry, 1952) model, the human preference distribution p^* can be expressed as:

$$p^*(\boldsymbol{y}_1 \succ \boldsymbol{y}_2|\boldsymbol{x}) = \frac{\exp(r^*(\boldsymbol{y}_1|\boldsymbol{x}))}{\exp(r^*(\boldsymbol{y}_1|\boldsymbol{x})) + \exp(r^*(\boldsymbol{y}_2|\boldsymbol{x}))} = \frac{1}{1 + \exp\left(r^*(\boldsymbol{y}_1|\boldsymbol{x}) - r^*(\boldsymbol{y}_2|\boldsymbol{x})\right)}$$

Assuming the access to a dataset $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}_w^{(i)}, \boldsymbol{y}_l^{(i)})\}_{i=1}^N$ sampled from p^*. We parameterize the reward as r_ϕ and estimate it via maximum log-likelihood:

$$\mathcal{L}_R = \sum_{i=1}^{N} \frac{1}{N} \log \sigma \left(r_\phi(\boldsymbol{y}_w^{(i)}|\boldsymbol{x}^{(i)}) - r_\phi(\boldsymbol{y}_l^{(i)}|\boldsymbol{x}^{(i)}) \right) \tag{1}$$

where σ is the sigmoid function. r_ϕ is initialized with π^{SFT} augmented by additional linear layers on top. Constraints like $\mathbb{E}\left[r(\boldsymbol{y}|\boldsymbol{x})\right] = 0$ might be incorporated to lower the variance.

- **RL Fine-Tuning Phase.** Prior work formulate the optimization problem as:

$$\max_{\pi_\theta} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}, \boldsymbol{y}\sim\pi_\theta(\cdot|\boldsymbol{x})} \left[r_\phi(\boldsymbol{y}|\boldsymbol{x}) - \beta D_{\text{KL}}(\pi_\theta(\cdot|\boldsymbol{x}) \| \pi^{\text{SFT}}(\cdot|\boldsymbol{x})) \right] \tag{2}$$

The $\beta D_{\text{KL}}(\pi_\theta(\cdot|\boldsymbol{x}) \| \pi^{\text{SFT}}(\cdot|\boldsymbol{x}))$ term is used to regulate the deviation from the SFT model, it is important to prevent the model from completely forget the world knowledge acquired in the pre-training stage. The standard approach is to directly employ PPO (Schulman et al., 2017; Ouyang et al., 2022) to optimize the modified reward $r_\phi(\boldsymbol{y}|\boldsymbol{x}) - \beta \left(\log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) - \log \pi^{\text{SFT}}(\boldsymbol{y}|\boldsymbol{x})\right)$.

[☞ T. Wu, et al., "Harnessing Relative Feedback for LLM Alignment," arxiv, 2023]

B. Large Language Model Families

1) GPT Family OpenAI

➢ **GPT-4** (March, 2023)

- Multimodal LLM ; it can take image and text as inputs and produce text outputs
- GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers
- Like early GPT models, GPT-4 was first pre-trained to predict next tokens on large text corpora, and then fine-tuned with RLHF to align model behaviors with human-desired ones.

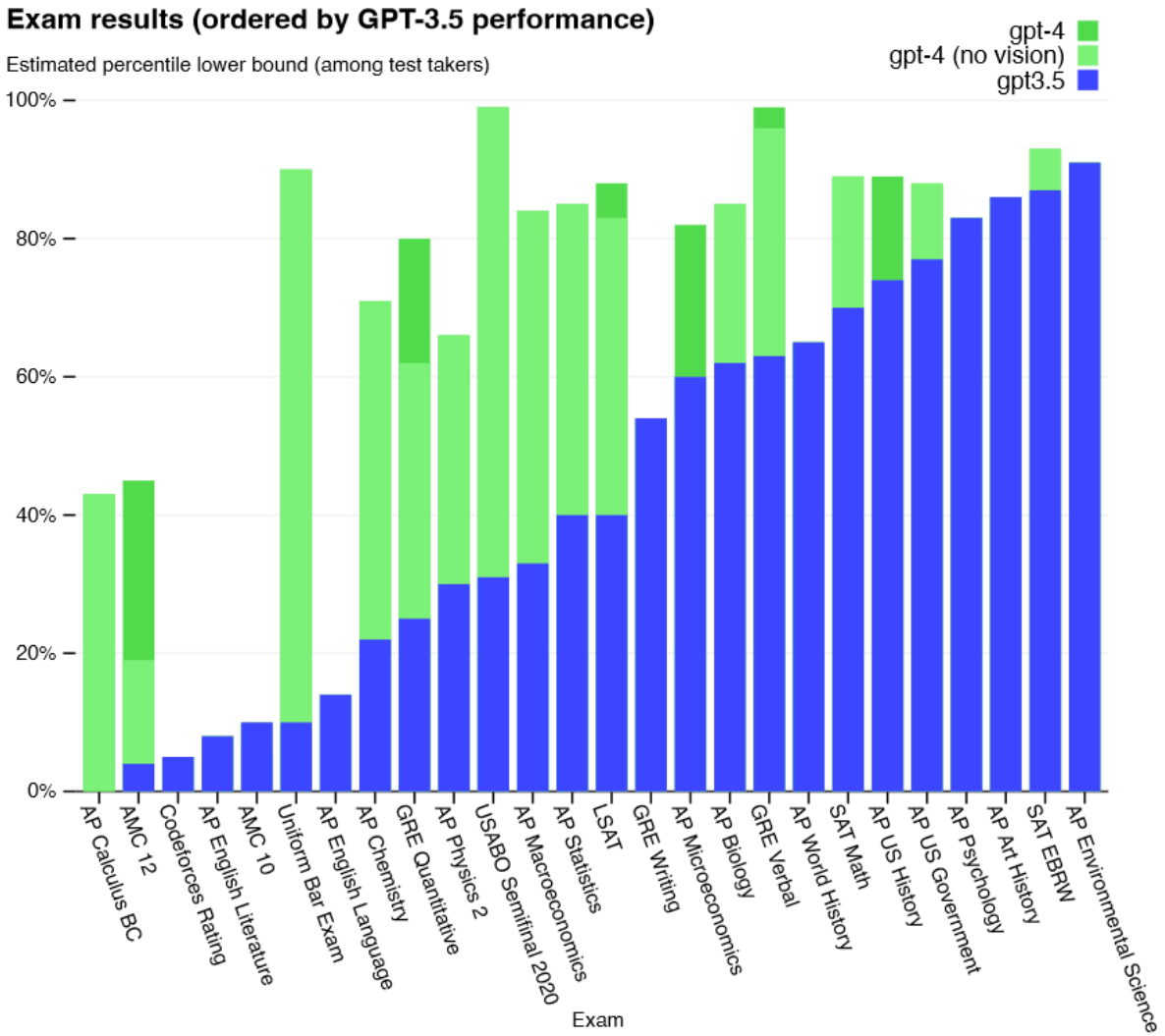


Fig. 11: GPT-4 performance on academic and professional exams, compared with GPT 3.5.

B. Large Language Model Families

[☞ 출처 Chatgpt o3; GPT-4 / GPT-4.5 / GPT-40]

1) GPT Family OpenAI

구분	GPT-4	GPT-4.5	GPT-4o (“omni”)
최초 공개	2023-03-14	2025-02-27 (리서치 프리뷰)	2024-05-13
주요 입력·출력	• **텍스트·이미지 입력** • **텍스트 출력**	• **텍스트·이미지 입력** • **텍스트 출력**	• **텍스트·이미지·오디오·비디오 입력** • **텍스트·이미지·오디오 출력 (단일 NN)**
지향점	• **범용 지식·추론의 첫 멀티모달 LLM**	• **대규모 사전학습(unsupervised) 확장 → 지식·사실성 강화**	• **실시간(≈320 ms) 다중모달 상호작용**
컨텍스트 윈도우	• **8K·32K** (API), 일부 128K(Plus 이상)	• **최대 128K**(Plus·Pro·API)	• **16K**(출시 시점)
대표 성능	• MMLU 0.86, 상위 10 % 변호사시험 통과 등	• GPT-4 대비 인간 선호도 ↑·환각 ↓·EQ ↑	• GPT-4 Turbo 수준 텍스트 + 다국어 ↑, 비전·음성 SOTA
API 가격 (입력/출력, 1M tokens)*	• *$30 / $60 (8K) ▶ Turbo $10 / $30 (128K) Open AI Help Center*	• *$75 / $150 (프리뷰) Informa TechTarget*	• *$5 / $20 (표준) · $0.60 / $2.40(4o mini) OpenAI*
강점	• 인간수준 추론, 이미지 수용 • RLHF 기반 안전성	• 더 넓은 지식·적은 환각 • “EQ” & 대화 자연도 강화	• 실시간 멀티모달(음성 ↔ 음성) • 2× 속도·½ 비용(4 Turbo 대비) OpenAI
한계	• 느린 응답·높은 비용	• 고비용, 속도·응답 지연, 멀티모달 제한(텍스트, 이미지), 복합 추론·코딩 성능 한계	• 텍스트 컨텍스트 16K 제한(확장 예정) • 일부 고난도 추론은 여전히 4.5·4.1이 우위

B. Large Language Model Families

[☞ 출처 Chatgpt o3; GPT-5]

1) GPT Family OpenAI

➢ **GPT-5** (2025 하반기 출시예정) ; **Frontier-level LLM – "LLM + Reasoning + Agent"** **막대한 연산/데이터/안전성 요구**

❖ 현재(2025-04-27) 기준으로 GPT-5는 공식 출시 전 단계이며, OpenAI가 일부 기업 고객에게만 기술 데모 시연, 내부 안전성("레드팀") 테스트 중

▪ 목표 및 기능 사양

① Reasoning : Chain-of-Thought + 복합 도구 호출이 결합된 "**적응형 Reasoning Engine**"

② Modality : 텍스트·이미지·음성·비디오 입출력(4o 수준의 실시간 멀티모달)에 더해 "**장면 이해·액션 계획**"(**에이전트 제어**) 강화 예상

③ Context : 4o(128 K)보다 더 큰 윈도우(업계는 **256K~1M Token 추정**) — Gemini 1.5 Ultra·Claude 3 Opus 대비 우위 확보 목적

④ Personalization(개인화) : 사용자별 저비용 미세조정·장기 메모리·프라이버시-존중형 온-디바이스 캐시 등 "**맞춤 AI**" 기능 탐색

⑤ 모델 통합 : GPT 계열(언어) + O 계열(실시간 Reasoning)을 단일 네트워크로 일원화하여 "**모델 선택 없는 ChatGPT**" 실현 예정

▪ 활용 시나리오

Mult-agent orchestration	에이전트가 하위 툴·API를 *자율 호출* → 복합 업무(리포트 작성 + 스프레드시트 편집 + 코드 배포) **원패스 자동화**
Realtime Multimodal UX	음성 ↔ 비디오 ↔ 텍스트가 끊김 없이 전환 → **콘택트센터·실시간 통역·AR 도우미** 등 지연 민감 서비스
장문·복합 문서 분석	100K+ 토큰 맥락 처리로 **규정집·법률·의료 기록** 같은 초장문 질의/추론 지원
개인 맞춤형 AI	사용자가 허용한 개인 데이터로 미세조정된 **온-디바이스 '마이크로 GPT'** 제공 가능성

❖ Advice Tips

- 4o mini / 4 Turbo / 4.5 조합으로 멀티모달·장문·고정밀 워크플로를 먼저 실험하고,GPT-5 API가 미리보기 형태로 열리면 안전성 가이드라인을 준수하며 점진적 마이그레이션을 준비하는 것을 권장함

B. Large Language Model Families

2) LLaMA Family Meta

- A collection of foundation language models, released by Meta
- LLaMA models are open-source, i.e., model weights are released to the research community under a noncommercial license.

➢ LLaMA-1 model (Feb. 2023)

- 7B~65B parameters
- Pre-trained on trillions (10^{12}) of tokens, collected from publicly available datasets
- Uses the transformer architecture of GPT-3, with a few minor architectural modifications, including (1) using a **SwiGLU** activation function instead of ReLU, (2) using **rotary positional embeddings** instead of absolute positional embedding, and (3) using **root-mean-squared layer normalization** instead of standard layer-normalization.

☞ H. Touvron, et al., "Llama: Open and efficient foundation language models," arXiv preprint arXiv:2302.13971, 2023.

➢ **LLaMA-2** Collection (July. 2023) with Microsoft

- Include both foundation language models and Chat models finetuned for dialog, so called, LLaMA-2 Chat
- 1) Pre-training LLaMA-2 using publicly available online data.
- 2) Initial version of LLaMA-2 Chat is built via **supervised fine-tuning**.
- 3) The model is iteratively refined using **RLHF**, rejection sampling and proximal policy optimization.

✓ In the RLHF stage, the accumulation of human feedback for revising the reward model is crucial to prevent the reward model from being changed too much

☞ H. Touvron, et al., "Llama 2: Open foundation and fine-tuned chat models," arXiv preprint arXiv:2307.09288, 2023.

B. Large Language Model Families

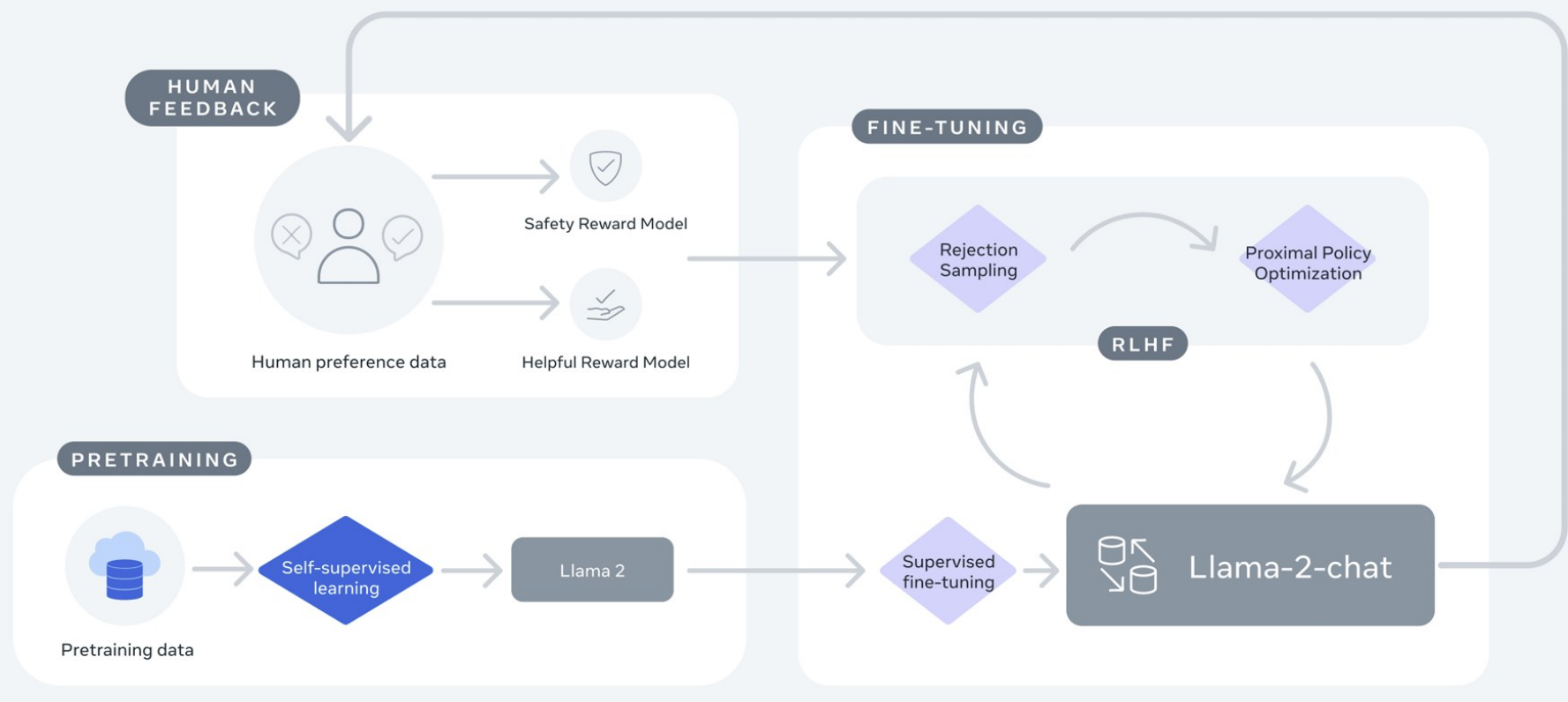


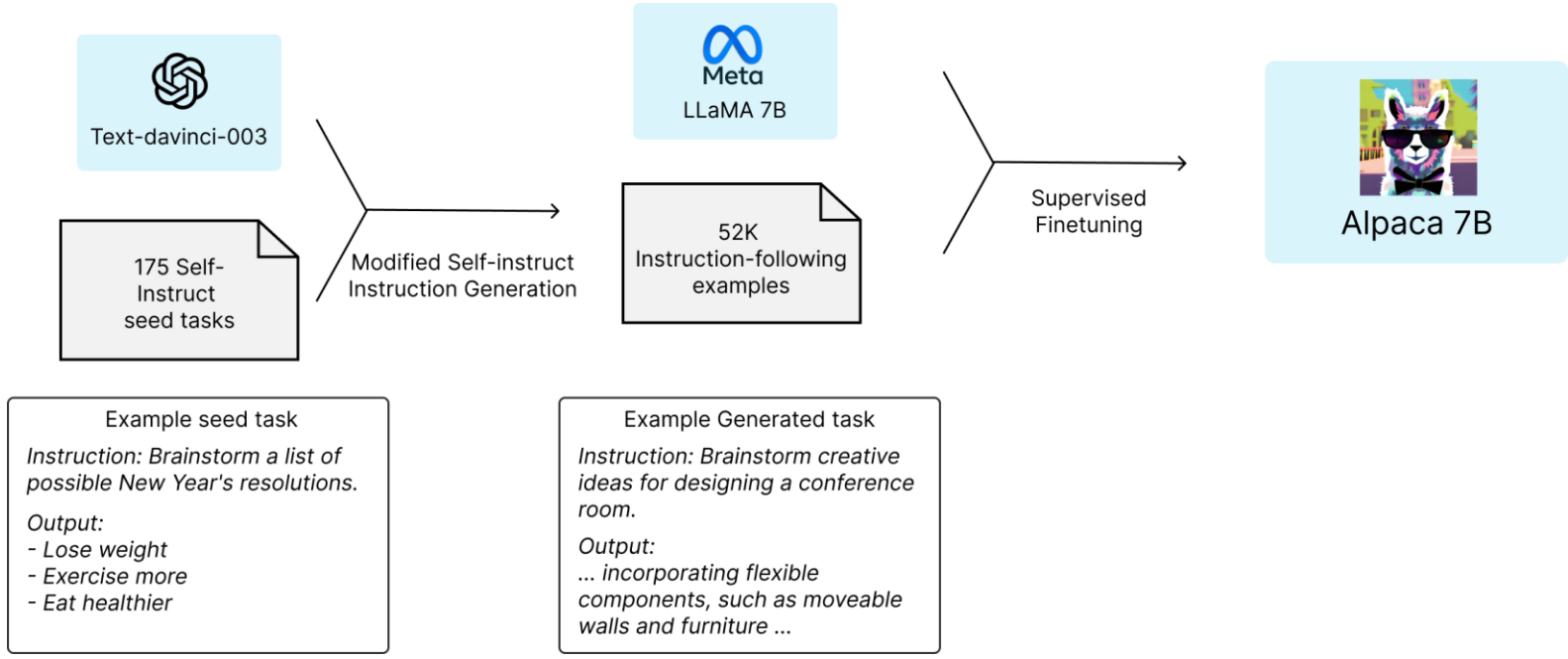
Figure 4: **Training of Llama 2-Chat**: This process begins with the **pretraining** of Llama 2 using publicly available online sources. Following this, we create an initial version of Llama 2-Chat through the application of **supervised fine-tuning**. Subsequently, the model is iteratively refined using Reinforcement Learning with Human Feedback (**RLHF**) methodologies, specifically through rejection sampling and Proximal Policy Optimization (PPO). Throughout the RLHF stage, the accumulation **of iterative reward modeling data** in parallel with model enhancements is crucial to ensure the reward models remain within distribution.

B. Large Language Model Families

2) LLaMA Family Meta

- **Alpaca** (2023, Stanford Center for Research on Foundation Models)

- Fine-tuned from the **LLaMA-7B** model using **52K instruction-following demonstrations** generated in the style of **self-instruct** using **GPT-3.5** (text-davinci-003).
- Very cost-effective for training, especially for academic research..



The figure illustrates how we obtained the Alpaca model. For the data, we generated instruction-following demonstrations by building upon the self-instruct method. We started with the 175 human-written instruction-output pairs from the self-instruct seed set. We then prompted text-davinci-003 to generate more instructions using the seed set as in-context examples. We improved over the self-instruct method by simplifying the generation pipeline (see details in GitHub) and significantly reduced the cost. Our data generation process results in 52K unique instructions and the corresponding outputs, which costed less than $500 using the OpenAI API.

☞ R. Taori, et al., “Alpaca: A strong, replicable instruction-following model,” Stanford Center for Research on Foundation Models, https://crfm.stanford.edu/2023/03/13/alpaca.html, vol. 3, no. 6,p. 7, 2023.

B. Large Language Model Families

2) LLaMA Family Meta

➢ **Vicuna** (March 2023, Vicuna Team)

- Vicuan team developed a 13B chat model, **Vicuna-13B**, by fine-tuning LLaMA on user-shared conversations collected from ShareGPT.
- More than 90% quality of OpenAI's ChatGPT and Google's Bard while outperforming LLaMA and Stanford Alpaca in more than 90% of cases

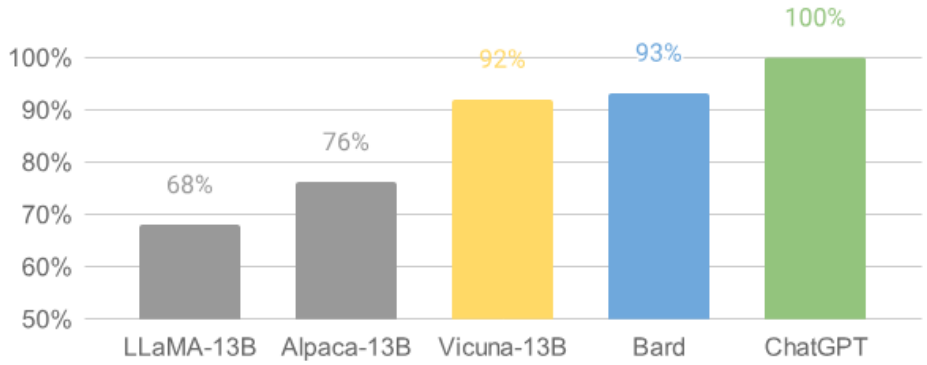


Figure 1. Relative Response Quality Assessed by GPT-4*

☞ Vicuna team, https://lmsys.org/blog/2023-03-30-vicuna/

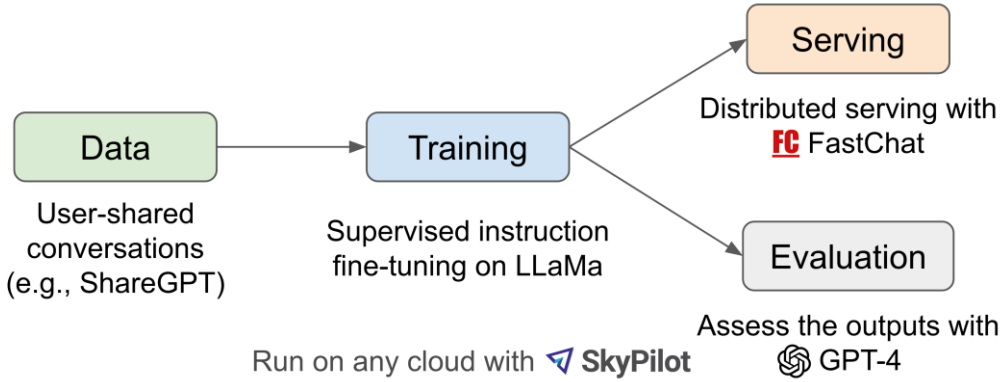


Figure 2. Workflow Overview

- Training recipe builds on top of Stanford's alpaca with the following improvements.
- **Multi-turn conversations**: We adjust the training loss to account for multi-turn conversations and compute the fine-tuning loss solely on the chatbot's output.
- **Memory Optimizations**: To enable Vicuna's understanding of long context, we expand the max context length from 512 in alpaca to 2048, which substantially increases GPU memory requirements. We tackle the memory pressure by utilizing *gradient checkpointing* and *flash attention*.
- **Cost Reduction via Spot Instance**: The ***40x*** larger dataset and ***4x*** sequence length for training poses a considerable challenge in training expenses. We employ **SkyPilot** managed spot to reduce the cost by leveraging the cheaper spot instances with auto-recovery for preemptions and auto zone switch. This solution slashes costs for training the 7B model from 140 and the 13B model from around 300.

B. Large Language Model Families

2) LLaMA Family Meta

➢ **Mistral-7B** (2023, Mistral AI team)

- Outperforms the best open-source 13B model (LLaMA-2-13B) across all evaluated benchmarks, and the best open-source 34B model (LLaMA-34B) in reasoning, mathematics, and code generation.
- Leverages ***grouped-query attention*** for faster inference, coupled with ***sliding window attention*** to effectively handle sequences of arbitrary length with a reduced inference cost.

- Architecture Overview

- **RMS Normalization** — replacing Layer Normalization
- **Rotary Position Embedding (RoPE)** — replacing Absolute Positional Encoding
- **Grouped Query Attention (GQA)** — replacing Multi-Head Attention
- **Sliding Window Attention (SWA)** — improving training and inference speed, particularly for long sequences
- **Rolling Buffer KV Cache** — improving training and inference speed, in conjunction with SWA
- **SwiGLU Activation Function** — replacing ReLU in the Feed Forward sub-layers

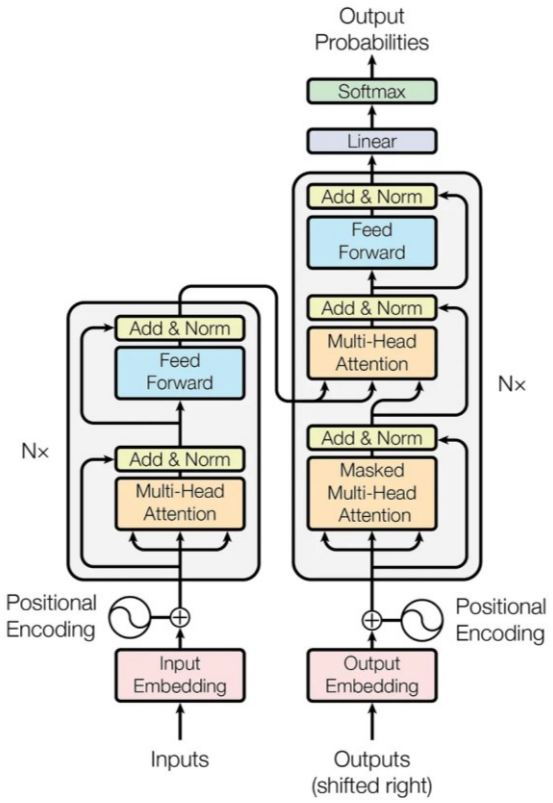
☞ A.Q. Jiang, et al., “Mistral 7b,” arXiv preprint, arXiv:2310.06825, 2023.

☞ Mistral 7B Explained: Towards More Efficient Language Models; https://medium.com/data-science/mistral-7b-explained-towards-more-efficient-language-models-7f9c6e6b7251

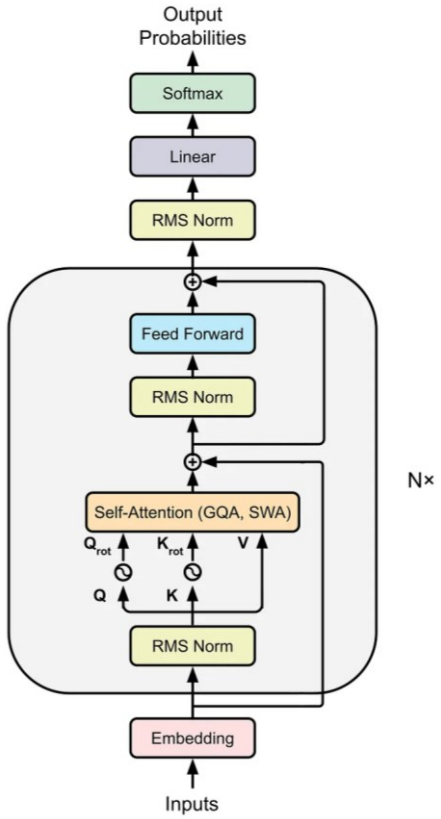
B. Large Language Model Families

2) LLaMA Family Meta

➢ **Mistral-7B** (2023, Mistral AI team)



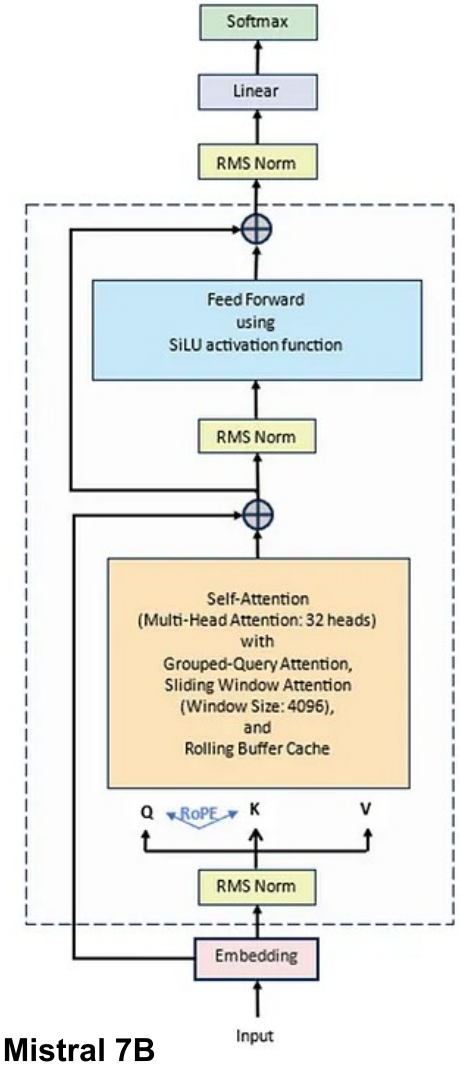
Transformer



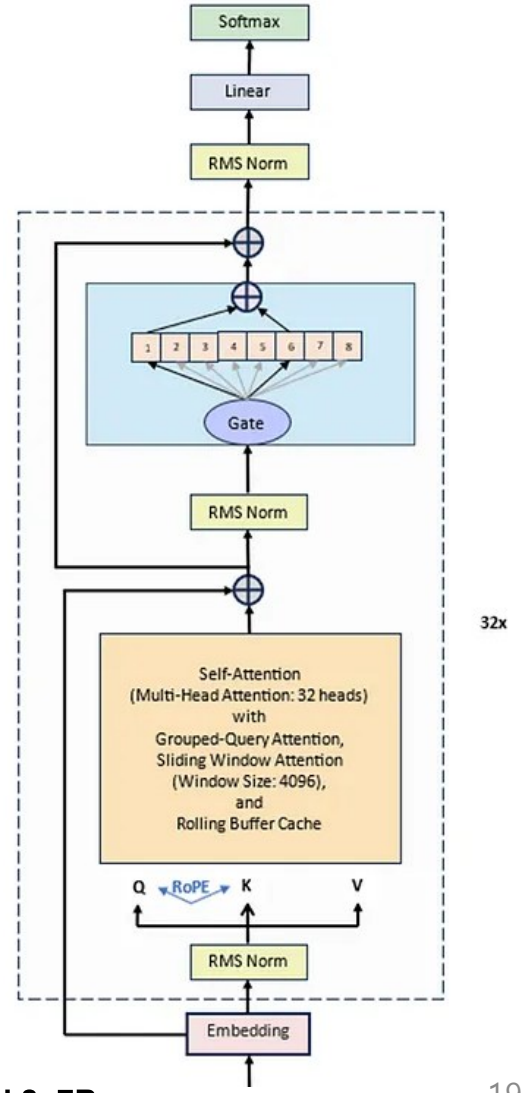
Mistral 7B

☞ Mistral 7B Explained: Towards More Efficient Language Models; https://medium.com/data-science/mistral-7b-explained-towards-more-efficient-language-models-7f9c6e6b7251

☞ Mistral 7B and Mixtral 8x7B; https://medium.com/@EleventhHourEnthusiast/paper-reviews-mistral-7b-and-mixtral-8x7b-e8f5a011ebbf



Mistral 7B



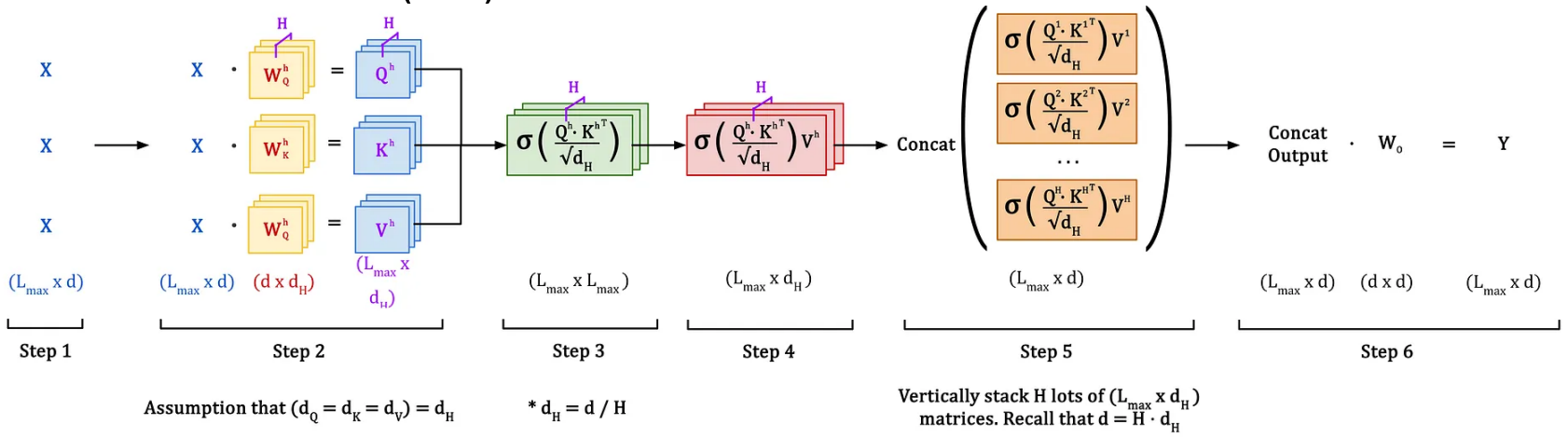
Mixtral 8x7B

☞ Mistral 7B Explained: Towards More Efficient Language Models; https://medium.com/data-science/mistral-7b-explained-towards-more-efficient-language-models-7f9c6e6b7251

B. Large Language Model Families

❖ **Grouped Query Attention (GQA)**

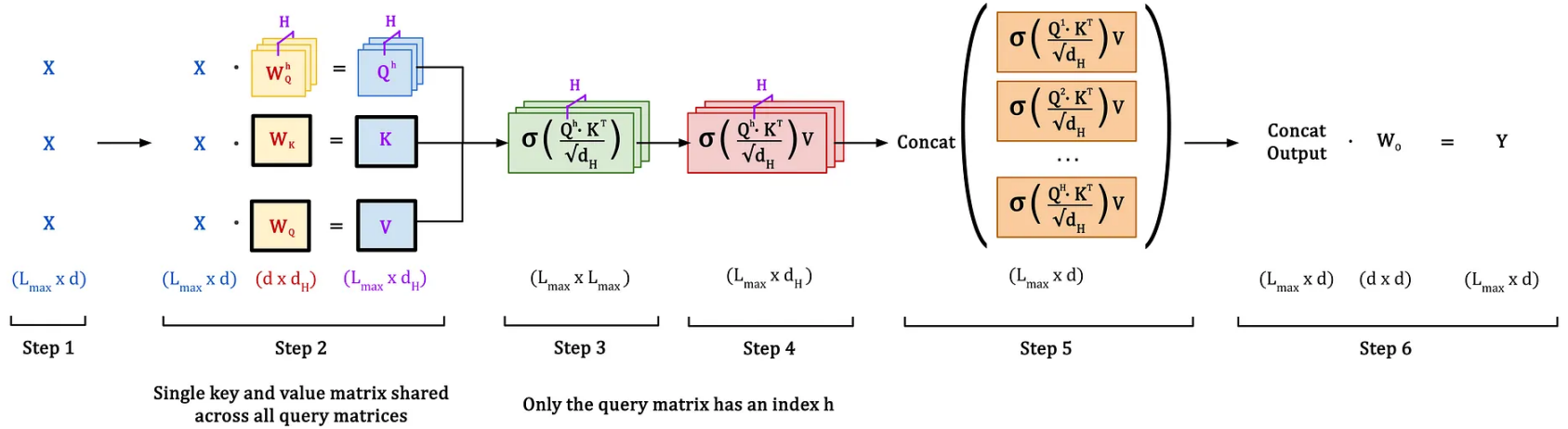
▪ **Multi-Head Attention (MHA)**



MHA : Limit of Memory-Intensive

large Key and Value matrices must be stored in memory for each attention head, causing a bottleneck that limits the overall model size that can be used with a given hardware setup

▪ **Multi-Query Attention (MHA)**



MQA

Fast Transformer Decoding: One Write-Head is All You Need, 2019

the same Key and Value matrices are shared across all heads, and only the Query matrices are head-specific.

B. Large Language Model Families

❖ **Grouped Query Attention (GQA)**

- GQA : Google Research, “GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints”, https://arxiv.org/abs/2305.13245, 2024

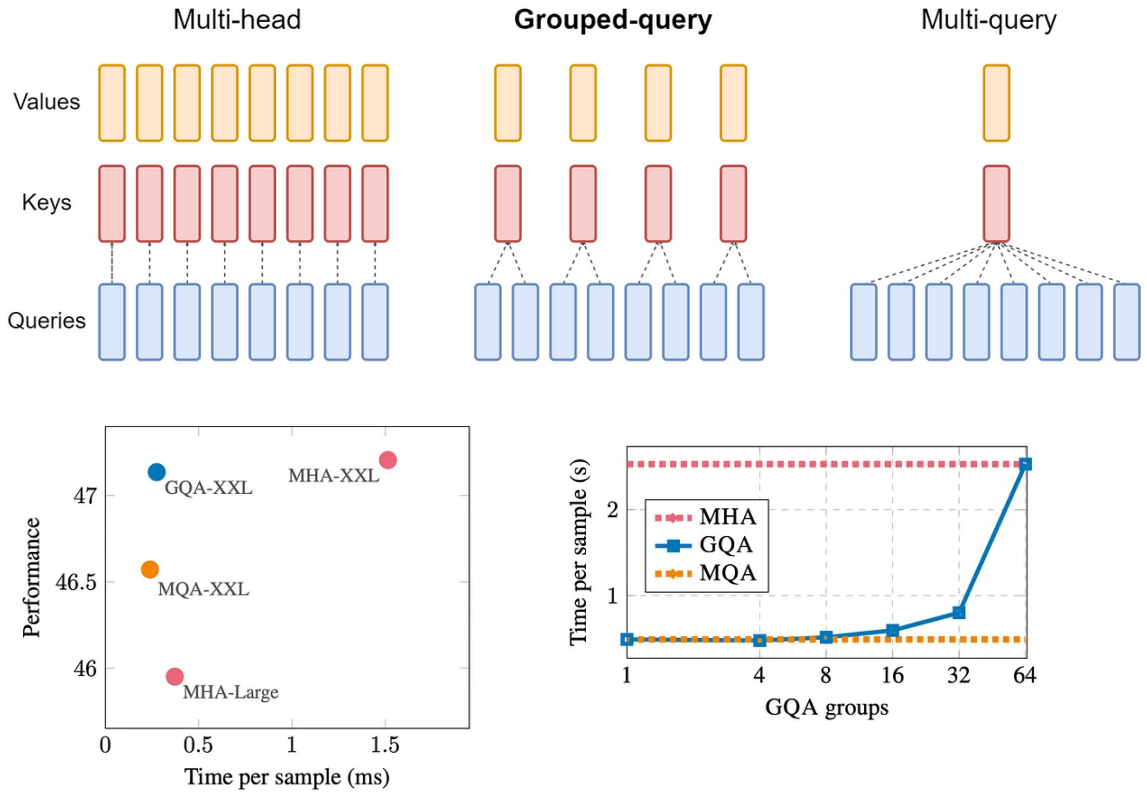
- **Key and Value matrices are shared between G groups of heads**, where the group size is determined by the user.

  1) If all the groups contain a single head (G=1), each head has its own unique Key and Value matrices, which is equivalent to MHA.

  2) If every head belongs to a single group (G=H), all the heads share the same Key and Value matrices, which is equivalent to MQA.

- The strength of GQA lies in selecting a group size such that the performance losses are minimal and the memory efficiency is much improved.

☞ Mistral 7B Explained: Towards More Efficient Language Models; https://medium.com/data-science/mistral-7b-explained-towards-more-efficient-language-models-7f9c6e6b7251



A comparison of the performance of Multi-Head, Multi-Query, and Grouped Query Attention. The left graph shows performance vs run time, showing that GQA achieves performance similar to MHA while maintaining a run time comparable to MQA. The right graph shows the relationship between the number of groups (G) in GQA and the run time, with **G=32** giving strong performance and a low run time. Image taken from [15].

☞ Mistral 7B Explained: Towards More Efficient Language Models; https://medium.com/data-science/mistral-7b-explained-towards-more-efficient-language-models-7f9c6e6b7251

B. Large Language Model Families

❖ Incremental inference

- **Incremental inference** as an optimisation technique, which utilises a **standard KV cache**, to improve the efficiency of LLMs as their sizes increase
- Only compute a query vector and small updates to the cached Key and Value matrices rather than recalculating the full Query, Key, and Value matrices at every timestep

1. Calculate $\mathbf{Q_h}$, **K**, and **V**: K and V matrices are stored in a KV cache
2. Predict $\mathbf{x_{new}}$: the first token of the output sequence
3. Calculate a corresponding query vector $\mathbf{q_{new,h}}$ $q_{\text{new},h} = x_{\text{new}} W_Q^h$
4. Attention Step: Combine q_(new, h) with the cached K and V matrices

$$\text{Attention}(q_{\text{new}}, K, V) = \text{softmax}\left(\frac{q_{\text{new}} K^T}{\sqrt{d_H}}\right) V$$

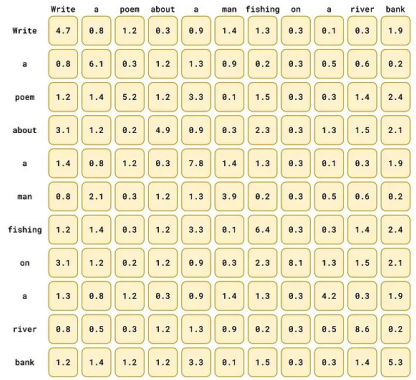
5. Updating the KV Cache: Compute the key and value vectors for the new token ($\mathbf{k_{new}}$ and $\mathbf{v_{new}}$)

$$k_{\text{new}} = x_{\text{new}} W_K, \quad v_{\text{new}} = x_{\text{new}} W_V$$

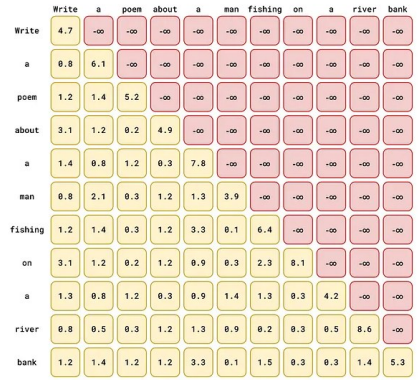
6. Repeating the Process: with the model predicting one token at a time, until the End of Sequence (EOS) token is generated.

❖ Sliding Window Attention (SWA)

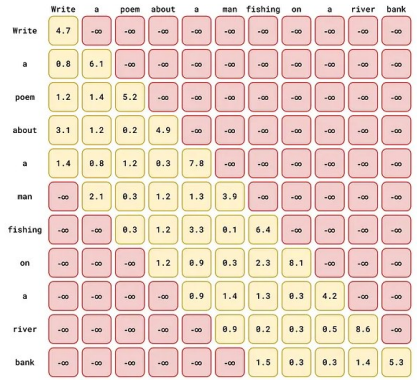
- GQA : Google Research, “GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints”, https://arxiv.org/abs/2305.13245, 2024



Vanilla attention



Causal attention



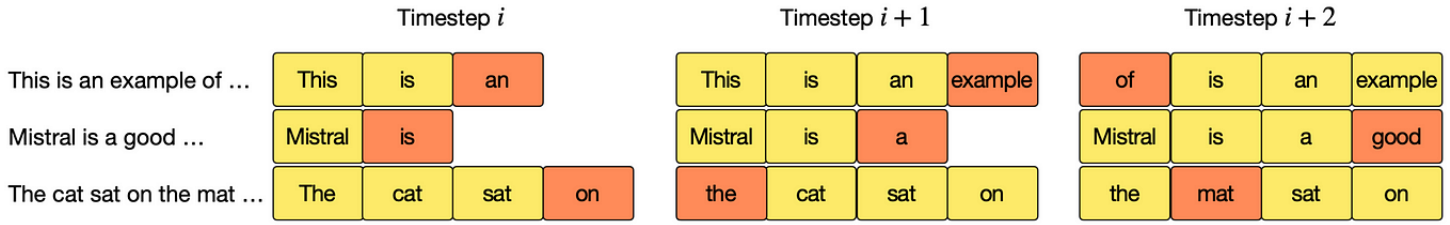
Sliding Window Attentio

☞ Mistral 7B Explained: Towards More Efficient Language Models; https://medium.com/data-science/mistral-7b-explained-towards-more-efficient-language-models-7f9c6e6b7251

B. Large Language Model Families

❖ Rolling Buffer KV Cache

- Rolling Buffer KV Cache extends this further by taking advantage of the sliding window in Sliding Window Attention.
- "**Rolling Buffer**" refers to the *Key* and *Value* matrices in the *cache* only storing information for tokens within the current attention window.
- the cache can "forget" tokens outside the local context, significantly reducing memory usage while maintaining the necessary information for accurate token generation.
- Enable the model to handle long inputs efficiently, making the **32,000-token context length** feasible without incurring excessive memory usage.



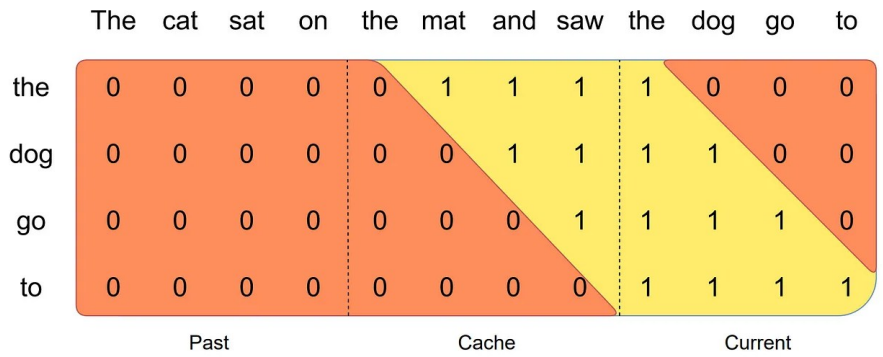
Rolling Buffer KV Cache for a window size of 4

● Pre-filling

- Pre-filling refers to **populating the KV Cache with the key and value vectors** for all tokens in the input sequence prior to incremental inference.
- Ensure that the static portion of the input sequence (e.g., a prompt) is fully processed ahead of time, reducing redundant computation when generating new tokens.

● Chunking

- Addresses the challenge of handling long sequence lengths by dividing the input into fixed-length sections called chunks, equal to the window size of the attention mechanism.



An overview of the KV Cache where the input sequence has been pre-filled across three chunks. Tokens in the final chunk can only attend to themselves and the previous chunk, as long as the tokens are within the local context window.

B. Large Language Model Families

2) LLaMA Family Meta

[☞출처 Chatgpt o3; LLaMA 계열 모델]

➢ **LLaMA Family Models Released from 2024**

Model	Parameter	Released	Features	Reference
Meta Llama 3	8 B / 70 B	2024-04-18	Llama 2 후속 · 성능 SOTA, 상업적 사용 가능(Community License)	Meta AI
Llama 3.1	8 B / 70 B	2024-07	**Context 128 K** 확장·다국어 강화	Meta AI
Llama 3.3	70 B	2024-12-06	동일 성능·GPU 4 GB 구동 가능한 경량("compute-efficient") 버전	PYMNTS.com
Code Llama 70 B (Base·Python·Instruct)	70 B	2024-02-06	코드 생성 특화, MIT-style 라이선스	LinkedInMedium
Llama Guard 2 8 B	8 B	2024-04	프롬프트·응답 안전 필터링 전용 모델	Huggingface Medium

➢ **Major LLaMA derived/professional tuning models**

Model	Base	Released	Features	Reference
WizardLM-2 (7 B·70 B·8×22 B)	Llama 3	2024-04-15	추론·멀티언어 강화, 인스트럭션 튜닝	Huggingface
LLaVA-NeXT 32 B	Llama 3	2024-07-16	비전·비디오 인스트럭션 어시스턴트	GitHub
InternVL2-Llama3 76 B	Llama 3	2025-04	대규모 멀티모달 연구용	Huggingface
Finance-Llama3 8 B	Llama 3	2025-02	금융 문서·질문응답 특화	Huggingface

B. Large Language Model Families

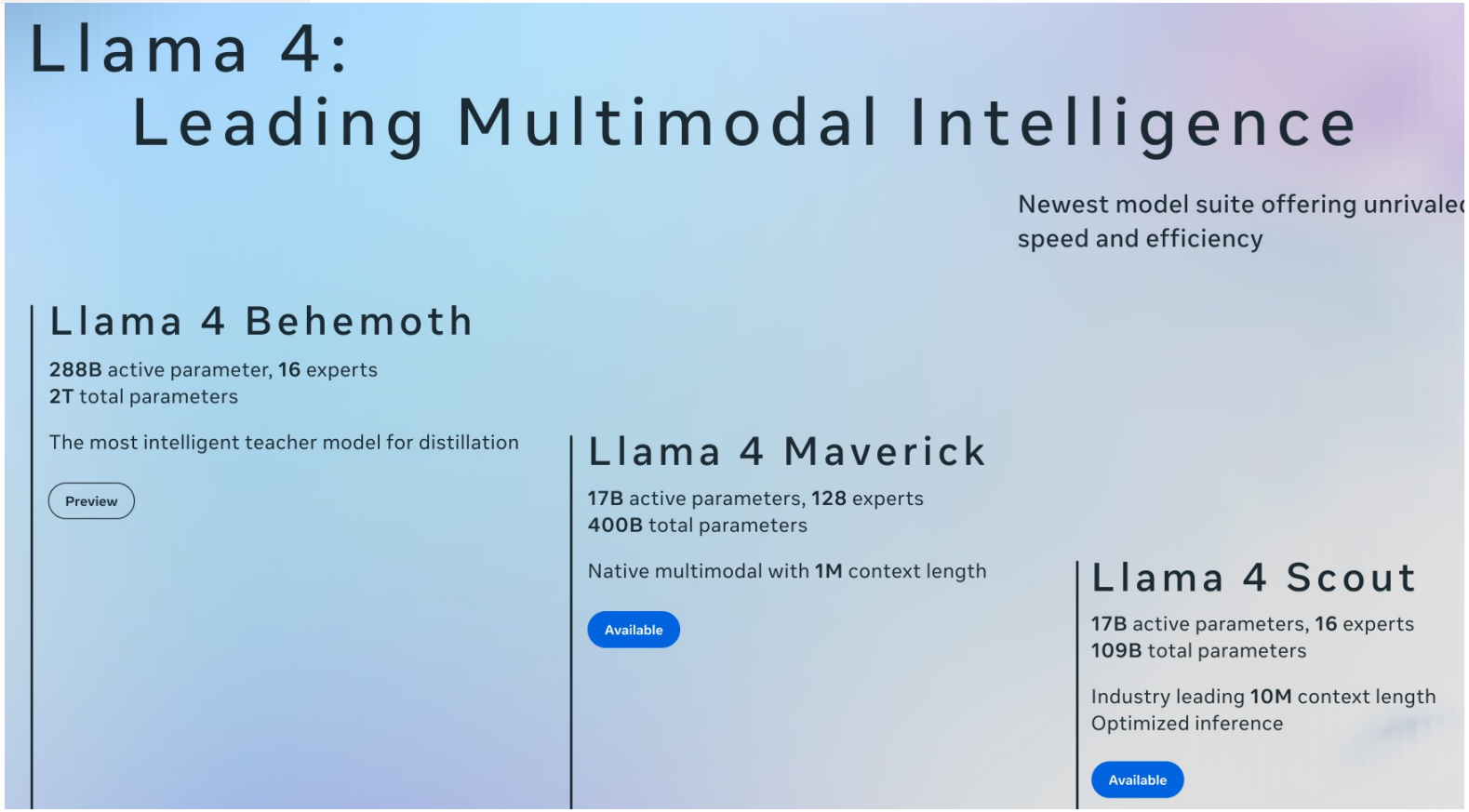
[☞ https://ai.meta.com/blog/llama-4-multimodal-intelligence/]

2) LLaMA Family Meta

- **Meta Llama 3: The most capable openly available LLM to date** (April 18, 2024)

- **The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation** (April 5, 2025)

Meta Llama 3 & Llama 4 Paper Review



Behemoth : An early preview (it's still training!) of the **Llama 4 teacher model** used to distill Llama 4 Scout and Llama 4 Maverick.

Maverick : **Industry-leading natively multimodal model** for **image and text understanding** with ***groundbreaking intelligence*** and ***fast responses*** at a low cost.

Scout : Class-leading natively multimodal model that offers **superior text and visual intelligence**, **single H100** GPU efficiency, and a **10M context window** for seamless long document analysis.

B. Large Language Model Families

[☞출처 Chatgpt o3; PaLM]

3) PaLM (Pathways Language Model) Family

- PaLM: Scaling Language Modeling with Pathways, arxiv, 2022
- Google Research가 '**Pathways**'라는 대규모 분산 학습 인프라 위에서 훈련한 초거대 Transformer 언어모델 계열임. 첫 논문(2022)은 540 B 파라미터를 가진 단일(dense) 모델을 소개하며, "모델·데이터·연산 세 축을 동시에 10× 이상 확장하면 복잡 추론 능력이 불연속적으로(i.e. "emergent") 향상된다"는 것을 입증함

- Main features
- Few-/Zero-shot 성능: 29개 자연어·코드 벤치마크에서 GPT-3 175 B를 큰 폭으로 상회.
- Chain-of-Thought(COT) 프롬프트를 처음으로 체계적으로 실험 → GSM8K 등 수학 문제에서 SOTA 기록
- 다국어·코드(C++, Python) 생성 모두 강력.
- BIG-bench 58 % 문제에서 인간 평균을 초과

단계	공개·업데이트	핵심 변화	현황(2025)
PaLM 1 (540 B) (2022-04)	*Dense* 540 B 파라미터, TPU-v4 × 6 144 "Pathways" 훈련	Few-/zero-shot·Chain-of-Thought(COT) 능력 증명	연구용 API만 유지, 신규 기능 추가 X
PaLM 2 (I/O 2023)	Gecko (모바일) ~ Unicorn(대형) 4 규모군·다국어·코드↑	Bard 초기 엔진, Vertex AI "Codey"·Workspace Duet 탑재	**Still-alive**: 2025-Q1 기준 Bard→Gemini 전환 뒤에도 Vertex AI에서 PaLM 2 Bison API 제공
PaLM 2-S (2023-10)	PaLM 2 Bison을 **Server-Side Streaming** 엔진으로 최적화 → 레이턴시 30 %↓	클라우드 문서 초안·Meet 실시간 요약 등에 적용	계속 서비스 中 (Gemini Pro 선택 가능)
Med-PaLM 2 (2023-12)	14 국가 의학 라이선스 시험 ~ Pass@60 % 달성, 팩트 검증 추가	Google Cloud Vertex AI "Preview" → 2025-Q1 **GA(일반공개)**	
Sec-PaLM 2 (2024-02)	보안 로그·악성코드 분석 특화, Mandiant 위협 인텔리전스 연동	Chronicle Security AI에 내장 → 2025-Q1 **GA**	
PaLM-E (Embodied) (2024-05)	언어+비전+로봇 센서 입력 통합, 562 B 파라미터	Alphabet X Everyday Robots 실험 지속; 2025-Q1 논문 후속편 제출 완료	
PaLM 2-Max(128 K) (2024-09)	Bison 아키텍처 ↔ *2-stage RAG*로 128 K 컨텍스트	Workspace Duet 문서·슬라이드 장문 요약에 사용	
PaLM 2-Flash (2025-02)	텍스트 전용 Micro-MoE, 4 K-32 K 윈도우·지연 <250 ms	Google Cloud Functions for AI(베타) 실시간 응답	

B. Large Language Model Families

[☞출처 Chatgpt o3; PaLM]

3) PaLM (Pathways Language Model) Family

- Gemini 통합 전략 (2024 → 2025)

항목	내용
Gemini 중심 전환	Google DeepMind는 2023 말 이후 "**Gemini=최상위 멀티모달·장문 라인, PaLM 2=안정적 텍스트·도메인 라인**"으로 이원화. Bard→Gemini Advanced 전환 후에도, Cloud Vertex AI는 **PaLM 2 Bison/Gecko** API를 계속 유지해 레거시 코드 호환 보장.
연구·특화 모델	의료·보안·로봇 등 **도메인 특화 파생**은 여전히 "PaLM" 브랜드 사용 (예: Med-PaLM 3 논문 pre-print 상반기 예정).
Edge AI	PaLM Gecko 계열은 2024 하반기 Pixel 9·Chromebook Plus에 배포 → 2025 Q1 "Gemini Nano"와 병행 업데이트.

- **PaLM** 라인은 2025년 현재 "**안정·도메인 특화**" 플랫폼으로 지속 운영 중이며,**일상 대화·멀티모달·초장문** 등 최첨단 기능은 **Gemini**로 컨버전스.
- Vertex AI 사용자는 2 트랙으로 안내
  - ✓ "신뢰성·비용·텍스트 전용" → PaLM 2 Bison/Flash
  - ✓ "장문·멀티모달" → Gemini 1.5 Pro/Flash

B. Large Language Model Families

[☞출처 Chatgpt o3; PaLM]

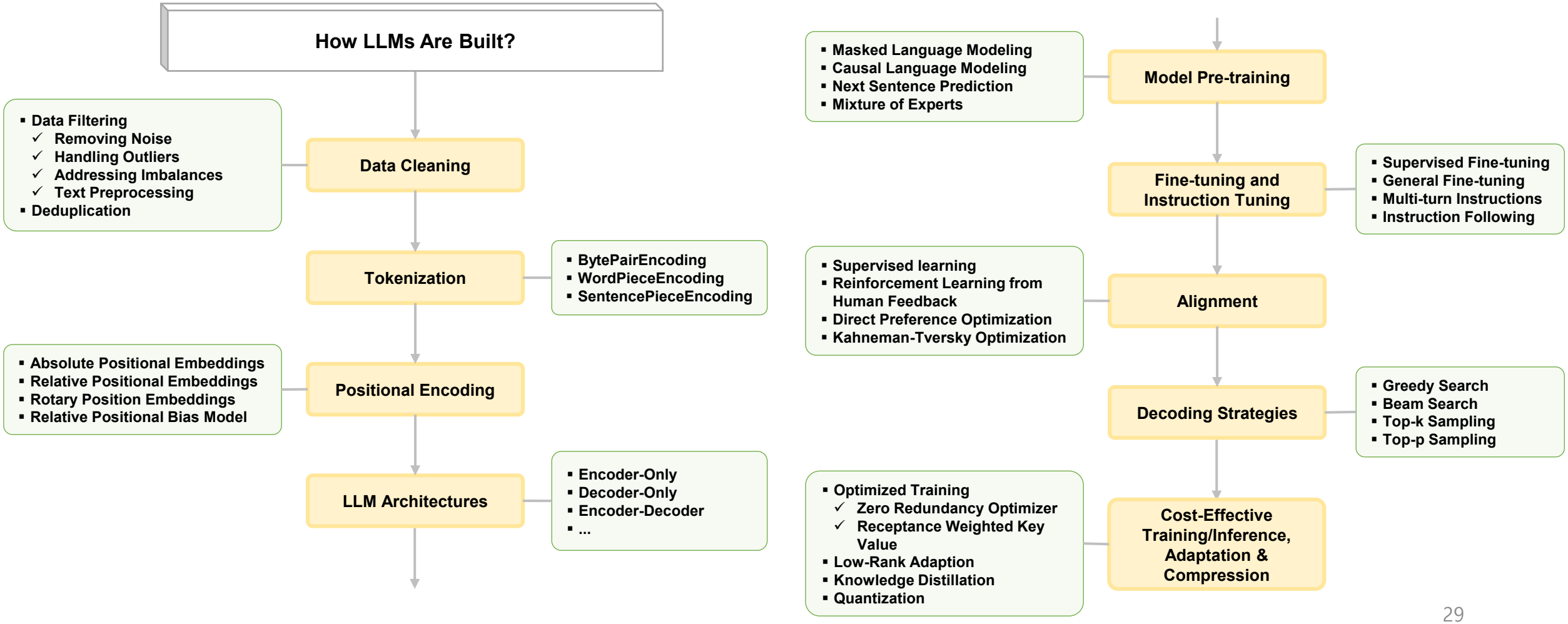
4) Gemini

- LaMDA 및 PaLM 2의 후속으로 구글 딥마인드에서 개발한 다중 모드(멀티모달) 대형 언어 모델 제품군임.
- Gemini family includes three versions: **Ultra** for highly-complex tasks, **Pro** for enhanced performance and deployability at scale, and **Nano** for on-device applications.
- Gemini는 "Multimodal + Long Context + Sparse Inference" 3대 축을 빠른 주기로 고도화하며, 2025년 2.5 세대에서 '**투명한 Reasoning**'과 Live Streaming Interface로 본격적인 에이전트 시대를 겨냥하고 있음.

Generation	Main Architecture	Parameter 활성 방식	Multimodal 처리	Maximum Context length	지향점
1.0 (Pro·Ultra)	**Dense Decoder-only Transformer**	모든 토큰이 전 층 참여	텍스트·코드·이미지·오디오·영상* 단일 시퀀스	32 K	"원-네트워크" 멀티모달 SOTA
1.5 Pro	**Sparse Mixture-of-Experts(MoE) Transformer**	100~200개 Expert 중 ≤ 8개 선택 (토큰별 라우팅)	동일 + 모달 interleave/대화형 I/O	128 K(표준) → **1 M**(프리뷰)	초장문 + 연산 효율 3-4×
1.5 Flash / 2.0 Flash	MoE 경량 파이프라인 + Layer Drop	"Thinking budget" 파라미터로 추론 깊이 자동 조절	스트리밍 멀티모달	32 K / 128 K	**지연·비용 최소화** API
2.5 Pro	**MoE v2** + "**Reasoning-path**" 블록	내부 step-by-step 상태 유지 → '생각 후 답변'	동일	**1 M** (2 M 예정)	**도구 호출·에이전트** 최적
1.0 Nano	**Distilled Dense** (≈ 1.8 B)	정적 활성	텍스트·이미지·음성 on-device	4 K	**모바일 AI Core** 저전력 Google DeepMind

How LLMS Are Built

- This section first reviews the popular **architectures** used for **LLMs**, and then discuss data and modeling techniques ranging from ***data preparation***, ***tokenization***, to ***pre-training***, ***instruction tuning***, and ***alignment***.



A. Dominant LLM Architectures

1) Transformer; Self-attention mechanism; capture long-term contextual information much more effectively using GPUs than the recurrence and convolution mechanisms

	Structure	Mechanism	Use Cases	Models
2) Encoder-Only	A stack of Transformer Encoder layers	• 입력된 텍스트 전체를 한 번에 받아 처리 • **Bidirectional Self-Attention** : 문장 내 모든 단어들이 서로의 관계를 파악하고 문맥을 이해함. 즉, 특정 단어의 의미를 파악할 때 그 단어의 앞뒤에 나오는 모든 단어를 참고함. • 주된 목적은 입력 텍스트에 대한 깊은 이해를 바탕으로 문맥 정보를 풍부하게 담은 표현(representation)을 생성함	• Text Classification (e.g., sentiment analysis, topic categorization) • Named Entity Recognition (NER) • Extractive Question Answering (finding the answer span within a given text) • Sentence Similarity	BERT, RoBERTa, ALBERT, ELECTRA
3) Decoder-Only	A stack of Transformer Decoder layers	• 텍스트를 순차적으로 생성하는데 특화 • **Masked Self-Attention** (also called **Causal Self-Attention**). 다음 단어를 예측할 때, 현재 예측 위치 뒤에 오는 단어들의 정보는 참고하지 않도록 Masking함. 오직 이전에 생성된 단어들만을 기반으로 다음 단어를 예측함 (자기회귀적, Autoregressive 방식) • 주어진 Prompt(Start Text)를 바탕으로 가장 확률 높은 다음 단어를 예측하고, 이를 다시 입력으로 사용하여 계속해서 다음 단어 생성	• **Natural Language Generation** (NLG) tasks ; Text Generation and Completion, Chatbots and Conversational AI, Creative Writing (stories, poems), Code Generation	**GPT-2,3,4, PaLM, LLaMA, Claude, Phi-3**
4) Encoder-Decoder	Both a stack of Encoder layers and a stack of Decoder layer	• **Encoder**: 입력 시퀀스 전체를 받아 Bidirectional Self-Attention을 통해 문맥 정보를 이해하고 압축된 표현을 생성. (예: 번역할 한국어 문장 입력) • **Decoder**: Encoder가 생성한 문맥 표현을 참조하여(Cross-Attention 사용), Masked Self-Attention을 통해 순차적으로 출력 시퀀스 생성함. (예: 영어 번역 문장 생성) • Encoder는 입력 의미 파악, Decoder는 의미 바탕으로 새로운 형식의 출력 생성	• Tasks involving transformation between sequences (Seq2Seq); **Machine Translation** (e.g., English to Korean), **Text Summarization**, **Generative Question Answering**	T5, BART, FLAN-T5, UL2, Gemma-Instruct

How LLMS Are Built

A. Dominant LLM Architectures

➢ Training & Inference Trade-offs

Aspect	Encoder-only	Decoder-only	Encoder-Decoder
Pre-training cost	Lower (predict < 15 % tokens)	**Higher** (predict every token)	Medium-high (two towers)
Inference latency	Single forward pass → low	**One forward pass per new token** → hi gh	Encoder pass + autoregressive decoder → moderate-high
Context length	Good for long **inputs**	Tricks (flash-attention, rolling cache) ext end limits	Encoder can be long; decoder typically s horter
Multimodal extension	Attach vision/audio encoders easily (e.g. CLIP)	Needs projection to token space	Used by Flamingo, Gemini: vision → enc oder, language → decoder

➢ Practical Selection Guide

Requirement	Recommended Family	Rationale
Rich, open-ended generation (chatbots, code, storytelling)	**Decoder-only**	Natural autoregressive output, strong world knowledge
Deep input understanding with low latency (retrieval, classification)	**Encoder-only**	Bi-directional context, fast inference
Conditional input→output transformation (translation, summarisatio n, editing)	**Encoder-Decoder**	Designed for seq-to-seq mapping with cross-attention

A. Dominant LLM Architectures

➢ **Current Trends & Hybrids**

- **Mixture-of-Encoders/Decoders** – e.g. CoDi-2: modality-specific encoders + shared language decoder.
- **Hybrid masking** – UL2, YAKE teach a single network both masked-LM and causal tasks, improving adaptability.
- **Long-context R&D** – FlashAttention-2, Ring-Attention, and MoE scaling ease context-window limits for all three families.
- **Agentic pipelines** – A common pattern couples an encoder-only retriever (fast embeddings) with a decoder-only reasoner/generator, achieving efficiency without sacrificing output fluency.

➢ **Key Takeaways**

- ***Encoder-only***: “Understand deeply and embed.
- ***Decoder-only***: “Extend context and create.”
- **Encoder-Decoder**: “Digest, transform, and respond.”

❖ Selecting the right family by use-case, latency budget, and compute resources leads to an optimal balance of performance and cost.

B. Data Cleaning

- Data cleaning techniques ; ***Filtering***, ***Deduplication*** (데이터중복제거), etc
- Example : **Falcon40B**
- *Properly filtered and deduplicated web data* alone can lead to powerful models
- Despite extensive filtering, they were able to obtain 5 trillion tokens from CommonCrawl.
- They released 600 billion tokens from our REFINEDWEB dataset, and 1.3/7.5B parameters language models trained on it.

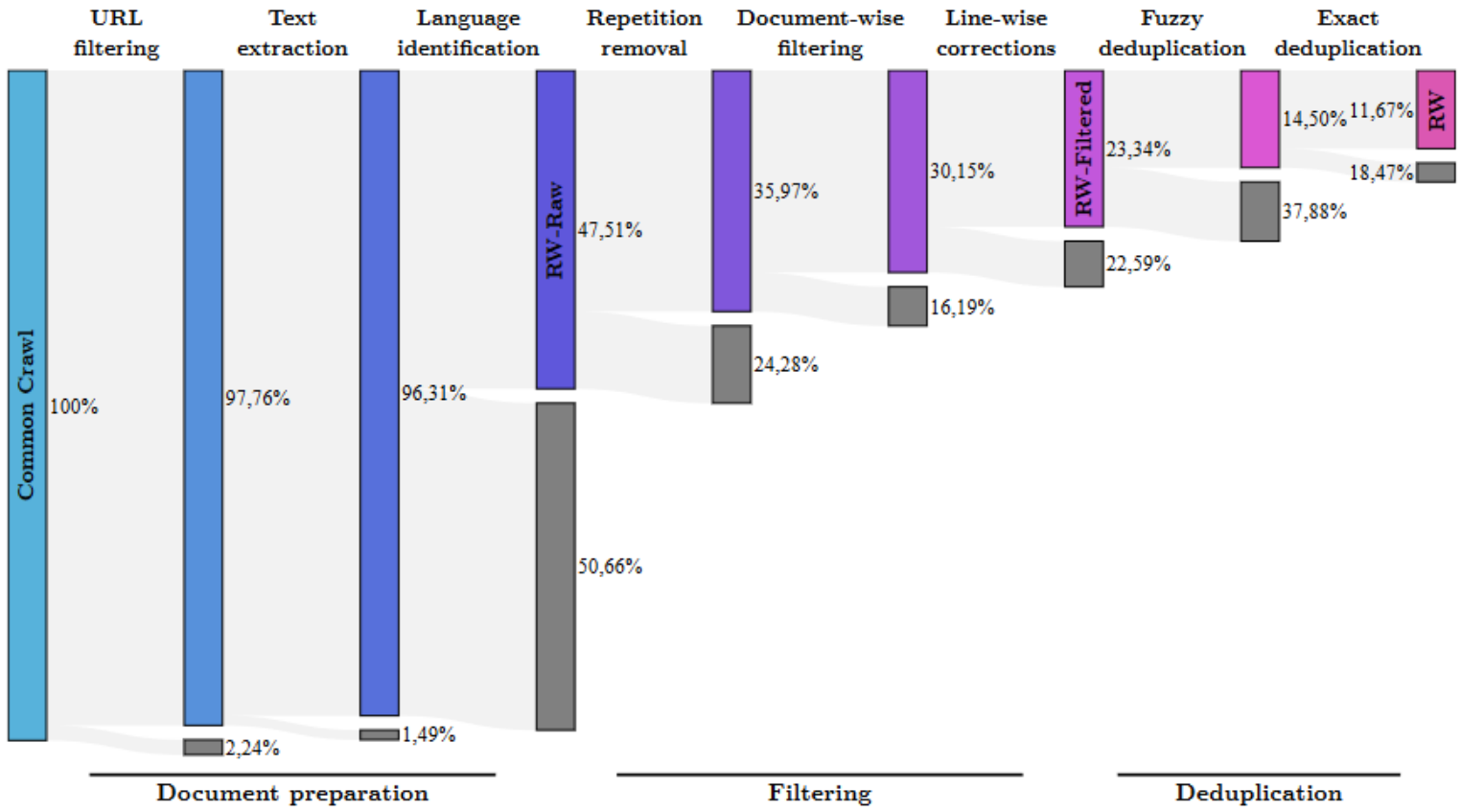


Fig. 27: Subsequent stages of Macrodata Refinement remove nearly 90% of the documents originally in CommonCrawl

B. Data Cleaning

[☞출처 Chatgpt o3; LLM Data Filtering

1) Data Filtering

▪ Removing Noise

- Eliminate irrelevant or noisy data that might impact the model's generalization ability well.
- Noisy samples (e.g., HTML boilerplate, garbled characters, spam) dilute learning signals, slow training, and can lead to incoherent or malicious generations.

- Crawling·Scraping 과정에서 유입된 HTML Tag, 광고, 중복 문단, Code Snipet (코드 조각), 깨진 문자 등을 제거해 LLM이 불필요한 패턴을 학습하지 않도록 함

- Techniques

✓ **Rule-based Preprocessing**: 정규식(Regex)·BeautifulSoup로 HTML/JS 제거
✓ **중복 필터링**: MinHash + Locality Sensitive Hashing(LSH)로 높은 Jaccard 유사도 문서 제거
✓ **언어 검**증: Langdetect / fastText ID로 목표 언어가 아닌 샘플 제거
✓ **Noise Scoring**: OpenAI Recoverability, Pile Deduplication 등 평가 지표로 하위 백분위 Cutoff

❖ LLM Tips

✓ **소스별 품질 태그**를 보존하면 학습 중 가중치 조정(샘플링 temperature) 시 유용
✓ **과도한 필터링**은 도메인 다양성·표현 폭을 손상시킬 수 있으므로 샘플링 비중 조절('mixing rate')로 보완

▪ Handling Outliers

- Detecting and mitigating data that is syntactically valid but statistically extreme (*very rare vocabulary*, *highly repetitive tokens*, *extremely long documents*, etc.).
- Outliers can *skew token distributions*, *hurt convergence*, or *dominate attention in small batches*, *biasing the model toward fringe patterns*.

- Unicode Broken, 터무니없는 길이, 구조적으로 일관성 없는 문단 등을 걸러 모델 수렴 안정화·파괴적 그래디언트 방지

- Techniques

✓ **길이 기반 필터**: token length / character length의 하위 1%·상위 99% 제거
✓ **Perplexity(PPL) 기반 검증**: Pretrained Small LM으로 PPL이 지나치게 높은 Sequence 제외 (값이 높으면 내용 무작위 가능성 ↑)
✓ **Rule-based schema check**: 예) JSONL dataset에서 필수 Key 누락, Field 타입 불일치 샘플 제거

❖ LLM Tips

✓ PPL threshold를 도메인별로 Tuning(예: 코드 vs. 소설)
✓ 길이 분포 리샘플링으로 장·단문 균형 확보

B. Data Cleaning

[☞출처 Chatgpt o3; LLM Data Filtering

1) Data Filtering

▪ Addressing Imbalances

- Ensuring the *corpus* is not *over- or under*-representing specific domains, languages, demographics, or temporal slices.
- Imbalance can cause representational bias, poor performance on low-resource dialects(방언), and ethical concerns.

- 특정 주제·스타일·언어·지리 또는 소수자 표현이 과소대표/과다대표 되는 편향을 완화해 응답 공정성·범용성 확보
- Techniques

✓ **Stratified Sampling & Reweighting**: 메타데이터 (label)별 샘플링 확률 조정
✓ **Data Augmentation**: Paraphrasing, Back-translation, Code-tuning 등으로 희소 클래스 증강
✓ **Filtering Weight Matrix**: 학습 단계에서 rarity score 사용하여 손실 스케일 조정

❖ LLM Tips

✓ Imbalance 해소는 '취약 집합(**performance-critical slice**)'을 기준으로 평가해야 과도한 균일화에 따른 모델 품질 저하를 막을 수 있음
✓ RLHF 파이프라인에서는 Preference Dataset 쪽 비율까지 같이 고려

▪ Text Preprocessing

- *Standardizing* text to a clean, consistent format before tokenization or subword segmentation.
- Uniform preprocessing *improves tokenizer efficiency*, *reduces vocabulary fragmentation*, and *eliminates trivial variability* the model would waste capacity memorizing.

- Tokenizer 친화적 입력으로 변환해 어휘 분해 손실 최소화, 불필요한 variance 축소
- Techniques

✓ **정규화**: NFC Unicode, 숫자·단위 표준화("5km"→"5 km"), 영어 대소문자 규칙화
✓ **Tokenizer 학습 재사용**: 기존 BPE(Byte-Pair Encoding)/SentencePiece vocab에 맞춰 rare token을 split하거나 신규 vocab training
✓ **언어별 처리**: 한국어 자음/모음 분리/합치기, 중국어 공백 삽입, 아랍어 diacritics (발음구별기호) 제거 등

❖ LLM Tips

✓ 오픈소스 Tokenizer 사용 시 normalization step (Byte Fallback, Unicode Nomalization 등)이 모델에 implicit하게 포함되는지 확인 필요
✓ Code dataset에서는 공백·들여쓰기 유지가 질적 성능에 직접적

B. Data Cleaning

[☞출처 Chatgpt o3; LLM Data Filtering

1) Data Filtering

- **Dealing with Ambiguities**

- Detecting and resolving *polysemy*, *abbreviations*, *entity co-reference*, or *conflicting labels* in supervised data.
- Ambiguous or contradictory examples confuse gradient signals, increasing perplexity and harming factual consistency.

- 동음이의어, 공동 출현 개체, 불완전 문맥 등으로 학습 시 의미 충돌을 최소화

- Techniques

✓ **Entity Linking**: 표준 지식그래프(Wikipedia/DBpedia/KoNLU)로 정규화; 링크 confidence 낮은 샘플 제외
✓ **Coreference Resolution**: 'he/she/it' 참조 chain 길이 제한, 실패 문장 제거
✓ **Promptable Disambiguation**: 소형 LM에게 disambiguation tag 부여(예: "<AMBIG class='bank/river'>") 후 메인 LLM 학습

❖ **LLM Tips**

✓ 완전 제거보다는 Annotation-style 보존이 후속 instruction-tuning 시 few-shot 사전 지식으로 활용될 수 있음
✓ RLHF 단계에서 Ambiguous Query를 넣어 Preference ranking으로 실제 품질을 점검하면 효과적

❖ **Practical Workflow Integration**

➢ **Pipeline Order**
- Noise removal → outlier detection → imbalance assessment → preprocessing → ambiguity resolution.
- Early steps shrink data volume, saving later compute.

➢ **Automation + Human-in-the-Loop**
- Combine large-scale heuristics or ML classifiers with targeted human audits (e.g., review all samples flagged as both "violence" and "kids" content).

➢ **Iterative Metrics**
- Track corpus-level stats (token entropy, domain mix) and model-level metrics (validation perplexity per slice) after each cleaning stage to quantify impact.

➢ **Documentation**
- Log filters, thresholds, and rationale. Transparent data cards help reproducibility and downstream trust.

❖ **참고 구현 Library / Toolkit**

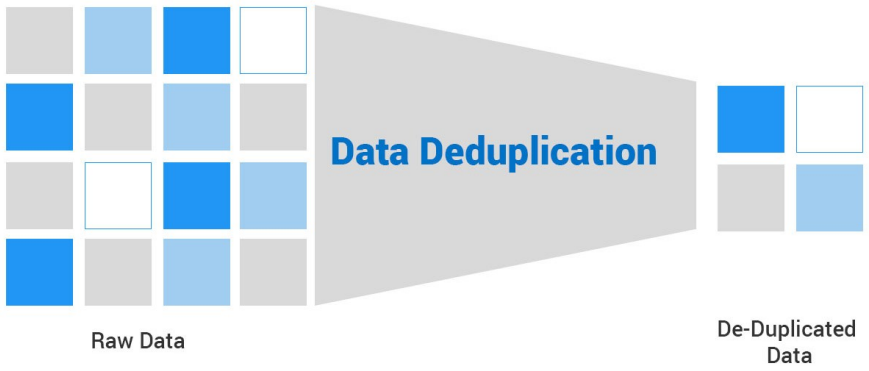
✓ OpenAI tiktoken / SentencePiece / Hugging Face tokenizers : Custom vocab & normalization
✓ LangChain TextCleaner, Cleanlab, Polaris : 노이즈·이상치 탐지 자동화
✓ Detoxify / Perspective API : 욕설·유해 콘텐츠 필터
✓ Apache Spark + UDF / Dask / Ray Data : 대규모 분산 전처리
✓ Weighting by DataComp, Dolma : 품질 점수 기반 재샘플링

B. Data Cleaning

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

2) Deduplication:

- Identify and remove *identical* or *highly similar* data entries (such as documents, paragraphs, or sentences) within the training dataset.
- Due to the massive scale of LLM datasets, the presence of duplicate data is common, making its removal essential.



[☞ 출처; https://www.beyondkey.com/master-data-deduplication]

- Importances of Deduplication
- **Increased Training Efficiency**: Training on the same data repeatedly wastes computational resources (time, cost). Removing duplicates allows the model to learn from a wider variety of data using fewer resources.
- **Reduced Model Bias**: Excessive repetition of certain data can cause the model to overlearn specific features or biases present in that data. Deduplication helps balance the data distribution, mitigating model bias.
- **Improved Model Performance and Generalization**: Duplicate data can lead the model to merely memorize specific patterns or phrases. Exposure to diverse data is crucial for enhancing the model's generalization ability and its capacity to handle novel inputs.
- **Prevention of Data Contamination**: If content present in the evaluation set also exists redundantly in the training set, the model's performance might be overestimated. Deduplication alleviates this data contamination issue.

B. Data Cleaning

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

2) Deduplication:

- Technical Methods

① **Exact-match Deduplication (완전중복제거)**

✓ **Hashing**: Compute fast hashes (SHA-1, xxHash) per document; identical hashes ⇒ duplicates.

✓ **Normalization**: Canonicalize whitespace, casing, markup to tighten equality criteria.

② **Near Deduplication (유사중복제거)**

✓ **Shingling**: Text is broken down into overlapping sequences of characters or words (k-shingles) to form sets.

✓ **MinHash**: To quickly estimate the similarity between ***shingle sets***, compact signatures summarizing the characteristics of each set are generated. This technique effectively *reduces dimensionality* while preserving metrics like *Jaccard similarity*.

✓ **Locality-Sensitive Hashing (LSH)**: Uses *MinHash signatures* to hash *similar items* into *the same buckets* with high probability. Instead of comparing all pairs of data, similarity calculations (e.g., Jaccard similarity) are performed only on candidate pairs within the same bucket, efficiently finding near-duplicates.

③ **Scaling techniques**

✓ **Blocking / partitioning**: Bucket by length, hash prefixes, language, domain to reduce pairwise comparisons.

✓ **Streaming dedup**: Online filtering via (hash → Bloom filter) during crawl-train pipelines.

✓ **Distributed execution**: MinHash + LSH on Spark, Ray, or Dask for 100 TB-scale datasets.

✓ **SimHash**: One 64/128-bit signature; Hamming distance ≤ n marks a duplicate pair.

✓ **Sentence/paragraph embeddings**: Compute SBERT/LaBSE embeddings; cluster vectors with cosine sim > τ (e.g., 0.95) and keep a centroid sample.

✓ **n-gram overlap metrics**: Use Jaccard or Sørensen-Dice; treat pairs above threshold θ (e.g., 0.8) as duplicates

B. Data Cleaning

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

2) Deduplication:

- Quality Control

- **Recall–precision trade-off**: Tune τ/θ *thresholds* against a human-labeled validation set to maximize F1.
- **Domain-specific rules**: For code corpora, strip comments/whitespace and compare token streams.

- Popular OSS toolchains

- **text-dedup**, Google's C4 **dedupe.py**, Hugging Face **datasets.filter_duplication**, Cohere's **near_dedup** scripts, etc.

- Open challenges

1) paraphrase 수준의 중복 감지 한계 (semantic dedup)
2) 다국어 혼합 코퍼스의 인코딩·토큰화 차이
3) GPT-작성 텍스트가 훈련셋에 반복 출현할 때 생기는 "self-feedback" 제거 등.

❖ LLM Tips

✓ **1-Pass Filtering → 2-Pass Review**: 먼저 해시 기반으로 exact duplicates를 제거한 뒤, 남은 후보에 대해 MinHash/SimHash를 적용하면 계산량을 대폭 줄일 수 있음

✓ **Hybrid Thresholding**: Length-aware τ 설정(긴 문서는 낮은 임계, 짧은 문서는 높은 임계)을 사용하면 짧은 boiler-plate를 과도 필터링하지 않고 긴 기사 중복을 포착할 수 있음.

✓ **Metadata Leverage**: URL slug, timestamps, and site IDs can serve as cheap blocking keys before textual similarity scoring.

How LLMS Are Built

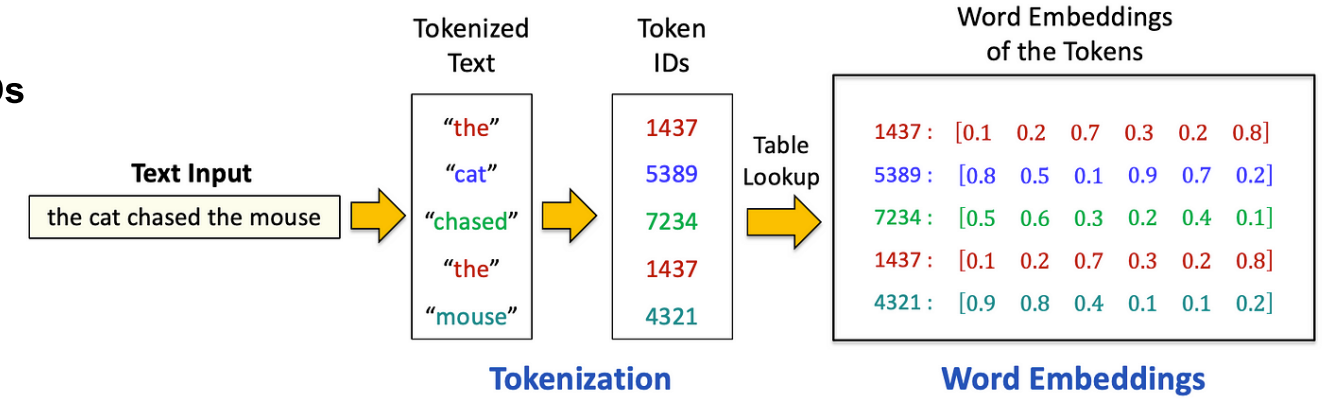
C. Tokenizations

- The step that converts raw text into a sequence of **vocabulary IDs** so the model can ingest **fixed-size integer tensors**.
- Purpose

- **Vocabulary curation (어휘한정)** – Reduces a virtually infinite string space into a manageable set of ≈10k ~ 100k symbols.
- **OOV(out-of-vocabulary) mitigation** – splits unseen words into sub-words/bytes.
- **Length accounting** – token count determines how much context (e.g., 32 k tokens) fits in memory.

- Units: Token은 공백 기준 단어(Word), 형태소, 더 작은 단위인 서브워드(Subword, 예: "Tokenization" -> "Token", "ization"), 또는 개별 문자(Character)가 될 수 있음. 현대 LLM에서는 주로 Subword 방식이 사용됨.
- Vocabulary(어휘집): Tokenizer는 사전에 구축된 '어휘집'을 가짐. 이 어휘집은 가능한 모든 token과 각 token에 해당하는 고유한 정수 ID를 매핑한 목록임.
- OOV(Out-of-Vocabulary): 어휘집에 없는 단어를 처리하는 능력도 중요하며, Subword 방식은 OOV 문제를 효과적으로 완화함. 모르는 단어도 학습된 Subword나 문자로 분해하여 표현할 수 있기 때문임

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication



[출처; https://medium.com/@lmpo/tokenization-and-word-embeddings-the-building-blocks-of-advanced-nlp-c203b78bfd07]

Pipeline :

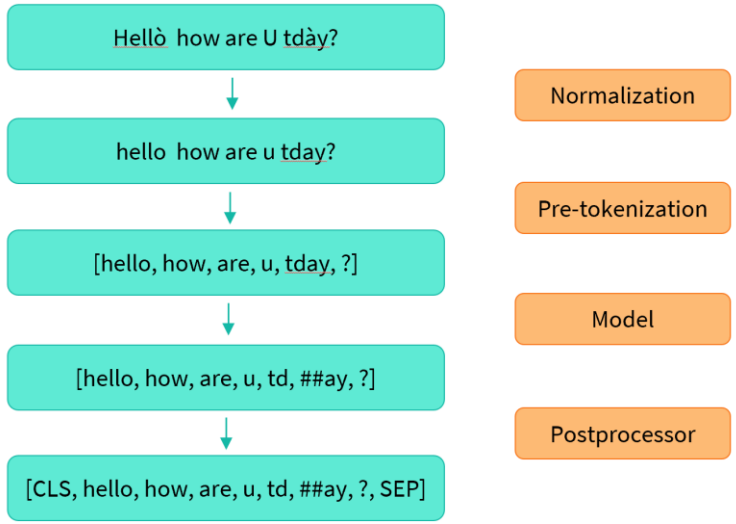
Text →

Normalizer →

Tokenizer →

Model →

Detokenizer.



C. Tokenizations

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

- Modern LLMs predominantly use **subword tokenization** algorithms that generate *vocabularies* and *segmentation rules* based on *statistical methods* and *data-driven learning*. Prominent examples include BPE, WordPiece, and SentencePiece.

Byte Pair Encoding (BPE)

- Iteratively finds the most frequent pair of adjacent bytes (or characters) in the data and merges them into a single new symbol (token)

- **[Tokenization Process]** Given new text, apply *the learned merge rules* greedily, starting from the most frequent pair, to segment it into a sequence of subwords.
- **[Characteristics]** Relatively simple to implement and generates subwords reflecting the statistical properties of the data. It has been commonly used in **GPT-series** models.
- [Implementations] **fastBPE**, Hugging **Face tokenizers** (“BPE”), OpenAI **tiktoken**, etc.

[참고] https://huggingface.co/learn/llm-course/chapter6/5?fw=pt

[Process]

1) Initialize the vocabulary with individual characters (or bytes).
2) Find the most frequently occurring adjacent pair of tokens in the training data.
3) Merge this pair into a new single token and add it to the vocabulary.
4) Repeat steps 2 and 3 for a predetermined number of merges or until the desired vocabulary size is reached.

BPE token learner algorithm

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

[출처; https://medium.com/@RobuRishabh/introduction-to-natural-language-processing-byte-pair-encoding-bpe-and-natural-language-toolkit-414c83f85b71

C. Tokenizations

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

● WordPiece Encoding

- A subword tokenization method similar to BPE, but with a different merging criterion.

→ WordPiece **merges the pair that maximizes the likelihood of the training data given the current vocabulary**.

- **[Tokenization Process]** Attempts to segment input words greedily into *the longest possible known subwords* from the vocabulary.
- **[Characteristics]** Tends to generate statistically more meaningful subwords by merging based on *likelihood*. It is primarily used in models developed by **Google**, such as BERT and RoBERTa.

- [토큰 규칙] 첫 subword는 원형, 후속 subword는 prefix "##" 부착(예: "playing" → "play ##ing"), Prefix 조건 덕분에 Decoding이 단순
- [장점] 고빈도 서브워드 vs Coverage 균형 → 희귀 형태·어근/접미 변화에 유연
- [한계] 공백을 사전 문자 토큰 "[CLS]"/"[SEP]" 뒤에 두거나 별도 처리해야 함

[Process]

1) Initialize the vocabulary with base characters.
2) Build a language model based on the training data and the current vocabulary.
3) Iteratively find the pair of existing tokens that, when merged, increases the likelihood of the training data the most. Mathematically, this is often approximated by maximizing

$$\frac{\text{count(pair)}}{\text{count(first\_token)} * \text{count(second\_token)}}.$$

4) Merge the selected pair into a new token and add it to the vocabulary.
5) Repeat steps 3 and 4 until the target vocabulary size is reached.

[참고] https://huggingface.co/learn/llm-course/chapter6/6?fw=pt

C. Tokenizations

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

- **SentencePiece Encoding**

- A ***language-independent tokenizer library and algorithm*** developed by Google. It is designed to **operate directly on raw text without requiring pre-tokenization** (like splitting by spaces).

- **[Characteristics]**

✓ **Language-Agnostic** (언어중립적): Easily applicable to languages that don't use spaces to delimit words (e.g., Korean, Japanese, Chinese) without extra preprocessing.

✓ **Reversibility**: Allows lossless reconstruction of the original text from the tokenized sequence.

✓ **Integrated Normalization**: Includes built-in text normalization options, such as Unicode normalization (e.g., NFKC).

✓ Widely used in various models like **T5**/**mT5**, **LLaMA family**, **XLNet, Gemma**, many others.

[Process]

1) **Algorithm Choice**: SentencePiece can internally use either **BPE** or a **Unigram language model**-based **tokenization** algorithm, specified by the user during vocabulary training.
2) **Space Handling**: Treats space as a normal character. It's often replaced by a special meta-symbol like (_), allowing perfect reconstruction (reversibility) of the original text (including spaces) when detokenizing.
3) **Unigram Mode**: (Another primary mode of SentencePiece)
   - Starts with a large vocabulary of candidate subwords (e.g., all substrings).
   - It then iteratively removes tokens that contribute least to minimizing the overall loss (*negative log-likelihood*) of the training data, based on a unigram language model assumption (each token occurs independently), until the desired vocabulary size is reached.
   - During tokenization, it finds the segmentation of the input text into tokens that maximizes the probability according to the trained unigram model, often using the *Viterbi* algorithm.

[참고] https://huggingface.co/learn/llm-course/chapter6/7?fw=pt

C. Tokenizations

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

● SentencePiece Encoding

- The unigram LM method, in contrast to the bottom-up construction process of BPE, begins with a superset of the final vocabulary, pruning it to the desired size.
- Unigram LM tokenization takes the vocabulary **V** and unigram LM parameters $\boldsymbol{\theta}$ and performs Viterbi inference to decode the segmentation with maximum likelihood under θ.

Algorithm 2 Unigram LM (Kudo, 2018)

Input: set of strings D, target vocab size k
procedure UNIGRAMLM(D, k)
  $V \leftarrow$ all substrings occurring more than once in D (not crossing words)
  while $|V| > k$ **do** ▷ Prune tokens
    Fit unigram LM θ to D
    for $t \in V$ **do** ▷ Estimate token 'loss'
      $L_t \leftarrow p_\theta(D) - p_{\theta'}(D)$
      where θ' is the LM without token t
    end for
    Remove $\min(|V| - k, \lfloor \alpha |V| \rfloor)$ of the tokens t with highest L_t from V,
    where $\alpha \in [0, 1]$ is a hyperparameter
  end while
  Fit final unigram LM θ to D
  return V, θ
end procedure

☞ Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates https://arxiv.org/abs/1804.10959
☞ Byte Pair Encoding is Suboptimal for Language Model Pretraining https://arxiv.org/abs/2004.03720

[Process]

1) **Algorithm Choice**: SentencePiece can internally use either **BPE** or a **Unigram language model**-based **tokenization** algorithm, specified by the user during vocabulary training.
2) **Space Handling**: Treats space as a normal character. It's often replaced by a special meta-symbol like (_), allowing perfect reconstruction (reversibility) of the original text (including spaces) when detokenizing.
3) **Unigram Mode**: (Another primary mode of SentencePiece)
   - Starts with a large vocabulary of candidate subwords (e.g., all substrings).
   - It then iteratively removes tokens that contribute least to minimizing the overall loss (*negative log-likelihood*) of the training data, based on a unigram language model assumption (each token occurs independently), until the desired vocabulary size is reached.
   - During tokenization, it finds the segmentation of the input text into tokens that maximizes the probability according to the trained unigram model, often using the *Viterbi* algorithm.

☞ https://github.com/google/sentencepiece
☞ Kudo, T., & Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. arXiv preprint arXiv:1808.06226.

D. Positional Encoding

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

- Technique used to **inject information about the position of tokens** within an input sequence into Transformer-based models like LLMs.
- Since the core **self-attention** mechanism is **permutation-invariant** (doesn't inherently consider order), **positional encoding** provides the necessary sequential context, allowing the model to **understand word order** (e.g., distinguish "cat eats fish" from "fish eats cat").
- This positional information, typically represented as vectors, is usually added to or combined with the token's input embeddings.

➢ 초기 모델들은 Absolute positional embeddings을 주로 사용했지만, 시퀀스 길이 일반화와 언어적 특성 반영의 중요성이 부각되면서 다양한 형태의 Relative positional encoding (임베딩 추가, RoPE, 편향 추가 등)이 활발히 연구되고 적용되고 있음

❖ **LLM Tips**

✓ **디코더-전용 LLM**(예: GPT-계열, LLaMA 등)에선 **RoPE** 또는 **ALiBi**가 사실상 표준

✓ **인코더-디코더**(T5류)나 **병렬 학습**이 중요한 환경은 **Relative Bias**가 널리 사용

✓ **컨텍스트 윈도우를 크게 늘릴 때**는 **RoPE + 주파수 스케일링(NTK/YaRN)**이나 **ALiBi**가 안전한 선택

D. Positional Encoding

- **Absolute Positional Embeddings**

  - Assigns a unique embedding vector to each absolute position index (e.g., 0, 1, 2, ...) in the sequence.

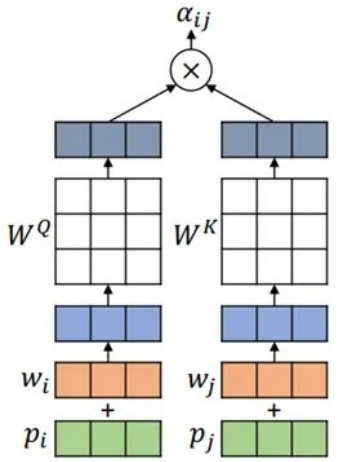
  - **Learned Embeddings**: An embedding matrix ($P \in \mathbb{R}^{L_{max} \times d_{model}}$) is defined, where L_{max} is the maximum sequence length and d_{model} is the model dimension. The vector P_i corresponding to position index i is learned along with other model parameters and added to the input embedding of the token at position i. (e.g., BERT, GPT-1/2/3)

  - **Fixed Sinusoidal Embeddings**: (in Transformer), Uses predefined sine and cosine functions of varying frequencies to generate unique vectors for each position. For instance, the $2i$-th and $(2i+1)$ -th dimension values for position pos are calculated as

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right), PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right).$$

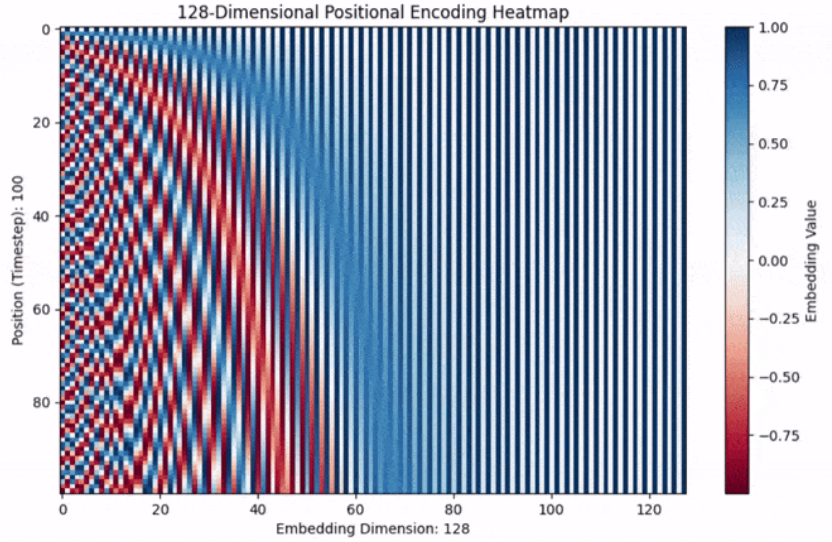
    A theoretical advantage is its potential to generate positional values for sequences longer than those seen during training.

  - **Pros & Cons**: Simple to implement. However, they may not generalize well to sequences longer than L_{max} (especially learned ones). They don't explicitly model relative distances between tokens, which can be crucial for capturing linguistic structures.



Absolute Positional Embeddings

G. Ke, et al, “Rethinking positional encoding in language pre-training,” arXiv, 2020.



PE; timestep is changed from 100 to 500.

You can see that consistent position information is maintained.

D. Positional Encoding

- **Relative Positional Embeddings** Shaw et al. (Google) Self-Attention with Relative Position Representations, arxiv, 2018

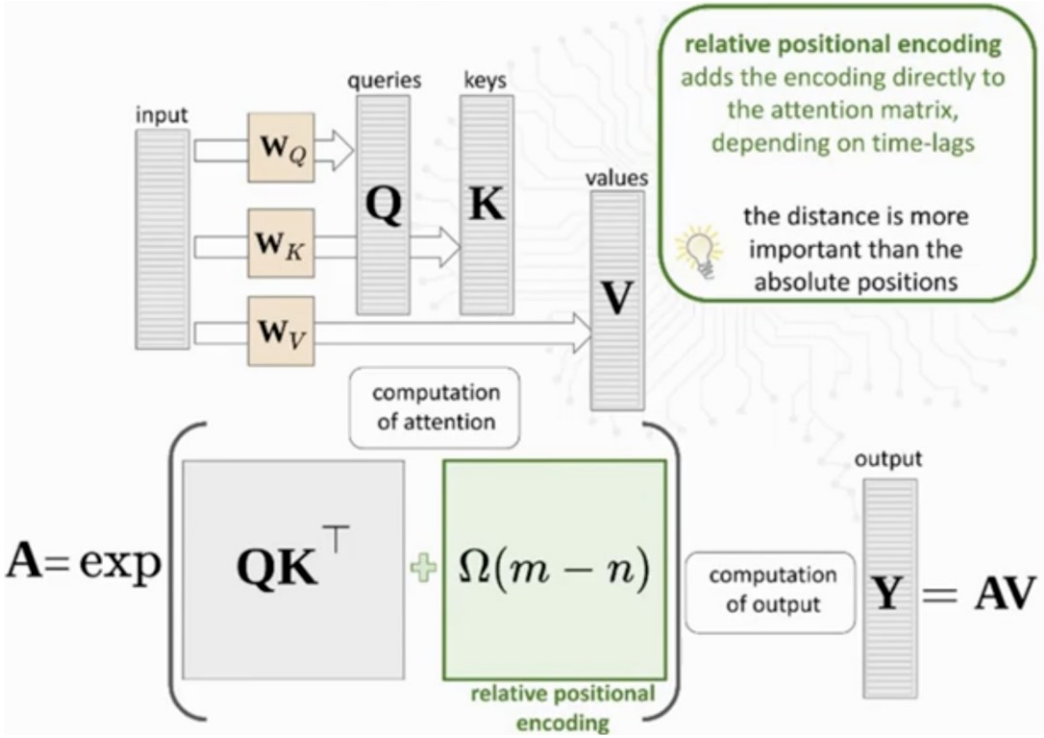
  - Models the **relative distance (offset) between pairs of tokens involved in the attention mechanism**. When a query token at position i attends to a key token at position j, an embedding corresponding to the relative distance $(i-j)$ is incorporated into the attention calculation.

  - Method : When computing the attention score between query q_i and key k_j, a **learnable embedding vector** $\boldsymbol{R_{i-j}}$ representing the relative distance $(i-j)$ is added. This embedding can modify the key or value vectors. e.g., ***transforming the attention score*** calculation like

$$e_{ij} = \frac{q_i(k_j+R_{i-j})^T}{\sqrt{d_k}}$$

Often, the relative distance $(i-j)$ is clipped to a maximum range $[-k, k]$ to limit the number of parameters.

- **Pros & Cons**: Explicitly models relative relationships, leading to better generalization across sequence lengths and aligning better with linguistic intuition. Implementation can be slightly more complex than absolute methods.



- First, relative positional information is supplied to the model as an additional component to the keys

$$e_{ij} = \frac{x_i W^Q \left(x_j W^K + a_{ij}^K\right)^T}{\sqrt{d_z}}$$

- Here α is an edge representation for the inputs x_i and x_j. The softmax operation remains unchanged from vanilla self-attention. Then relative positional information is supplied again as a sub-component of the values matrix:

$$z_i = \sum_{j=1}^{n} \alpha_{ij}\left(x_j W^V + a_{ij}^V\right)$$

- Instead of simply combining semantic embeddings with absolute positional ones, relative positional information is added to keys and values on the fly during attention calculation.

D. Positional Encoding

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

- **Rotary Position Embeddings (RoPE)** Su et al. RoFormer: Enhanced Transformer with Rotary Position Embedding, 2021

  - Incorporates relative position information by **rotating** the ***query*** and ***key*** vectors based on their **absolute positions** before the attention score computation.

① Divide the query q and key k vectors into 2D subspaces.

② For a token at position m, apply a rotation matrix R_m to its query q_m and key k_m vectors within these 2D subspaces. The rotation angle depends on the position m and the dimension index within the vector.

③ The core insight is that the dot product between rotated vectors, $< \boldsymbol{R_m q_m, R_n k_n} >$ (related to the attention score), is designed to depend only on the original vectors q_m, k_n, and the relative position $m - n$ (utilizing the property $R_m^T R_n = R_{m-n}$).

- **Pros & Cons**: Effectively injects relative position information without adding positional values directly to input embeddings. Known for good performance and length extrapolation capabilities. (e.g., PaLM, Llama, GPT-NeoX). It is mathematically more involved than other methods.

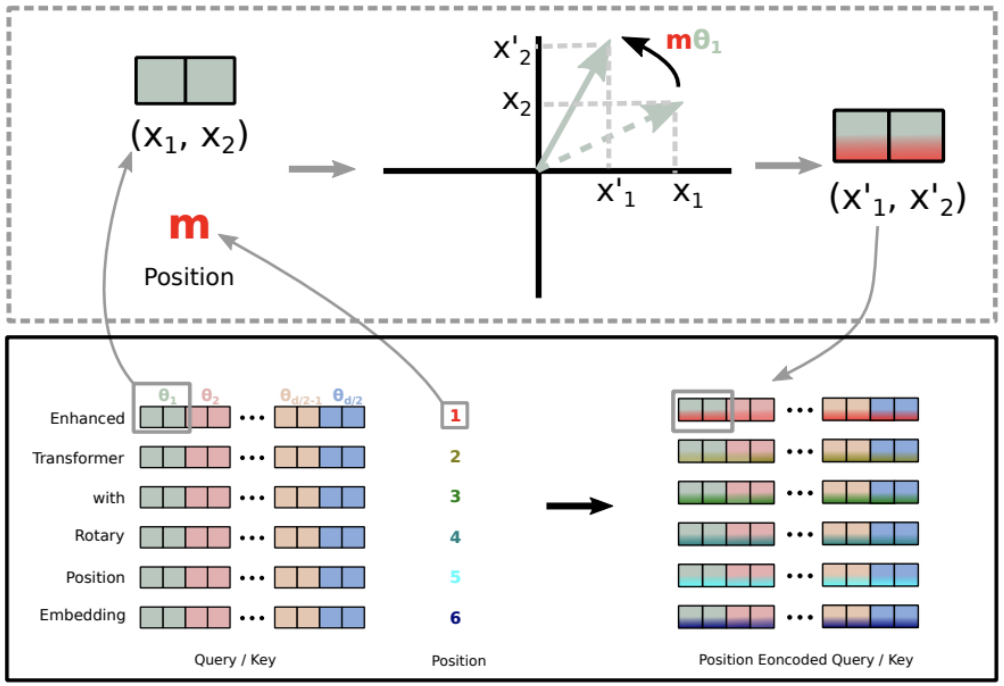


Figure 1: Implementation of Rotary Position Embedding(RoPE).

D. Positional Encoding

[☞출처 Chatgpt o3 & Gemini 2.5 pro; LLM Deduplication

- **Rotary Position Embeddings (RoPE)** Su et al. RoFormer: Enhanced Transformer with Rotary Position Embedding, 2021

[General form]

In order to generalize our results in 2D to any $\boldsymbol{x}_i \in \mathbb{R}^d$ where d is even, we divide the d-dimension space into $d/2$ sub-spaces and combine them in the merit of the linearity of the inner product, turning $f_{\{q,k\}}$ into:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m \tag{14}$$

where

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

is the rotary matrix with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$. A graphic illustration of RoPE is shown in Figure **(1)**. Applying our RoPE to self-attention in Equation (2), we obtain:

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_q \boldsymbol{x}_m)^\intercal (\boldsymbol{R}^d_{\Theta,n} \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}^\intercal \boldsymbol{W}_q R^d_{\Theta,n-m} \boldsymbol{W}_k \boldsymbol{x}_n \tag{16}$$

where $\boldsymbol{R}^d_{\Theta,n-m} = (\boldsymbol{R}^d_{\Theta,m})^\intercal \boldsymbol{R}^d_{\Theta,n}$. Note that $\boldsymbol{R}^d_\Theta$ is an orthogonal matrix, which ensures stability during the process of encoding position information. In addition, due to the sparsity of R^d_Θ, applying matrix multiplication directly as in Equation (16) is not computationally efficient; we provide another realization in theoretical explanation.

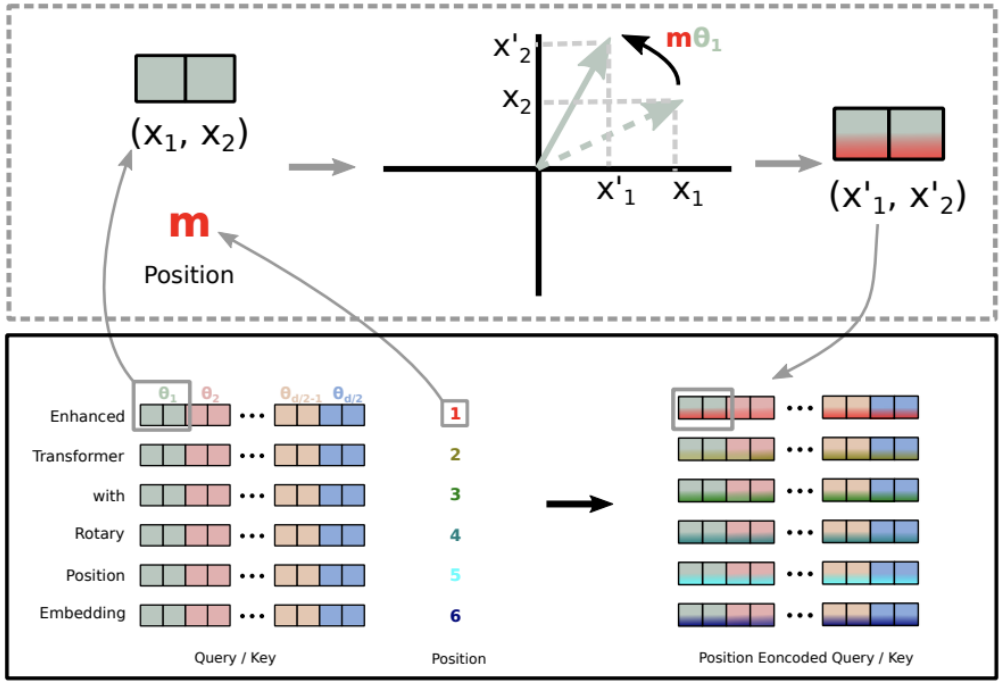


Figure 1: Implementation of Rotary Position Embedding(RoPE).

D. Positional Encoding

● T5 Bias

C. Raffel, Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, arxiv, 2019

- A simpler form of relative positional encoding, used in models like **T5.**
- Instead of incorporating relative embeddings into key/value interactions, **a learnable scalar bias based on the relative distance between tokens** is added directly to the **attention logits** (**pre-softmax attention scores**).

① Calculate the relative distance $i - j$ between the query position i and key position j.

② Map this relative distance to a corresponding learnable scalar bias value b_{i-j}. Often, distances are grouped into buckets to manage the number of bias parameters efficiently.

③ Add this bias to the attention logits: $logit(q_i, k_j) = \frac{q_i k_j^T}{\sqrt{d_k}} + b_{i-j}$

- **Pros & Cons**: Simple, computationally efficient, and effective at capturing local biases. Demonstrated good performance in models like T5. Its capacity to represent complex long-range relative dependencies might be somewhat limited compared to other relative methods.

● Alibi (Attention with Linear Biases)

O. Press, et al, “Train short, test long: Attention with linear biases enables input length extrapolation,” arXiv, 2021 (ICLR2022)

- 트랜스포머 모델이 훈련 시 사용된 시퀀스 길이보다 훨씬 긴 시퀀스를 추론 시에도 효과적으로 처리할 수 있도록, 즉 길이 외삽(**Length Extrapolation**) 성능을 높이기 위해 고안된 위치 인코딩 방식

[Mechanism]

- **No Positional Embeddings**: Completely removes explicit positional embeddings that are typically added to the token embeddings at the input layer of the Transformer.
- **Bias Added to Attention Scores**: Instead, Introduces a bias term directly into the attention score calculation, right before the softmax function is applied.
- **Fixed Linear Bias**: This bias is not a learned parameter. It's a static, ***predefined penalty*** that decreases linearly with the distance between the query and key tokens.

D. Positional Encoding

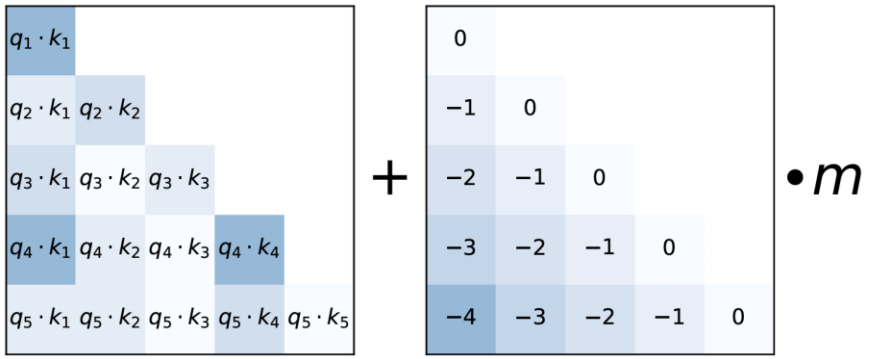
- **Alibi (Attention with Linear Biases)**

[Method]

- When calculating the attention score between a query vector q_i (at position i) and a key vector k_j (at position j) within a specific attention head h, ALiBi modifies the standard dot-product attention as follows:.

$$AttentionScore(q_i, k_j) = \frac{q_i k_j^T}{\sqrt{d_k}} + m_h \cdot (j - i)$$

- $\frac{q_i k_j^T}{\sqrt{d_k}}$: This is the standard scaled dot-product attention score.
- $(j - i)$: This represents the **relative distance** between the key token's position (j) and the query token's position (i).
- m_h : This is a **head-specific scalar slope** that is **fixed** (predetermined before training, not learned). It's a hyperparameter of the model architecture.



When computing attention scores for each head, our linearly biased attention method, ALiBi, adds a constant bias (right) to each attention score (q_i, k_j,left). As in the unmodified attention sublayer, the softmax function is then applied to these scores, and the rest of the computation is unmodified. **m is a head-specific scalar** that is set and not learned throughout training.

❖ **기울기 m_h 결정** : m_h 값은 각 Attention Head마다 다르게 설정됨.

논문에서는 N개의 Head가 있을 때, Head h(h=1,...,N)의 기울기를 $m_h = -1/2^k$ 형태로 설정. 여기서 k는 h값에 따라 결정되며 (예: k=h 또는 [h/c] 등), 결과적으로 일부 헤드는 작은 기울기(약한 거리 페널티, 더 넓은 범위 주의 집중 가능)를 갖고, 다른 헤드는 큰 기울기(강한 거리 페널티, 지역적 정보에 집중)를 갖도록 설계됨. 이 값들은 hyperparameter임

[Why ALiBi Enables Length Extrapolation (Intuition)]

- Inductive Bias Towards Locality (지역편향성 도입) : 거리가 먼 토큰 간의 어텐션 스코어에 직접적인 페널티를 부여
- Independence from Maximum Length : 편향 값이 오직 상대 거리에만 의존하고 미리 정해진 상수 m_h 를 사용. 훈련 시 사용된 최대 시퀀스 길이에 대한 정보가 필요 없음
- Adaptive Attention Patterns : 각 헤드마다 다른 기울기 m_h 가짐 (어떤 헤드는 지역 정보에 집중하거나 약한 패널티로 전역 정보에 집중)

Mastering LLM Techniques: Text Data Processing

https://developer.nvidia.com/blog/mastering-llm-techniques-data-preprocessing/

E. Model Pre-training

- Pre-training : Very first step in LLM training pipeline,
  - Helps LLMs to acquire fundamental language understanding capabilities
  - During pre-training, LLM is trained on a massive amount of (usually) *unlabeled texts*, usually in a *self-supervised manner*.

Autoregressive Language Modeling (ALM)

- Predicting the next token in a sequence based on the preceding tokens.

[Basic Principles]

- Given a sequence of n tokens $x_1, \cdots, x_n$, the model tries to predict next token x_{n+1} in an auto-regressive fashion. One popular loss function in this case is the log-likelihood of predicted tokens

$$\mathscr{L}_{ALM}(x) = \sum_{i=1}^{N} p(x_{i+n}|x_i, ..., x_{i+n-1})$$

[Representative Models]

- GPT series (GPT-1, GPT-2, GPT-3, GPT-4), PaLM, LLaMA, and BLOOM are prominent examples of autoregressive language models. They have demonstrated outstanding performance across a wide range of natural language processing tasks, including text generation, summarization, translation, and question answering.

[Implementation in LLMs (Primarily using Transformer Decoders):]

- Modern LLMs typically implement autoregressive language modeling using the decoder part of the Transformer architecture.
- ***Input and Output***: The model takes a sequence of preceding tokens $(x_1, \cdots, x_{t-1})$ as input and outputs **a probability distribution over the entire vocabulary for what the next token x_t might be**.
- ***Causal Masking / Look-ahead Mask***: The self-attention mechanism in Transformer decoders employs a ***'causal mask***.' This mask prevents the model from "seeing" or attending to subsequent tokens (i.e., x_{t+1}, x_{t+2}, ...) when predicting the current token x_t. This enforces the autoregressive property by preventing information leakage from the future. Each token can only attend to itself and preceding tokens.
- ***Training Objective***: During pre-training, the model is trained on a large text corpus to accurately predict the next token. The primary loss function used is ***Cross-Entropy Loss***, which minimizes the difference between the model's predicted probability distribution for the next token and the actual next token in the training data.

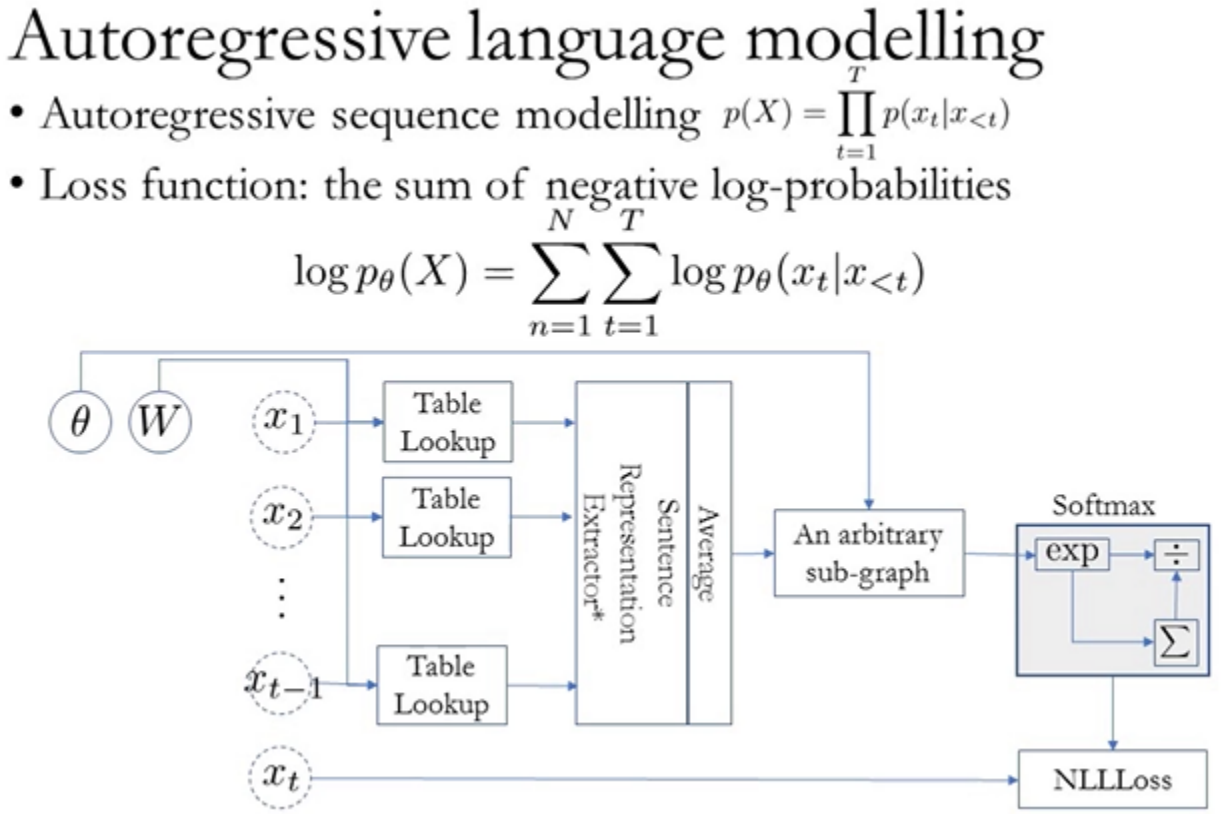
E. Model Pre-training

- **Autoregressive Language Modeling (ALM)**

[Text Generation (Inference)]

- Autoregressive models are highly effective for text generation.

1) An initial prompt or a special start-of-sequence token is provided as input.
2) The model processes this input and generates a probability distribution for the next token. A token is then selected from this distribution (e.g., through sampling or by picking the highest probability token – *greedy decoding*).
3) The selected token is appended to the input sequence.
4) This process is repeated, with the model generating one token at a time, until a desired length is reached or an end-of-sequence (EOS) token is generated.



[Simple Text-Based Diagram Example]

```
[Input Sequence]          [LLM (Autoregressive)]      [Predicted Next Token Distribution & Choice]
----------------------    ------------------------    ------------------------------------------
[<BOS>]                   -->  MODEL (predicting...)  -->   P("The"| <BOS>)                  --> "The“
[<BOS>, "The"]            -->  MODEL (predicting...)  -->   P("quick"| <BOS>,"The")         --> "quick“
[<BOS>, "The", "quick"]   -->  MODEL (predicting...)  -->   P("brown"| <BOS>,"The","quick") --> "brown“
...and so on...
```

E. Model Pre-training

Masked Language Modeling (MLM)

- MLM works by masking some tokens in an input sentence and then training the model to predict these masked tokens using their *surrounding* ***bidirectional context***.
- If we denote the *masked/corrupted samples* in the sequence x, as $\tilde{x}$, then the training objective can be written as:

$$\mathscr{L}_{MLM}(x) = \sum_{i=1}^{N} p(\tilde{x}|x\backslash\tilde{x})$$

[Concept & Core Idea]

- Randomly select some tokens from an input text sequence, replace them with a special [MASK] token or transform them in other ways, and then train the model to predict the original identity of these tokens.
- Crucially, the model doesn't just process information from left to right; it utilizes **both the left and right context of the masked word to make its prediction**. This allows the model to learn **deep bidirectional** relationships between words and their context.

[Training Process]

➢ **Input Preparation (Masking Strategy)**:

1) A certain percentage of tokens (e.g., **15%**) are randomly selected from the input sentence.
2) These selected tokens are modified according to the following rules (as in **BERT**):
   - About **80%** of them are replaced with a special [MASK] token. (e.g., "I went to the [MASK] today.")
   - About **10%** are replaced with another random token from the vocabulary. (e.g., "I went to the apple today." - if the original word was "store"). This helps the model learn to distinguish correct words from contextually incorrect ones.
   - About **10%** are kept as the original token. (e.g., "I went to the store today."). This biases the model towards the true observed word and aids in learning its representation..

➢ **Model Training**

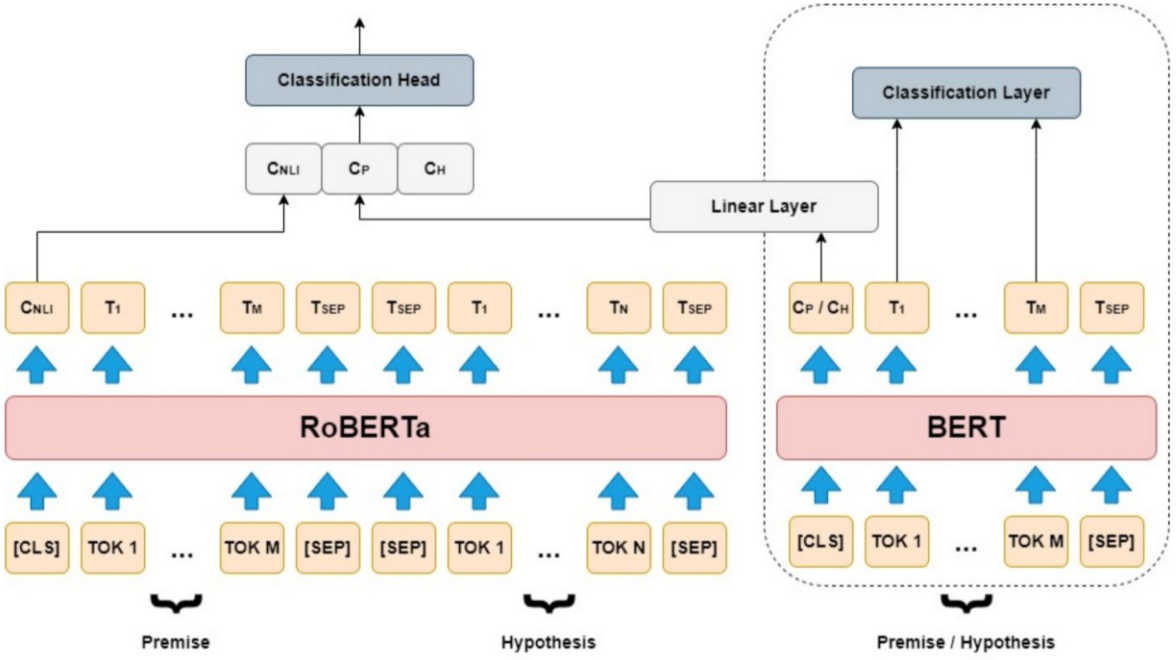
- Masked input sequence is fed into the model (typically the Encoder part of a Transformer).
- The objective is to predict the original tokens at the [MASK]ed positions.
- At the output layer, for each masked position, the model computes a probability distribution over the entire vocabulary. The model is trained to minimize the cross-entropy loss between this predicted distribution and the actual original token. The loss is calculated only for the masked tokens.

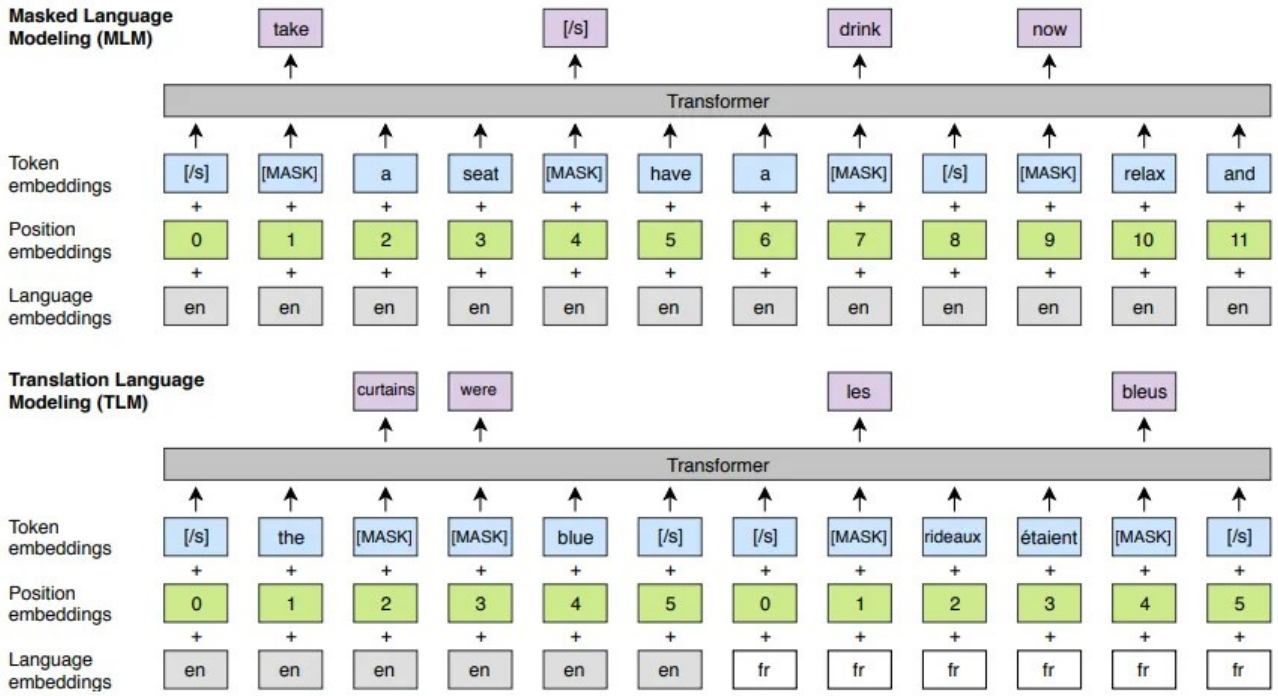
E. Model Pre-training

- **Masked Language Modeling (MLM)**

[Representative Models]

- **BERT** (Bidirectional Encoder Representations from Transformers): The quintessential MLM-based model.
- **RoBERTa** (Robustly Optimized BERT Pretraining Approach): Improves upon BERT by optimizing its training procedure and using more data.
- **ALBERT** (A Lite BERT for Self-supervised Learning of Language Representations): A parameter-efficient version of BERT.





MLM objective uses continuous text streams rather than sentence pairs. TLM extends MLM to parallel sentence pairs, encouraging alignment of English and French representations during masked word prediction.

E. Model Pre-training

Mixture of Experts (MoE)

[Core Idea]

- Instead of a single, dense network processing every token, an MoE layer uses multiple "expert" sub-networks. For each incoming token, a smaller "**gating network**" or "**router**" dynamically chooses which few experts (often just one or two out of many) should process that specific token. The other experts remain inactive for that token.

[Key Components]

- **Experts**: Typically smaller neural networks. In Transformer-based LLMs, the Feed-Forward Network (FFN) layers within a Transformer block are often replaced by multiple, parallel FFN "experts." Each expert can, in theory, specialize in different types of *data*, *patterns*, or *tasks*.
- **Gating Network (Router)**: A small, learnable neural network that takes the token representation as input and outputs a set of scores or probabilities indicating which expert(s) are most suitable for processing that token. Commonly, a "**Top-K**" routing strategy is used, where only the K experts with the highest scores are selected (K is often 1 or 2).

[How it Works During Pre-training]

1) When a token's representation reaches an MoE layer, the gating network calculates which expert(s) to send it to.
2) Only the selected expert(s) are activated and perform computations for that token. This is called ***sparse activation***.
3) The outputs of the activated expert(s) are then combined, usually via a weighted sum based on *the gating network's scores*, to produce the final output of the MoE layer.
4) The overall pre-training *objective* (e.g., autoregressive language modeling or masked language modeling) remains the same, but the model learns to utilize these sparse expert layers.

[Advantages in Pre-training]

- **Massive Parameter Scalability**: Allows models to have ***trillions*** of parameters in total, significantly increasing their capacity.
- **Computational Efficiency (per token)**: The FLOPs per token are roughly equivalent to a much smaller dense model because only a fraction of the total parameters are used for any given token.
- **Potential for Specialization**: Experts can learn specialized functions, potentially leading to better performance on diverse tasks.

E. Model Pre-training

- **Mixture of Experts (MoE)**

[Challenges in Pre-training]

- **Load Balancing**: Ensuring that all experts receive a roughly equal amount of training examples. If some experts are underutilized, they don't learn effectively. Auxiliary loss functions are often used to encourage balanced routing.
- **Training Stability**: MoE models can sometimes be more challenging to train stably than dense models.
- **Communication Overhead**: If experts are distributed across different devices, the communication for routing and aggregating results can be a bottleneck.
- **Memory Requirements**: All expert parameters still need to be stored, which can be substantial, even if only a few are active at a time.

E. Model Pre-training

[Representation MoE Model]

Switch Transformers

W. Fedus, et al, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” The Journal of Machine Learning Research, vol. 23, no. 1, pp. 5232–5270, 2022.

[Core idea]

- Replace the dense Feed-Forward Network (FFN) layers in a standard Transformer with a Switch FFN layer
- This layer consists of multiple independent FFN "experts," and for each input token, a router network dynamically selects only one of these experts to process it. This maintains a manageable computational cost per token while vastly increasing the total number of model parameters.

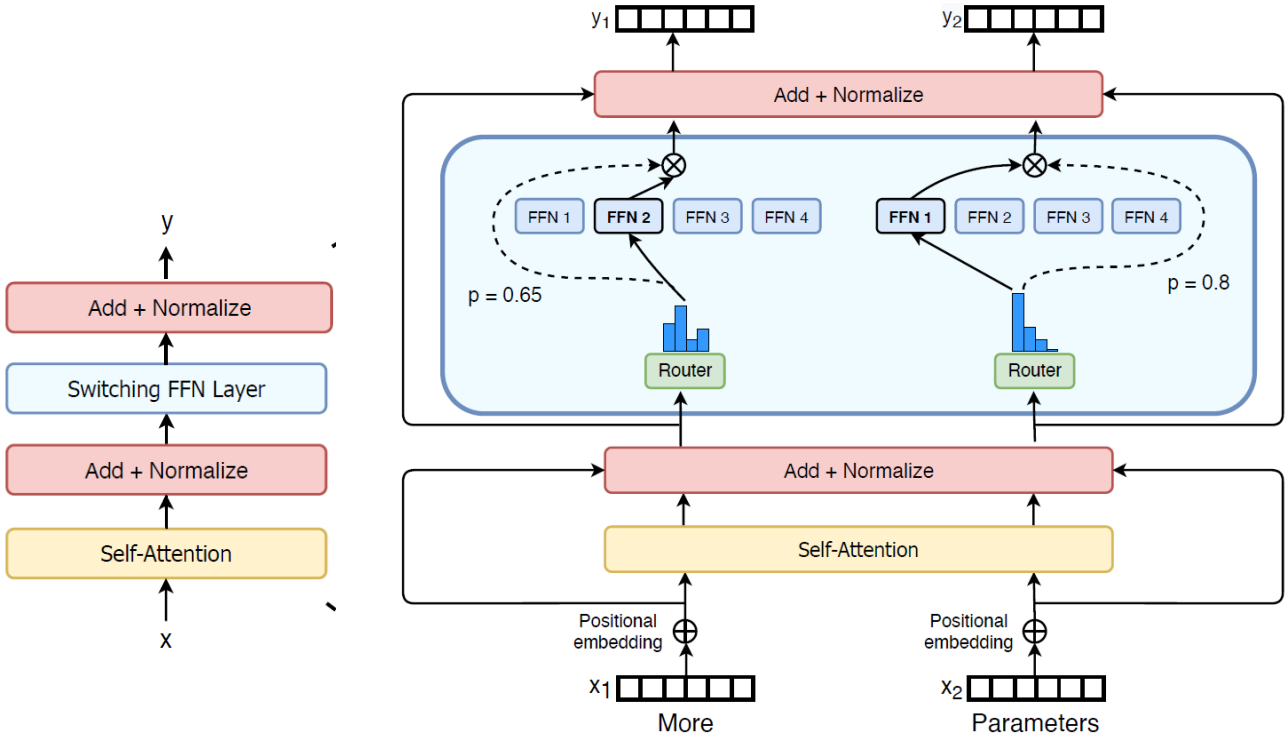


Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = \More" and x_2 = \Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).