

Bios 740- Chapter 8. Large Language Models

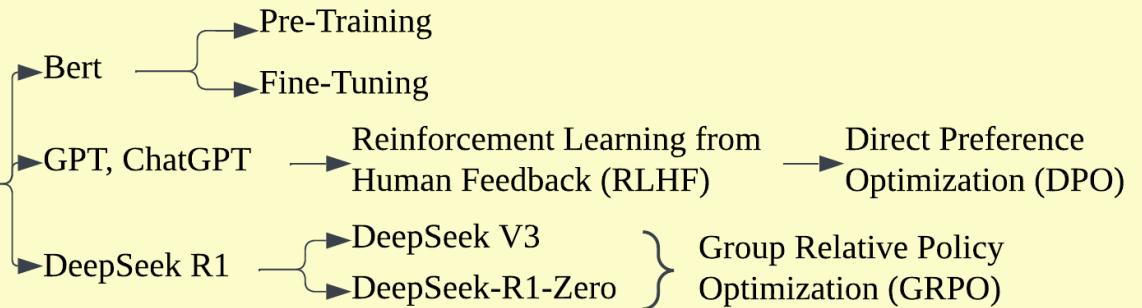
Acknowledgement: Many thanks to Mrs. Shan Gao and Mr. Runpeng Dai for preparing some of these slides. I also drew on material from the lecture presentations of Stanford CS224w and StanfordCS231n as well as content generated by ChatGPT.

Overview of This Chapter

1. Introduction

- ▶ Language Model → Large Language Models (LLMs)
- ▶ Why Large?
- ▶ Evolution of LLMs
- ▶ LLMs in Biomedicine

2. Popular Large Language Model Architectures



3. Make LLMs More Suitable for Your Downstream Application

- ▶ No Training Required
 - ▶ Zero-shot / Few-shot Learning
 - ▶ Chain of Thought (CoT)
- ▶ Training Required
 - ▶ Prompt Tuning
 - ▶ Prefix Tuning
 - ▶ Low-Rank Adaption (LoRA)

4. Challenges

- ▶ Deploy LLMs Locally

5. Applications

- ▶ Fine-Tune LLMs with Our Datasets
- ▶ Retrieval-Augmented Generation (RAG)
- ▶ AI Agents

Content

1. Introduction to Large Language Models

2. Popular Large Language Model Architectures

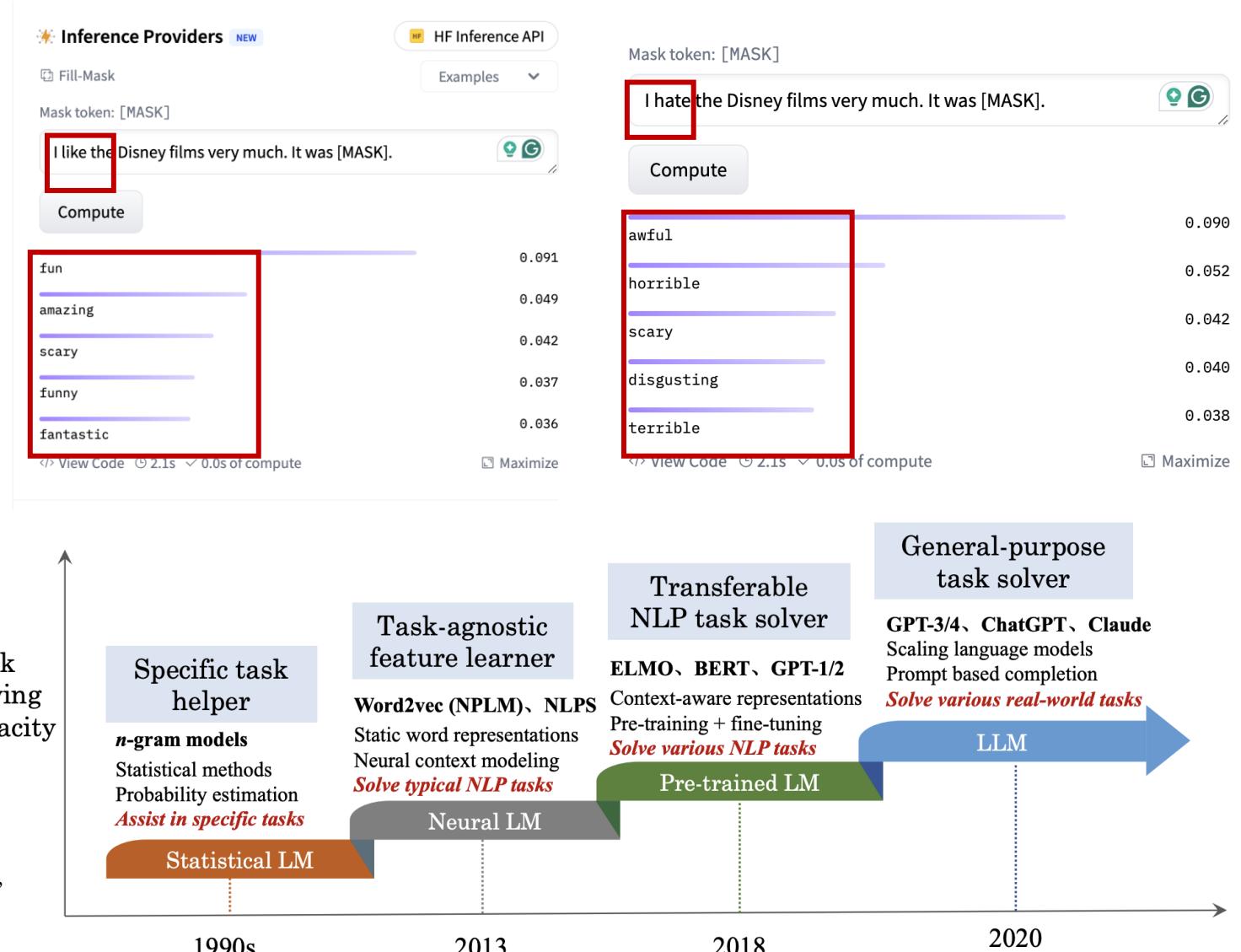
3. Make LLMs More Suitable for Your Downstream Application

4. Challenges

5. Applications

Language Models >> Large LMs (LLMs)

- **Definition:** A language model aims to predict the probability of the occurrence of a token or a sequence of tokens.
- The probability prediction of a language model is closely related to **context** and **corpus** information.
- Language model is **not a new technical** concept specially for LLMs, but has evolved with the advance of artificial intelligence over the decades.
- **Definition:** LLMs have billions of Task solving capacity parameters, trained on massive corpora.

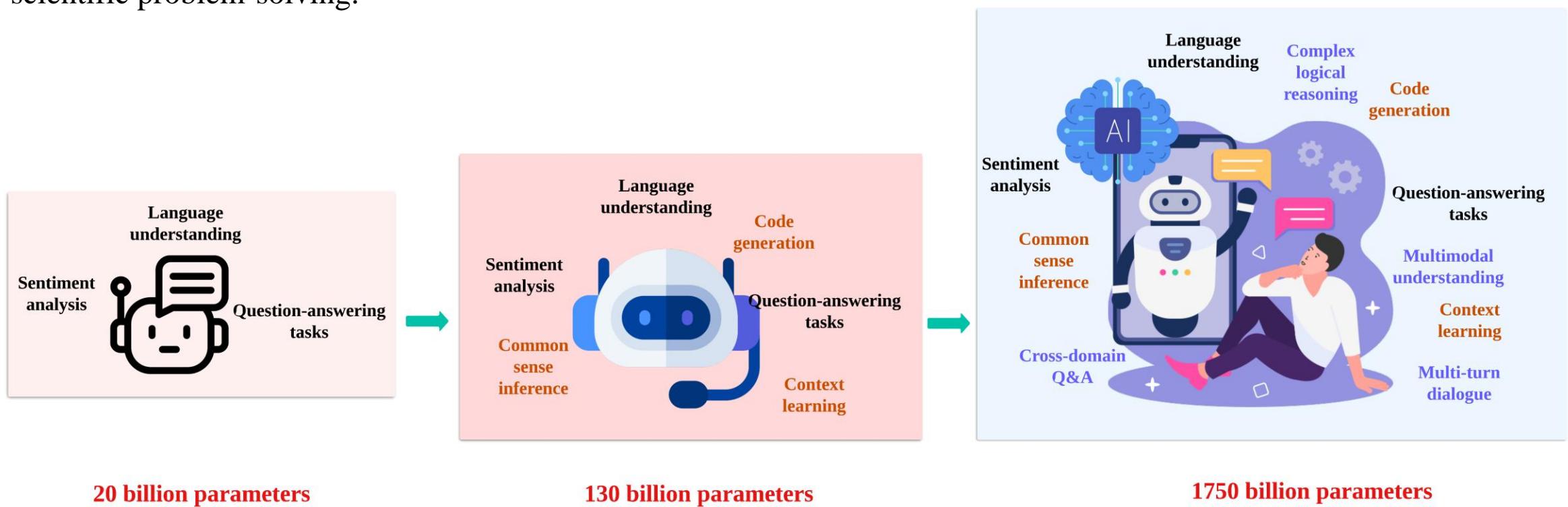


Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., . . . Wen, J. (2023). A Survey of Large Language Models. *ArXiv*. <https://arxiv.org/abs/2303.18223>

Evolution brings new abilities

Key Point: With the continuous iteration and updates of LLMs, the range of problems they can solve has become increasingly rich, demonstrating some new abilities.

- **Growing Complexity:** Each new LLM version shows improvements in reasoning, creativity, and context handling.
- **Emergent Properties:** Larger, more diverse training corpora lead to surprising capabilities (e.g., zero-shot translation, chain-of-thought prompting).
- **Broadening Applications:** Beyond text generation, LLMs now assist in code completion, legal drafting, and even scientific problem-solving.

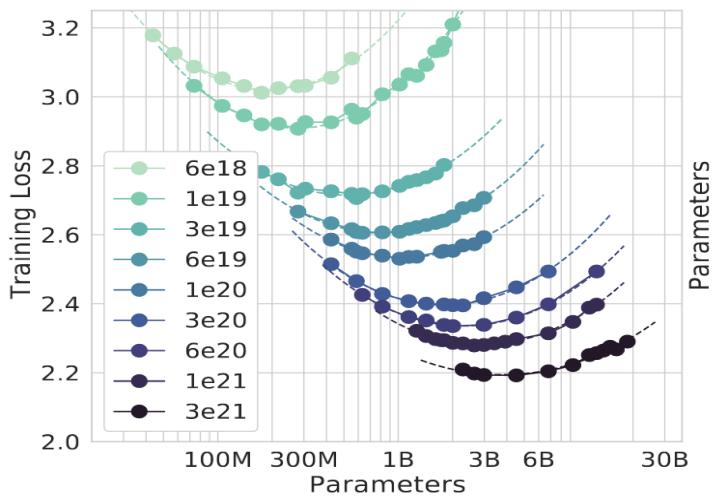
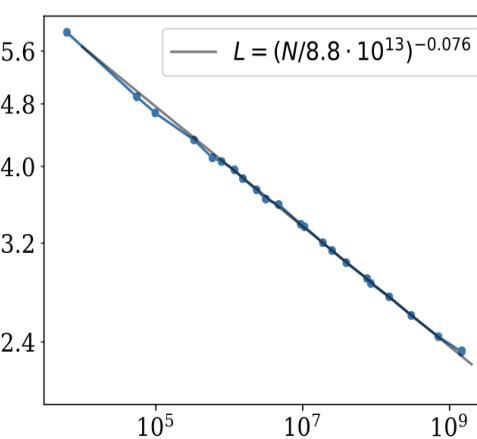
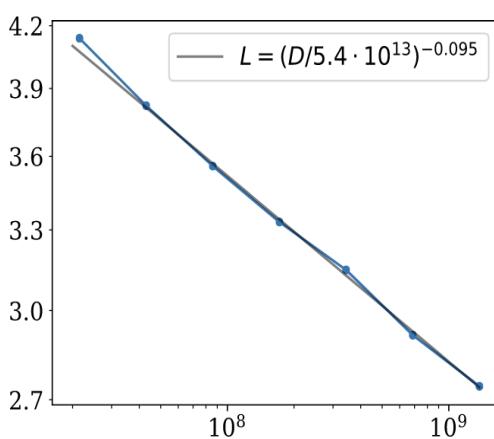
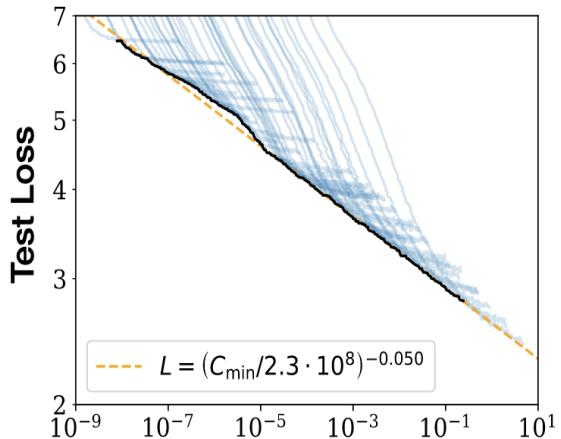


20 billion parameters

130 billion parameters

1750 billion parameters

Scaling Law



Conducted large-scale experiments on Transformer-based models.

- Found that data and model size are equally crucial.
- **Balanced approach:** Sufficiently large dataset and model scale are needed for robust performance across tasks.
- **Reinforced the synergy:** more data = model can generalize; bigger model = can leverage that data better



Scaling Law:

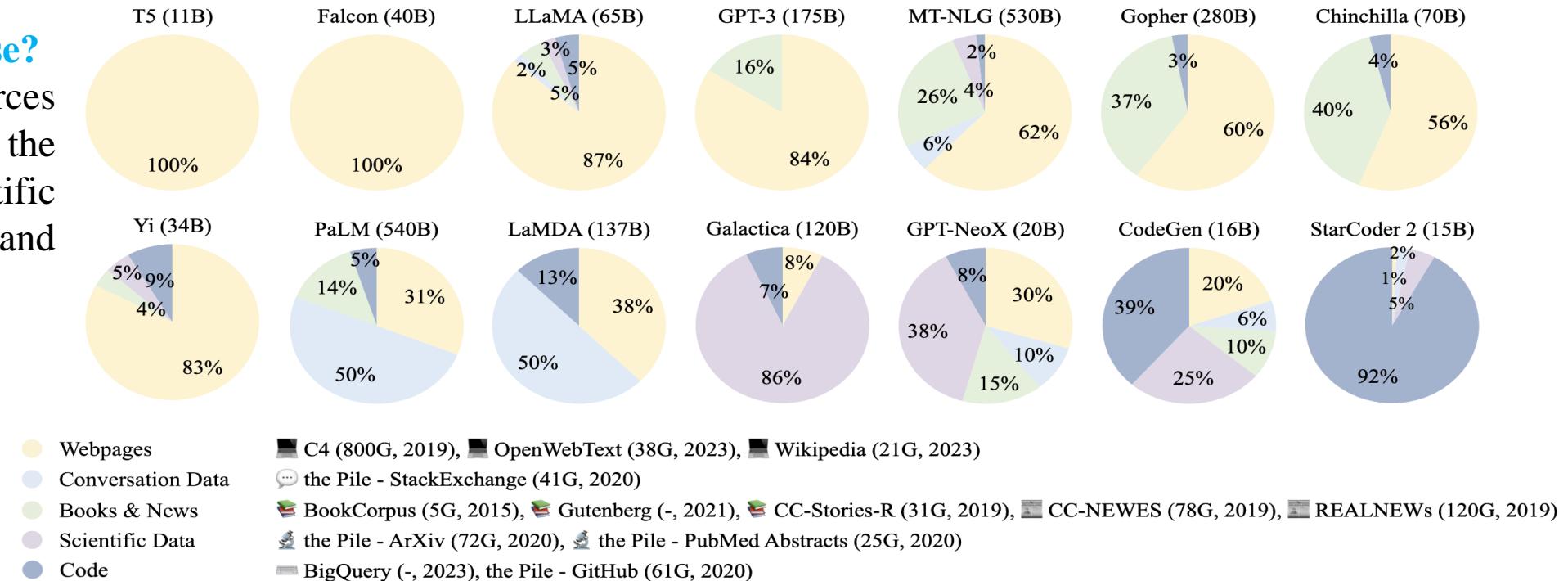
Empirical finding that language modeling performance improves continuously with larger model size, bigger datasets, and more compute.

- Context: Introduced prominently by OpenAI (2020) —“bigger is better” trend.
- Beyond a certain scale, emergent behaviors (complex reasoning, zero-shot skills) often appear.

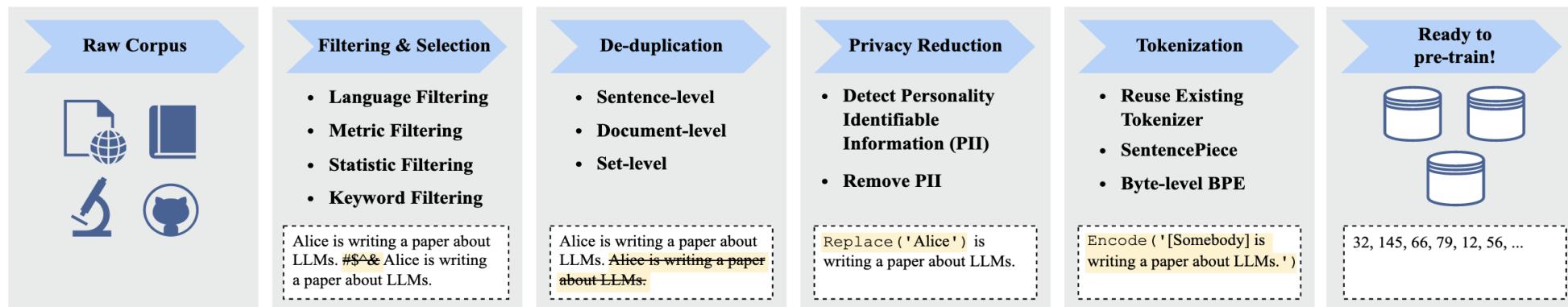
Large-Scale Data for Pretraining

What data do LLMs use?

Typical data sources include text from the internet, books, scientific data, code repositories, and more.



Data preprocessing pipeline



Timeline of Existing LLMs

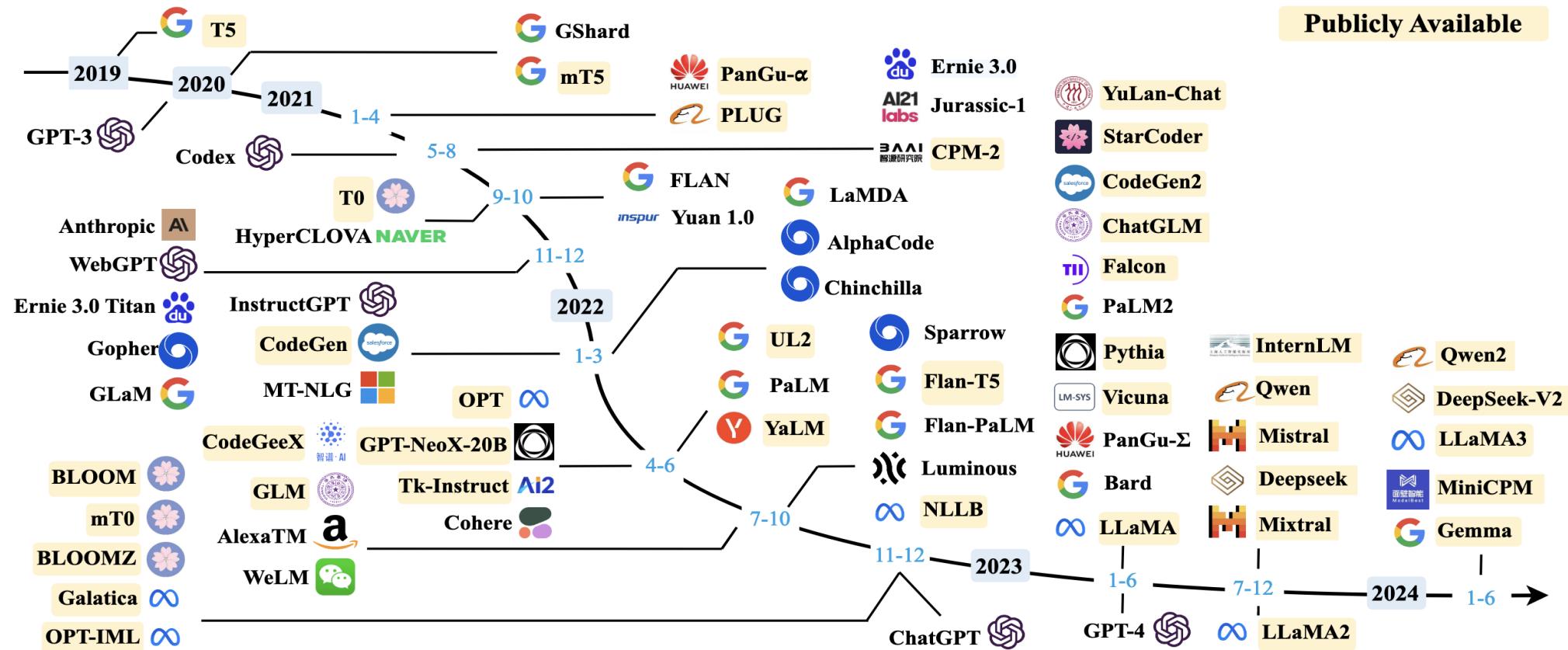


Fig. 3: A timeline of existing large language models (having a size larger than 10B) in recent years. The timeline was established mainly according to the release date (*e.g.*, the submission date to arXiv) of the technical paper for a model. If there was no corresponding paper, we set the date of a model as the earliest time of its public release or announcement. We mark the LLMs with publicly available model checkpoints in yellow color. Due to the space limit of the figure, we only include the LLMs with publicly reported evaluation results.

OpenAI LLM Series Timeline

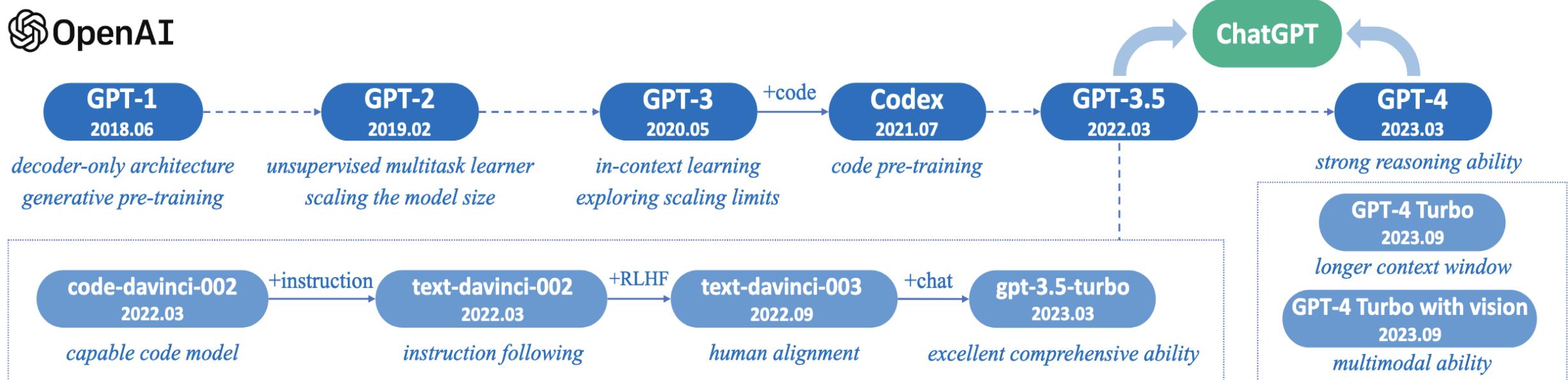
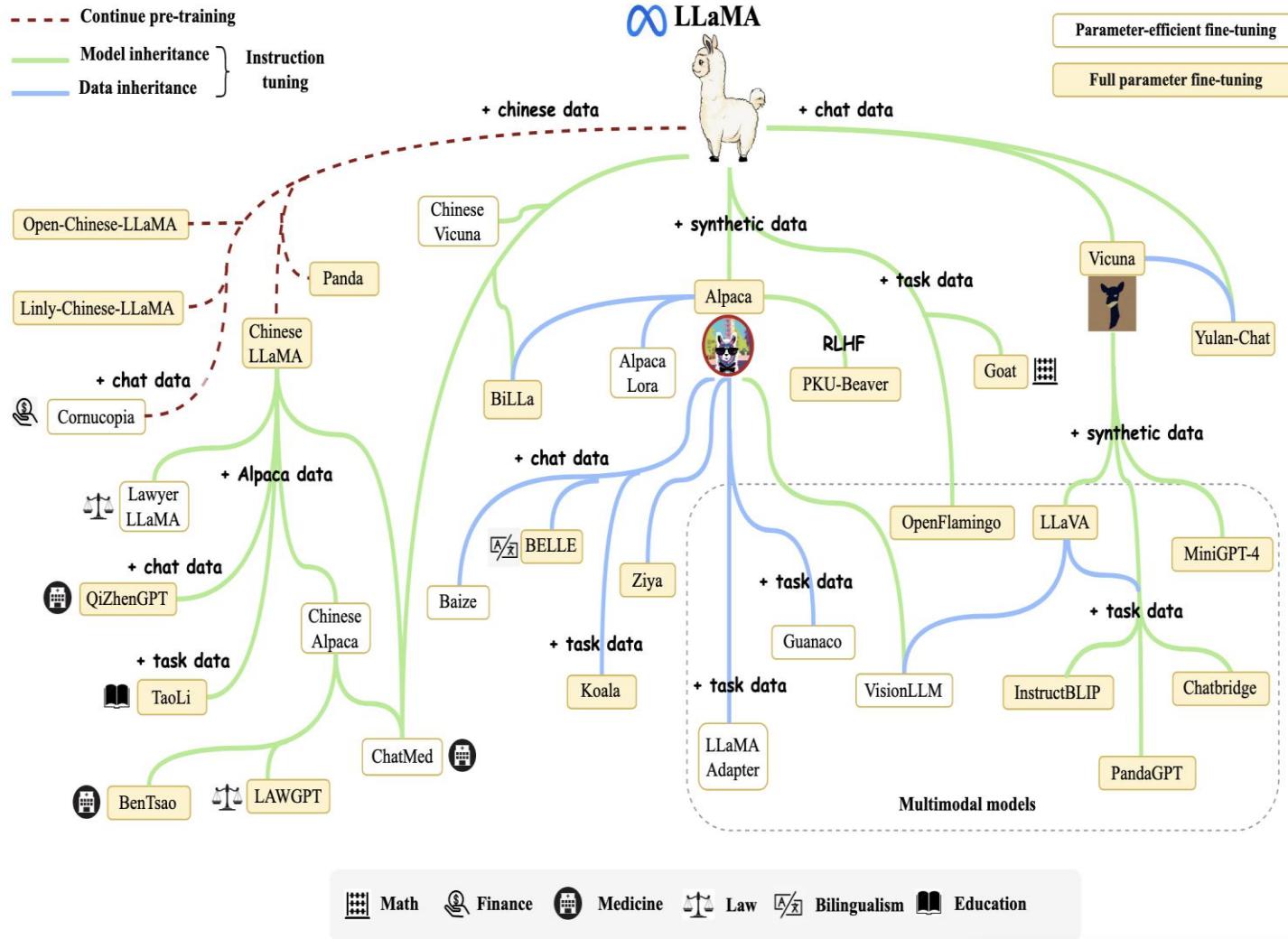


Fig. 4: A brief illustration for the technical evolution of GPT-series models. We plot this figure mainly based on the papers, blog articles and official APIs from OpenAI. Here, *solid lines* denote that there exists an explicit evidence (e.g., the official statement that a new model is developed based on a base model) on the evolution path between two models, while *dashed lines* denote a relatively weaker evolution relation.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., . . . Wen, J. (2023). A Survey of Large Language Models. *ArXiv*. <https://arxiv.org/abs/2303.18223>

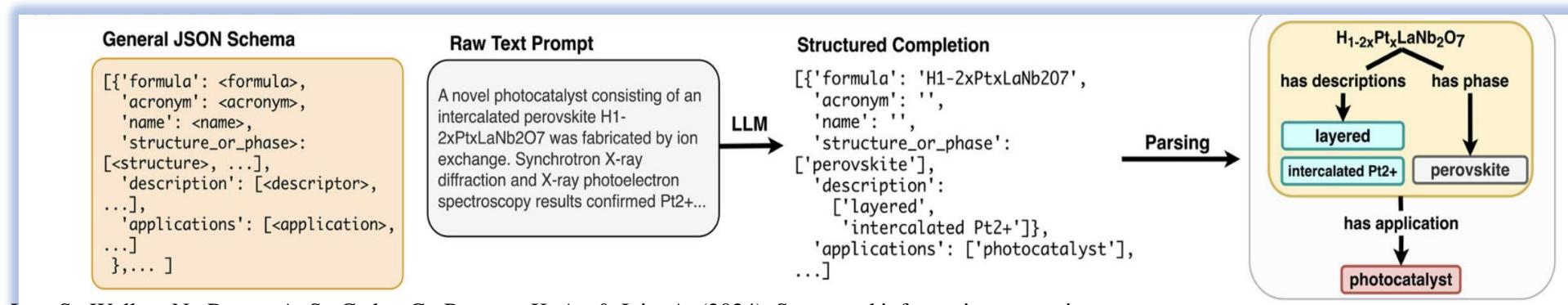
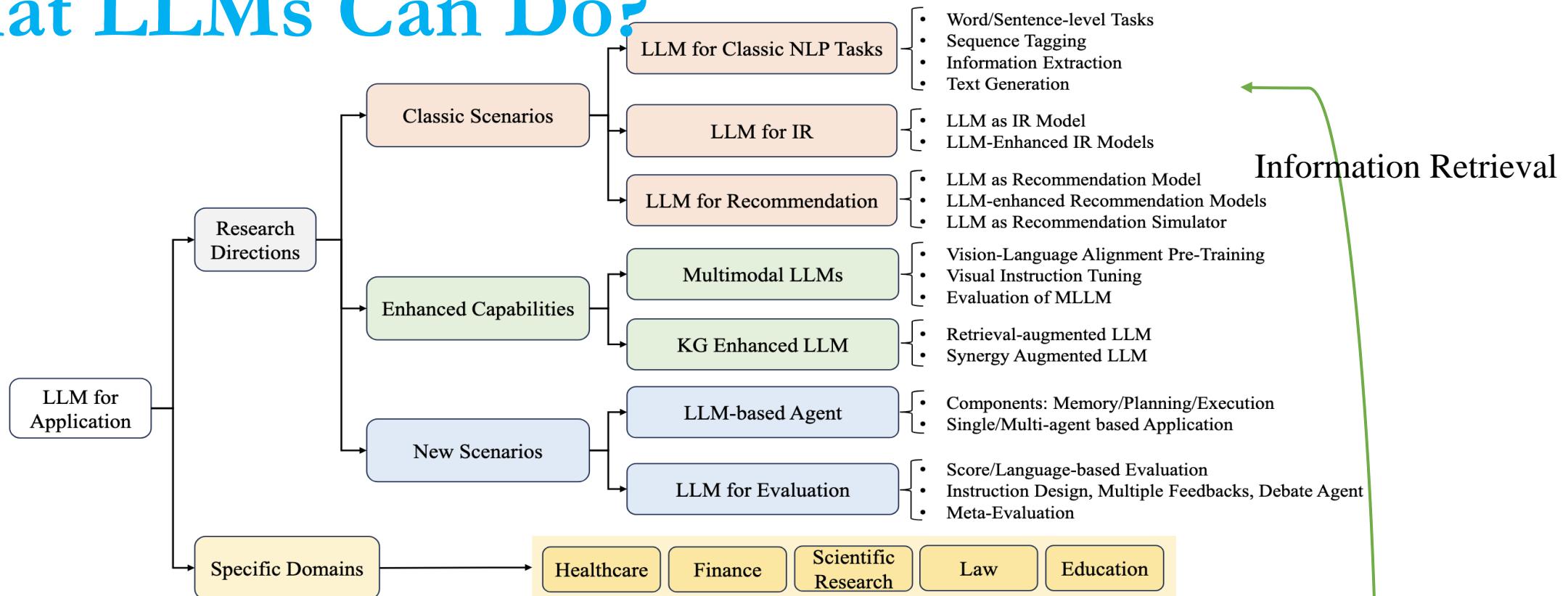
Large Language Model Meta AI (LLaMA)

An Evolutionary Graph of the Research Work Conducted on LLaMA



- **Purpose:** To provide a robust, scalable language model architecture.
- **Capabilities:** Natural language understanding, text generation, reasoning, and more.
- Known for High efficiency and scalability compared to previous models.
- 2023: Initial release of LLaMA with a focus on efficient language modeling.
- 2024: LLaMA 2 introduced, featuring improved accuracy and efficiency.
- 2024: Integration of fine-tuning techniques, including instruction tuning.
- 2024: Emergence of multimodal capabilities, enabling both text and visual tasks.
- Ongoing: Research into expanding real-world applications.

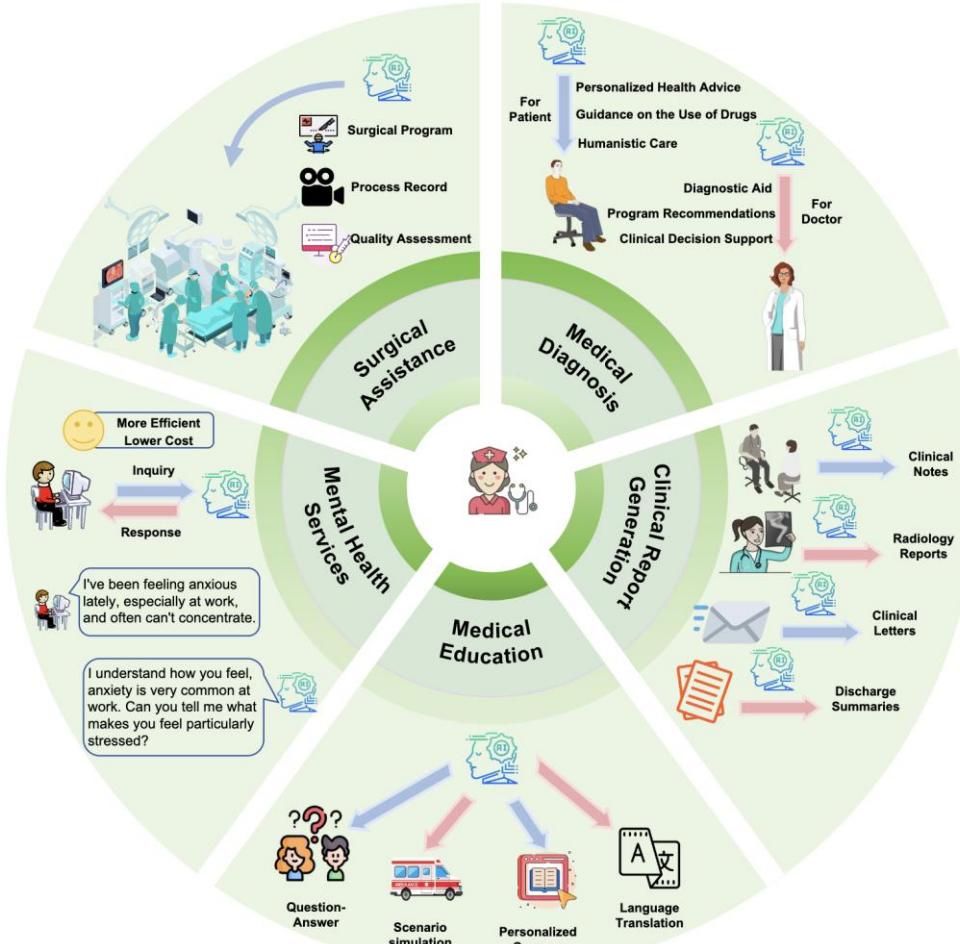
What LLMs Can Do?



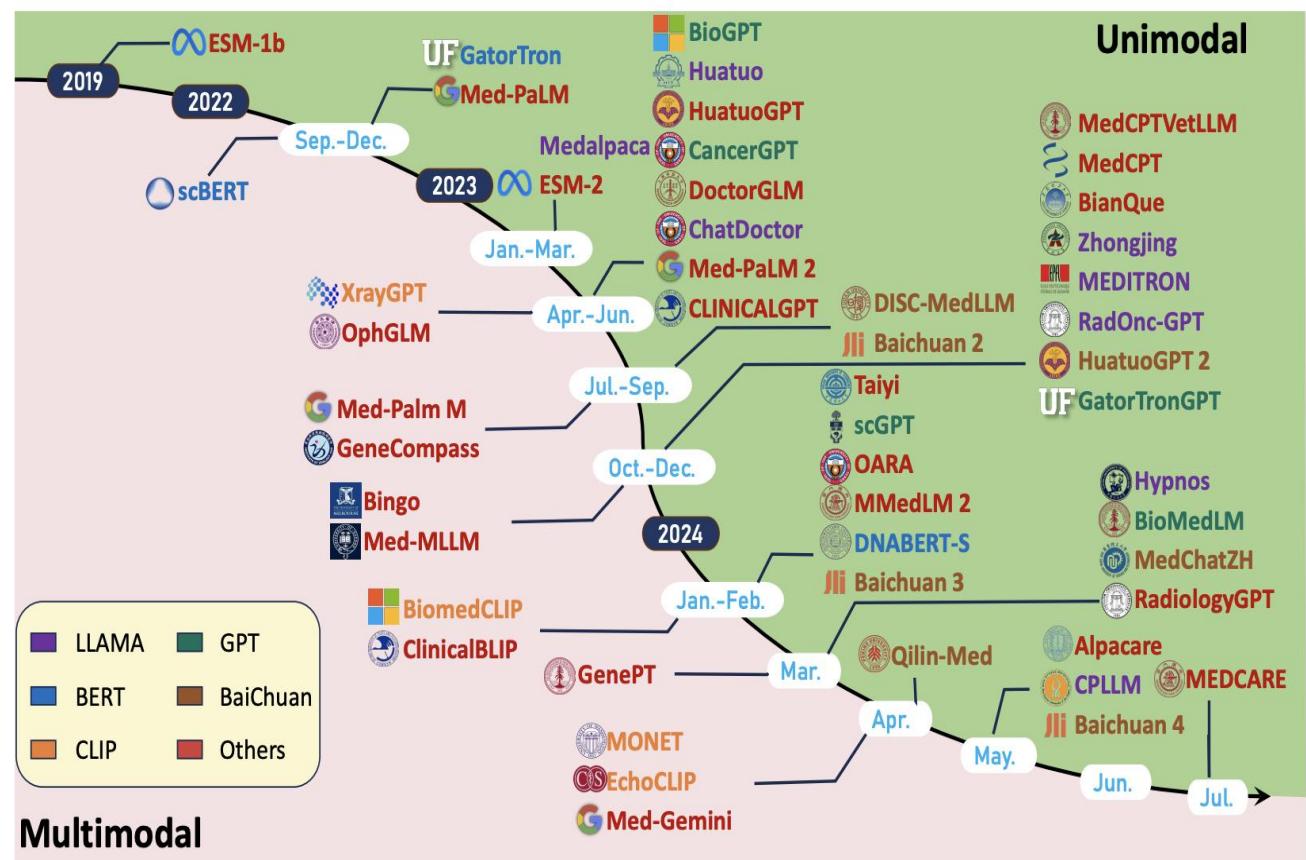
Dagdelen, J., Dunn, A., Lee, S., Walker, N., Rosen, A. S., Ceder, G., Persson, K. A., & Jain, A. (2024). Structured information extraction from scientific text with large language models. *Nature Communications*, 15(1), 1-14. <https://doi.org/10.1038/s41467-024-45563-x>

LLMs in Biomedicine

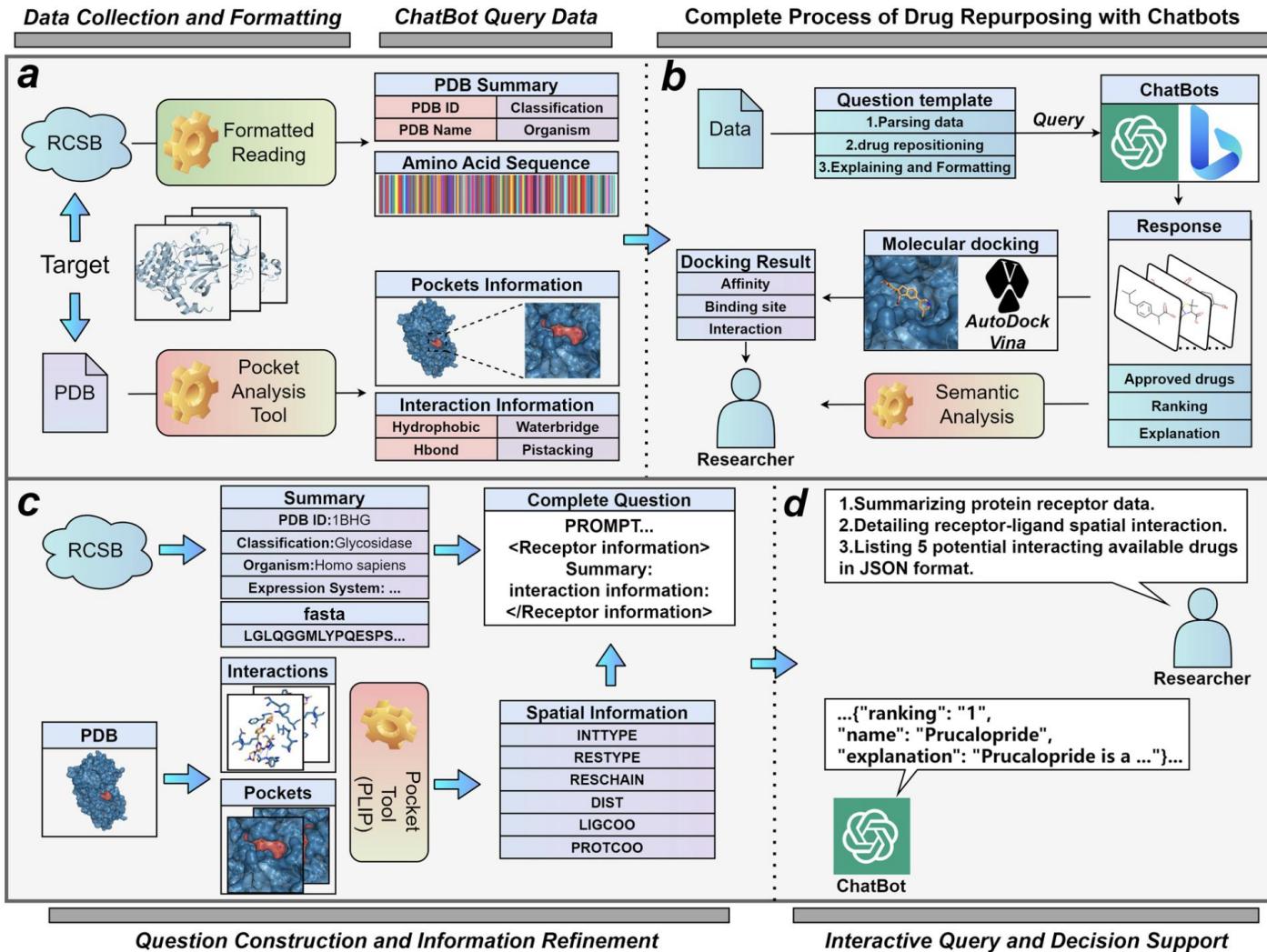
Overview of potential applications of LLMs in medicine.



Chronological overview of LLMs and their variants in biomedical applications from 2019 to 2024



LLMs in Biomedicine



DrugReAlign leverages LLMs and multi-source prompts to efficiently **repurpose existing drugs** by integrating extensive knowledge on targets and drugs.

DrugReAlign Framework and Detailed Flowchart.

- a. Construction of multisource prompts for targets
- b. Screening of potential drugs interacting with targets based on LLMs and multisource prompt information.
- c. Example of Prompt Construction for Specific Targets
- d. Interactive querying and decision support using LLMs

Content

1. Introduction to Large Language Models

2. Popular Large Language Model Architectures

3. Make LLMs More Suitable for Your Downstream Application

4. Challenges

5. Applications

The core of LLMs ---- Transformer



Transformer models have diversified into **Encoder**, **Decoder**, and **Encoder-Decoder branches**, driven by advancements from companies like Microsoft, Google, Meta, OpenAI, and Eleuther AI.

- ❖ Encoder models like BERT focus on language understanding, while Decoder models like GPT are aimed at generation.
- ❖ Encoder-Decoder models combine both functionalities, excelling in tasks like translation and summarization.
- ❖ The trend shows increasing specialization and improvement in both generative and comprehension tasks as the models evolve.

Encoder-only models — BERT

What is BERT?

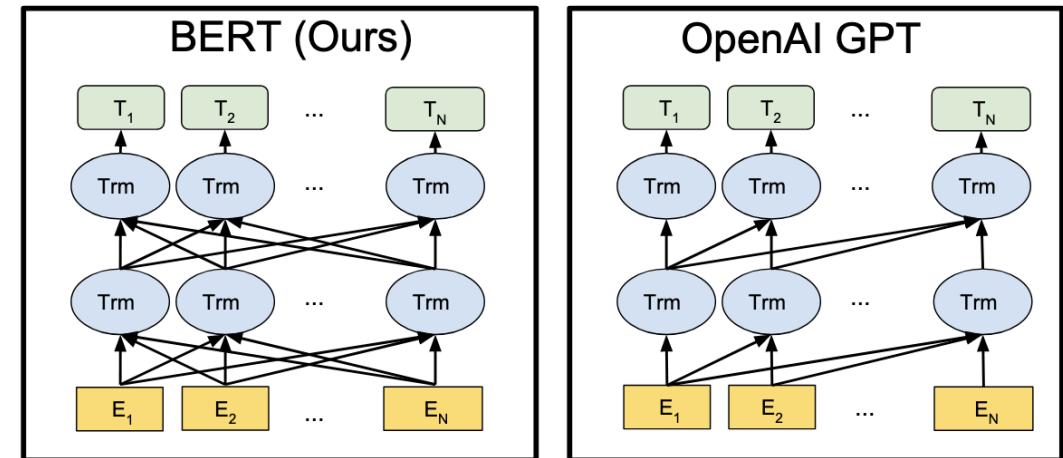
BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language representation model with an encoder-only architecture based on the Transformer's encoder module.

Key Features:

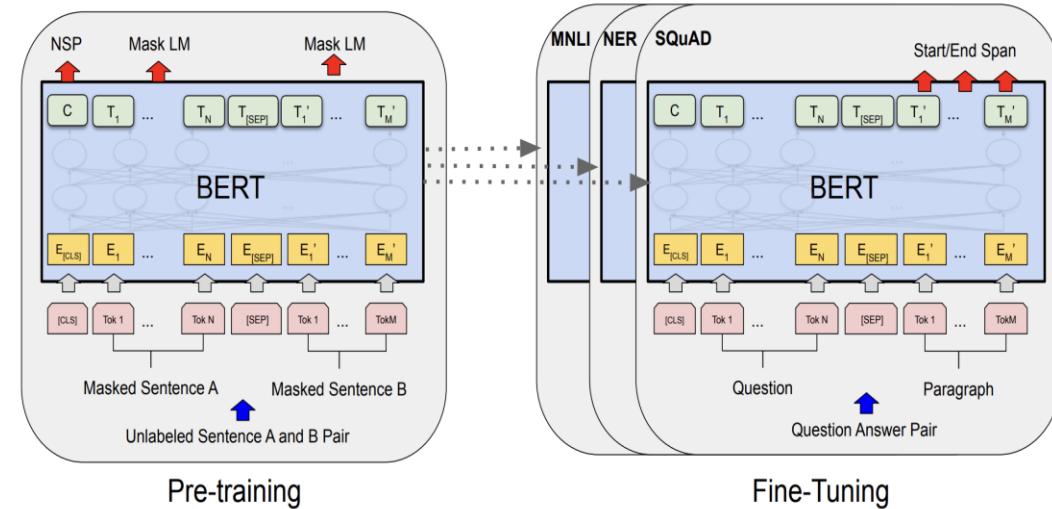
- **Bidirectional Encoding:** Uses bidirectional attention to understand **context from both directions**. Suitable for **understanding-based tasks** like classification and question answering.
- **Pre-training + Fine-tuning:** Pre-trained on large-scale unsupervised data, then fine-tuned on specific tasks.

Difference with OpenAI GPT

GPT: Uses unidirectional attention to predict the **next word** in a sequence. Suitable for **generation-based tasks** like text completion and conversation.



BERT uses a **bidirectional Transformer**.
OpenAI GPT uses a **left-to-right Transformer**.

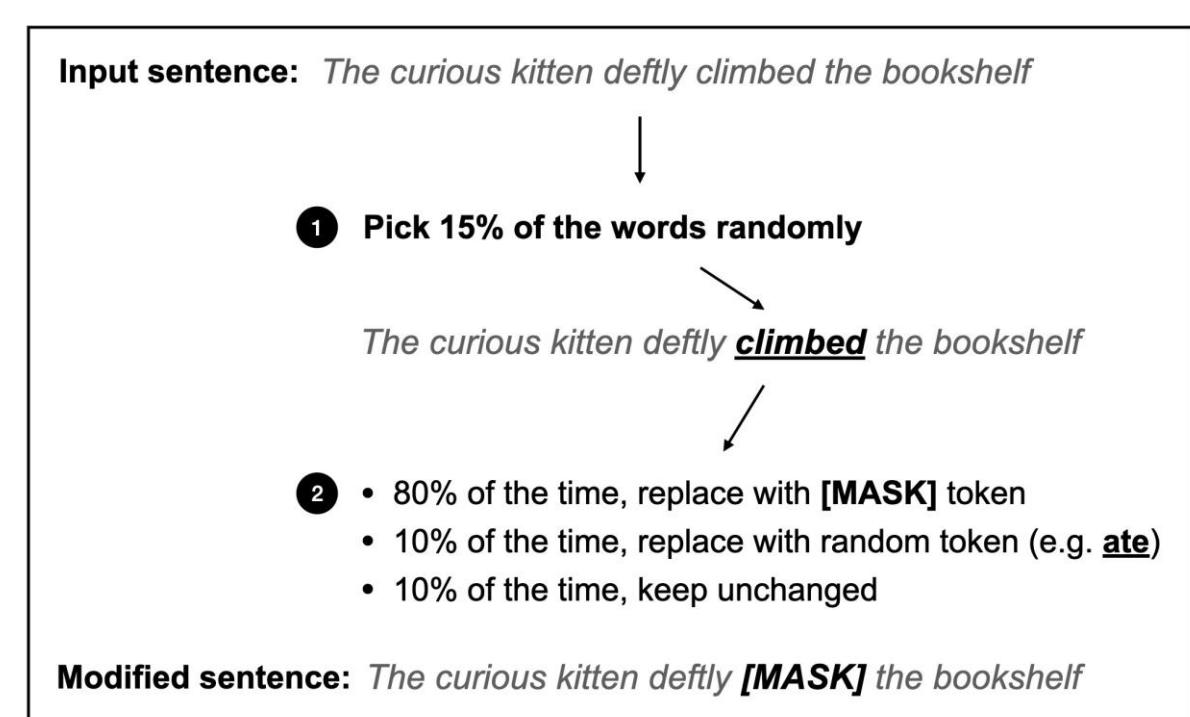


BERT Training Methods — Pre-training

➤ Task1: Masked Language Model (MLM)

- Randomly masks the input words and predicts the original words.
- 80% of masked tokens are replaced with [MASK], 10% are replaced with a random word, and 10% remain unchanged to reduce pre-training and fine-tuning mismatch.

◆ The main idea behind masked language modeling is to mask (or replace) random word tokens in the input sequence and then train the model **to predict the original masked tokens based on the surrounding context**.



BERT Training Methods — Pre-training

➤ Task2: Next Sentence Prediction (NSP)

Predicts whether two sentences are consecutive: 50% of examples are actual consecutive sentences (**IsNext**), 50% of examples are randomly chosen, non-consecutive sentences (**NotNext**).

Helps the model learn textual coherence and is particularly useful for tasks like **Question Answering (QA)** and **Natural Language Inference (NLI)**.

📌 Example:

✓ IsNext:

Sentence A: “She finished her homework.”

Sentence B: “Then she went out to play with her friends.”

Prediction: IsNext

✗ NotNext:

Sentence A: “She finished her homework.”

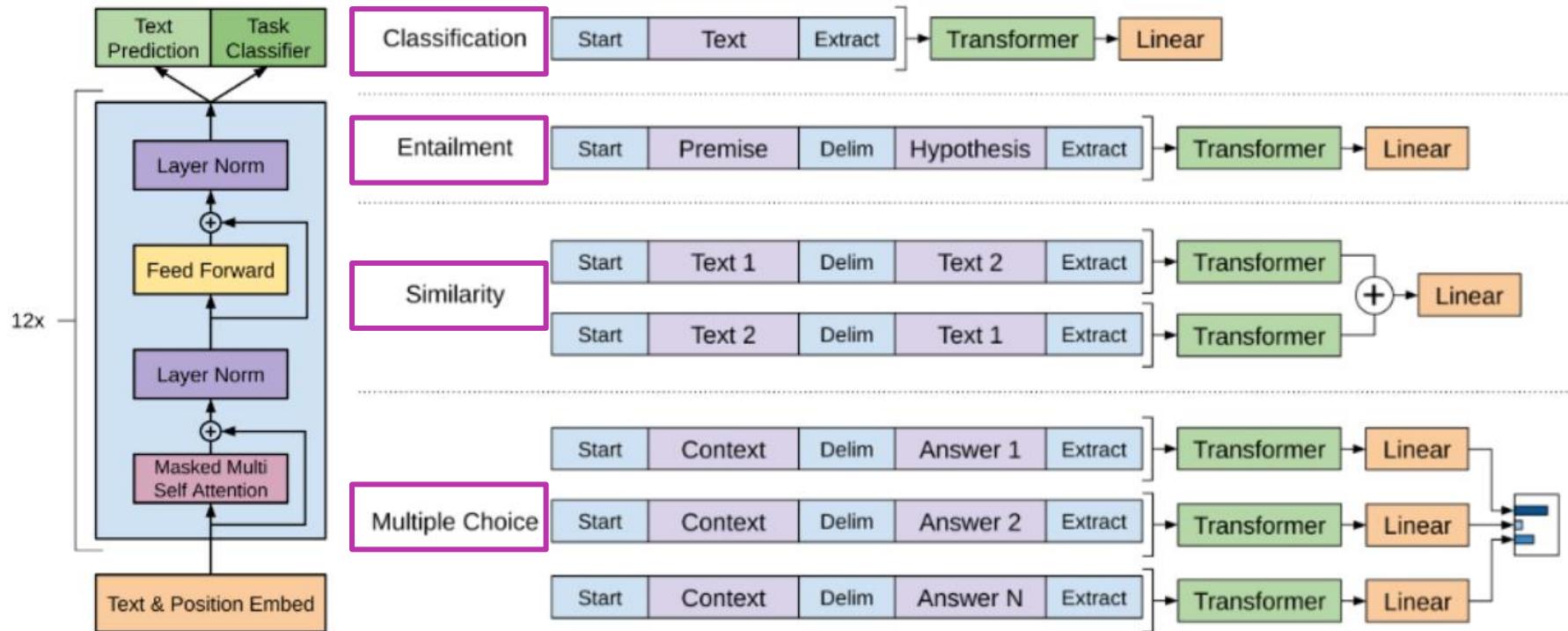
Sentence B: “The stock market showed an unexpected rise today.”

Prediction: NotNext

The above two tasks do not require manually annotated data, allowing for a large corpus to be available for training.

BERT Training Methods — Fine-tuning

Same architecture, different tasks: A simple output layer is added for tasks such as classification, question answering (QA), and Named Entity Recognition (NER).



Sentiment classification using BERT

```
import torch
from transformers import BertModel, BertTokenizer, BertForMaskedLM
# Load pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
masked_model = BertForMaskedLM.from_pretrained('bert-base-uncased')
# Example text
text = "BERT is a powerful language model that learns text representations through bidirectional Transformer architecture."
# Tokenize and convert to model input format
inputs = tokenizer(text, return_tensors="pt")
# Get BERT outputs
with torch.no_grad():
    outputs = model(**inputs)
# Get the [CLS] token representation (commonly used for classification tasks)
last_hidden_states = outputs.last_hidden_state
cls_embedding = last_hidden_states[:, 0, :]
print(f"CLS embedding shape: {cls_embedding.shape}")
# Simple sentiment analysis example (assuming we have a pre-trained classification head)
class SentimentClassifier(torch.nn.Module):
    def __init__(self, hidden_size=768):
        super().__init__()
        self.classifier = torch.nn.Linear(hidden_size, 2) # Binary classification: positive/negative
    def forward(self, embeddings):
        return self.classifier(embeddings)
# Initialize classifier
classifier = SentimentClassifier()
# Use [CLS] embedding for sentiment prediction
logits = classifier(cls_embedding)
predicted_class = torch.argmax(logits, dim=1).item()
print(f"Predicted sentiment: {'Positive' if predicted_class == 1 else 'Negative'}")
```

Decoder-only models GPT family

Transformer-based architecture for all GPT models, with variations in model depth and embedding size.

Key milestones:

- **GPT-1 (2018):** Introduced generative pretraining for NLP tasks, achieving SOTA results.
- **GPT-2 (2019):** Scaled up parameters and training data, pioneering Prompt Learning.
- **GPT-3 (2020): 1,750B parameters,** demonstrated strong performance in code generation and mathematics.

Key Objectives of GPT Models:

1. Improve performance on NLP tasks.
2. Enhance generalization to non-NLP tasks (e.g., coding, math).

Note: OpenAI's GPT models are **not open-source** but offer public access for testing.

Model	Release Date	Layers	Heads	Embedding Size	Parameters	Training Data Size
GPT-1	June 2018	12	12	768	117M	~5GB
GPT-2	Feb 2019	48	-	1600	1.5B	40GB
GPT-3	May 2020	96	96	12,888	175B	45TB

How to improve GPT?

Instruct Learning vs. Prompt Learning

Challenge 1: Pretrained models **lack transparency**, raising concerns about **bias**.

Challenge 2: Need alignment with **human preferences**.

For Challenge 1:

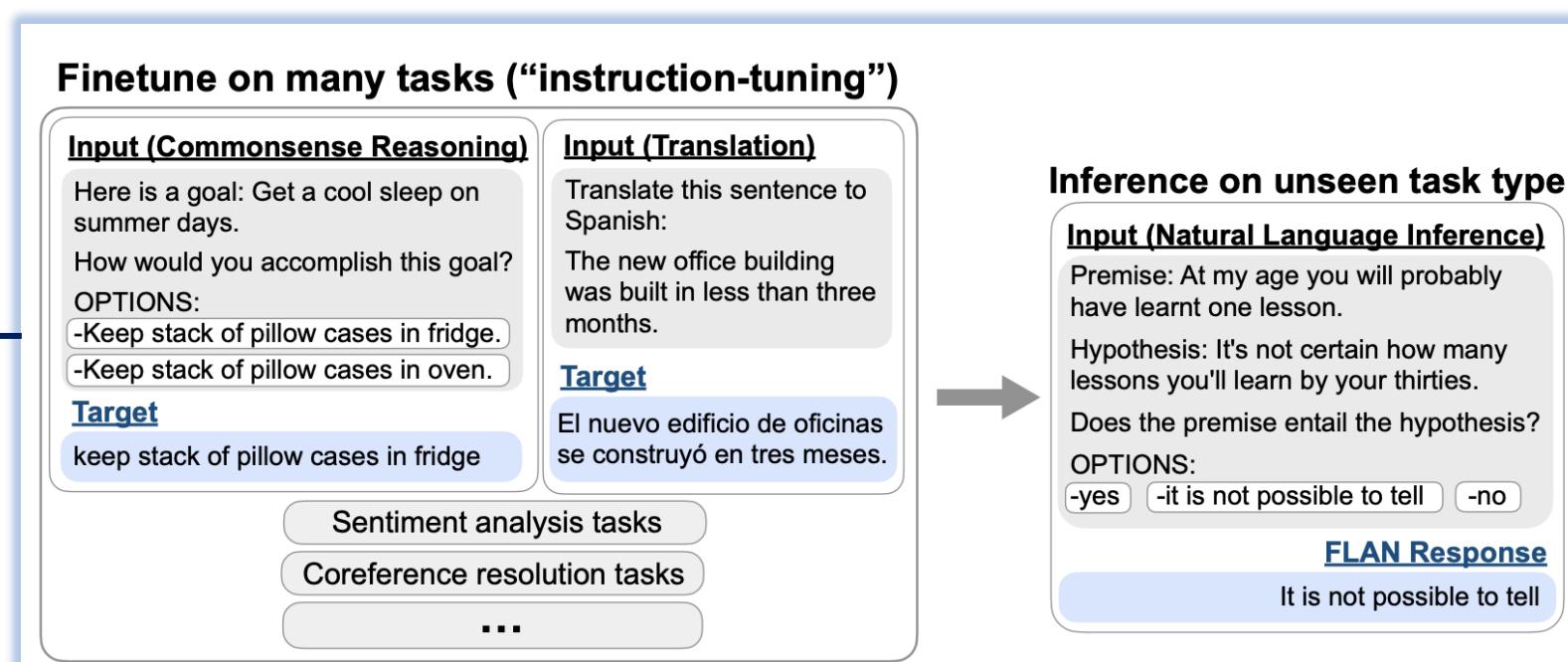
- **Prompt Learning:**

Activates a model's **completion ability**.

📌 Example: *This necklace is too _____.*

- **Instruct Learning:** ←

Activates a model's **understanding ability** by using explicit instructions.



Key Advantages of Instruct Learning

- ✓ **Better generalization:** Enables **zero-shot** learning after multi-task fine-tuning.
- ✓ **More effective than prompt learning**, which is **task-specific** and lacks adaptability.

RL from Human Feedback (RLHF)

For Challenge 2: Focus on **3H** principles:

- **Helpful** (Improves usability)
- **Honest** (Enhances trustworthiness)
- **Harmless** (Mitigates harmful outputs)

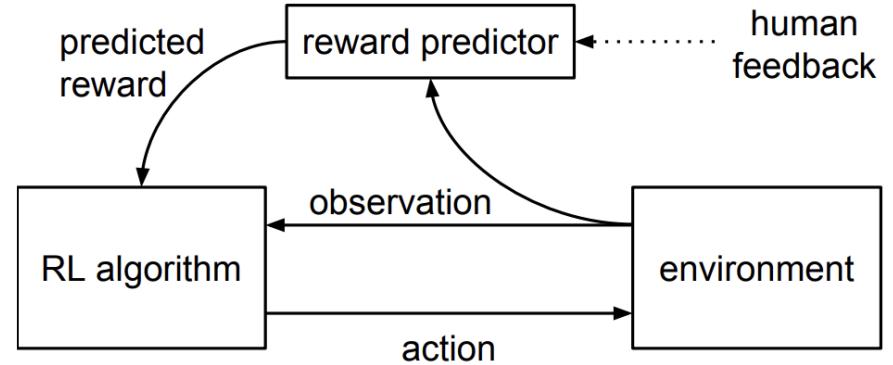
Reinforcement Learning Basics

- Uses **rewards** as feedback, similar to loss functions in traditional training.
- Rewards can be **more flexible** than loss functions but are **not differentiable**, requiring an approximation.

RLHF Process

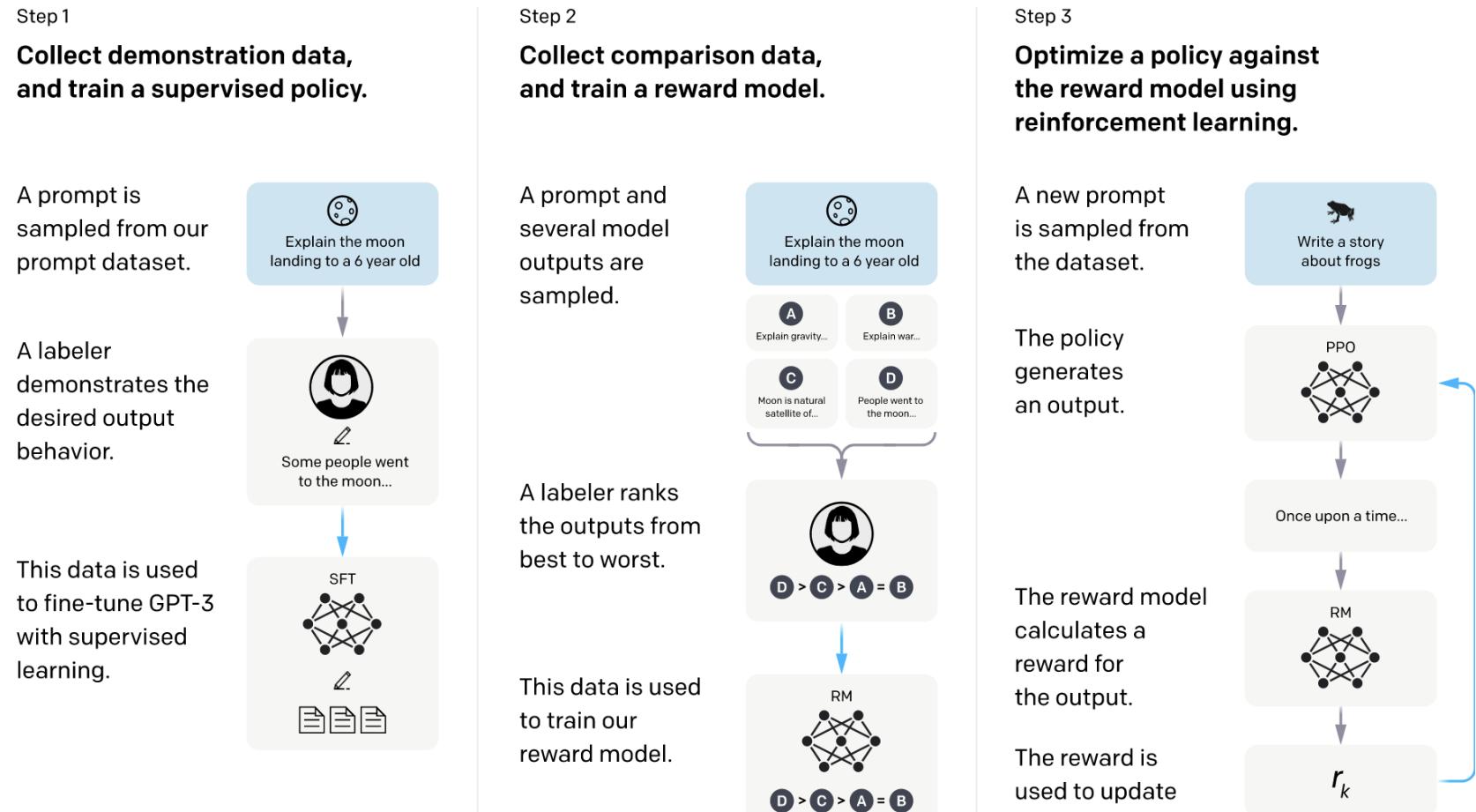
1. **Human annotations** provide reward signals.
2. **Model samples multiple outputs**, and **human preferences** guide reward learning.
3. **Reinforcement Learning optimizes the model** to maximize human-aligned responses.

>>> **InstructGPT & ChatGPT**



InstructGPT/ChatGPT

- Training language models to follow instructions with human feedback. (**RLHF**)
- A diagram illustrating the three steps of our method:
 1. Supervised fine-tuning (SFT),
 2. Reward model (RM) training,
 3. Reinforcement learning via proximal policy optimization (PPO) on this reward model.



Human Feedback: Essential for aligning model outputs with user expectations.

Reinforcement Learning: Enables the model to generate higher-quality and more aligned responses.

Iterative Process: Combines supervised fine-tuning with reinforcement learning for enhanced model performance.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., . . . Lowe, R. (2022). Training language models to follow instructions with human feedback. *ArXiv*. <https://arxiv.org/abs/2203.02155>

Dataset Collection

1. Supervised Fine-Tuning (SFT) Dataset

Purpose: Fine-tune GPT-3 using **instruction-response pairs**.

Sources:

- **User data** from OpenAI's Playground.
- **40 trained labelers** hired by OpenAI.

Table 6: Dataset sizes, in terms of number of prompts.

SFT Data			RM Data			PPO Data		
split	source	size	split	source	size	split	source	size
train	labeler	11,295	train	labeler	6,623	train	customer	31,144
train	customer	1,430	train	customer	26,584	valid	customer	16,185
valid	labeler	1,550	valid	labeler	3,488			
valid	customer	103	valid	customer	14,399			

2. Reward Model (RM) Dataset

Purpose: Train a **reward model** to guide content alignment.

Process:

- Model generates candidate responses.
- **Labelers rank responses** based on quality & bias.
- **Lower scores for biased/harmful content**, discouraging unwanted outputs.

3. PPO (Proximal Policy Optimization) Dataset

Purpose: Used in the third stage of RLHF training.

Source: User-generated queries from GPT-3's API.

Table 1: Distribution of use case categories from our API prompt dataset.

Use-case	(%)
Generation	45.6%
Open QA	12.4%
Brainstorming	11.2%
Chat	8.4%
Rewrite	6.6%
Summarization	4.2%
Classification	3.5%
Other	3.5%
Closed QA	2.6%
Extract	1.9%

Direct Preference Optimization (DPO)

Challenges with RLHF:

- Complexity:** Requires multiple LM training steps and continuous policy sampling.
- High computational costs.**

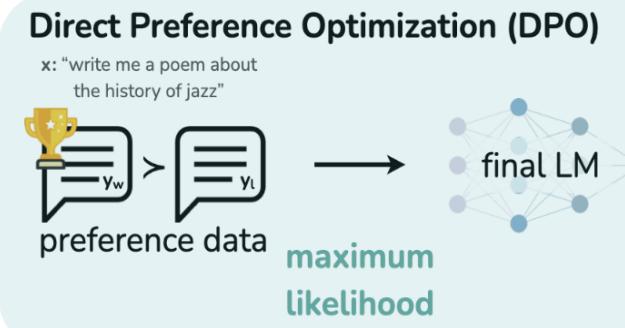
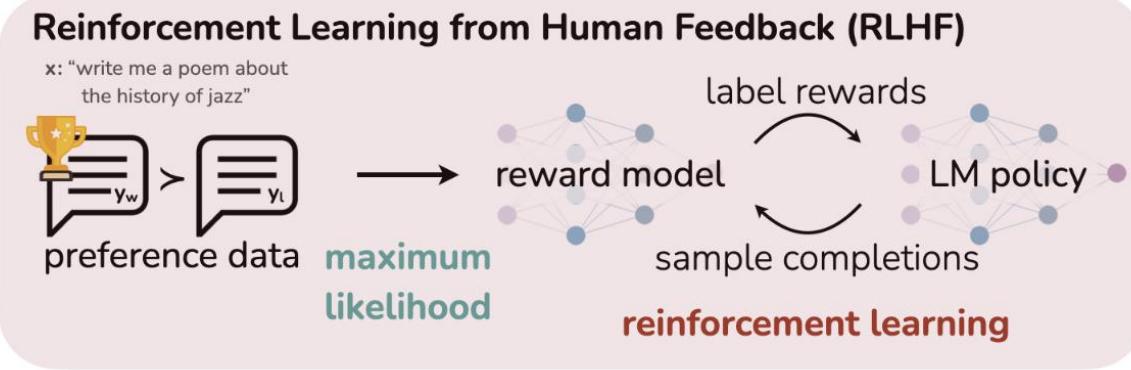
What is DPO?

Avoiding RL

- A simpler alternative to RLHF** that directly optimizes a language model for human preferences.
- No explicit reward modeling or reinforcement learning required.**

Key Advantages of DPO

- Optimizes the same objective as RLHF** (reward maximization with KL-divergence constraint).
- Updates model probability distribution:**
 - **Increases log probability of preferred responses.**
 - **Uses per-example importance weighting** to prevent model degeneration.
- Relies on a theoretical preference model** (e.g., **Bradley-Terry model**) to align reward functions with empirical data.



DeepSeek-R1

Why DeepSeek-R1 is important?

- **Cost-Effective Excellence:** DeepSeek-R1 achieves reasoning performance comparable to OpenAI's o1 model but at a significantly reduced cost.
- **Open-Source Ecosystem Contributions:**
 - Reinforcement Learning (RL)
 - Supervised Fine-Tuning (SFT)
 - GRPO Algorithm: Eliminates the need for a separate critic model, enhancing RL efficiency.
 - Integrated Reward Mechanism: Combines metrics such as accuracy, language consistency, helpfulness, and harmlessness to ensure well-rounded model performance.
 - Multi-Domain Data Mixing Strategy: Improves adaptability across diverse tasks, enabling the model to perform effectively in various scenarios.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., . . . Zhang, Z. (2025). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *ArXiv*. <https://arxiv.org/abs/2501.12948>

nature

Explore content ▾ About the journal ▾ Publish with us ▾ Subscribe

[nature](#) > [news](#) > article

NEWS | 30 January 2025

How China created AI model DeepSeek and shocked the world

Government policies, generous funding and a pipeline of AI graduates have helped Chinese firms create advanced LLMs.

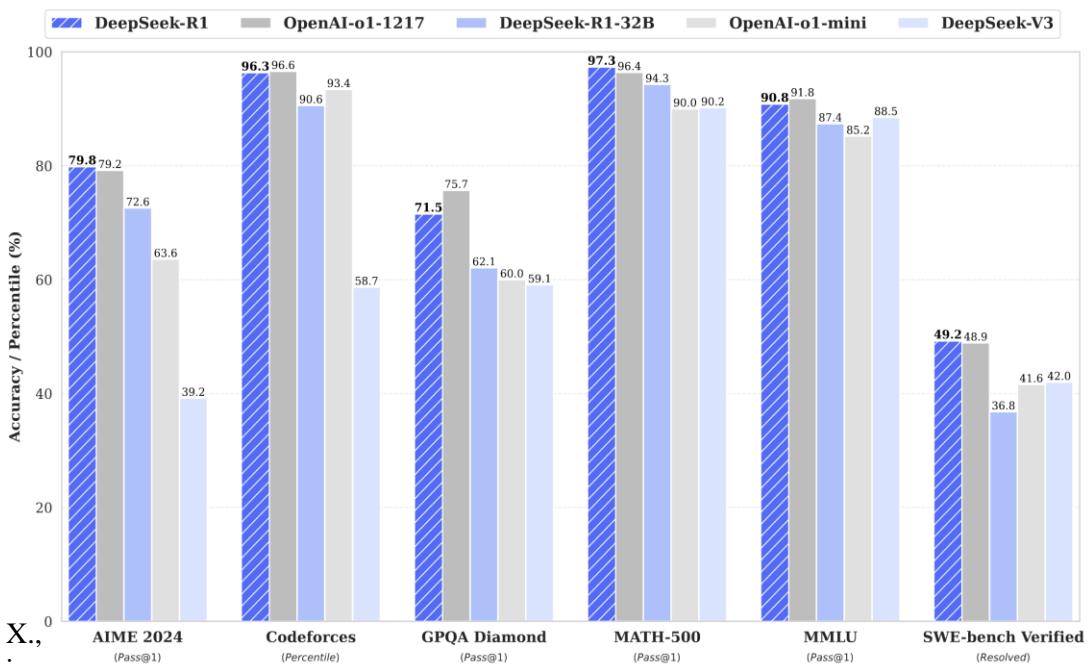
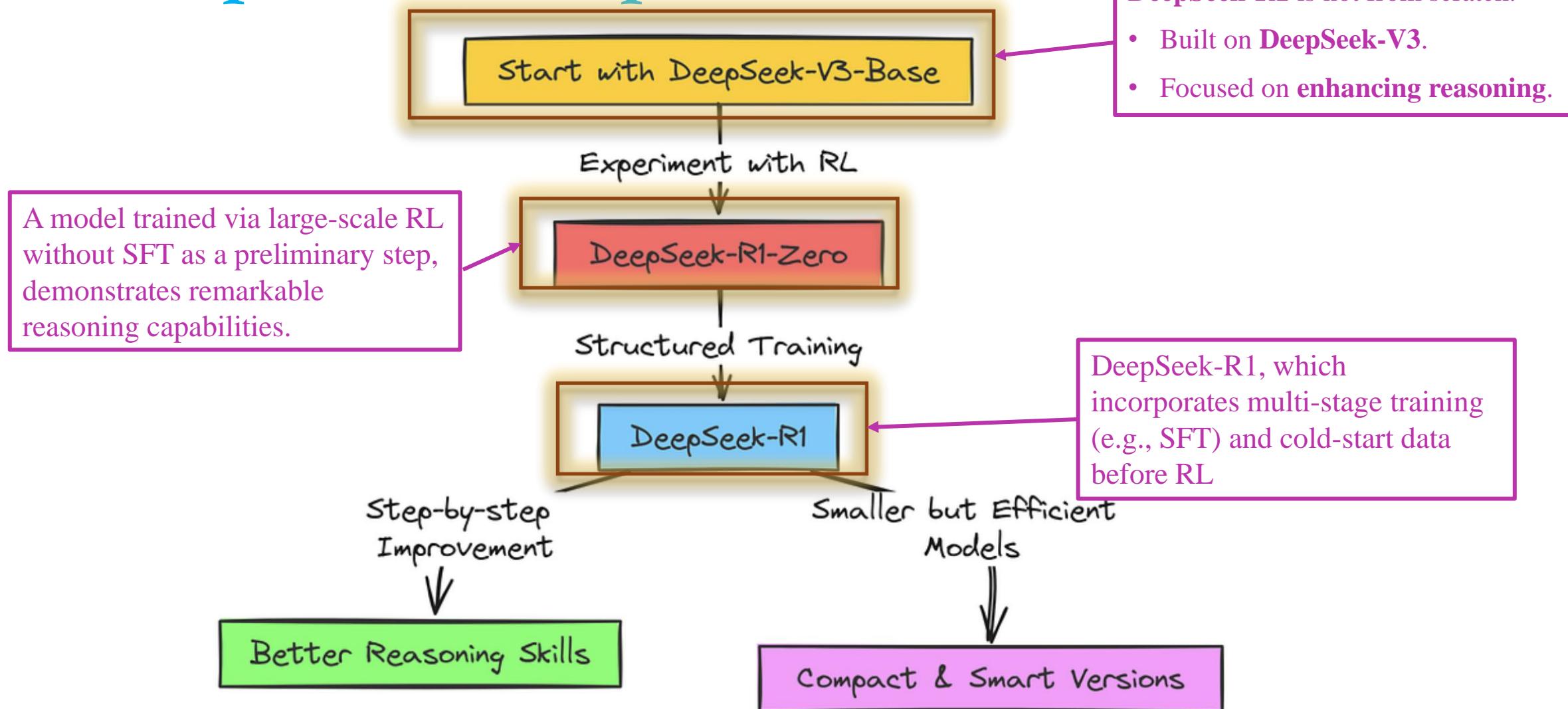


Figure 1 | Benchmark performance of DeepSeek-R1.

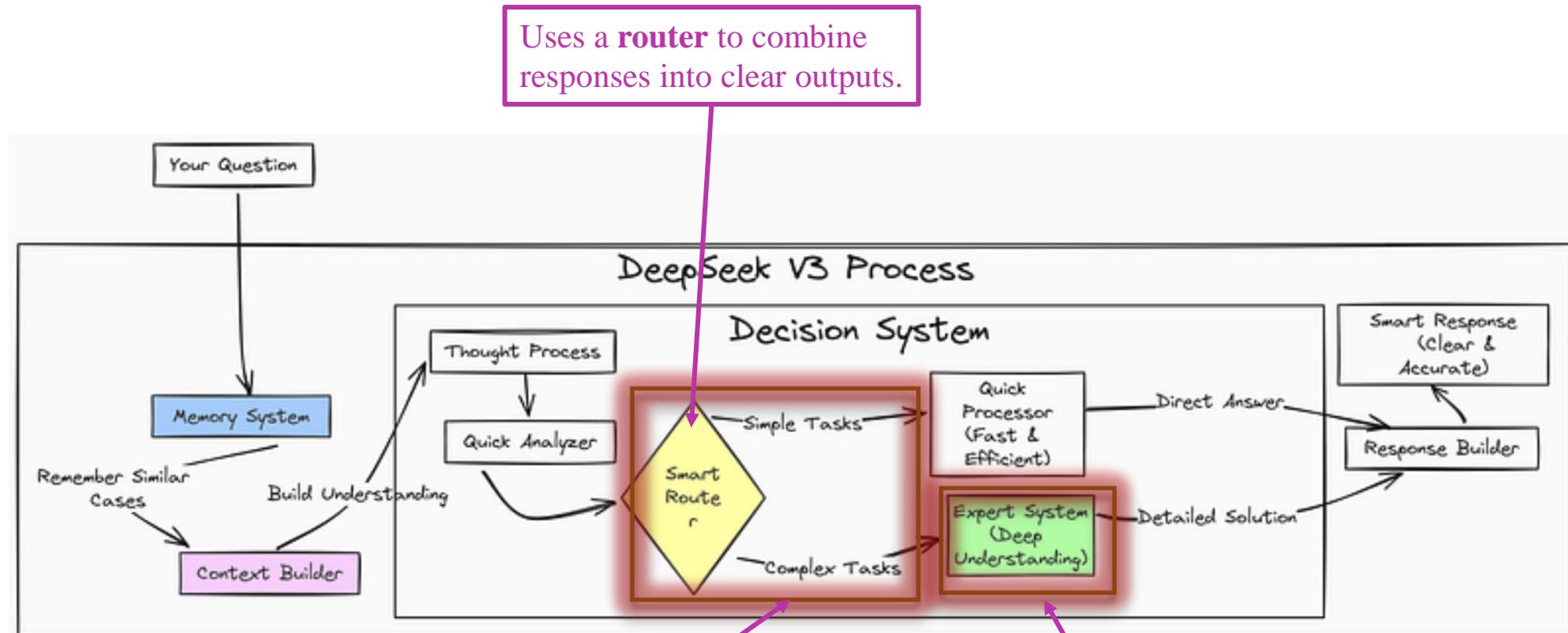
DeepSeek-R1 Pipeline



DeepSeek Implementation Quick Overview (Created by [Fareed Khan](#))

<https://levelup.gitconnected.com/drawing-deepseek-r1-architecture-and-training-process-from-scratch-72043da33955>

How DeepSeek V3 Thinks?



DeepSeek V3 Architecture (Created by Fareed Khan)

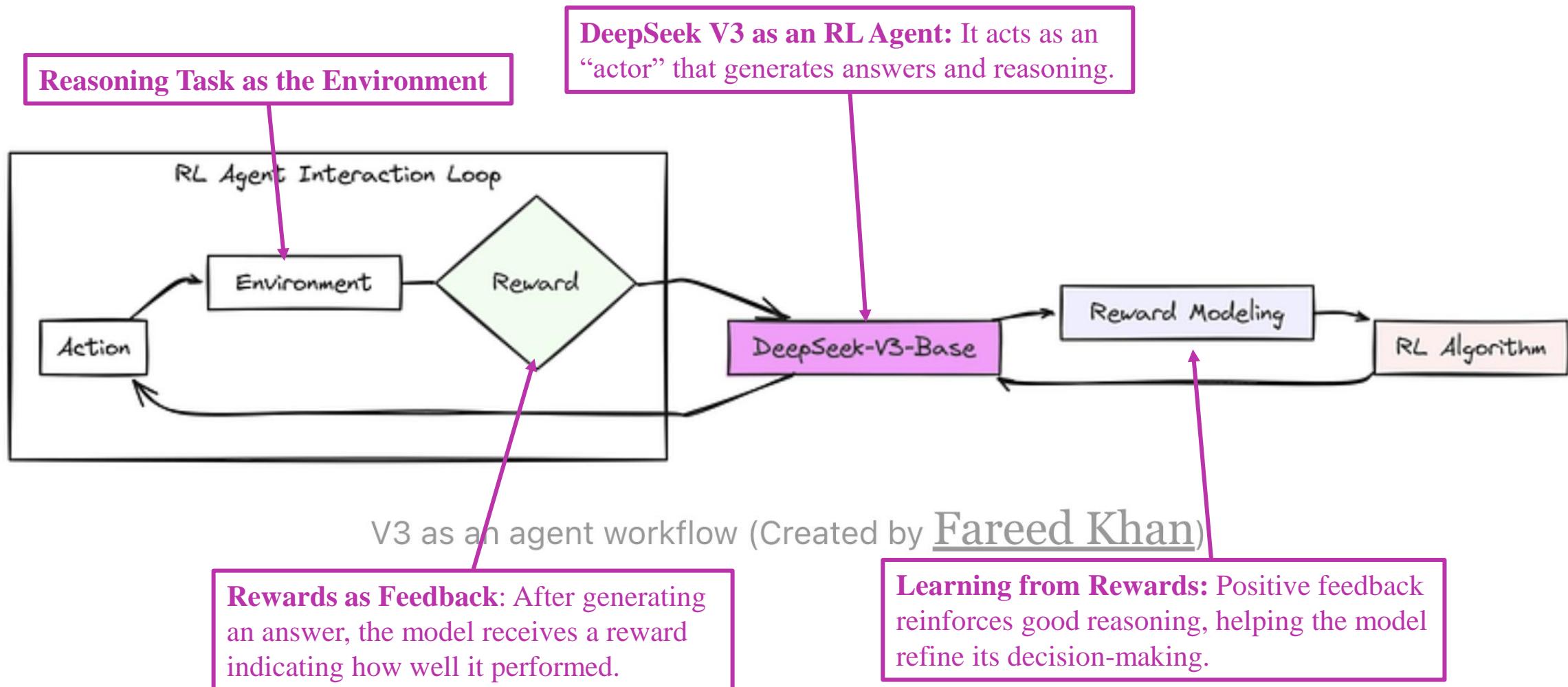
DeepSeek V3 follows two paths:

- **Quick processor** for simple tasks.
- **Expert system** for complex problems.

A Mixture of Experts (MoE) model, dynamically selecting the best expert for each task.

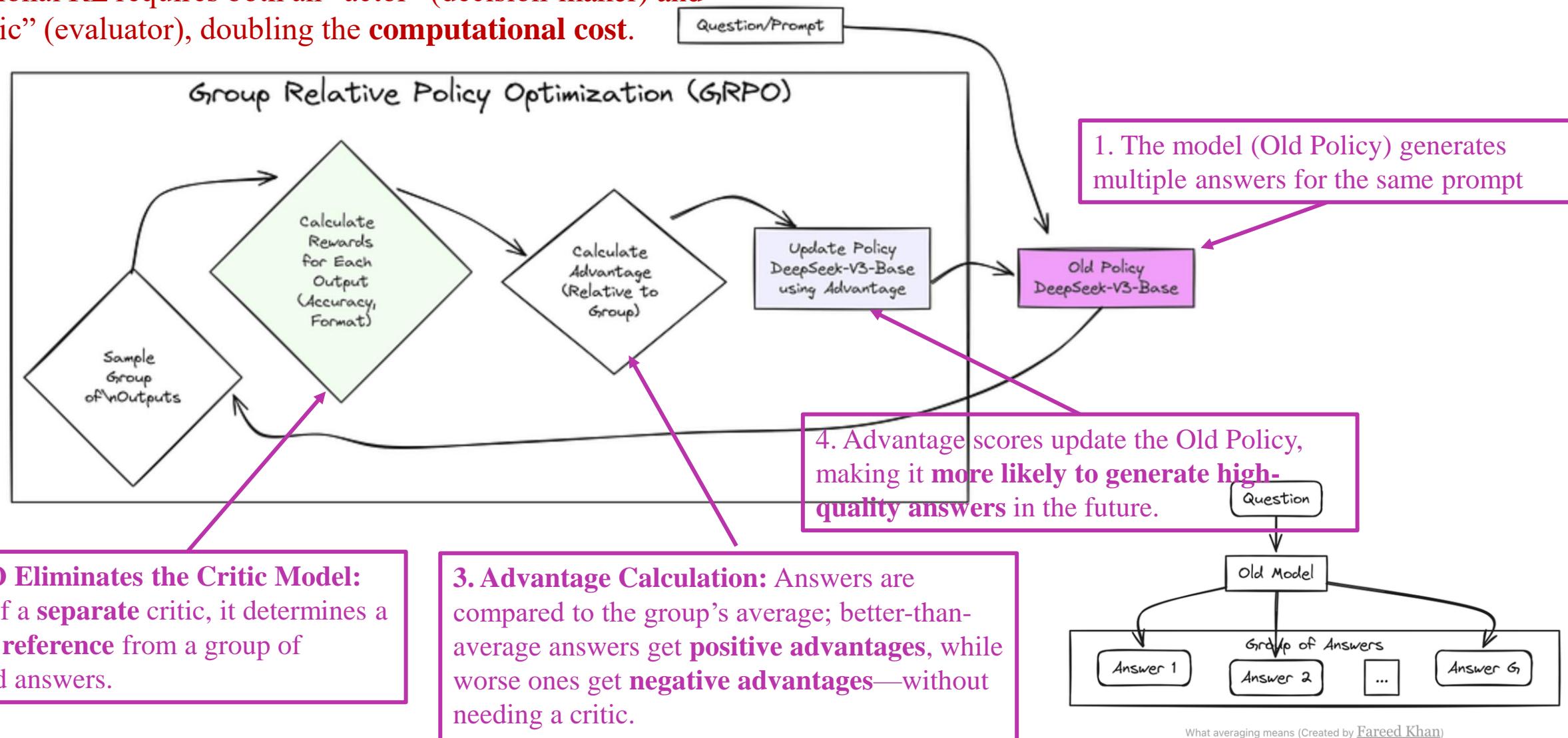
DeepSeek-R1-Zero

DeepSeek V3 is the Policy Model (Actor) In RL Setup



GRPO (Group Relative Policy Optimization)

Traditional RL requires both an “actor” (decision-maker) and a “critic” (evaluator), doubling the **computational cost**.



GRPO

- In order to save the training costs of RL, we adopt GRPO, which foregoes the critic model that is typically the same size as the policy model, and estimates the baseline from group scores instead.

Specifically, for each question q , GRPO samples **a group of outputs** $\{o_1, o_2, \dots, o_G\}$ from the **old policy** $\pi_{\theta_{old}}$ and then optimizes **the policy model** π_θ by maximizing the following objective:

$$\begin{aligned} \mathcal{J}_{GRPO}(\theta) &= \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)] \\ &\quad \frac{1}{G} \sum_{i=1}^G \left(\min \left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)} A_i, \text{clip} \left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_i \right) - \beta \mathbb{D}_{KL}(\pi_\theta || \pi_{ref}) \right), \end{aligned} \quad (1)$$

$$\mathbb{D}_{KL}(\pi_\theta || \pi_{ref}) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1, \quad (2)$$

where ε and β are hyper-parameters, and A_i is the advantage, computed using a group of rewards $\{r_1, r_2, \dots, r_G\}$ corresponding to the outputs within each group:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_G\})}{\text{std}(\{r_1, r_2, \dots, r_G\})}. \quad (3)$$

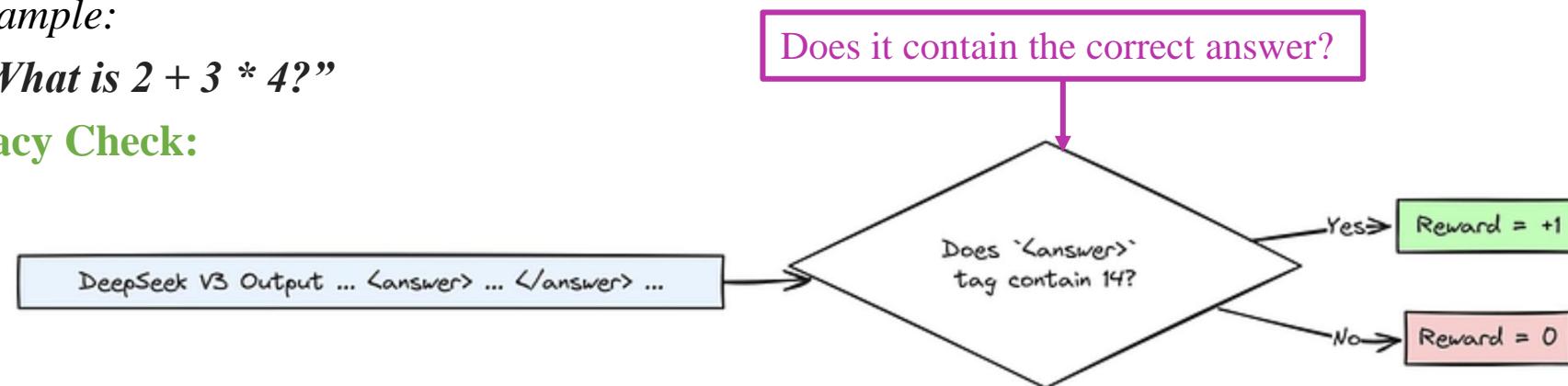
Reward Modeling for DeepSeek R1 Zero

- The reward is the source of the training signal, which decides the optimization direction of RL (Simple Rule-Based Rewards).
- Accuracy rewards:** The accuracy reward model evaluates whether the response is correct.
- Format Rewards:** enforces the model to put its thinking process between ‘<think>’ and ‘</think>’ tags.

📌 For example:

*prompt: “What is 2 + 3 * 4?”*

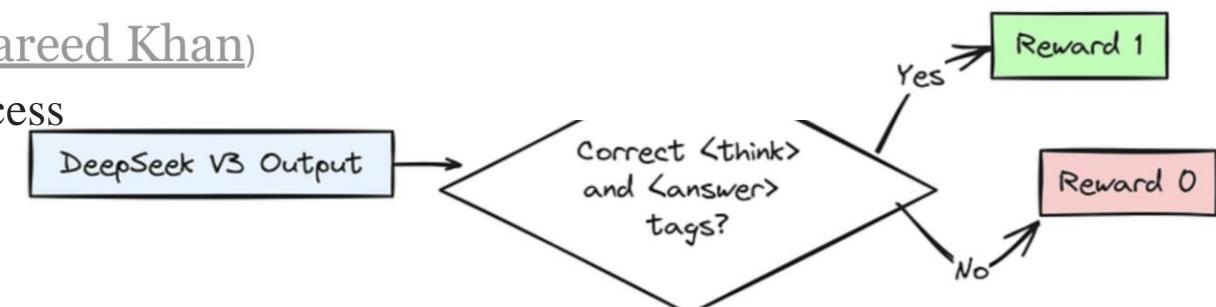
1. Accuracy Check:



2. Format Check:

Rule based check (Created by Fareed Khan)

Check if the model output *correctly* encloses the reasoning process within <think>...</think> and the final answer within <answer>...</answer>.



Training Template for Reward

This template acts as a blueprint, instructing **DeepSeek-V3-Base** on how to structure its responses during the Reinforcement Learning process.

- 📌 For the example problem, the **input** would look like:

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant **first thinks** about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within **<think> </think>** and **<answer> </answer>** tags respectively, i.e., **<think>** reasoning process here **</think>** **<answer>** answer here **</answer>**. User: **What is $2 + 3 * 4$?**.
Assistant:

We expect the model to generate an **output** that conforms to the template, like:

```
<think>
Order of operations: multiply before add.  $3 * 4 = 12$ .  $2 + 12 = 14$ 
</think>
<answer>
14
</answer>
```

The DeepSeek team intentionally kept this template simple and focused on structure, not on telling the model how to reason.

An Example

1. Generating Outputs Using the Old Policy (DeepSeek-V3-Base)

Before RL updates, the model **samples multiple responses** ($G = 4$) for a given prompt.

📌 Example outputs for *What is $2 + 3 \times 4$?*:

- o1: Incorrect order of operations. ✗
- o2: Correct answer with reasoning. ✓
- o3: Correct answer but missing reasoning tags. ✓
- o4: Incorrect reasoning and answer. ✗

2. Reward Assignment (Rule-Based System)

Each output is evaluated based on:

- **Accuracy Reward:** Whether the final answer is correct.
- **Format Reward:** Proper reasoning structure with `$` and `$` tags.

Correct and well-structured outputs receive **higher rewards**, while incorrect or incomplete outputs receive **lower rewards**.

3. Computing the Advantage Score

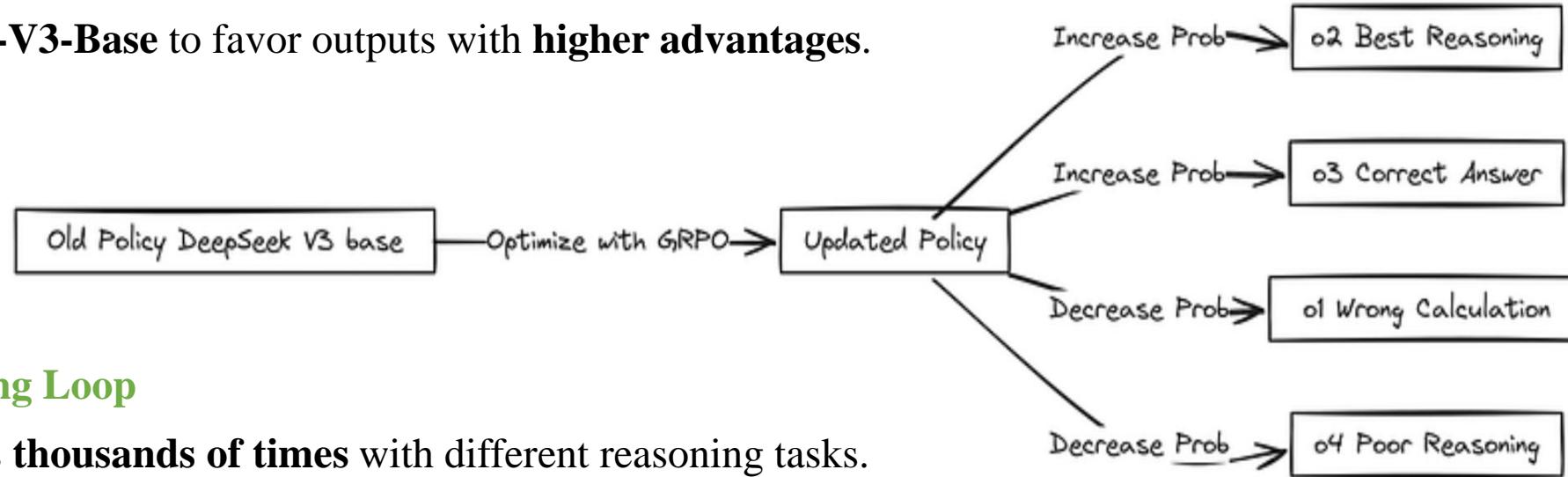
Mean reward and standard deviation are calculated across all sampled outputs.

- **Advantage score** is determined by comparing each output's reward against the average:
- **Positive advantage:** Encouraged (o2, o3). ✓
- **Negative advantage:** Discouraged (o1, o4). ✗

An Example

4. Policy Update via GRPO

Updates **DeepSeek-V3-Base** to favor outputs with **higher advantages**.



5. Iterative Training Loop

The process repeats **thousands of times** with different reasoning tasks.

Over time, **DeepSeek-R1 Zero** improves by:

- Applying correct order of operations.
- Providing logical reasoning steps.
- Consistently using the correct format.

GRPO working (Created by Fareed Khan)

DeepSeek-R1 Zero to DeepSeek-R1

Two main problems with DeepSeek-R1 Zero:

1. The tags captured reasoning, but the output was often **difficult for humans to read and analyze**.
2. When given multi-lingual prompts, the model sometimes **mixed languages within the same response**.

Two key questions arise:

1. Can reasoning performance be further improved with a small amount of high-quality data as a cold start?
2. How to train a user-friendly model with clear **Chains of Thought (CoT)** and **strong general capabilities**?

➤ Next Steps:

- To fix these issues, researchers refined **DeepSeek-R1 Zero** into **DeepSeek-R1** 
 - **Goal: Better reasoning + Better output**
 - **Pipeline Structure: Four-stage training process.**
- ✓ Cold Start >> Ensures a structured, readable format with a summary at the end of each response.
- ✓ Reasoning-oriented Reinforcement Learning
- ✓ Rejection Sampling and Supervised Fine-Tuning
- ✓ Non-Reasoning data

Step 1: Cold Start Data Collection (1)

1. Few-Shot Prompting with Long CoT:

- Provided step-by-step Chain-of-Thought (CoT) examples.
- Used | special_token | markers to clearly structure reasoning and answers.

📌 For $2 + 3 * 4$?, they might show prompts like this:

Problem Examples with Solutions:

Problem: What's the square root of 9 plus 5?

Solution: | special_token | First, find the square root of 9, which is 3. Then, add 5 to 3. $3 + 5$ equals 8. | special_token | Summary: The answer is 8.

Problem: Train travels at 60 mph for 2 hours, how far?

Solution: | special_token | Use the formula: Distance = Speed times Time. Speed is 60 mph, Time is 2 hours.

Distance = $60 * 2 = 120$ miles. | special_token | Summary: Train travels 120 miles.

Problem: What is $2 + 3 * 4$?

Solution:

Output:

| special_token | Following order of operations (PEMDAS/BODMAS), do multiplication before addition. So, first calculate $3 * 4 = 12$. Then, add 2 to 12. $2 + 12 = 14$. | special_token | Summary: The answer is 14.

Cold Start Data Collection (2)

2. Direct Prompting

Directly prompting the model to not only solve problems but also to explicitly show its reasoning step-by-step and then double-check its answer.

📌 For What is $2 + 3 * 4$? the prompt could be:

Problem: Solve this, show reasoning **step-by-step**, and verify: What **is** $2 + 3 * 4$?

And they do expect an output that includes both reasoning steps and a verification part:

| special_token | Reasoning: To solve $2 + 3 * 4$, I need to use order of operations. Multiplication comes before addition. Step 1: Calculate $3 * 4 = 12$. Step 2: Add 2 to the result from step 1: $2 + 12 = 14$. Verification: Checking order of operations again, yes, multiplication is before addition. Calculation looks right. | special_token | Summary: The answer is 14.

3. Refining the Results Through Post-processing by Human Annotators

Step 2: Reasoning-Oriented RL in DeepSeek-R1

After fine-tuning on cold-start data, the model undergoes large-scale RL training similar to DeepSeek-R1-Zero (**Same GRPO algorithm**, but with an **upgraded Reward System**).

◆ **Focus areas:** reasoning-intensive tasks (coding, mathematics, science, logic).

- **Key Upgrade: Language Consistency Reward**

Fixes the R1 Zero issue where reasoning mixed multiple languages. (If a question is asked in English, the reasoning and answer **must also be in English**.)

- **RL Training Loop**
 - a. **Generate multiple outputs** using the model.
 - b. **Assign rewards**, now including **Language Consistency Reward**.
 - c. **Use GRPO** to calculate **advantage scores**.
 - d. **Train the model** to favor high-reward responses.
 - e. **Repeat** the process to refine reasoning and consistency.

Step 3: Rejection Sampling & SFT

SFT after RL Convergence:

- Uses the converged checkpoint to collect new SFT data.
- Unlike the initial cold-start data, which primarily focuses on reasoning, this stage incorporates data from other domains to **enhance the model's capabilities in writing, role-playing, and other general-purpose tasks**.

Fine-tune DeepSeek-V3-Base again using the data generated by DeepSeek-R1-Zero.

Reasoning Data Collection:

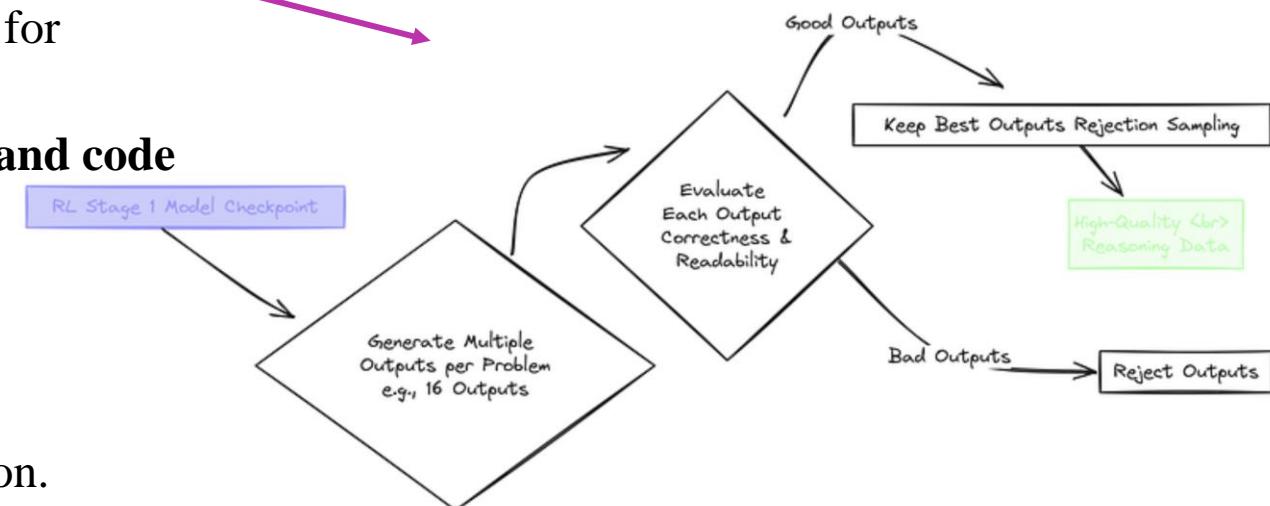
- Curated reasoning prompts **with rejection sampling** from the RL-trained checkpoint.
- Expanded dataset includes generative reward models for judgment.
- Filters out **mixed-language CoT, long paragraphs, and code blocks** for better readability.

>> 600k reasoning-related samples collected.

Non-Reasoning Data Collection:

Covers writing, factual QA, self-cognition, and translation.

>> 200k non-reasoning samples collected.



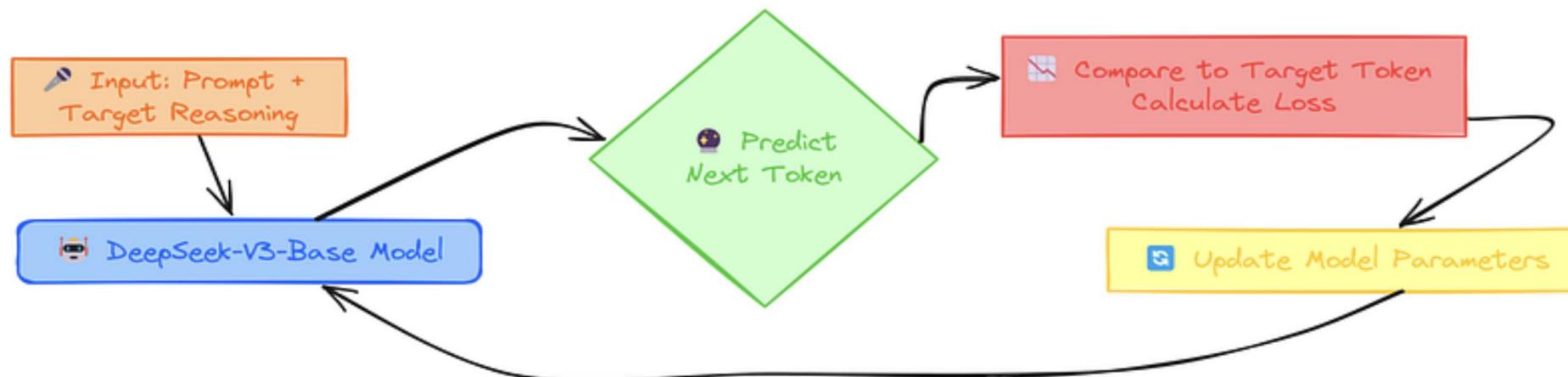
Rejection Sampling (Created by [Fareed Khan](#))

Supervised Fine-Tuning

800k total samples used for fine-tuning DeepSeek-V3-Base over two epochs.

Training Process (Step-by-Step):

1. **Input Prompt + Target Reasoning** → Given to the model as a training example.
2. **Predict Next Token** → Model generates the next word in the reasoning sequence.
3. **Compare to Target Token (Calculate Loss)** → Measures how far the prediction is from the correct token.
4. **Update Model Parameters** → Backpropagation + optimizer adjust model weights.
5. **Repeat Process** → Loops over thousands of examples, refining the model's structured reasoning skills.



SFT Process (Created by [Fareed Khan](#))

<https://levelup.gitconnected.com/drawing-deepseek-r1-architecture-and-training-process-from-scratch-72043da33955>

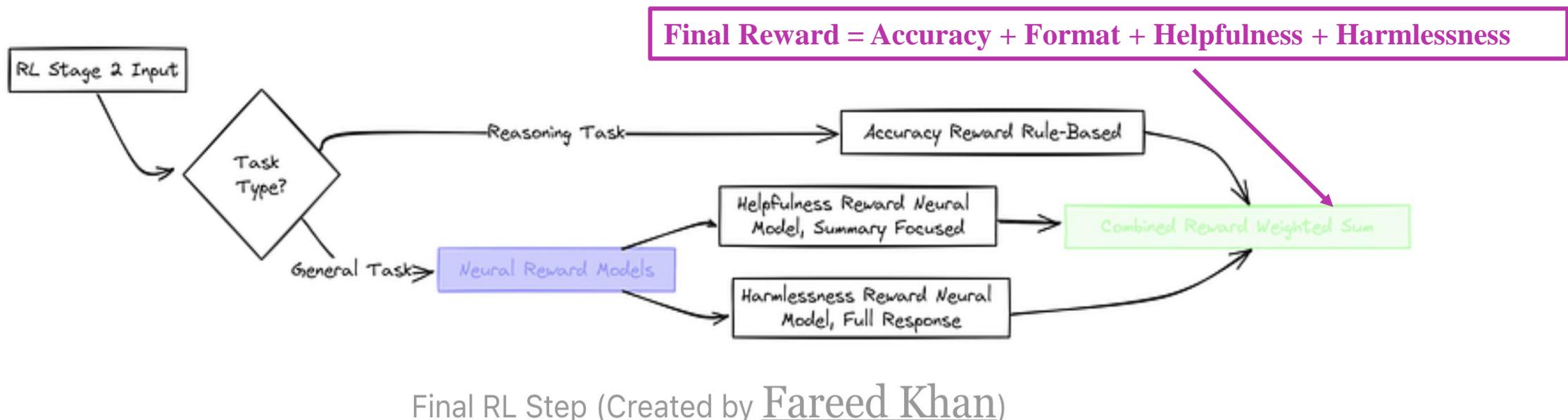
Step 4: RL for All Scenarios

- alignment with human Preferences

Implement a secondary RL stage aimed at **improving the model's helpfulness and harmlessness** while simultaneously refining its reasoning capabilities.

Expanded Reward System

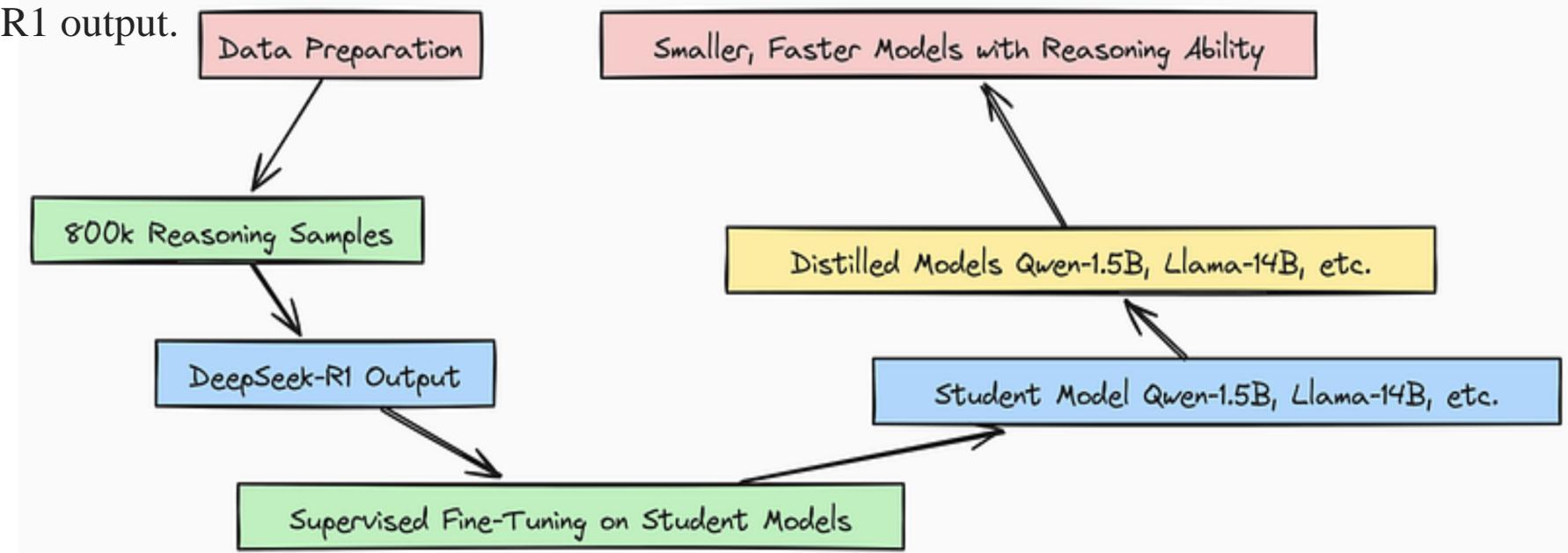
- **Helpfulness:** Checks if the response provides **useful context**, beyond just the answer. Ensures that summaries (if included) are informative.
- **Harmlessness:** Evaluates if the response is **safe, unbiased, and appropriate**.



Distillation

To equip more efficient **smaller models** with reasoning capabilities like DeepSeek-R1, we directly fine-tuned open-source models like **Qwen** and **Llama** using the **800k samples curated with DeepSeek-R1**.

- **DeepSeek-R1 Output:** For each sample, the output from the teacher model (DeepSeek-R1) is used as the target for the student model.
- **Supervised Fine-Tuning (SFT):** The student models (e.g., Qwen-1.5B, Llama-14B) are fine-tuned on these 800k samples to match the DeepSeek-R1 output.



Distillation of R1 (Created by Fareed Khan)

<https://levelup.gitconnected.com/drawing-deepseek-r1-architecture-and-training-process-from-scratch-72043da33955>

LLM breakdown

Here we take Qwen2.5/0.5b as an example to show the structure, tokenization and embedding of LLMs

We can see the model architecture by printing the model

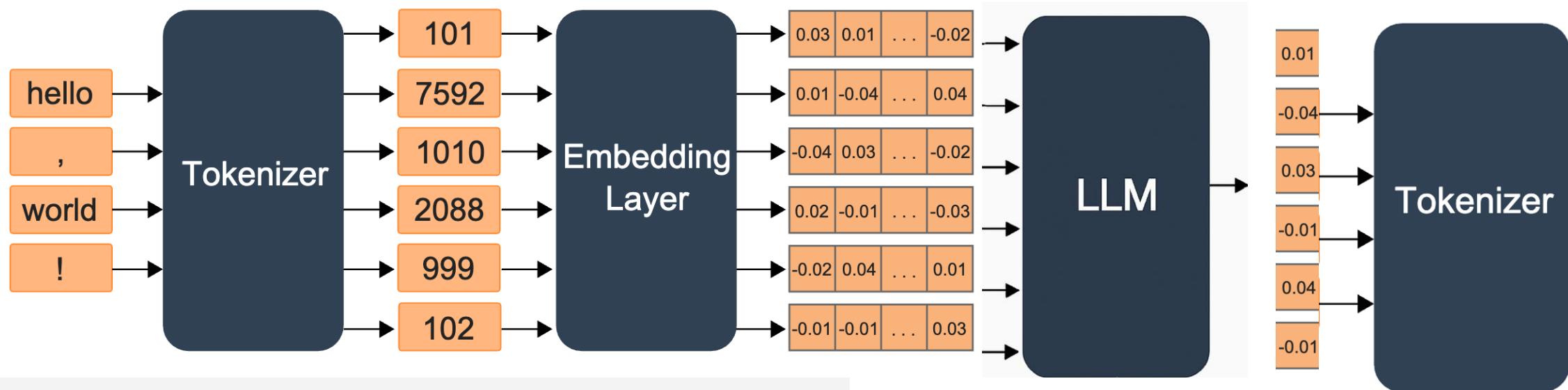
```
name= "Qwen/Qwen2.5-0.5B"  
model = AutoModelForCausalLM.from_pretrained(name)  
tokenizer = AutoTokenizer.from_pretrained(name)  
print(model)
```

What are all those modules?

What is the tokenizer?

```
Qwen2ForCausalLM(  
    (model): Qwen2Model(  
        (embed_tokens): Embedding(151936, 896)  
        (layers): ModuleList(  
            (0-23): 24 x Qwen2DecoderLayer(  
                (self_attn): Qwen2SdpaAttention(  
                    (q_proj): Linear(in_features=896, out_features=896, bias=True)  
                    (k_proj): Linear(in_features=896, out_features=128, bias=True)  
                    (v_proj): Linear(in_features=896, out_features=128, bias=True)  
                    (o_proj): Linear(in_features=896, out_features=896, bias=False)  
                    (rotary_emb): Qwen2RotaryEmbedding()  
                )  
                (mlp): Qwen2MLP(  
                    (gate_proj): Linear(in_features=896, out_features=4864, bias=False)  
                    (up_proj): Linear(in_features=896, out_features=4864, bias=False)  
                    (down_proj): Linear(in_features=4864, out_features=896, bias=False)  
                    (act_fn): SiLU()  
                )  
                (input_layernorm): Qwen2RMSNorm((896,), eps=1e-06)  
                (post_attention_layernorm): Qwen2RMSNorm((896,), eps=1e-06)  
            )  
            (norm): Qwen2RMSNorm((896,), eps=1e-06)  
        )  
        (lm_head): Linear(in_features=896, out_features=151936, bias=False)  
    )
```

LLM breakdown



```
input_text = "I love UNC"
```

```
inputs = tokenizer(input_text, return_tensors="pt")
```

Tokenization: Word -> Int

```
outputs = model(**inputs)
```

Forward passing

```
last_token_logits = outputs.logits[0, -1, :]
```

```
probs = F.softmax(last_token_logits, dim=-1)
```

Distribution of next token

```
top_probs, top_indices = torch.topk(probs, 10)
```

```
top_tokens = [tokenizer.decode([idx]) for idx in top_indices]
```

```
for token, prob in zip(top_tokens, top_probs):
    print(f'{token}: {prob.item():.4f}')
```

Decode: Int -> Word

LLM breakdown - Tokenization

The essence of tokenization lies in segmenting a continuous text sequence into discrete fundamental units, or *tokens*. This process is akin to creating a "vocabulary" for the computer, enabling it to understand and generate text like humans do—by combining basic units. A good tokenization algorithm needs to strike a balance across multiple dimensions, including vocabulary size, semantic integrity, and the handling of out-of-vocabulary (OOV) words.

Word level tokenization:

- The vocabulary size is enormous, leading to substantial computational costs.
- Cannot deal with new words (OOV)

Oxford English Dictionary lists around 171,476 words currently in use—this doesn't even account for other languages.

Character level tokenization:

- The vocabulary size is small and there is no OOV issue.
- The text sequence becomes very long, increasing the difficulty of processing.

Subword level tokenization:

- Strike a balance. Keep common words intact, while rare words are split into smaller units.
- Balances vocabulary size and semantic representation capability.
- **Used by LLMs**

LLM breakdown - Tokenization

Tiktokenizer by OpenAI

Tokenization methods used by main-stream models:

Model	Tokenizer
BERT	WordPiece
GPT-2, GPT-3	BPE (tiktoken)
T5	SentencePiece
ALBERT	SentencePiece
XLNet	SentencePiece
GPT-4	tiktoken

Tiktokenizer

The screenshot shows the Tiktokenizer interface for the gpt-3.5-turbo model. It displays two input fields: 'System' (containing 'You are a helpful assistant') and 'User' (containing 'Content'). The total 'Token count' is 18, and the 'Price per prompt' is \$0.000018. Below the inputs, an 'Add message' button is visible. The tokenized output is shown in a large text area, which includes tokens for system and user messages, and tokens for the AI's response. A checkbox for 'Show whitespace' is present at the bottom. The URL https://tiktokenizer.vercel.app/ is also visible.

gpt-3.5-turbo

You are a helpful assistant

Content

Token count
18

Price per prompt
\$0.000018

<|im_start|>system
You are a helpful assistant<|im_end|>
<|im_start|>user
<|im_end|>
<|im_start|>assistant

[100264, 9125, 198, 2675, 527, 264, 11190, 18328, 100265, 198, 100264, 882, 198, 100265, 198, 100264, 78191, 198]

Show whitespace

Built by dqbd. Created with the generous help from Diagram.



<https://tiktokenizer.vercel.app/>

LLM breakdown – Tokenization -BPE

Byte Pair Encoding (BPE)

1. Start with a base vocabulary of individual characters.
2. Iteratively merge the most frequent adjacent pairs of tokens.
3. And we continue like this until we reach the desired vocabulary size.

Suppose we have the following word frequencies: ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

Step 1: Initialize

Split all words into characters:

("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)

Step 2: First Merge

Most frequent pair: ("u", "g") → 20 occurrences → Merge into "ug"

Updated corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)

Now we have a smaller vocabulary.

Step 3: Continue Merging

Continue merging until the desired vocabulary size is reached

Reference: <https://huggingface.co/learn/nlp-course/en/chapter6/5>

Content

1. Introduction to Large Language Models
2. Popular Large Language Model Architectures
- 3. Make LLMs More Suitable for Your Downstream Application**
4. Challenges
5. Applications

Optimizing LLMs for Downstream Tasks

1. No Training Required (Prompt Engineering)

- **Zero-shot / Few-shot:** Providing descriptions and examples.
- **Chain of Thought (CoT):** Enhancing logical reasoning.
- **Prompt Optimization:** Iterative refinement of prompts.
 - ◆ **Pros:** No computational cost, easy to implement.
 - ◆ **Cons:** Limited effectiveness for complex tasks.

2. Training Required (Fine-Tuning)

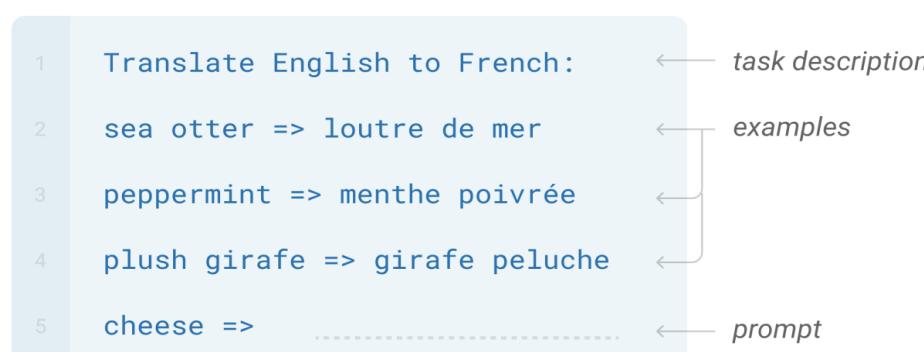
- **Full Fine-Tuning:** Adjusting all model parameters for task specificity.
- **Parameter-Efficient Fine-Tuning (PEFT, e.g., LoRA, Adapters):** Modifying only small parts of the model.
- **Instruction Tuning:** Enhancing task adaptability.
- **Domain Adaptation:** Training on specialized datasets (e.g., biomedical data).
 - ◆ **Pros:** High precision, better task adaptation.
 - ◆ **Cons:** Requires computational resources, risk of overfitting.

In-Context Learning (ICL)

Technique	Definition	Key Feature
In-Context Learning (ICL)	LLMs learn tasks from provided examples or instructions within the prompt, without retraining.	No weight updates, adapts based on context.
Few-Shot Prompting	A small set of examples guides the model's response.	Demonstrations help model understand task format.
Zero-Shot Prompting	The model performs a task without any examples.	Relies solely on generalization capability.

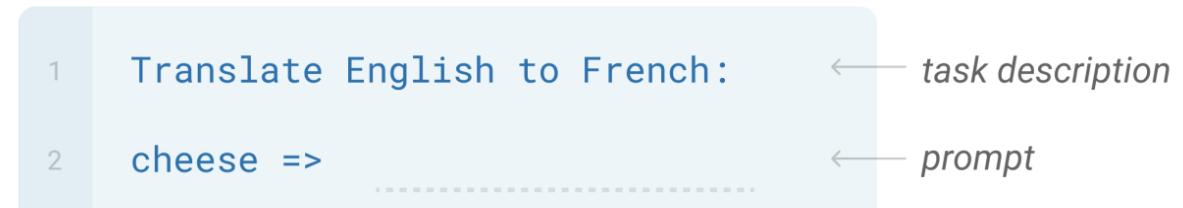
Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



- ◆ Few-shot improves accuracy, while zero-shot relies on model's pre-learned knowledge

Useful tips for designing prompts (1)

Ingredient	Collected Prompts
Task Description	<p>T1. Make your prompt as detailed as possible, e.g., "Summarize the article into a short paragraph within 50 words. The major storyline and conclusion should be included, and the unimportant details can be omitted."</p> <p>T2. It is helpful to let the LLM know that it is an expert with a prefixed prompt, e.g., "You are a sophisticated expert in the domain of computer science."</p> <p>T3. Tell the model more what it should do, but not what it should not do.</p> <p>T4. To avoid the LLM to generate too long output, you can just use the prompt: "Question: Short Answer: ". Besides, you can also use the following suffixes, "in a or a few words", "in one of two sentences".</p>
Input Data	<p>I1. For the question required factual knowledge, it is useful to first <u>retrieve relevant documents</u> via the search engine, and then <u>concatenate them into the prompt</u> as reference.</p> <p>I2. To highlight some important parts in your prompt, please <u>use special marks</u>, e.g., quotation ("") and line break (\n). You can also use both of them for emphasizing.</p>
Contextual Information	<p>C1. For complex tasks, you can clearly describe the required intermediate steps to accomplish it, e.g., "Please answer the question step by step as: Step 1 - Decompose the question into several sub-questions, . . ."</p> <p>C2. If you want LLMs to provide the score for a text, it is necessary to provide a <u>detailed description about the scoring standard</u> with examples as reference.</p> <p>C3. When LLMs generate text according to some context (e.g., making recommendations according to purchase history), instructing them with the explanation about the generated result conditioned on context is helpful to improve the quality of the generated text.</p> <p>C4. An approach similar to tree-of-thoughts but can be done in one prompt: e.g., Imagine three different experts are answering this question. All experts will write down one step of their thinking, then share it with the group of experts. Then all experts will go on to the next step, etc. If any expert realizes they're wrong at any point then they leave. The question is</p>

Useful tips for designing prompts (2)

Demonstration

- D1. Well-formatted in-context exemplars are very useful, especially for producing the outputs with complex formats.
- D2. For few-shot chain-of-thought prompting, you can also use the prompt “Let’s think step-by-step”, and the few-shot examples should be separated by “\n” instead of full stop.
- D3. You can also retrieve similar examples in context to supply the useful task-specific knowledge for LLMs. To retrieve more relevant examples, it is useful to first obtain the answer of the question, and then concatenate it with the question for retrieval.
- D4. The diversity of the in-context exemplars within the prompt is also useful. If it is not easy to obtain diverse questions, you can also seek to keep the diversity of the solutions for the questions.
- D5. When using chat-based LLMs, you can decompose in-context exemplars into multi-turn messages, to better match the human-chatbot conversation format. Similarly, you can also decompose the reasoning process of an exemplars into multi-turn conversation.
- D6. Complex and informative in-context exemplars can help LLMs answer complex questions.
- D7. As a symbol sequence can typically be divided into multiple segments (e.g., $i_1, i_2, i_3 \rightarrow i_1, i_2$ and i_2, i_3), the preceding ones can be used as in-context exemplars to guide LLMs to predict the subsequent ones, meanwhile providing historical information.
- D8. Order matters for in-context exemplars and prompts components. For very long input data, the position of the question (first or last) may also affect the performance.
- D9. If you can not obtain the in-context exemplars from existing datasets, an alternative way is to use the zero-shot generated ones from the LLM itself

Other Designs

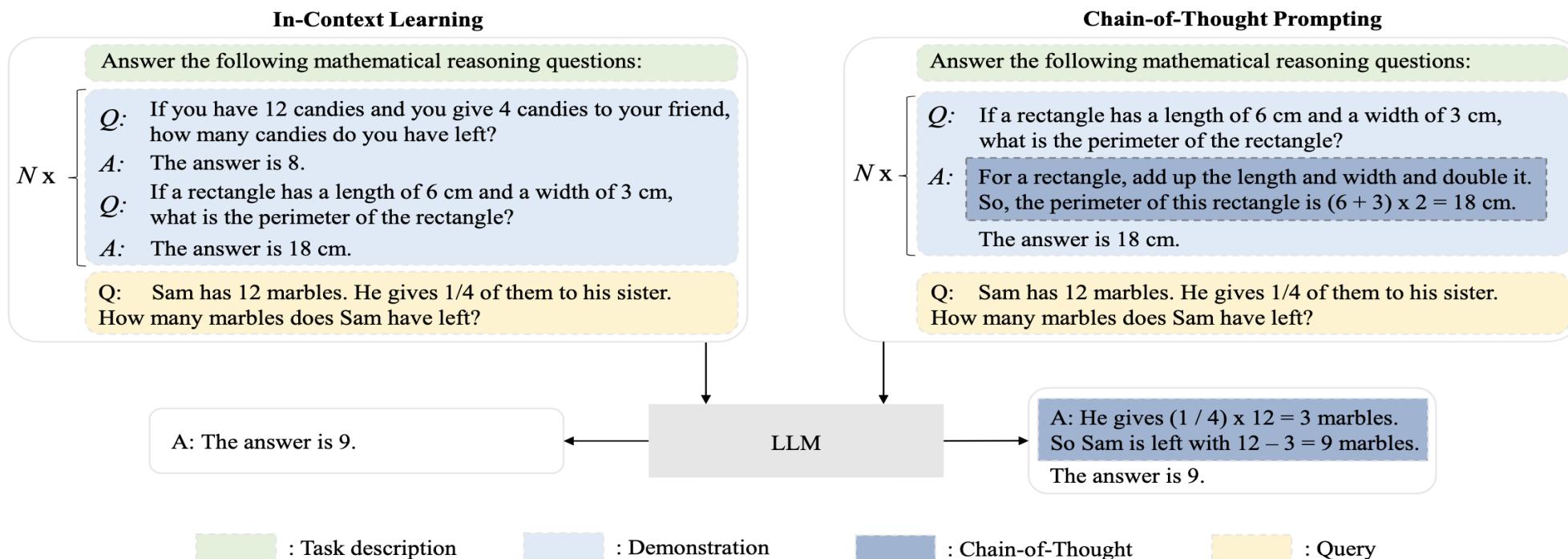
- O1. Let the LLM check its outputs before draw the conclusion, e.g., “Check whether the above solution is correct or not.”
- O2. If the LLM can not well solve the task, you can seek help from external tools by prompting the LLM to manipulate them. In this way, the tools should be encapsulated into callable APIs with detailed description about their functions, to better guide the LLM to utilize the tools.
- O3. The prompt should be self-contained, and better not include pronouns (e.g., it and they) in the context.
- O4. When using LLMs for comparing two or more examples, the order affects the performance a lot.
- O5. Before the prompt, assigning a role for the LLM is useful to help it better fulfill the following task instruction, e.g., “I want you to act as a lawyer .
- O6. OpenAI models can perform a task better in English than other languages. Thus, it is useful to first translate the input into English and then feed it to LLMs.
- O7. For multi-choice questions, it is useful to constrain the output space of the LLM. You can use a more detailed explanation or just imposing constraints on the logits.
- O8. For sorting based tasks (e.g., recommendation), instead of directly outputting the complete text of each item after sorting, one can assign indicators (e.g., ABCD) to the unsorted items and instruct the LLMs to directly output the sorted indicators.

ICL vs. Chain-of-Thought (CoT)

Limitations of ICL:

- **Poor Generalization for Complex Tasks:** Struggles with multi-step reasoning or problems requiring deeper logical connections.
- Performance depends on the number and quality of demonstration examples.

CoT: Instead of simply constructing the prompts with input-output pairs like ICL, CoT prompting further incorporates intermediate reasoning steps, which serve as the bridge between inputs and outputs.



CoT vs. Direct Reasoning

Direct Reasoning

Input

A small sunflower has 3 dozen seeds and a large sunflower has 50% more seeds than a small sunflower. How many sunflower seeds are there altogether?

Output

54



Chain-of-Thoughts (Few-Shot)

Input

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been $21 - 15 = 6$. The answer is 6.

Q: A small sunflower has 3 dozen seeds and a large sunflower has 50% more seeds than a small sunflower. How many sunflower seeds are there altogether?

A:

Chain-of-Thoughts (Zero-Shot)

Input

A small sunflower has 3 dozen seeds and a large sunflower has 50% more seeds than a small sunflower. How many sunflower seeds are there altogether? Let's think step by step.

Output

A small sunflower has $3 \times 12 = 36$ seeds. A large sunflower has 50% more seeds than a small sunflower, which is $36 \times 0.5 = 18$ more seeds. So a large sunflower has $36 + 18 = 54$ seeds. Altogether, there are $36 + 54 = 90$ sunflower seeds.

The answer is 90.



Exemplars

Question

Instruction

Rationale

Answer

Direct Reasoning for Agent

Input

You are in the middle of a room. Looking quickly around you, you see a drawer 2, a shelf 6, a shelf 9, a cabinet 2, a sofa 1, a safe 1, a garbage can 1, and a armchair 1.

Your task is to: put the pillow on sofa.

Output

> go to drawer 2
You arrive at loc 4. The drawer 2 is closed.
> Open the drawer
You open the drawer 2. The drawer 2 is open. In it, you see a pillow 1.
> take pillow 1 from drawer 2
Nothing happens
> go to sofa 1
Nothing happens
> put pillow 1 in/on sofa 1
Nothing happens



Chain-of-Thought for Agent

Input

You are in the middle of a room. Looking quickly around you, you see a drawer 2, a shelf 6, a shelf 9, a cabinet 2, a sofa 1, a safe 1, a garbage can 1, and a armchair 1.

Your task is to: put the pillow on sofa.

Output

First I need to find the pillow which is more likely to be in the drawer 2.
> go to drawer 2
You arrive at loc 4. The drawer 2 is closed.
The drawer is closed. I should open it first
> Open the drawer
You open the drawer 2. The drawer 2 is open. In it, you see a pillow 1.
> take pillow 1 from drawer 2
You pick up the pillow 1 from the drawer 2.
Now I found the pillow. I should put it on the sofa.
> go to sofa 1
You arrive at loc 3.
> put pillow 1 in/on sofa 1
You won!



CoT refers to a series of intermediate reasoning steps that are generated to solve a problem or arrive at an answer

Formulation Shifts of CoT

Chain-of-Thoughts

Question

Rationale

Answer

A small sunflower has 3 dozen seeds and a large sunflower has 50% more seeds than a small sunflower. How many sunflower seeds are there altogether?

A small sunflower has $3 \times 12 = 36$ seeds. A large sunflower has 50% more seeds than a small sunflower, which is $36 \times 0.5 = 18$ more seeds. So a large sunflower has $36 + 18 = 54$ seeds. Altogether, there are $36 + 54 = 90$ sunflower seeds.

The answer is 90.

Program-of-Thoughts

Question Input

```
# Number of seeds in a small sunflower (3 dozen)
seeds_in_small_sunflower = 3 * 12

# Calculate the number of seeds in a large sunflower (50% more than small sunflower)
seeds_in_large_sunflower = seeds_in_small_sunflower * 1.5

# Calculate the total number of sunflower seeds
total_seeds =
seeds_in_small_sunflower +
seeds_in_large_sunflower

# Print the result
print("Total number of sunflower seeds:", total_seeds)
```

The answer is 90.

Table-of-Thoughts

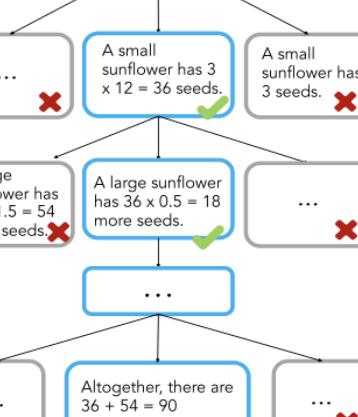
Question Input

step	subquestion	process	result
0	How many seeds does small sunflower have?	A small sunflower has $3 \times 12 = 36$ seeds	36 seeds
1	How many seeds does large sunflower have?	A large sunflower has $36 \times 1.5 = 54$ seeds	54 seeds
2	How many sunflower seeds are there altogether	Altogether, there are $36 + 54 = 90$ sunflower seeds	90 seeds

The answer is 90.

Tree-of-Thoughts

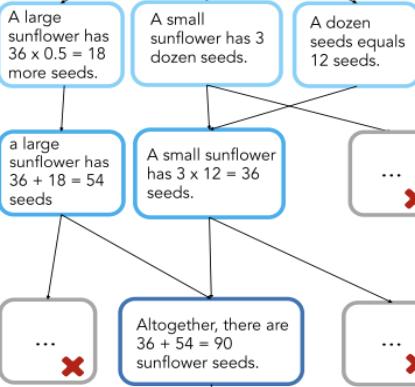
Question Input



The answer is 90.

Graph-of-Thoughts-Rationale

Question Input



The answer is 90.

Pre-training vs. Fine-Tuning

1. Definitions & Objectives

- **Pre-Training:** Learns general patterns (syntax, semantics) from large unlabeled datasets.

Goal: Captures universal knowledge.

- **Fine-Tuning:** Adapts a pre-trained model for specific tasks using smaller labeled datasets.

Goal: Enhances task-specific alignment (e.g., coding styles, domain jargon).

2. Technical Differences

- **Pre-Training:** Trains all parameters from scratch.

- **Fine-Tuning:**

➤ **Full Fine-Tuning:** Adjusts all parameters (risk of catastrophic forgetting).

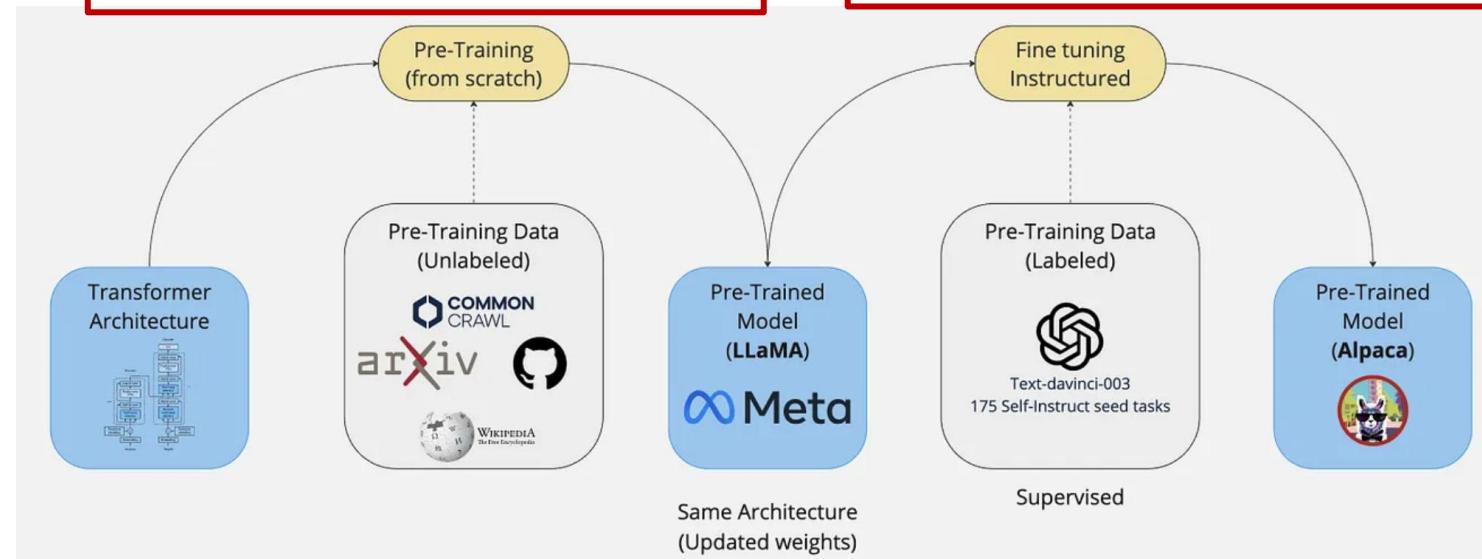
➤ **PEFT (Parameter-Efficient Fine-Tuning):** Updates only key subsets (e.g., LoRA, adapters) to retain core knowledge.

Challenges for Pre-training:

- High computational cost
- Risk of dataset-specific biases

Challenges for Fine-tuning:

- Balancing task-specific adaptation with knowledge retention
- Overfitting



PEFT - A Smarter Approach to Fine-Tuning

Why PEFT?

- **Large models** achieve remarkable performance but demand **high computational resources**.
- **Full fine-tuning** updates billions of parameters, making **customization costly and impractical**.
- **PEFT** optimizes efficiency by modifying only a **small subset of parameters** while leveraging pre-trained knowledge.

Key Advantages:

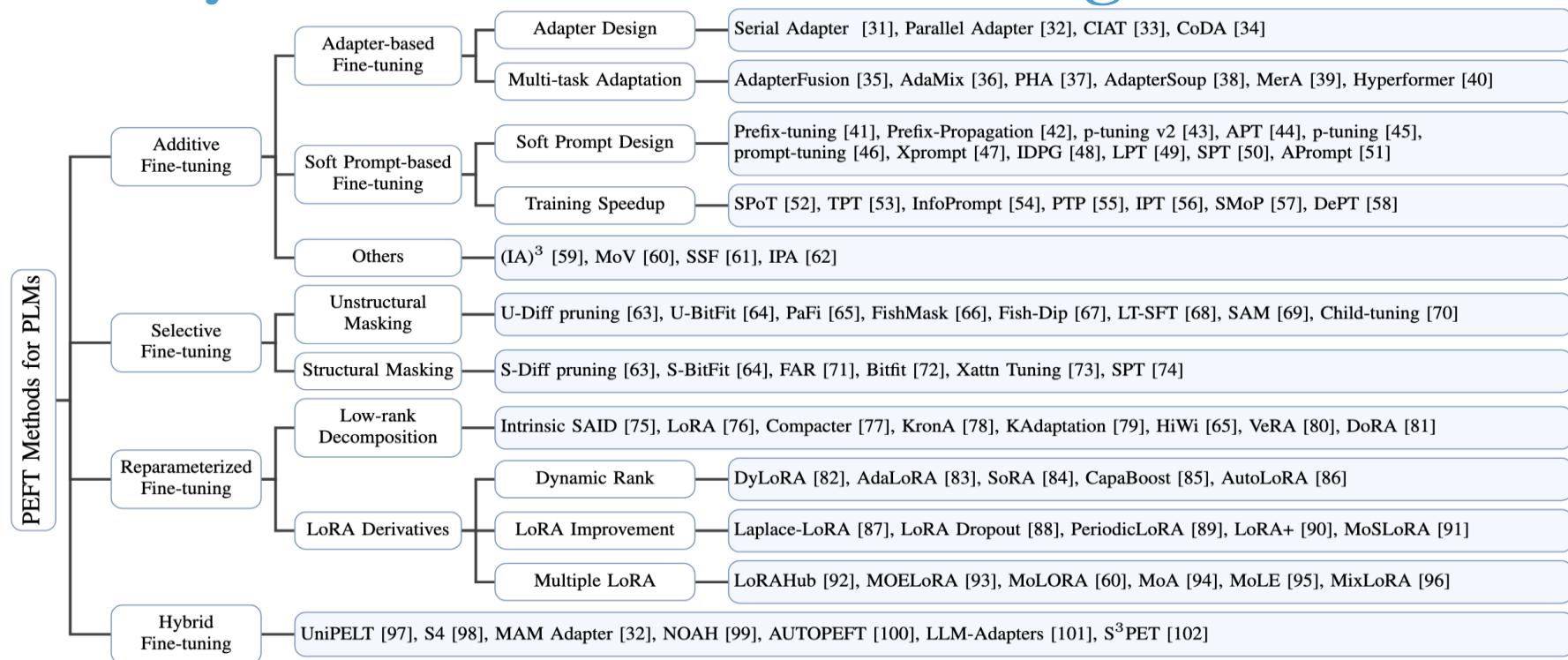
- **Lower Computational Cost:** Reduces GPU/TPU memory usage and processing power.
 - **Faster Training:** Updates fewer parameters, enabling quicker adaptation.
 - **Preserves Pre-Trained Knowledge:** Prevents catastrophic forgetting while enhancing task-specific performance.
 - **Balances Efficiency & Performance:** Adapts models effectively without full retraining.
- Different types of PEFT algorithms**
- The diagram illustrates three types of PEFT algorithms:

 - (a) Additive PEFT:** Shows a gray "Frozen" block and a pink "Learnable" block. Arrows point from both blocks to a central "Combine" block, which then points to an "Output" block.
 - (b) Selective PEFT:** Shows a gray "Frozen" block and a pink "Learnable" block. Arrows point from the "Frozen" block to an "Output" block, and from the "Learnable" block to the "Frozen" block.
 - (c) Reparameterization PEFT:** Shows a gray "Frozen" block and a pink "Learnable" block. Arrows point from the "Learnable" block to the "Frozen" block, which then points to an "Output" block. An "Input (train)" block also points to the "Learnable" block.

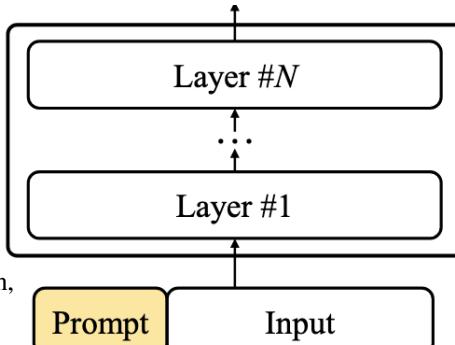
Legend: Frozen Learnable

Han, Z., Gao, C., Liu, J., Zhang, J., & Zhang, S. Q. (2024). Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey. *ArXiv*.
<https://arxiv.org/abs/2403.14608>

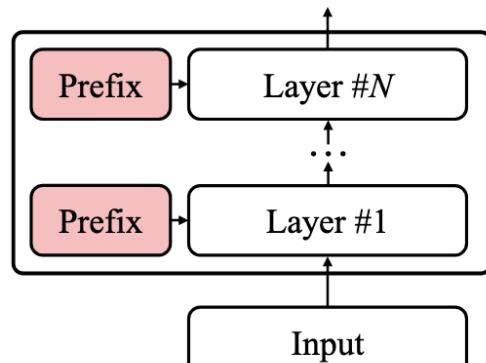
Taxonomy of PEFT Methods for Large Models



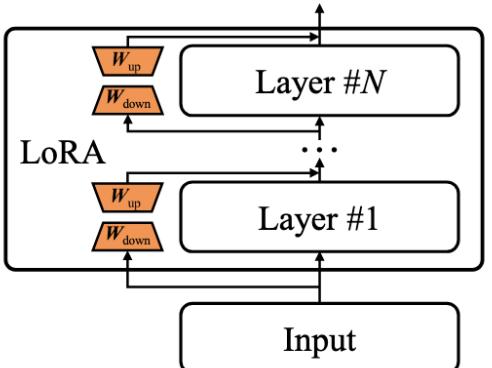
Prompt Tuning



Prefix Tuning



Low-Rank Adaption (Lora)

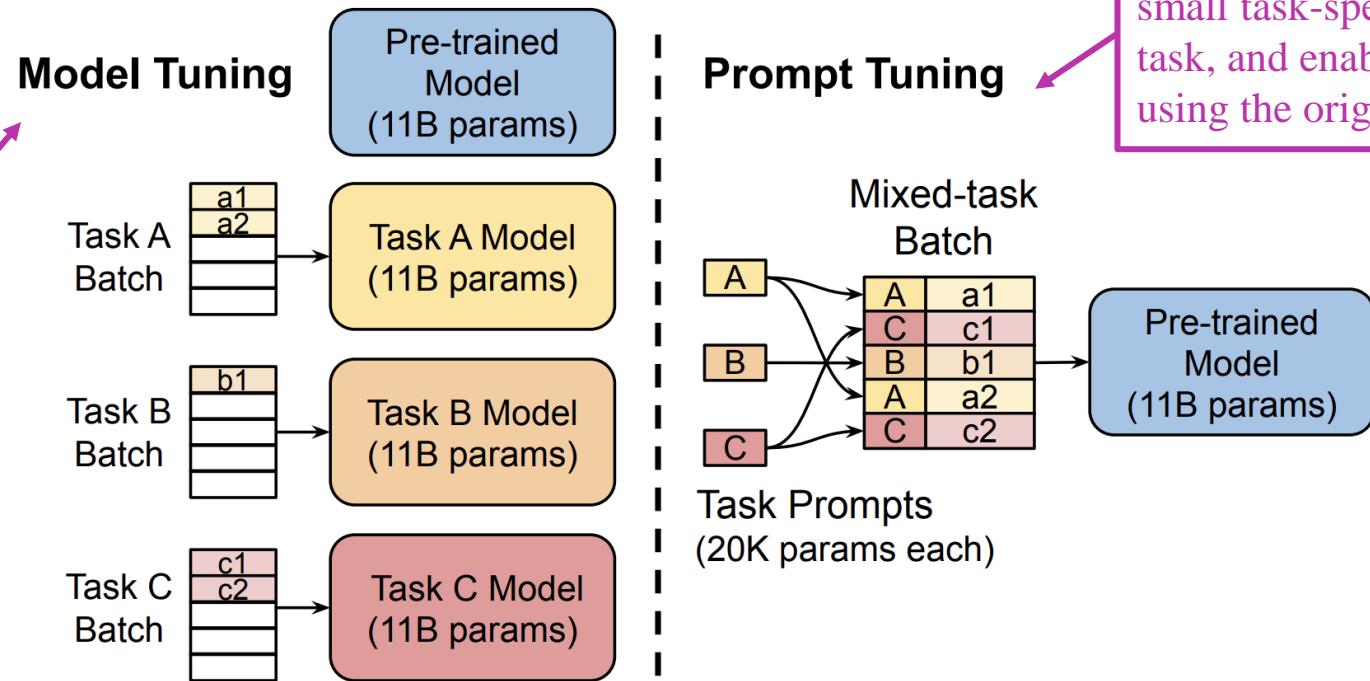


- The Three Types of **PEFT** We Will Introduce

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., . . . Wen, J. (2023). A Survey of Large Language Models. *ArXiv*. <https://arxiv.org/abs/2303.18223>

Prompt Tuning

Model tuning requires making a task-specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches.



What is Prompt-Tuning?

- Prompt tuning adjusts a set of extra parameters, known as "**soft prompts**," which are integrated into the model's input processing.
- Functions as a form of **information augmentation** by explicitly specifying the task and expected output.

How Prompting Works?

- Tells the model what task to perform and what kind of response to generate.

Prompt tuning only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pretrained model.

How Prompt Tuning Works?

Prompt-Tuning Pipeline Using T5 (Encoder-Decoder Model)

1. Given a sequence of n tokens $\{x_1, x_2, \dots, x_n\}$, The T5 first embeds them into a matrix: $X_e \in R^{ne}$, where e is the embedding dimension.
2. Soft-prompts are trainable parameters: $P_e \in R^{pe}$, where p is the prompt length.

3. How Soft-Prompting Works?

- **Concatenation:** The soft prompt P_e is prepended to the embedded input: $[P_e; X_e] \in R^{\{(p+n) \times e\}}$
- **Encoding & Decoding:** This combined matrix flows through the encoder-decoder as usual.
- **Training Objective:** The model maximizes the probability of output Y, only the prompt parameters P_e are updated, keeping the pretrained model frozen.

Advantages:

- Higher Memory Efficiency
- Multi-Task Capability
- Scaling Properties

Prefix-Tuning: Optimizing Continuous Prompts for Generation

What is Prefix-Tuning?

- A **lightweight alternative** to fine-tuning for **natural language generation (NLG)** tasks.
- Inspired by **prompting**, but instead of real tokens, it uses **learnable task-specific vectors** (prefix).

Intuition Behind Prefix-Tuning

Inspired by Prompting: Proper context can steer a language model (LM) without changing its core parameters.

Example: To generate *Obama*, the model is more likely to predict it if given *Barack* as context.

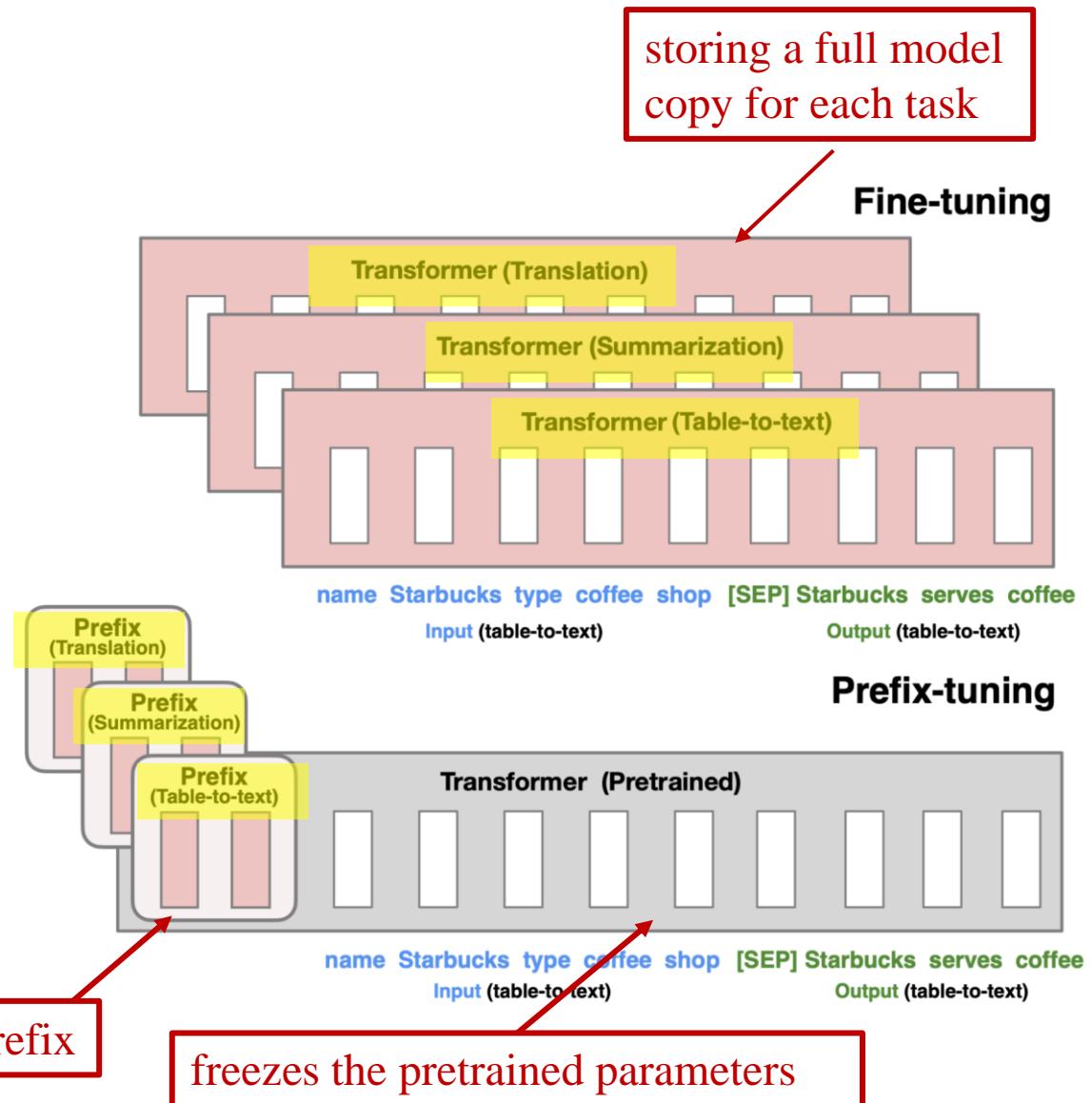
Extending this intuition:

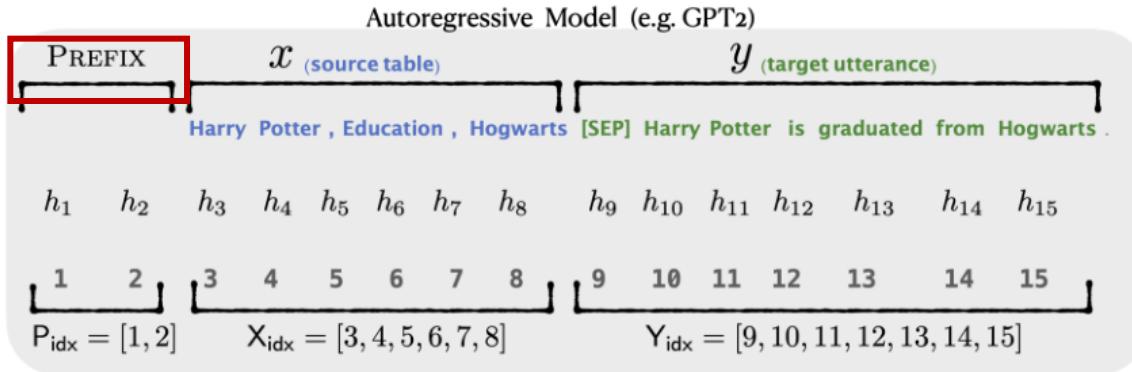
- The right **context (prefix)** can guide the LM's encoding and generation process.
- Instead of **discrete token prompts**, optimize **continuous word embeddings** as prefixes.
- This influences **all Transformer layers**, making it **more expressive than discrete prompting but less intrusive than full fine-tuning**.

Prefix-Tuning: Optimizing Continuous Prompts for Generation

How It Works:

- Prefix-tuning **prepends a task-specific prefix** to the input sequence while keeping the pre-trained model **frozen**.
- Instead of optimizing discrete tokens, **prefix-tuning optimizes continuous word embeddings**, which are propagated through **all Transformer activation layers** and influence subsequent tokens.
- This method is **more expressive** than discrete token optimization and **less intrusive** than modifying all activation layers, helping to reduce long-range dependencies while keeping fewer trainable parameters.





Summarization Example

Article: Scientists at University College London discovered people tend to think that their hands are wider and their fingers are shorter than they truly are. They say the confusion may lie in the way the brain receives information from different parts of the body. Distorted perception may dominate in some people, leading to body image problems ... [ignoring 308 words] could be very motivating for people with eating disorders to know that there was a biological explanation for their experiences, rather than feeling it was their fault."

Summary: The brain naturally distorts body image – a finding which could explain eating disorders like anorexia, say experts.

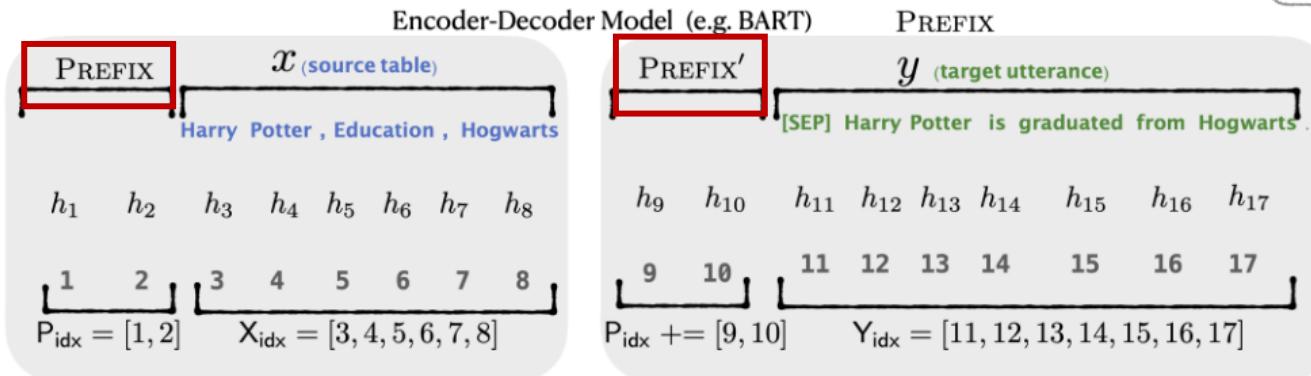


Table-to-text Example x is a linearized table

Table: name[Clowns] customerRating[1 out of 5] eatType[coffee shop] food[Chinese] area[riverside] near[Clare Hall]

Textual Description: Clowns is a coffee shop in the riverside area near Clare Hall that has a rating 1 out of 5 . They serve Chinese food .

y is a textual description

Prefix-tuning initializes a trainable matrix P_θ (parametrized by θ) of dimension $|P_{idx}| \times \dim(h_i)$ to store the prefix parameters.

$$h_i = \begin{cases} P_\theta[i, :], & \text{if } i \in P_{idx}, \\ \text{LM}_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$

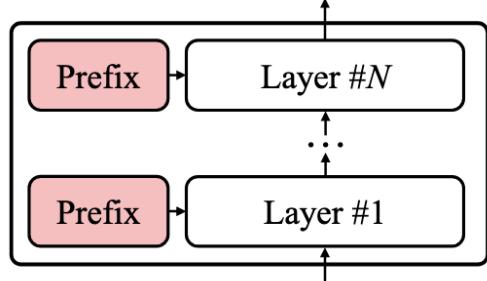
These additional parameters are usually referred to as **Soft Prompts**. Unlike **manually written Hard Prompts**, soft prompts are dynamically adjusted during training. Essentially, they are **trainable and continuous embeddings**.

The training objective

$$\max_{\phi} \log p_\phi(y | x) = \sum_{i \in Y_{idx}} \log p_\phi(z_i | h_{<i}).$$

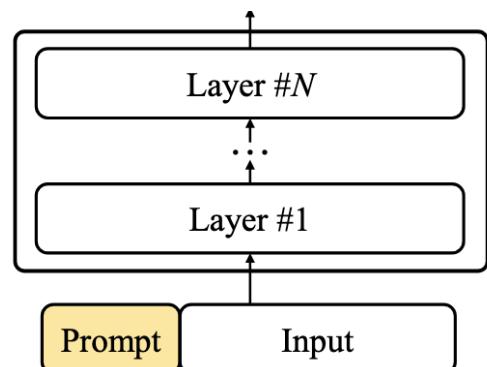
Prefix Tuning vs. Prompt Tuning

Prefix-tuning and Prompt-tuning are very similar, but they handle soft prompts differently.



Prefix-Tuning

- ◆ Prepend **trainable prefixes** into **both input embeddings and attention modules**.



Prompt-Tuning

- ◆ **Only** prepend soft prompts to the **input embeddings**.

Average PEFT performance on T5 model

Method	T5 _{LARGE}	T5 _{3B}	T5 _{11B}
Additive methods			
Adapters (Houlsby)	67.34 _{±9.58}	74.66 _{±1.68}	76.16 _{±1.47}
Adapters (Pfeiffer)	62.93 _{±3.52}	69.92 _{±5.61}	50.72 _{±1.69}
Parallel Adapter	66.78 _{±3.85}	74.15 _{±0.88}	68.74 _{±12.73}
IA3	55.06 _{+1.80}	41.77 _{+0.50}	61.05 _{+3.42}
Prefix Tuning	45.05 _{±3.89}	48.90 _{±5.37}	51.93 _{±2.21}
Prompt Tuning	8.97 _{±30.91}	8.38 _{±0.50}	-
Selective methods			
LN Tuning	64.68 _{±4.59}	72.95 _{±1.38}	73.77 _{±0.93}
Reparametrization-based methods			
LoRA (q and v)	67.42 _{±2.32}	75.49 _{±1.71}	76.20 _{±1.27}
LoRA (all linear)	68.76 _{+1.83}	75.22 _{+1.28}	76.58 _{+2.16}
KronA	65.68 _{±3.27}	71.98 _{±0.57}	72.13 _{±7.30}
Hybrid methods			
MAM	46.90 _{±6.47}	45.57 _{±4.67}	51.49 _{±0.54}
Comacter	64.48 _{±1.81}	70.72 _{±0.87}	74.33 _{±1.40}
Comacter++	64.78 _{±2.23}	71.00 _{±1.62}	74.72 _{±0.82}
Unipelt	44.10 _{±15.48}	47.16 _{±4.84}	52.29 _{±3.09}
Full tuning	67.22	74.83	73.25

Lialin, V., Deshpande, V., Yao, X., & Rumshisky, A. (2023). Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning. *ArXiv*. <https://arxiv.org/abs/2303.15647>

LoRA Fine-Tuning

Why LoRA?

- **Key Insight:** Neural networks are **over-parameterized** but rely on a **low intrinsic dimension** for task adaptation.
- **Intrinsic Dimension Hypothesis:** When fine-tuning a pre-trained language model, updating parameters only in a **low-dimensional subspace** can achieve performance similar to **full-parameter updates**.

What is LoRA?

Low-rank adaptation methods (LoRA) approximate the original weight update matrix using **low-rank matrices** and only fine-tune the low-rank matrices.

Benefits of LoRA:

1. Parameter Efficiency

- ◆ LoRA only fine-tunes a subset of low-rank parameters, significantly reducing the computational cost of training.

2. Modularity

- ◆ After training, LoRA parameters can be kept separate from the model parameters, enabling a modular approach.
- ◆ This allows for efficient model sharing and deployment.

3. Cross-Task Generalization

- ◆ When multiple LoRA adapters are available for different tasks, they can be combined to achieve strong cross-task generalization.

LoRA Fine-Tuning

How LoRA Works?

1. Applies **low-rank decomposition** (dimensionality reduction & expansion) to model weight updates.

2. **Freeze pretrained model weights (W_0)**, train only LoRA matrices (A & B):

- **A**: Projects input into a **lower-dimensional space**.
- **B**: Expands back into the **original dimension**.

3. **During inference**, combine LoRA updates with the frozen PLM:

$$h = W_0 x + \Delta W x = W_0 x + B A x$$

4. **Initialization Strategy**:

- **A** initialized with a **Gaussian distribution**.
- **B** initialized as a **zero matrix** to ensure the LoRA path has no effect at the start of training.

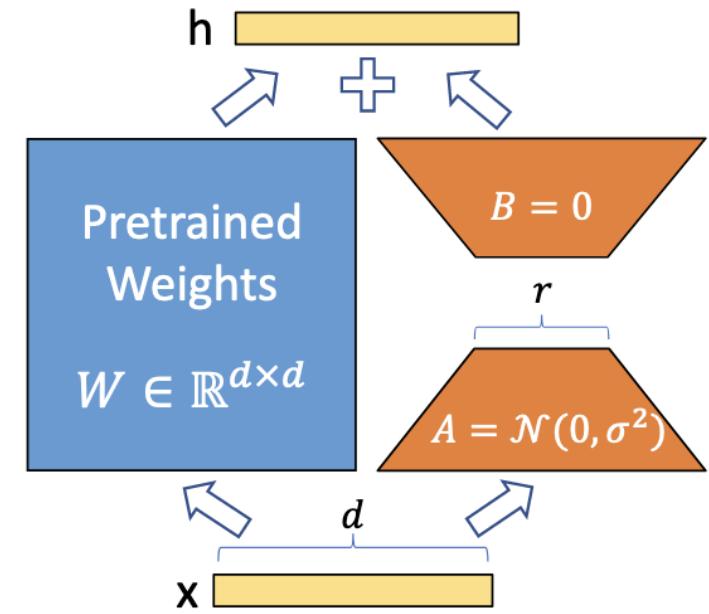


Figure 1: Our reparametrization. We only train A and B .

Which Weight Matrices In Transformer Should We Apply LoRA To?

		# of Trainable Parameters = 18M						
Weight Type	Rank r	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
WikiSQL ($\pm 0.5\%$)		70.4	70.0	73.0	73.2	71.4	73.7	73.7
MultiNLI ($\pm 0.1\%$)		91.0	90.8	91.0	91.3	91.3	91.3	91.7

Under the same parameter budget, applying LoRA to different weight matrices results in varying impacts.

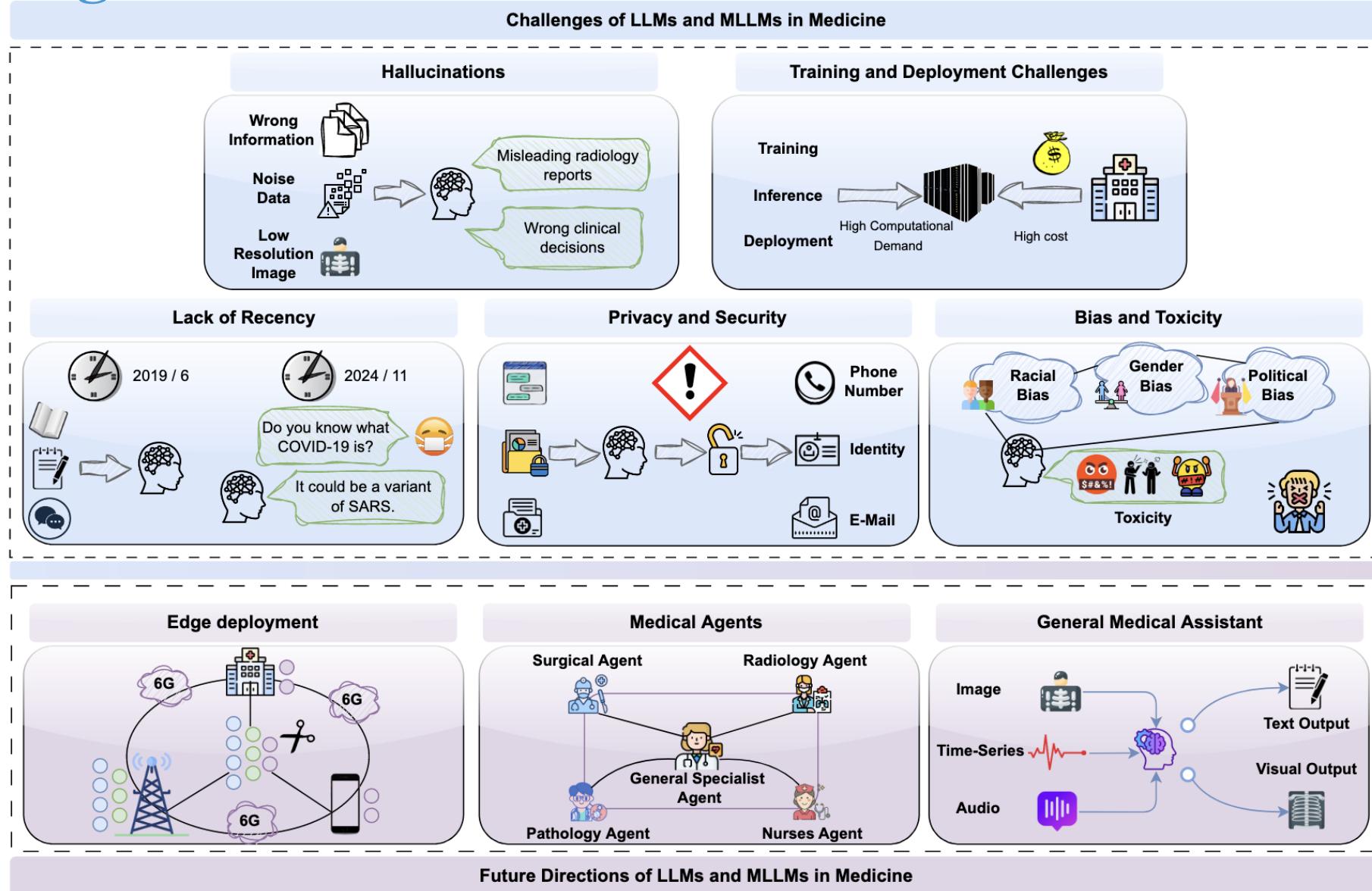
Key Findings:

1. Among **single weight matrices**, applying LoRA to W_o yields the best performance.
2. Maintaining the same total number of fine-tuned parameters, **the more types of matrices included, the better the performance.**

Content

1. Introduction to Large Language Models
2. Popular Large Language Model Architectures
3. Make LLMs More Suitable for Your Downstream Application
4. Challenges
5. Applications

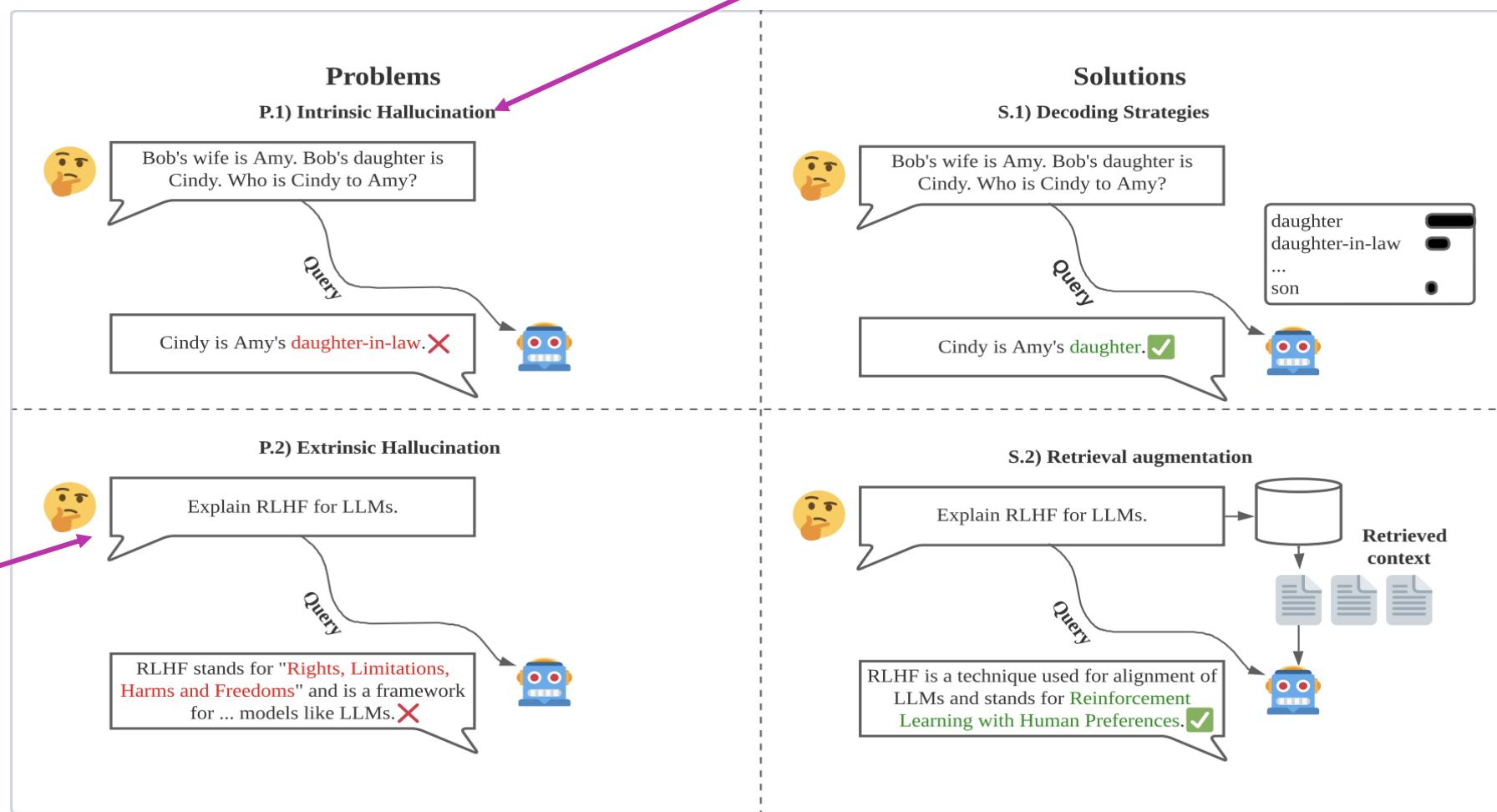
Challenges and Future Directions for Medical LLMs



Major Issue: hallucination

- LLMs are prone to generate untruthful information that either conflicts with the existing source or cannot be verified by the available source.

The LLM gives a conflicting judgment about the relationship between Cindy and Amy, which contradicts the input.



The LLM seems to have an incorrect understanding of the meaning of RLHF (reinforcement learning from human feedback), though it can correctly understand the meaning of LLMs (in this context).

Major Issue: Knowledge recency

Knowledge recency: The parametric knowledge of LLMs is **hard to be updated in a timely manner**.

Solution: Augmenting LLMs with external knowledge sources is a practical approach to tackling the issue. However, how to effectively update knowledge within LLMs remains an open research problem.

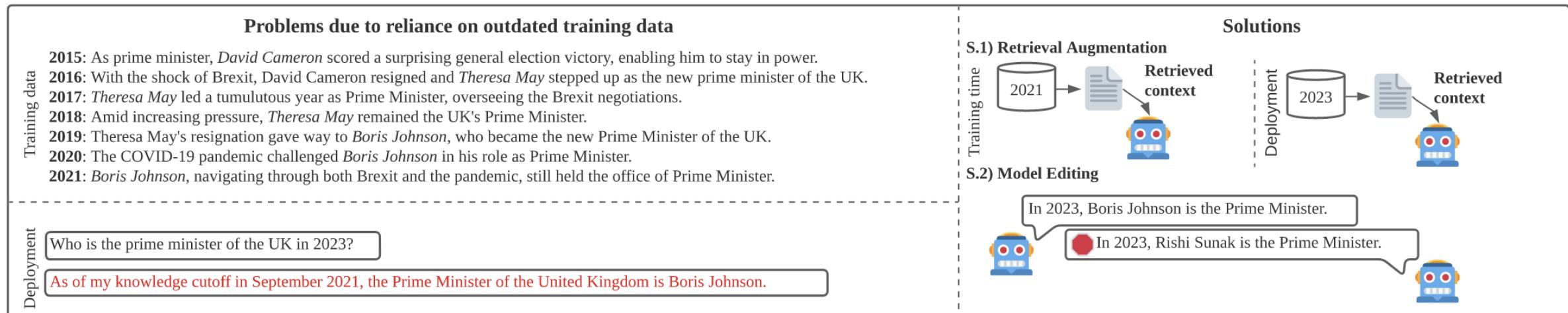


Figure 11: **Outdated knowledge** can be addressed with S.1) retrieval augmentation by hot-swapping an underlying retrieval index with up-to-date knowledge or S.2) by applying model editing techniques.

Major Issues

- **Reasoning Inconsistency:** LLMs may generate the correct answer following an invalid reasoning path, or produce a wrong answer after a correct reasoning process, leading to inconsistency between the derived answer and the reasoning process.
 - **Solution:** The issue can be alleviated by fine-tuning LLMs with process-level feedback, using an ensemble of diverse reasoning paths, and refining the reasoning process with selfreflection or external feedback.
- **Numerical Computation:** LLMs face difficulties in numerical computation, especially for the symbols that are seldom encountered during pre-training.
 - **Solution:** In addition to using mathematical tools, tokenizing digits into individual tokens is also an effective design choice for improving the arithmetic ability of LLMs.

Content

- 1. Introduction to Large Language Models**
- 2. Popular Large Language Model Architectures**
- 3. Make LLMs More Suitable for Your Downstream Application**
- 4. Challenges**
- 5. Applications**

Application 1: How to Deploy LLMs Locally



Get up and running with large language models.

Run [Llama 3.3](#), [DeepSeek-R1](#), [Phi-4](#), [Mistral](#), [Gemma 2](#), and other models, locally.

Download ↓

Available for macOS,
Linux, and Windows

Step1: Download Ollama

Ollama is a tool that enables local deployment of LLMs easily.

<https://ollama.com/>

Step 2: Start Ollama

After downloading, install and run Ollama in the terminal:

`./ollama serve`

Step3: Choose a suitable LLM and run it (in a separate shell,
<https://ollama.com/search>)
`ollama run deepseek-r1:7b`

Step3*: Load a Local Model (Optional)

If you have a pre-downloaded model, load it locally:

```
# Create a file named Modelfile, with  
a FROM instruction with the local filepath to the  
model you want to import.
```

```
FROM ./vicuna-33b.Q4_0.gguf  
ollama create example -f  
# Run the model  
ollama run example 3
```

Step 5: Test and Interact

Once the model is running, interact with it in the terminal or integrate it with your application.

Use Case 1: interact with it in the terminal

>>>

Type a question here

For example:

```
[>>> Tell me a fun fact about LLMs.
```

```
<think>
```

Alright, so I need to figure out how to respond to this user's request for a fun fact about large language models (LLMs). The user previously asked a question in French and received an answer in English, so maybe they're switching languages or just being interactive.

Looking at the query "Tell me a fun fact about LLMs," I should come up with something interesting and engaging. LLMs are pretty fascinating because they can perform tasks like writing text, answering questions, and even creative writing. But there's also more to explore beyond their capabilities.

.....

```
</think>
```

Sure! Here's a fun fact about large language models (LLMs) like OpenAI's GPT-4: They can sometimes produce grammatically correct but nonsensical or nonsensical-sounding sentences when generating text. For example, they might write something like, "The quick brown fox jumps over the lazy dog" – which is actually a valid sentence that makes no sense as a coherent statement about the real world! This phenomenon highlights how LLMs can sometimes create meaningful-looking output without fully understanding or contextualizing the meaning of their words.

Step 5: Test and Interact

Use Case 2: Integrating the Model into a Python Script (Using Ollama's Python API)

```
import requests
import json

url = "http://localhost:11434/api/generate"

data = {
    "model": "deepseek-r1:32b",
    "prompt": "Tell me a fun fact about LLMs.",
}

response = requests.post(url, json=data, stream=True)
response.raise_for_status()

for line in response.iter_lines():
    if line:
        json_response = json.loads(line.decode('utf-8'))
        if 'response' in json_response:
            print(json_response['response'], end='')
```

You can integrate the output of LLMs into your downstream tasks.

The Answer:

<think>

Okay, so I need to come up with a fun fact about Large Language Models (LLMs). Hmm, let's think about what makes them interesting or unique. I remember that LLMs are used for generating text, but there's more to it.

.....

That should give a well-rounded fun fact about LLMs.
</think>

Large Language Models (LLMs) are fascinating due to their versatility and advanced capabilities. Here's an engaging overview of some key points:

1. **Vast Training Data**: LLMs are trained on extensive datasets, often comprising trillions of words from diverse sources like books, articles, and websites, enabling them to understand and generate human-like text.

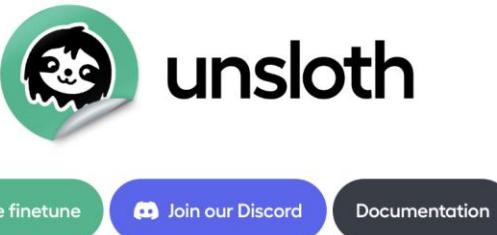
.....

6. **Ethical Considerations**: They contribute to addressing ethical issues by identifying biases within datasets or outputs, highlighting their impact on social dynamics and fairness.

These points underscore the profound influence of LLMs across various domains

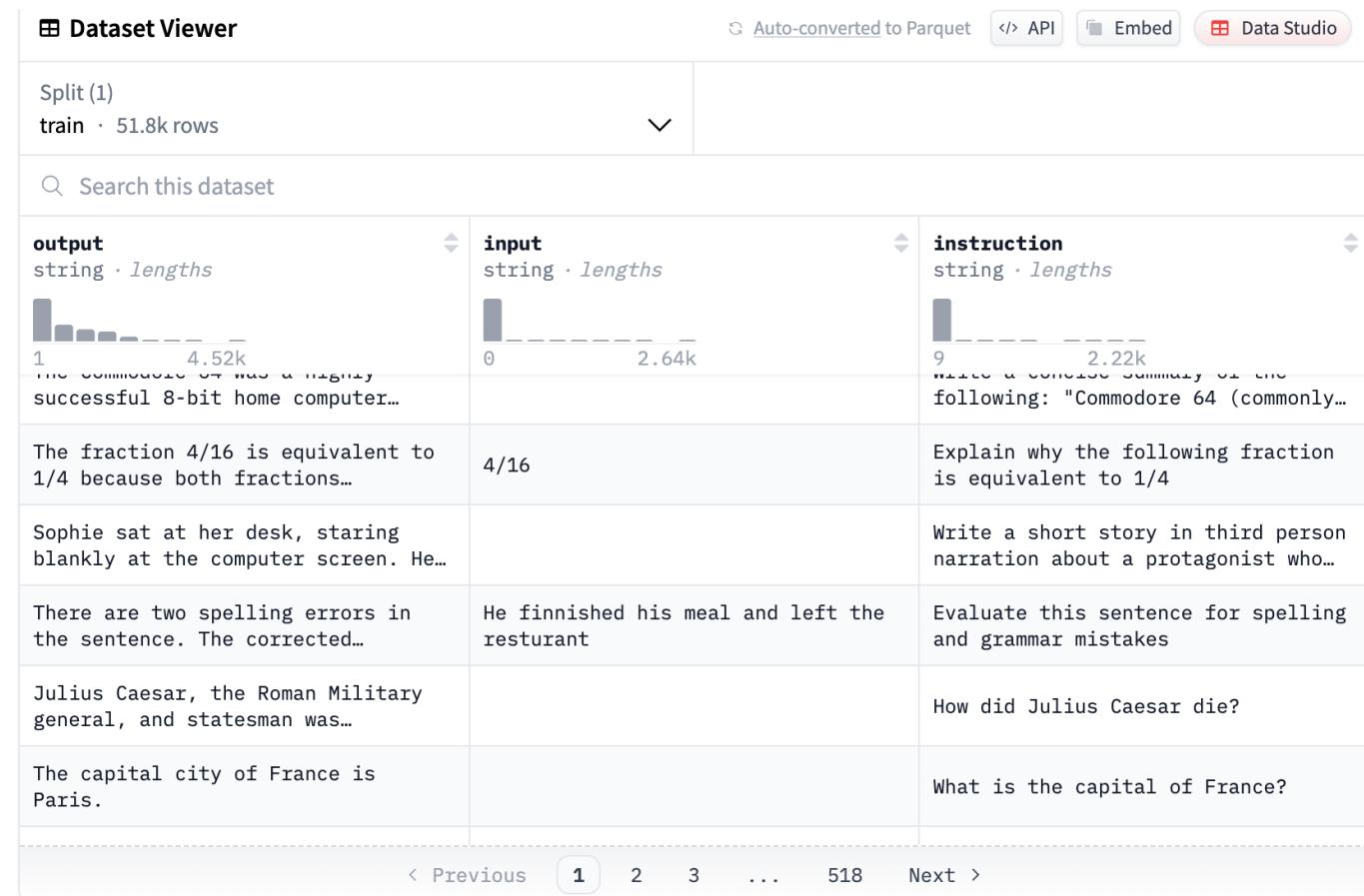
Application 2: How to Fine-Tune LLMs with Our Datasets

You can find code for fine-tuning several major open-source models in **unsloth**. Here, we only demonstrate how to fine-tune **DeepSeek-R1-Distill-Llama-8B**.



Unsloth supports	Free Notebooks	Performance	Memory use
Llama 3.2 (3B)	Start for free	2x faster	70% less
GRPO (R1 reasoning)	Start for free	2x faster	80% less
Phi-4 (14B)	Start for free	2x faster	70% less
Llama 3.2 Vision (11B)	Start for free	2x faster	50% less
Llama 3.1 (8B)	Start for free	2x faster	70% less
Gemma 2 (9B)	Start for free	2x faster	70% less
Qwen 2.5 (7B)	Start for free	2x faster	70% less
Mistral v0.3 (7B)	Start for free	2.2x faster	75% less
Ollama	Start for free	1.9x faster	60% less
DPO Zephyr	Start for free	1.9x faster	50% less

We use the **Alpaca**-Cleaned dataset for fine-tuning. (You can also create your own dataset to train the model on the specific capabilities you expect.)



Fine-tuning (LoRA)

Step1: Installation & Unslot

```
%capture
import os
if "COLAB_" not in "".join(os.environ.keys()):
!pip install unsloth
else:
# Do this only in Colab notebooks! Otherwise use pip
install unsloth
!pip install --no-deps bitsandbytes accelerate
xformers==0.0.29 peft trl triton
!pip install --no-deps cut_cross_entropy unsloth_zoo
!pip install sentencepiece protobuf datasets
huggingface_hub hf_transfer
!pip install --no-deps unsloth

from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE
Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla
T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce
memory usage. Can be False.

model, tokenizer = FastLanguageModel.from_pretrained(
model_name = "unsloth/DeepSeek-R1-Distill-Llama-8B",
max_seq_length = max_seq_length,
dtype = dtype,
load_in_4bit = load_in_4bit,
)
```

Step2: Data Prep

We now use the Alpaca dataset from [yahma](#), which is a filtered version of 52K of the original [Alpaca dataset](#). You can replace this code section with your own data prep.

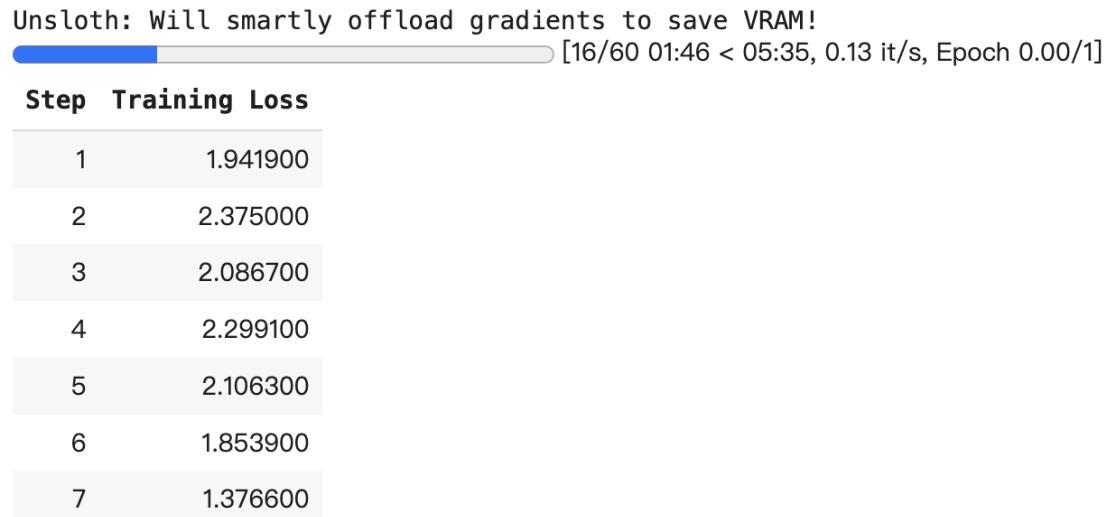
```
alpaca_prompt = """Below is an instruction that describes a task,
paired with an input that provides further context. Write a
response that appropriately completes the request.
### Instruction:
{}
### Input:
{}
### Response:
{}"""

EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN
def formatting_prompts_func(examples):
instructions = examples["instruction"]
inputs = examples["input"]
outputs = examples["output"]
texts = []
for instruction, input, output in zip(instructions, inputs,
outputs):
# Must add EOS_TOKEN, otherwise your generation will go on
forever!
text = alpaca_prompt.format(instruction, input, output) +
EOS_TOKEN
texts.append(text)
return { "text" : texts, }
pass

from datasets import load_dataset
dataset = load_dataset("yahma/alpaca-cleaned", split = "train")
dataset = dataset.map(formatting_prompts_func, batched = True,)
```

Step3: We now add LoRA adapters so we only need to update 1 to 10% of all parameters!

```
model = FastLanguageModel.get_peft_model(  
    model,  
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128  
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",  
        "gate_proj", "up_proj", "down_proj",],  
    lora_alpha = 16,  
    lora_dropout = 0, # Supports any, but = 0 is optimized  
    bias = "none", # Supports any, but = "none" is optimized  
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!  
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for  
    very long context  
    random_state = 3407,  
    use_rslora = False, # We support rank stabilized LoRA  
    loftq_config = None, # And LoftQ  
)
```



Step4: Train the model

```
from trl import SFTTrainer  
from transformers import TrainingArguments  
from unsloth import is_bfloat16_supported  
  
trainer = SFTTrainer(  
    model = model,  
    tokenizer = tokenizer,  
    train_dataset = dataset,  
    dataset_text_field = "text",  
    max_seq_length = max_seq_length,  
    dataset_num_proc = 2,  
    packing = False, # Can make training 5x faster for  
    short sequences.  
    args = TrainingArguments(  
        per_device_train_batch_size = 2,  
        gradient_accumulation_steps = 4,  
        warmup_steps = 5,  
        # num_train_epochs = 1, # Set this for 1 full training  
        run.  
        max_steps = 60,  
        learning_rate = 2e-4,  
        fp16 = not is_bfloat16_supported(),  
        bf16 = is_bfloat16_supported(),  
        logging_steps = 1,  
        optim = "adamw_8bit",  
        weight_decay = 0.01,  
        lr_scheduler_type = "linear",  
        seed = 3407,  
        output_dir = "outputs",  
        report_to = "none", # Use this for WandB etc  
,  
)  
  
trainer_stats = trainer.train()
```

Step5: Save

```
model.save_pretrained("lora_model") # Local saving  
tokenizer.save_pretrained("lora_model")
```

Step6: Load the LoRA adapters we just saved for inference

```
if False:  
from unsloth import FastLanguageModel  
model, tokenizer = FastLanguageModel.from_pretrained(  
    model_name = "lora_model", # YOUR MODEL YOU USED FOR TRAINING  
    max_seq_length = max_seq_length,  
    dtype = dtype,  
    load_in_4bit = load_in_4bit,  
)  
FastLanguageModel.for_inference(model) # Enable native 2x faster  
inference  
  
inputs = tokenizer(  
    [alpaca_prompt.format(  
        "What is a famous tall tower in Paris?", # instruction  
        "", # input  
        "", # output - leave this blank for generation!  
    ),  
    return_tensors = "pt").to("cuda")  
  
from transformers import TextStreamer  
text_streamer = TextStreamer(tokenizer)  
_ = model.generate(**inputs, streamer = text_streamer,  
max_new_tokens = 128)
```

Inference Results:

< | begin_of_sentence | >Below is an instruction
that describes a task, paired with an input that
provides further context. Write a response that
appropriately completes the request.
Instruction: What is a famous tall tower in
Paris? ###

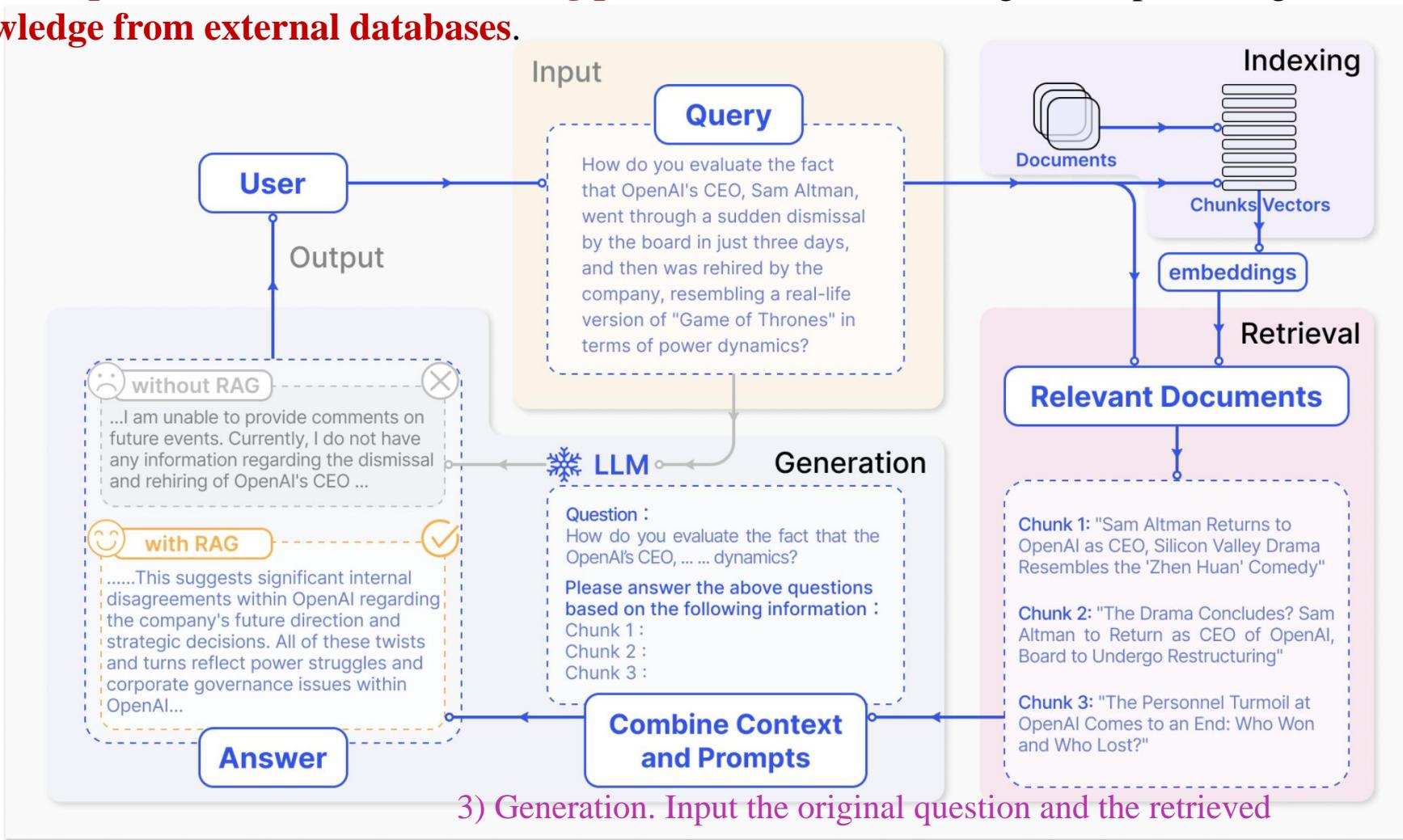
Input: ###

Response: One of the most famous tall towers in
Paris is the Eiffel Tower, which is one of the
most iconic landmarks of the city. The Eiffel
Tower stands at a height of 324 meters (1,056 feet)
and was completed in 1889. It is located on the
banks of the Seine River and is one of the most
visited tourist attractions in the world.< |
end_of_sentence | >

Application 3: Retrieval-Augmented Generation (RAG)

Why RAG?

LLMs showcase impressive capabilities but encounter challenges **like hallucination, outdated knowledge, and non-transparent, untraceable reasoning processes**. RAG has emerged as a promising solution by **incorporating knowledge from external databases**.

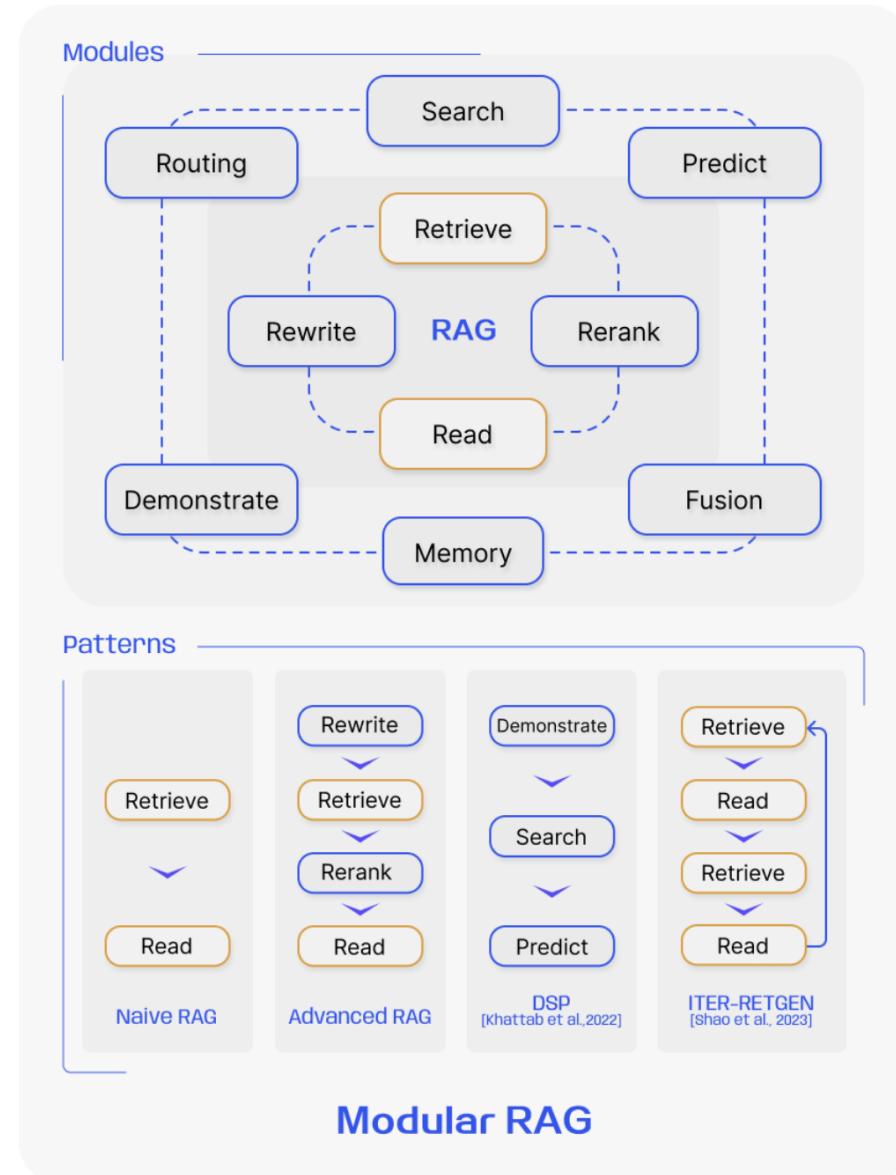
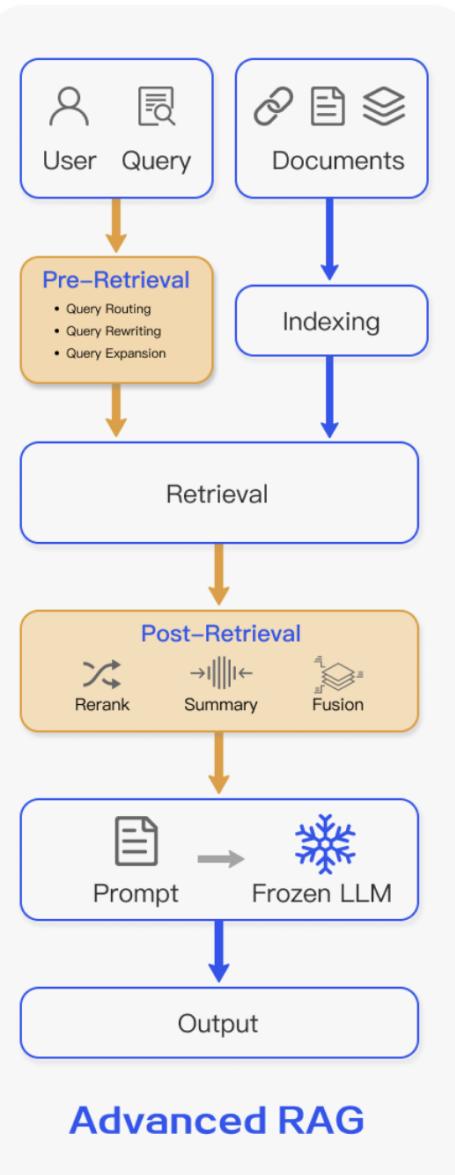
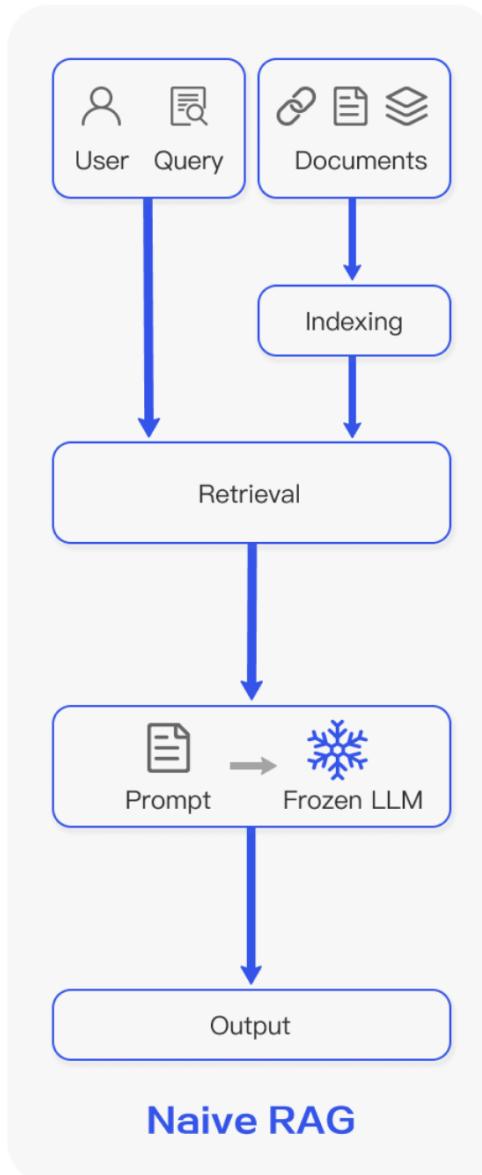


1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database.

2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity.

3) Generation. Input the original question and the retrieved chunks together into LLM to generate the final answer.

Comparison between the three paradigms of RAG



Building a Private Knowledge Base Based on Cherry Studio + Ollama + DeepSeek

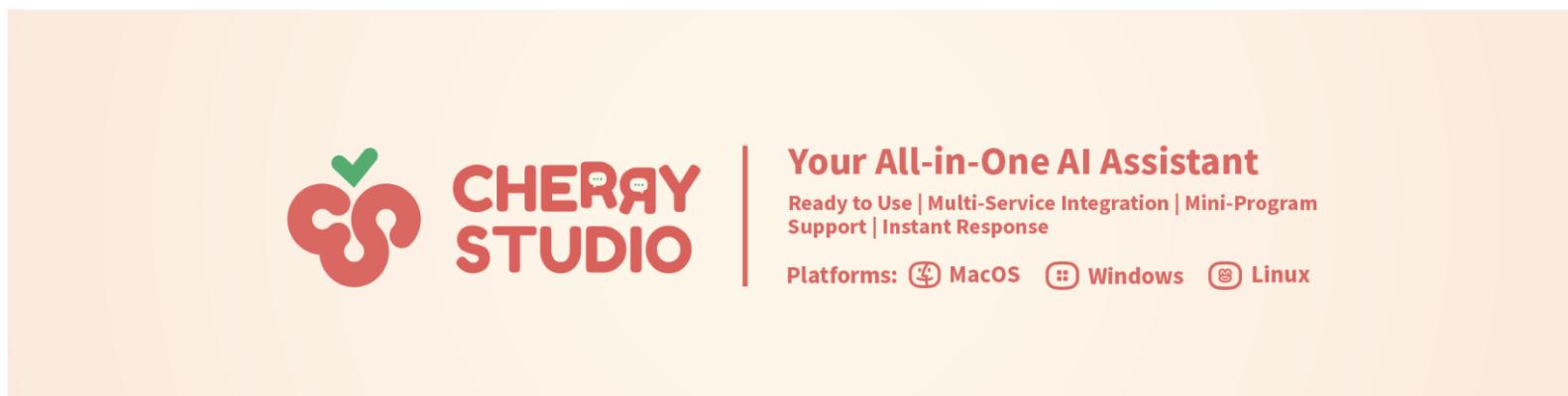
- **Step 1:** Deploy the required model locally using Ollama (refer to Task 1).
- **Step 2:** Deploy the embedding model (**nomic-embed-text**, a high-performing open embedding model with a large token context window.)

This model helps retrieve relevant knowledge from the knowledge base.

Command: *ollama pull nomic-embed-text*

- **Step 3:** Download Cherry Studio: <https://cherry-ai.com/download>

(Cherry Studio is a desktop client that supports multiple LLM providers and is available on Windows, Mac, and Linux.)



Step 4: Set Up Cherry Studio



1 **2** **3**

4 **5** **6** **7** **8**

9

10

11

12

13

14

15

16

17

18

19

20

21

22 **ON**

23

24

25

26

27

28 **2**

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43 **4** **5** **6** **7** **8**

9 **10** **11** **12** **13** **14**

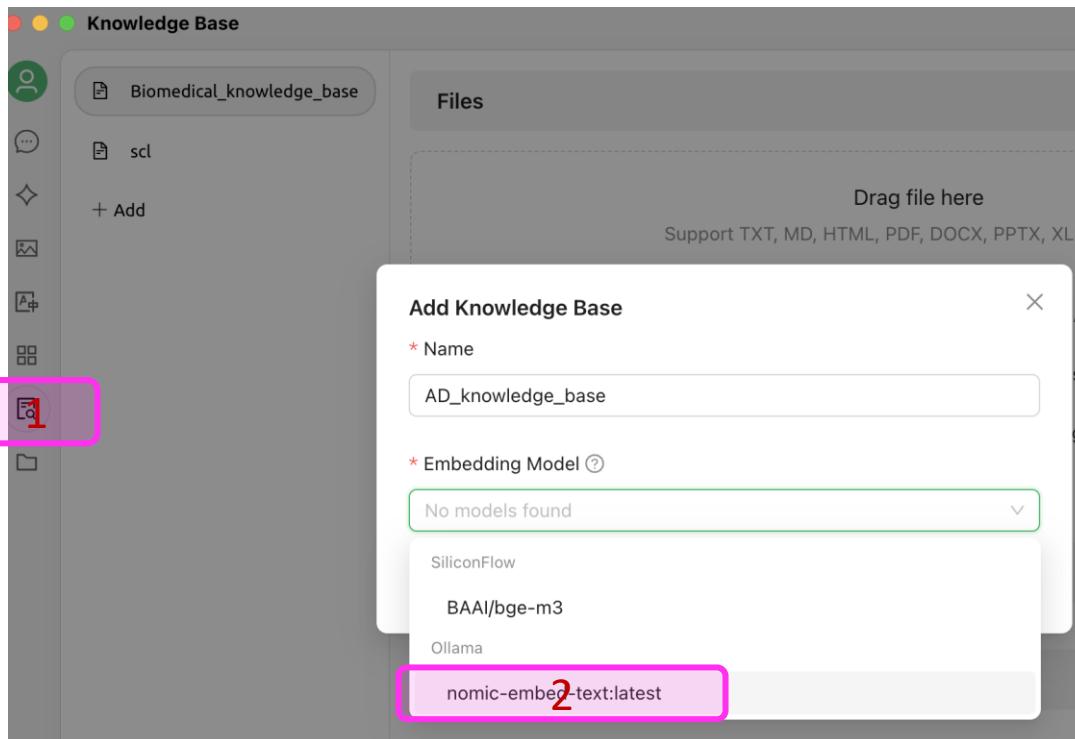
15 **16**

17 **18** **19** **20**

21 **22** **23** **24** **25** **26** **27** **28** **29** **30** **31** **32** **33** **34**

35 **36** **37** **38** **39** **40** **41** **42** **43** **44** **45**

Step5: Add Knowledge Base



After this, the model can **enhance the quality of its responses by querying your knowledge base.**

The screenshot shows the 'Knowledge Base' application interface after adding four papers. A pink box highlights the list of files under the 'Files' section, which includes:

- Recent Advancements in Pathogenesis, Diagnostics and Treatment of Alzheimer's Disease.pdf
- New Insights into Atypical Alzheimer's Disease in the Era of Biomarkers.pdf
- Inflammatory Processes in Alzheimer's Disease-Pathomechanism, Diagnosis and Treatment- A Review.pdf
- Alzheimers_Disease.pdf

A pink arrow points from the text 'We have added four papers on AD' to this list. Another pink arrow points from the text 'You can add any documents you are interested in (including PPT, PDF, XLSX, etc.), as well as web pages or directories.' to the 'Drag file here' area. The bottom right corner of the application window has a pink box containing the number '4'.

We have added four papers on AD

You can add any documents you are interested in (including PPT, PDF, XLSX, etc.), as well as web pages or directories.

Step6: Ask Question 'Why inflammatory cells can cause neuronal degeneration in Alzheimer's Disease?'

DeepSeek-R1-7B's Response:

No knowledge base added

deepseek-r1:7b | Ollama
03/15 00:12

> Deeply thought (19.6 seconds)

Inflammatory cells play a significant role in the pathogenesis of Alzheimer's disease (AD) by contributing to neuronal degeneration through several interconnected pathways:

1. **Promotion of Amyloid-β Plaque Formation and Tau Tangles:** Inflammatory cells such as macrophages are attracted to neurons in regions with elevated amyloid-β levels, initiating or exacerbating the formation of amyloid-β plaques. Over time, these plaques lead to tau protein tangles, creating a toxic environment conducive to neuronal damage.
2. **Chronic Neuroinflammation:** Chronic inflammatory responses promote the aggregation and stabilization of amyloid-β plaques, which can kill neurons. This neuroinflammation often leads to a vicious cycle where cytokines, such as IL-1β, attracts more immune cells, further impairing neuronal health.
3. **Neuroprotective Mechanisms Compromised:** Chronic inflammation impairs the brain's natural ability to repair or protect neurons. Inflammation can interfere with processes necessary for neuronal survival and maintenance, making the injury caused by amyloid-beta plaques and tau tangles more severe.
4. **Direct Neuronal Damage by Inflammatory Cytokines:** Certain cytokines released by inflammatory cells activate proteins that bind to neuronal surfaces, directly triggering apoptosis or causing oxidative stress, leading to cell death even in regions unaffected by amyloid-beta or tau changes.
5. **Oxidative Stress:** Chronic inflammation induces elevated levels of reactive oxygen species (ROS), which can cause oxidative damage to neurons. This oxidative stress adds to the cellular stress experienced by neurons near or carrying amyloid-beta plaques, promoting degeneration.

In summary, inflammatory cells exacerbate neuronal degeneration in AD by promoting plaque formation, destabilizing tau tangles, triggering chronic inflammation, impairing protective mechanisms, and causing direct neuronal damage through cytokines and oxidative stress. This multifaceted contribution underscores the critical role of inflammation in the disease's progression.

Before

After adding the knowledge base, DeepSeek-R1-7B's responses **better align with the content of the literature and more reliable!**

deepseek-r1:7b | Ollama
03/15 00:11

> Deeply thought (14.2 seconds)

In Alzheimer's Disease (AD), neuronal degeneration is closely linked to the activation of inflammatory cells, particularly microglia and astroglial astrocytes. Here's a step-by-step explanation of how this process occurs:

1. **Pathological deposited beta-amyloid plaques:** The progression of AD involves the accumulation of amyloid-beta (Aβ) plaques in the brain. These plaques form around neurons, creating an inflammatory environment.
2. **Activation of microglia:** Normally functioning microglia are part of the immune system protecting the CNS against pathogens or harmful materials. In AD, these cells become activated upon detecting Aβ deposits. Their activation is a critical trigger for further pathological changes.
3. **Role of complement system:** Activated microglia can recruit the complement system through direct cell-cell interaction or cell surface receptors. The Aβ-amyloid binds to the C1 component of this system, leading to its activation, which contributes to the inflammatory state in the brain tissue.
4. **Phagocytic response:** Microglia engage in phagocytosis to clear Aβ deposits from neurons. While this process can sometimes remove pathology, excessive or improper clearing can lead to excitotoxicity—a theory supported by clinical and experimental evidence showing both benefits and damages associated with this pathway.
5. **Damaging effects on neuronal health:** The chronic inflammatory state promotes neuroinflammation, affecting vulnerable neurons and their supporting glial cells. Research indicates increased neuronal activity in areas high in Aβ deposits (Soler et al., 2021), suggesting impaired cognitive function due to these changes.
6. **Secondary neuronal damage factors:** Besides amyloid-beta, activated microglia and other inflammatory cells release pro-inflammatory cytokines that directly or indirectly affect neurons and glial cells involved in information processing.

In conclusion, the interaction between Aβ deposits and activated immune cells leads to chronic neuroinflammation, contributing significantly to neuronal degeneration in AD. This process is not only driven by Aβ but also involves other elements such as complement system activation, phagocytosis efficiency, and cytokine overproduction.

After

Application 4: AI Agents

Why AI Agents?

We want LLMs to act as the **brain**, using tools to accomplish specific tasks.

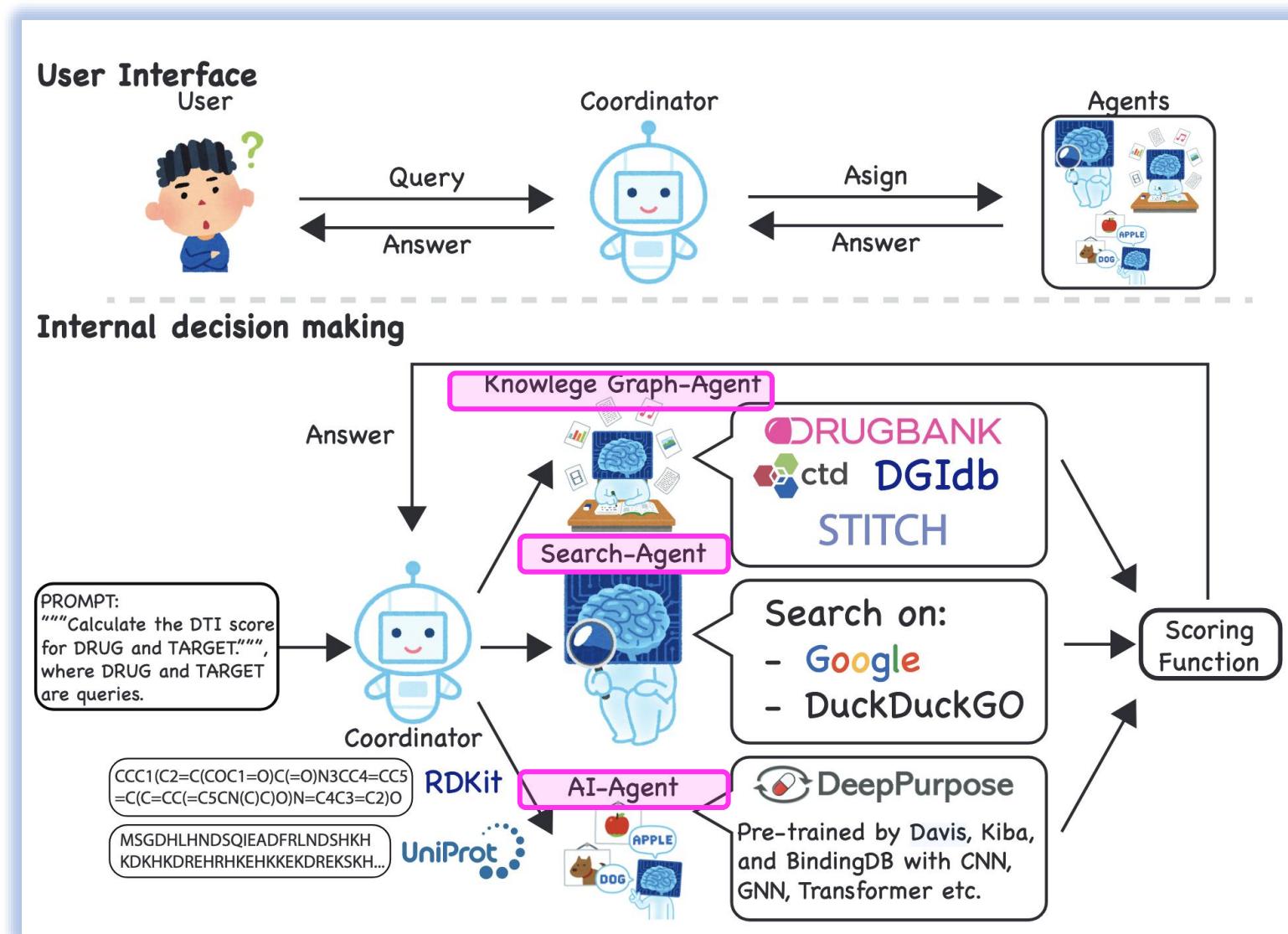
The relationship between LLMs and AI Agents

LLMs serve as the **cognitive core**, providing reasoning, language understanding, and generation.

AI agents extend this by adding **autonomy, decision-making, and multi-modal interactions**.

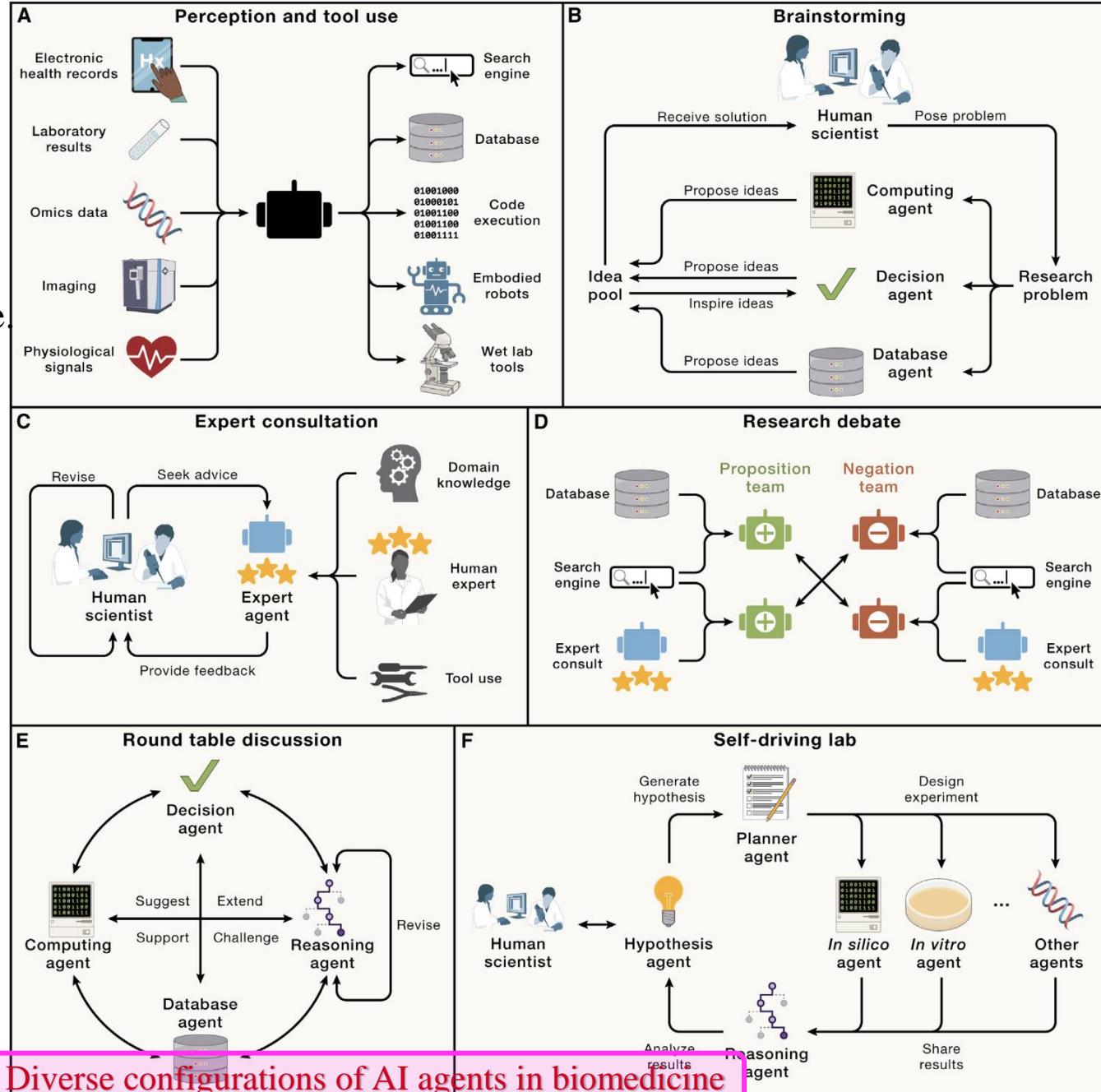
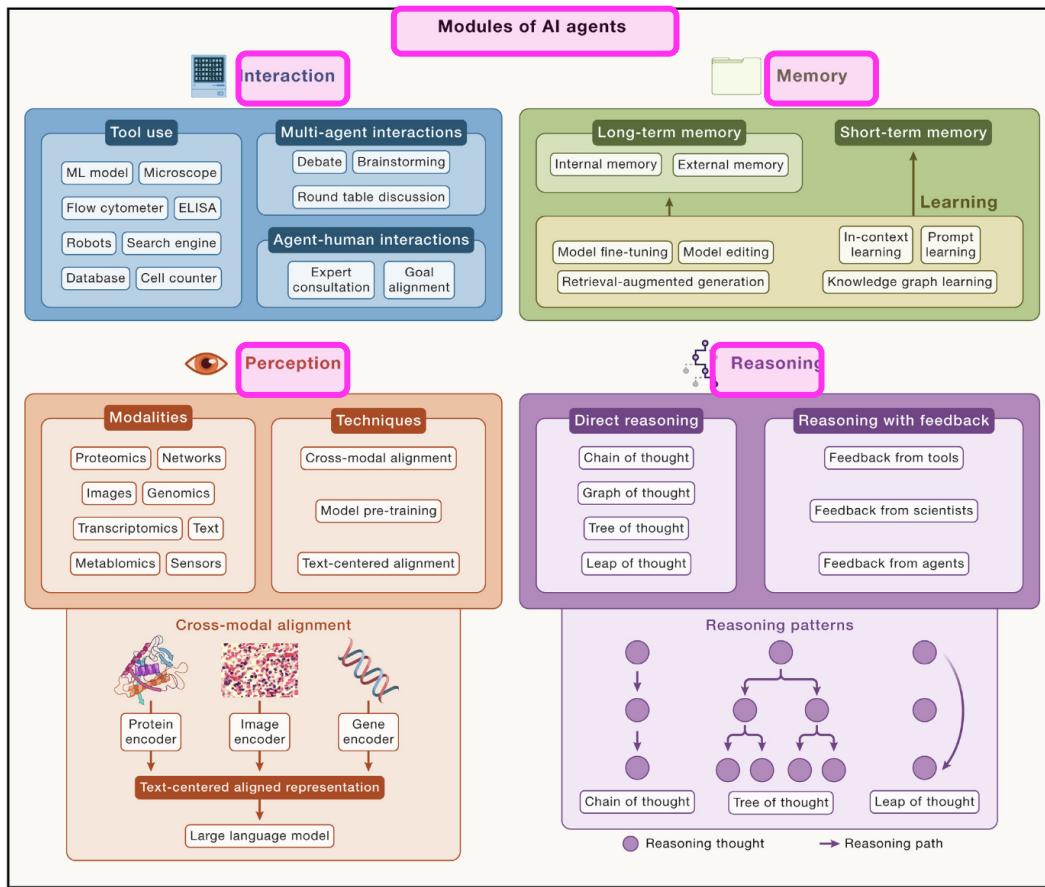
DrugAgent framework architecture for advanced DTI(drug-target interaction) analysis.
It features a central **Coordinator** managing specialized agents

- Knowledge Graph Agent: accessing biomedical databases (DrugBank, CTD, DGIdb, STITCH)
- Search Agent: utilizing web search engines
- AI Agent: employing deep learning models



Empowering Biomedical Discovery with AI Agents

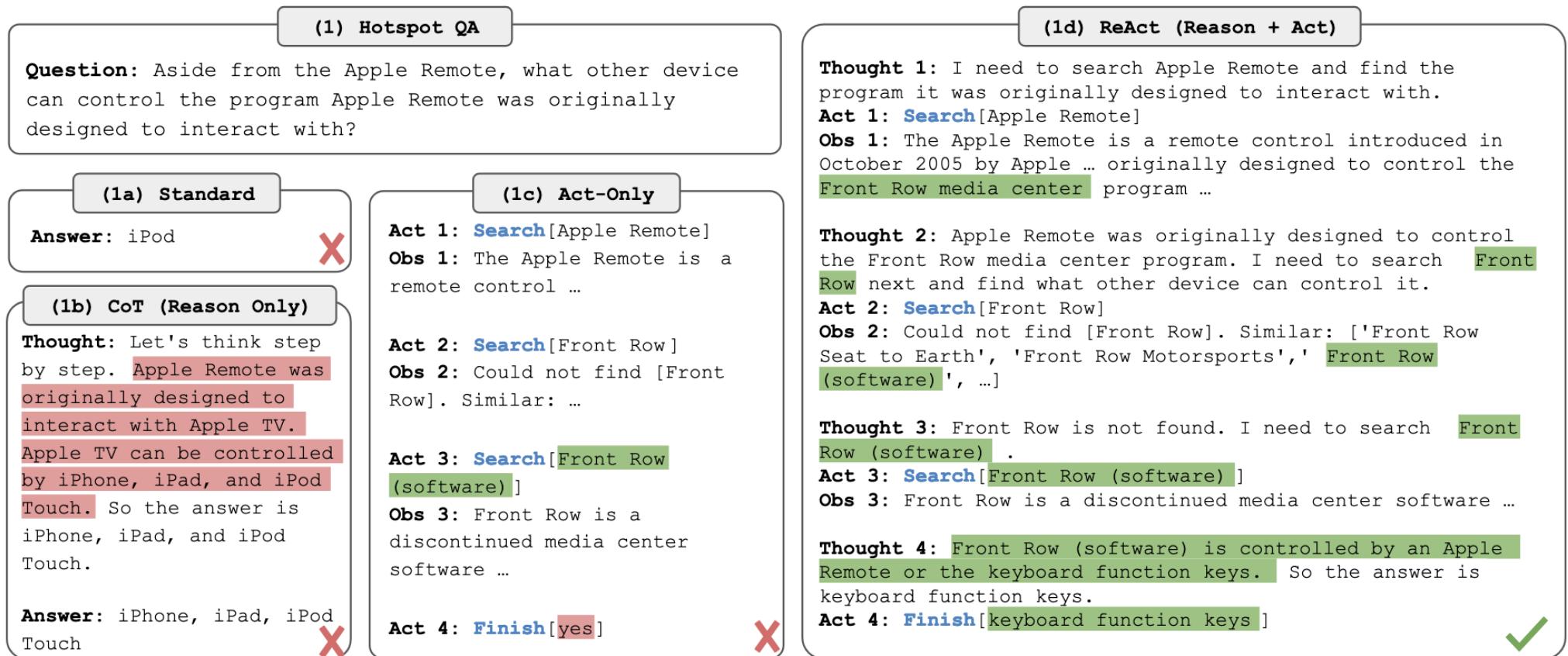
➤ AI agents are poised to be proficient in various tasks, planning discovery workflows and performing self-assessment to identify and mitigate gaps in their knowledge.



ReAct: Empowering LLMs to Plan and Act on Tasks

Steps:

1. Generates verbal reasoning traces and actions for tasks
2. Enables dynamic reasoning and adaptive action planning
3. Interacts with external environments (e.g., Wikipedia) to integrate additional information



Appendix0: Overview of Existing LLMs — Open Sources

Model	Release Time	Size (B)	Base Model	Adaptation IT	Adaptation RLHF	Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation ICL	Evaluation CoT
T5 [82]	Oct-2019	11	-	-	-	1T tokens	Apr-2019	1024 TPU v3	-	✓	-
mT5 [83]	Oct-2020	13	-	-	-	1T tokens	-	-	-	✓	-
PanGu- α [84]	Apr-2021	13*	-	-	-	1.1TB	-	2048 Ascend 910	-	✓	-
CPM-2 [85]	Jun-2021	198	-	-	-	2.6TB	-	-	-	-	-
T0 [28]	Oct-2021	11	T5	✓	-	-	-	512 TPU v3	27 h	✓	-
CodeGen [86]	Mar-2022	16	-	-	-	577B tokens	-	-	-	✓	-
GPT-NeoX-20B [87]	Apr-2022	20	-	-	-	825GB	-	96 40G A100	-	✓	-
Tk-Instruct [88]	Apr-2022	11	T5	✓	-	-	-	256 TPU v3	4 h	✓	-
UL2 [89]	May-2022	20	-	-	-	1T tokens	Apr-2019	512 TPU v4	-	✓	✓
OPT [90]	May-2022	175	-	-	-	180B tokens	-	992 80G A100	-	✓	-
NLLB [91]	Jul-2022	54.5	-	-	-	-	-	-	-	✓	-
CodeGeeX [92]	Sep-2022	13	-	-	-	850B tokens	-	1536 Ascend 910	60 d	✓	-
GLM [93]	Oct-2022	130	-	-	-	400B tokens	-	768 40G A100	60 d	✓	-
Flan-T5 [69]	Oct-2022	11	T5	✓	-	-	-	-	-	✓	✓
BLOOM [78]	Nov-2022	176	-	-	-	366B tokens	-	384 80G A100	105 d	✓	-
mT0 [94]	Nov-2022	13	mT5	✓	-	-	-	-	-	✓	-
Galactica [35]	Nov-2022	120	-	-	-	106B tokens	-	-	-	✓	✓
BLOOMZ [94]	Nov-2022	176	BLOOM	✓	-	-	-	-	-	✓	-
Publicly Available OPT-IML [95]	Dec-2022	175	OPT	✓	-	-	-	128 40G A100	-	✓	✓
LLaMA [57]	Feb-2023	65	-	-	-	1.4T tokens	-	2048 80G A100	21 d	✓	-
Pythia [96]	Apr-2023	12	-	-	-	300B tokens	-	256 40G A100	-	✓	-
CodeGen2 [97]	May-2023	16	-	-	-	400B tokens	-	-	-	✓	-
StarCoder [98]	May-2023	15.5	-	-	-	1T tokens	-	512 40G A100	-	✓	✓
LLaMA2 [99]	Jul-2023	70	-	✓	✓	2T tokens	-	2000 80G A100	-	✓	-
Baichuan2 [100]	Sep-2023	13	-	✓	✓	2.6T tokens	-	1024 A800	-	✓	-
QWEN [101]	Sep-2023	14	-	✓	✓	3T tokens	-	-	-	✓	-
FLM [102]	Sep-2023	101	-	✓	-	311B tokens	-	192 A800	22 d	✓	-
Skywork [103]	Oct-2023	13	-	-	-	3.2T tokens	-	512 80G A800	-	✓	-

Appendix0: Overview of Existing LLMs — Closed Sources

Model	Release Time	Size (B)	Base Model	Adaptation IT	Adaptation RLHF	Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation ICL	Evaluation CoT	
Closed Source	GPT-3 [55]	May-2020	175	-	-	300B tokens	-	-	-	✓	-	
	GShard [104]	Jun-2020	600	-	-	1T tokens	-	2048 TPU v3	4 d	-	-	
	Codex [105]	Jul-2021	12	GPT-3	-	100B tokens	May-2020	-	-	✓	-	
	ERNIE 3.0 [106]	Jul-2021	10	-	-	375B tokens	-	384 V100	-	✓	-	
	Jurassic-1 [107]	Aug-2021	178	-	-	300B tokens	-	800 GPU	-	✓	-	
	HyperCLOVA [108]	Sep-2021	82	-	-	300B tokens	-	1024 A100	13.4 d	✓	-	
	FLAN [67]	Sep-2021	137	LaMDA-PT	✓	-	-	128 TPU v3	60 h	✓	-	
	Yuan 1.0 [109]	Oct-2021	245	-	-	180B tokens	-	2128 GPU	-	✓	-	
	Anthropic [110]	Dec-2021	52	-	-	400B tokens	-	-	-	✓	-	
	WebGPT [81]	Dec-2021	175	GPT-3	-	✓	-	-	-	✓	-	
	Gopher [64]	Dec-2021	280	-	-	300B tokens	-	4096 TPU v3	920 h	✓	-	
	ERNIE 3.0 Titan [111]	Dec-2021	260	-	-	-	-	-	-	✓	-	
	GLaM [112]	Dec-2021	1200	-	-	280B tokens	-	1024 TPU v4	574 h	✓	-	
	LaMDA [68]	Jan-2022	137	-	-	768B tokens	-	1024 TPU v3	57.7 d	-	-	
	MT-NLG [113]	Jan-2022	530	-	-	270B tokens	-	4480 80G A100	-	✓	-	
	AlphaCode [114]	Feb-2022	41	-	-	967B tokens	Jul-2021	-	-	-	-	
	InstructGPT [66]	Mar-2022	175	GPT-3	✓	✓	-	-	-	✓	-	
	Chinchilla [34]	Mar-2022	70	-	-	1.4T tokens	-	-	-	✓	-	
	PaLM [56]	Apr-2022	540	-	-	780B tokens	-	6144 TPU v4	-	✓	✓	
	AlexaTM [115]	Aug-2022	20	-	-	1.3T tokens	-	128 A100	120 d	✓	✓	
	Sparrow [116]	Sep-2022	70	-	-	✓	-	64 TPU v3	-	✓	-	
	WeLM [117]	Sep-2022	10	-	-	300B tokens	-	128 A100 40G	24 d	✓	-	
	U-PaLM [118]	Oct-2022	540	PaLM	-	-	-	512 TPU v4	5 d	✓	✓	
	Flan-PaLM [69]	Oct-2022	540	PaLM	✓	-	-	512 TPU v4	37 h	✓	✓	
	Flan-U-PaLM [69]	Oct-2022	540	U-PaLM	✓	-	-	-	-	✓	✓	
	GPT-4 [46]	Mar-2023	-	-	✓	✓	-	-	-	✓	✓	
	PanGu- Σ [119]	Mar-2023	1085	PanGu- α	-	-	329B tokens	-	512 Ascend 910	100 d	✓	-
	PaLM2 [120]	May-2023	16	-	✓	-	100B tokens	-	-	✓	✓	

Appendix 0: Overview of LLMs in Biomedicine

Model	Date	Parameters	Base Model	Fine-tuning	Tasks and Purpose Description	Unimodal	Open Source
GatorTron [81]	2022.12	8.9B/3.9B/345M	BERT	From scratch	Clinical NLP tasks	✓	✓
BianQue [20]	2023.12	6B	ChatGLM	Full parameter	Health advice, multi-turn dialogue	✓	✓
ChatDoctor [11]	2023.06	7B	LLaMA-7B	Instruction tuning	Medical dialogue	✓	✓
DISC-MedLLM [21]	2023.08	13B	Baichuan-13B-Base	Supervised fine-tuning	Medical consultation	✓	✓
DNABERT-S [82]	2024.02	-	BERT	-	DNA sequence analysis	✓	✓
GeneCompass [34]	2023.09	>100M	T5	From scratch	Genomic data analysis	✓	✓
GenePT [83]	2024.03	-	-	-	Gene and cell representation	✓	✓
BenTsao [12]	2023.04	7B	LLaMA	Instruction tuning	Chinese biomedical tasks	✓	✓
HuatuoGPT [10]	2023.05	7B	Bloomz-7b1-mt	Supervised fine-tuning, RLAIF	Medical exams, research queries	✓	✓
Med-PaLM [29]	2022.12	540B	Flan-PaLM	Prompt tuning	Medical knowledge evaluation	✓	-
MedChatZH [84]	2024.03	7B	BaiChuan	Prompt tuning	Chinese medical dialogue	✓	✓
Radiology-GPT [85]	2024.03	7B	Alpaca-7B	Instruction tuning, LoRA	Radiology report generation	✓	✓
RadOnc-GPT [86]	2023.11	-	LLaMA2	Instruction tuning, LoRA	Radiation treatment planning	✓	-
scBERT [9]	2022.09	-	BERT	From scratch	Single-cell RNA analysis	✓	✓
scGPT [87]	2024.02	-	Transformer	From scratch	Single-cell multi-omics analysis	✓	✓
Taiyi [88]	2024.02	7B	Qwen-7B-base	Supervised fine-tuning	Multilingual biomedical NLP	✓	✓
OARA [89]	2024.02	7B	Vicuna v1.5	LoRA	Surgical/anesthetic education	✓	-
Med-PaLM 2 [8]	2023.05	340B	PaLM2	Instruction tuning, LoRA	Advanced medical Q&A	✓	✓
Hypnos [90]	2024.03	7B	LLaMA	LoRA	Anesthesiology tasks	✓	-
VetLLM [91]	2023.12	7B	Alpaca-7B	LoRA	Veterinary diagnosis	✓	✓
BioMedLM [19]	2024.03	2.7B	GPT-2	From scratch	Biomedical Q&A	✓	✓
CancerGPT [28]	2023.04	124M	GPT	K-SHOT	Drug synergy prediction	✓	-
ESM-2 [92]	2023.03	15B	-	From scratch	Protein structure prediction	✓	✓
HuatuoGPT II [93]	2023.11	7/13B	Baichuan2-7/13B-Base	Instruction tuning	TCM tasks	✓	✓
DoctorGLM [94]	2023.04	6B	ChatGLM	LoRA	Chinese medical Q&A	✓	✓
MedCPT [95]	2023.11	-	-	-	Biomedical information retrieval	✓	✓
BioGPT [27]	2023.04	-	GPT-2	From scratch	Biomedical text generation	✓	✓
GatorTronGPT [17]	2023.11	5B/20B	GPT-3	From scratch	Medical text synthesis	✓	✓
MEDITRON [96]	2023.11	7B/70B	LLaMA-2	Instruction tuning	Medical text comprehension	✓	✓
ClinicalGPT [97]	2023.06	7B	BLOOM-7B	LoRA	Clinical tasks	✓	-
Qilin-Med [98]	2024.04	-	Baichuan	-	Multi-stage medical training	✓	-
MedAlpaca [99]	2023.01	7/13B	LLaMA	LoRA	Open-source medical LLM	✓	-
Alpacare [100]	2024.05	-	-	Instruction tuning	Medical instruction following	✓	✓
Zhongjing [71]	2023.12	13B	Ziya-LLaMA-13B-v13	Supervised fine-tuning	TCM Q&A	✓	✓
Cplm [101]	2024.05	13B/2.7B	LLaMA2, PubMedGPT	LoRA	Clinical prediction	✓	✓
MMedLM 2 [68]	2024.02	7B	InternLM	LoRA	Multilingual medical Q&A	✓	✓
AlphaFold 3 [102]	2024.05	-	-	-	Protein structure prediction	✓	-
Bingo [103]	2023.11	15B	ESM-2	From scratch	Protein-coding gene prediction	✓	
BiomedCLIP [104]	2024.01	-	CLIP	-	Multimodal biomedical tasks	✓	
Med-PaLm M [66]	2023.07	12B/84B/562B	Palm-E	Instruction tuning	Multimodal medical analysis	✓	
MONET [105]	2024.04	-	CLIP	From scratch	Medical image annotation	✓	
XrayGPT [106]	2023.06	-	MedCLIP, Vicuna	Modality alignment	Chest X-ray analysis	✓	
Med-MLLM [107]	2023.12	-	-	Multi-stage training	X-ray representation learning	-	
EchoCLIP [108]	2024.04	-	OpenCLIP	From scratch	Echocardiogram interpretation	✓	
OphGLM [109]	2023.06	6B	ChatGLM	Instruction tuning	Ophthalmology diagnosis	✓	
ClinicalBLIP [110]	2024.02	3B	InstructBLIP	LoRA	Radiology report generation	-	
Med-Gemini [111]	2024.04	-	Gemini	Instruction tuning	Multimodal medical analysis	-	
BioMedGPT [112]	2023.08	Instruction tuning	LLaMA2	-	Biomedical question answering	✓	

Appendix 1: Representative basic and advanced abilities and corresponding representative datasets for evaluating

Level	Ability	Task	Dataset
Basic	Language Generation	Language Modeling	Penn Treebank [540], WikiText-103 [541], the Pile [166], LAMBADA [252]
		Conditional Text Generation	WMT'14,16,19,20,21,22 [542–547], Flores-101 [548], DiaBLA [549], CNN/DailyMail [550], XSum [551], WikiLingua [552]
		Code Synthesis	APPS [376], HumanEval [105], MBPP [223], CodeContest [114], MTPB [86], DS-1000 [554], ODEX [555]
	Knowledge Utilization	Closed-Book QA	Natural Questions [556], ARC [557], TruthfulQA [558], Web Questions [559], TriviaQA [560], PIQA [561], LC-quad2.0 [562], GrailQA [563], KQapro [564], CWQ [565], MKQA [566], ScienceQA [567]
		Open-Book QA	Natural Questions [556], OpenBookQA [568], ARC [557], TriviaQA [560], Web Questions [559], MS MARCO [569], QASC [570], SQuAD [571], WikiMovies [572]
	Knowledge Completion		WikiFact [573], FB15k-237 [574], Freebase [575], WN18RR [576], WordNet [577], LAMA [578], YAGO3-10 [579], YAGO [580]
Complex Reasoning	Knowledge Reasoning		CSQA [506], StrategyQA [199], HotpotQA [581], ARC [557], BoolQ [582], PIQA [561], SIQA [583], HellaSwag [584], WinoGrande [585], COPA [586], OpenBookQA [568], ScienceQA [567], proScript [587], ProPara [588], ExplaGraphs [589], ProofWriter [590], EntailmentBank [591], ProOntoQA [592]
			CoinFlip [33], ReverseList [33], LastLetter [33], Boolean Assignment [593], Parity [593], Colored Object [70], Penguins in a Table [70], Repeat Copy [436], Object Counting [436]
	Mathematical Reasoning		MATH [362], GSM8k [198], SVAMP [594], MultiArith [595], ASDiv [505], MathQA [596], AQUA-RAT [597], MAWPS [598], DROP [599], NaturalProofs [600], PISA [601], miniF2F [602], ProofNet [603]

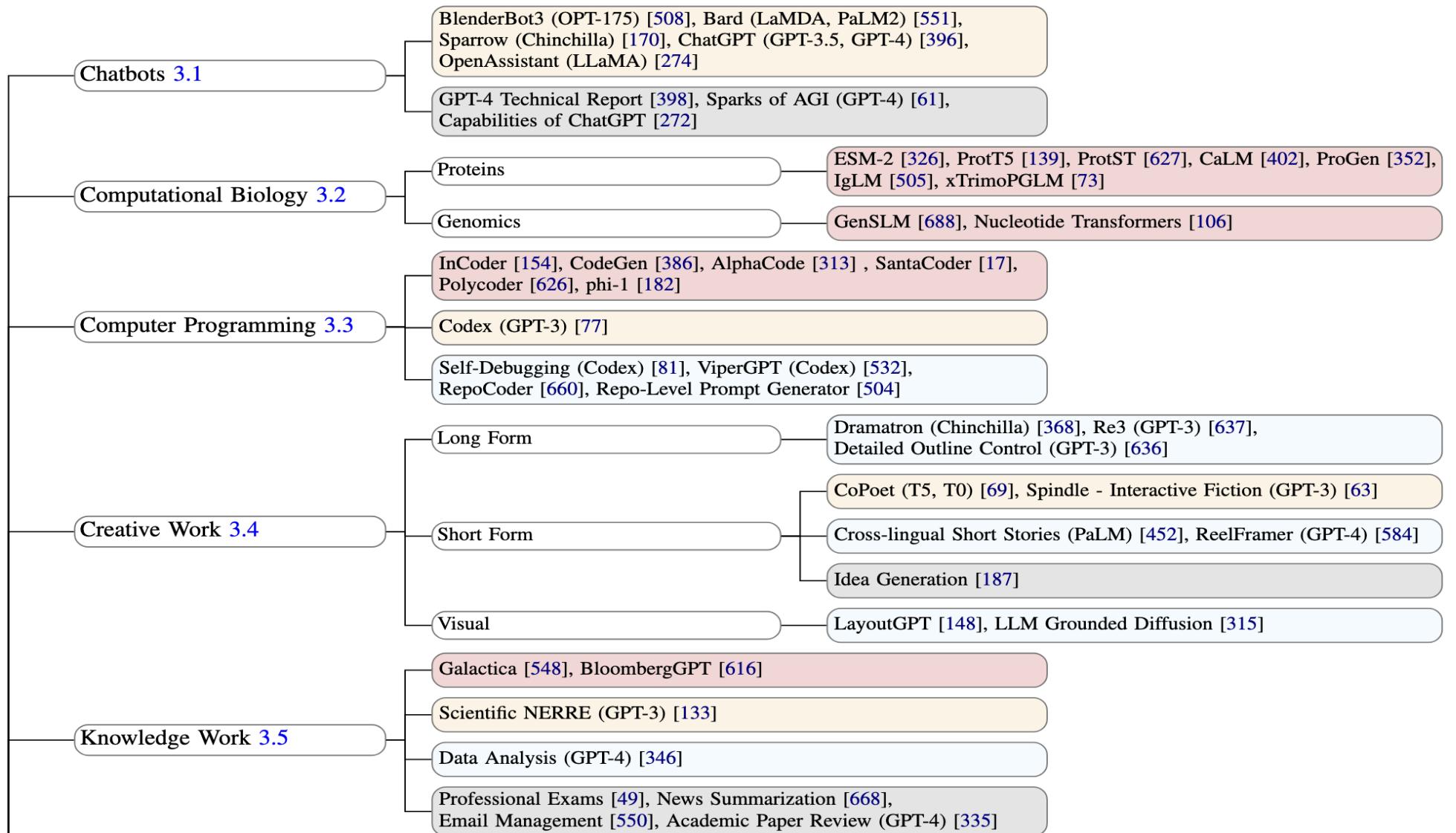
Appendix 1: Representative basic and advanced abilities and corresponding representative datasets for evaluating

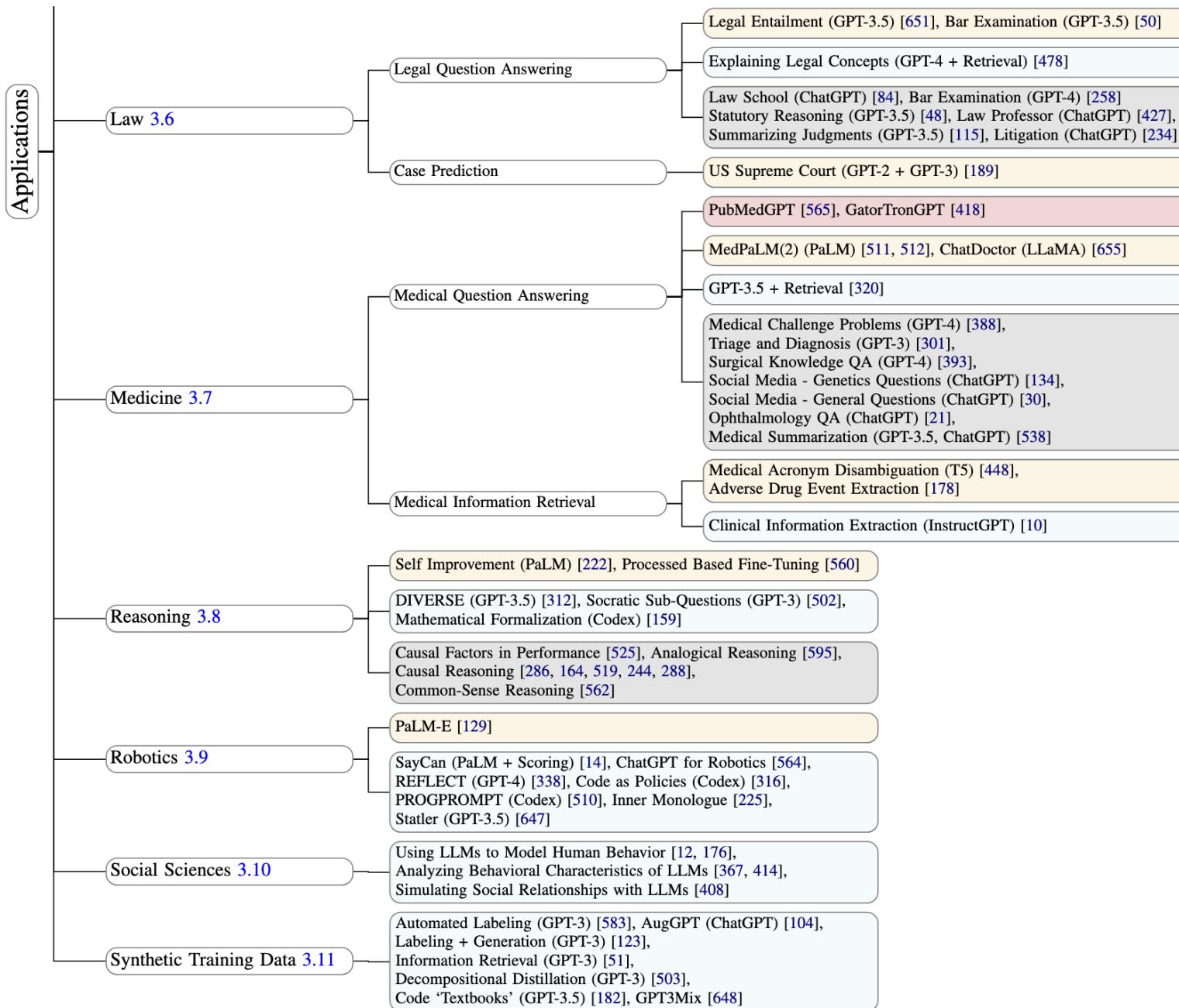
		Honestness	TruthfulQA [558], HaluEval [604]
		Helpfulness	HH-RLHF [183]
		Harmlessness	HH-RLHF [183], Crows-Pairs [605] WinoGender [606], RealToxicityPrompts [607]
Advanced	Interaction with External Environment	Household	VirtualHome [608], BEHAVIOR [609], ALFRED [610], ALFWORLD [611]
		Website Environment	WebShop [612], Mind2Web [613]
		Open World	MineRL [614], MineDojo [615]
Tool Manipulation	Search Engine		HotpotQA [581], TriviaQA [560], Natural Questions [556]
		Code Executor	GSM8k [198], TabMWP [616], Date Understanding [70]
	Calculator		GSM8k [198], MATH [362], CARP [617]
		Model Interface	GPT4Tools [618], Gorilla [619]
	Data Interface		WebQSP [620], MetaQA [621], WTQ [622] WikiSQL [623], TabFact [624], Spider [625]

Appendix 2: Benchmarks and Evaluation

Method	Evaluation	Model Types	Abilities/Domain	Data Source
Benchmark	MMLU [362]	Base/Fine-tuned/Specialized	General	Human exam/practice
	BIG-bench [70]	Base/Fine-tuned/Specialized	General	Human annotation
	HELM [522]	Base/Fine-tuned/Specialized	General	Benchmark collection
	Open LLM Leaderboard [709]	Base/Fine-tuned/Specialized	General	Benchmark collection
	AGIEval [710]	Base/Fine-tuned/Specialized	General	Human exam/practice
	MMCU [711]	Base/Fine-tuned/Specialized	General	Human exam/practice
	M3KE [712]	Base/Fine-tuned/Specialized	General	Human exam/practice
	C-Eval [713]	Base/Fine-tuned/Specialized	General	Human exam/practice
	Xiezhi [714]	Base/Fine-tuned/Specialized	General	Human exam/practice
	OpenCompass [715]	Base/Fine-tuned/Specialized	General	Benchmark collection
	Chain-of-Thought Hub [716]	Base/Fine-tuned	General	Benchmark collection
	KoLA [717]	Base/Fine-tuned	Knowledge utilization	Web
	ARB [718]	Fine-tuned	Complex reasoning	Human exam/practice
	APIBench [719]	Base/Fine-tuned	Tool manipulation	Web
	APIBank [720]	Fine-tuned	Tool manipulation	Synthesis
	ToolAlpaca [721]	Base/Fine-tuned	Tool manipulation	Synthesis
	T-Bench [722]	Fine-tuned	Tool manipulation	Synthesis
	ToolBench [723]	Fine-tuned	Tool manipulation	Synthesis
	BOLAA [724]	Base/Fine-tuned	Environment interaction	Benchmark collection
Human	AgentBench [725]	Base/Fine-tuned	Environment interaction	Human annotation/Synthesis
	HaluEval [604]	Base/Fine-tuned	Human alignment	Human annotation/Synthesis
	PromptBench [726]	Base/Fine-tuned	Robustness	Benchmark collection
	HumanEval [105]	Base/Fine-tuned/Specialized	Code synthesis	Human annotation
	MultiMedQA [354]	Specialized	Healthcare	Benchmark collection
	FLUE [727]	Specialized	Finance	Benchmark collection
	LegalBench [728]	Specialized	Legal	Human annotation
Model	Chatbot Arena [729]	Base/Fine-tuned/Specialized	Human Alignment	Human annotation
	SciBench [730]	Fine-tuned	Complex reasoning	Human exam/practice
	AlpacaEval [731]	Fine-tuned	Instruction following	Synthesis
	MT-bench [729]	Fine-tuned	Human alignment	Human annotation
	TrustGPT [732]	Base/Fine-tuned	Human alignment	Benchmark collection
	LMExamQA [733]	Base/Fine-tuned	Knowledge utilization	Synthesis
	ChatEval [734]	Base/Fine-tuned	Knowledge utilization	Benchmark collection

Appendix 3: LLM Applications





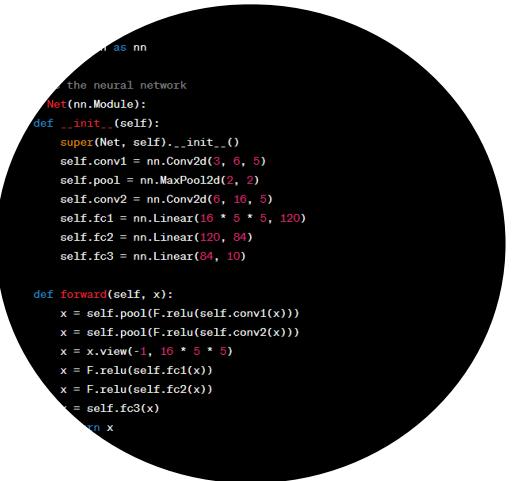
References

- Ansuini, A., Laio, A., Macke, J. H., & Zoccolan, D.** Intrinsic dimension of data representations in deep neural networks. In Proceedings of the 33rd International Conference on Neural Information Processing Systems 6111–6122 (Curran Associates Inc., Red Hook, NY, USA, 2019).
- Brown, T. B. et al.** Language Models are Few-Shot Learners. Preprint at <https://doi.org/10.48550/arXiv.2005.14165> (2020).
- Chaves, J. M. Z. et al.** Tx-LLM: A Large Language Model for Therapeutics. Preprint at <https://doi.org/10.48550/arXiv.2406.06316> (2024).
- Christiano, P. et al.** Deep reinforcement learning from human preferences. Preprint at <https://doi.org/10.48550/arXiv.1706.03741> (2023).
- Dagdelen, J. et al.** Structured information extraction from scientific text with large language models. *Nat. Commun.* 15, 1418 (2024).
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K.** BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Preprint at <https://doi.org/10.48550/arXiv.1810.04805> (2019).
- DeepSeek-AI et al.** DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. Preprint at <https://doi.org/10.48550/arXiv.2501.12948> (2025).
- Gao, S. et al.** Empowering biomedical discovery with AI agents. *Cell* 187, 6125–6151 (2024).
- Gao, Y. et al.** Retrieval-Augmented Generation for Large Language Models: A Survey. Preprint at <https://doi.org/10.48550/arXiv.2312.10997> (2024).
- Han, Z., Gao, C., Liu, J., Zhang, J., & Zhang, S. Q.** Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey. Preprint at <https://doi.org/10.48550/arXiv.2403.14608> (2024).
- Hoffmann, J. et al.** Training Compute-Optimal Large Language Models. Preprint at <https://doi.org/10.48550/arXiv.2203.15556> (2022).
- Hu, E. J. et al.** LoRA: Low-Rank Adaptation of Large Language Models. Preprint at <https://doi.org/10.48550/arXiv.2106.09685> (2021).
- Inoue, Y., Song, T., & Fu, T.** DrugAgent: Explainable Drug Repurposing Agent with Large Language Model-based Reasoning. Preprint at <https://doi.org/10.48550/arXiv.2408.13378> (2024).
- Kaddour, J. et al.** Challenges and Applications of Large Language Models. Preprint at <https://doi.org/10.48550/arXiv.2307.10169> (2023).
- Kaplan, J. et al.** Scaling Laws for Neural Language Models. Preprint at <https://doi.org/10.48550/arXiv.2001.08361> (2020).
- Li, X. L. & Liang, P.** Prefix-Tuning: Optimizing Continuous Prompts for Generation. Preprint at <https://doi.org/10.48550/arXiv.2101.00190> (2021).
- Lester, B., Al-Rfou, R., & Constant, N.** The Power of Scale for Parameter-Efficient Prompt Tuning. Preprint at <https://doi.org/10.48550/arXiv.2104.08691> (2021).
- Lialin, V., Deshpande, V., Yao, X., & Rumshisky, A.** Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning. Preprint at <https://doi.org/10.48550/arXiv.2303.15647> (2024).

References

- Lu, Z. et al.** Large language models in biomedicine and health: current research landscape and future directions. *J. Am. Med. Inform. Assoc.* 31, 1801–1811 (2024).
- Ouyang, L. et al.** Training language models to follow instructions with human feedback. Preprint at <https://doi.org/10.48550/arXiv.2203.02155> (2022).
- OpenAI et al.** GPT-4 Technical Report. Preprint at <https://doi.org/10.48550/arXiv.2303.08774> (2024).
- Rafailov, R. et al.** Direct Preference Optimization: Your Language Model is Secretly a Reward Model. Preprint at <https://doi.org/10.48550/arXiv.2305.18290> (2024).
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O.** Proximal Policy Optimization Algorithms. Preprint at <https://doi.org/10.48550/arXiv.1707.06347> (2017).
- Wang, C. et al.** A Survey for Large Language Models in Biomedicine. Preprint at <https://doi.org/10.48550/arXiv.2409.00133> (2024).
- Wei, J. et al.** Finetuned Language Models Are Zero-Shot Learners. Preprint at <https://doi.org/10.48550/arXiv.2109.01652> (2022).
- Wei, J. et al.** DrugReAlign: a multisource prompt framework for drug repurposing based on large language models. *BMC Biol.* 22, 226 (2024).
- Xiao, H. et al.** A Comprehensive Survey of Large Language Models and Multimodal Large Language Models in Medicine. *Inf. Fusion* 117, 102888 (2025).
- Yao, S. et al.** ReAct: Synergizing Reasoning and Acting in Language Models. Preprint at <https://doi.org/10.48550/arXiv.2210.03629> (2023).
- Yenduri, G. et al.** GPT (Generative Pre-Trained Transformer)—A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. *IEEE Access* 12, 54608–54649 (2024).
- Zhao, W. X. et al.** A Survey of Large Language Models. Preprint at <https://doi.org/10.48550/arXiv.2303.18223> (2024).

How to succeed in this course?



Practice



Discuss



Explore



Visualize



Ask