

机器学习

第十一讲:大语言模型

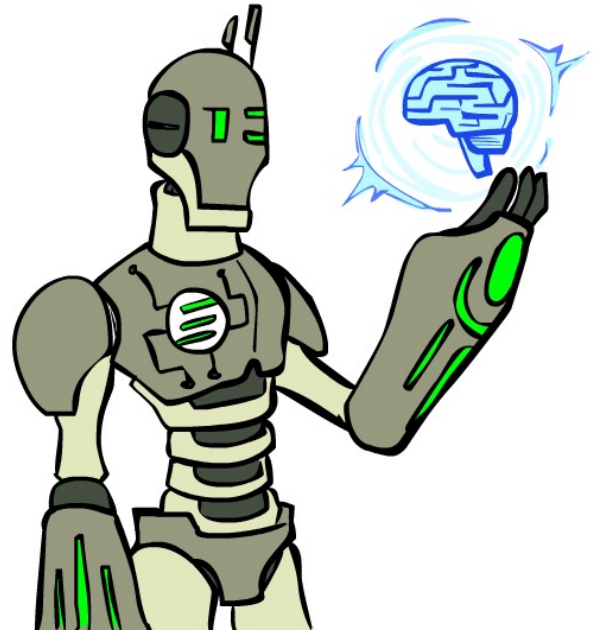
顾小东

上海交通大学计算机学院



Today

- Pre-trained Language Models
- Large Language Models



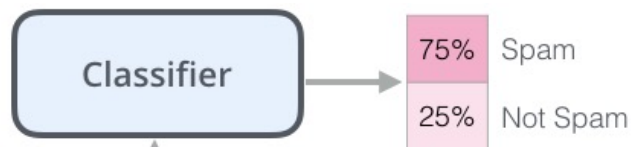
预训练语言模型

在**超大规模**文本语料(如电子书、维基等)上进行**无监督**训练



Phase 1: 预训练

在特定下游任务 (如垃圾邮件分类)的标注数据上进行**有监督**训练

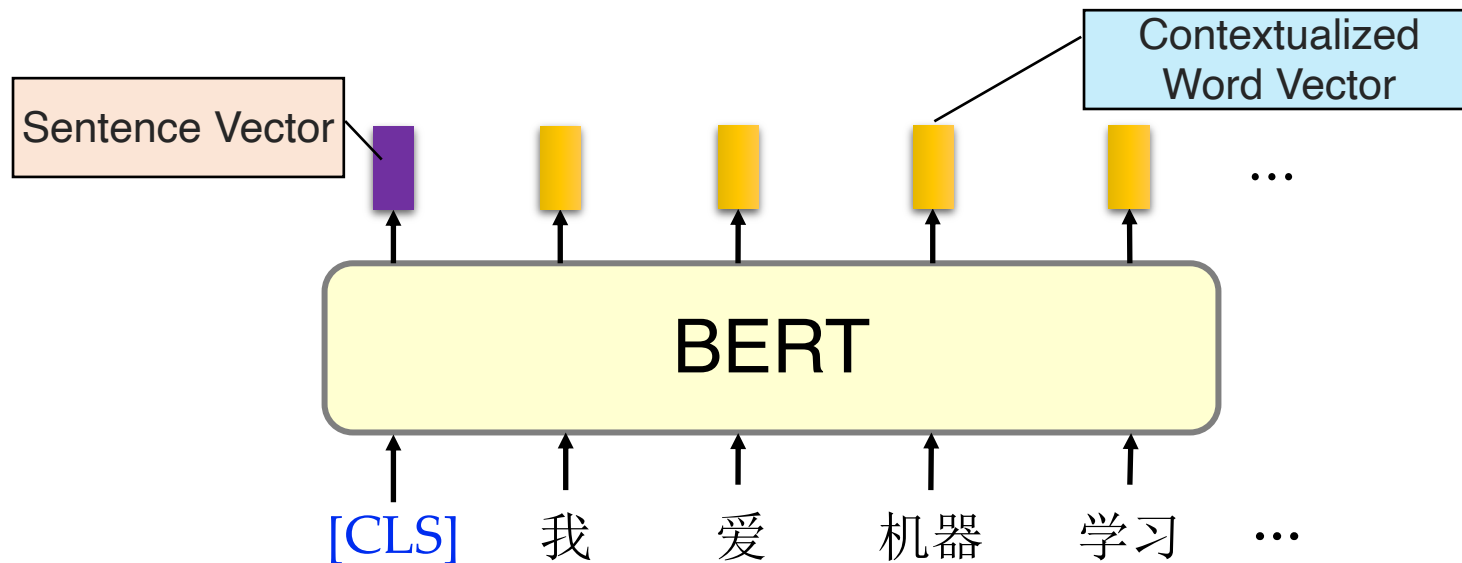


Phase 2: 微调

BERT – Bidirectional Encoder Representations from Transformers

一种Transformer编码器

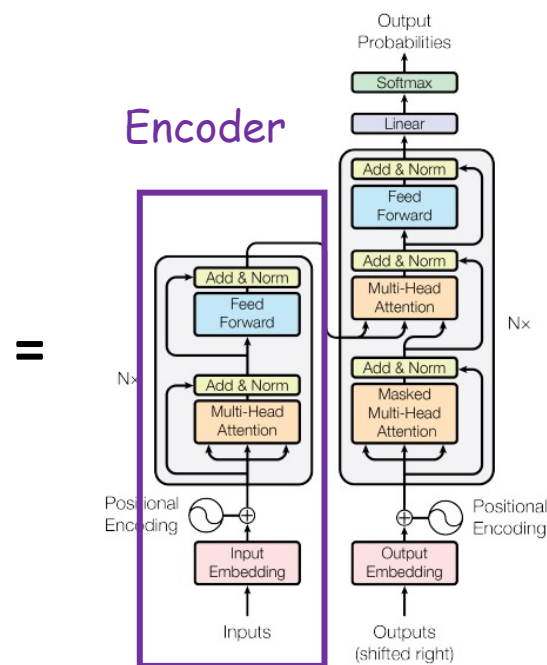
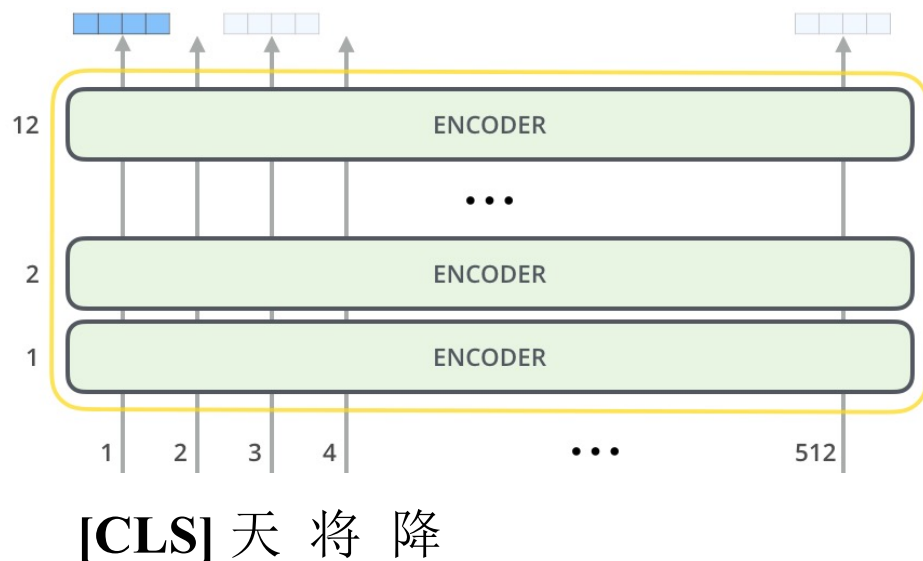
- 用来学习单词和句子的向量表征
- 在大规模文本语料上预训练，然后
- 在小规模特定任务数据集(e.g., classification, QA.)上微调



模型结构

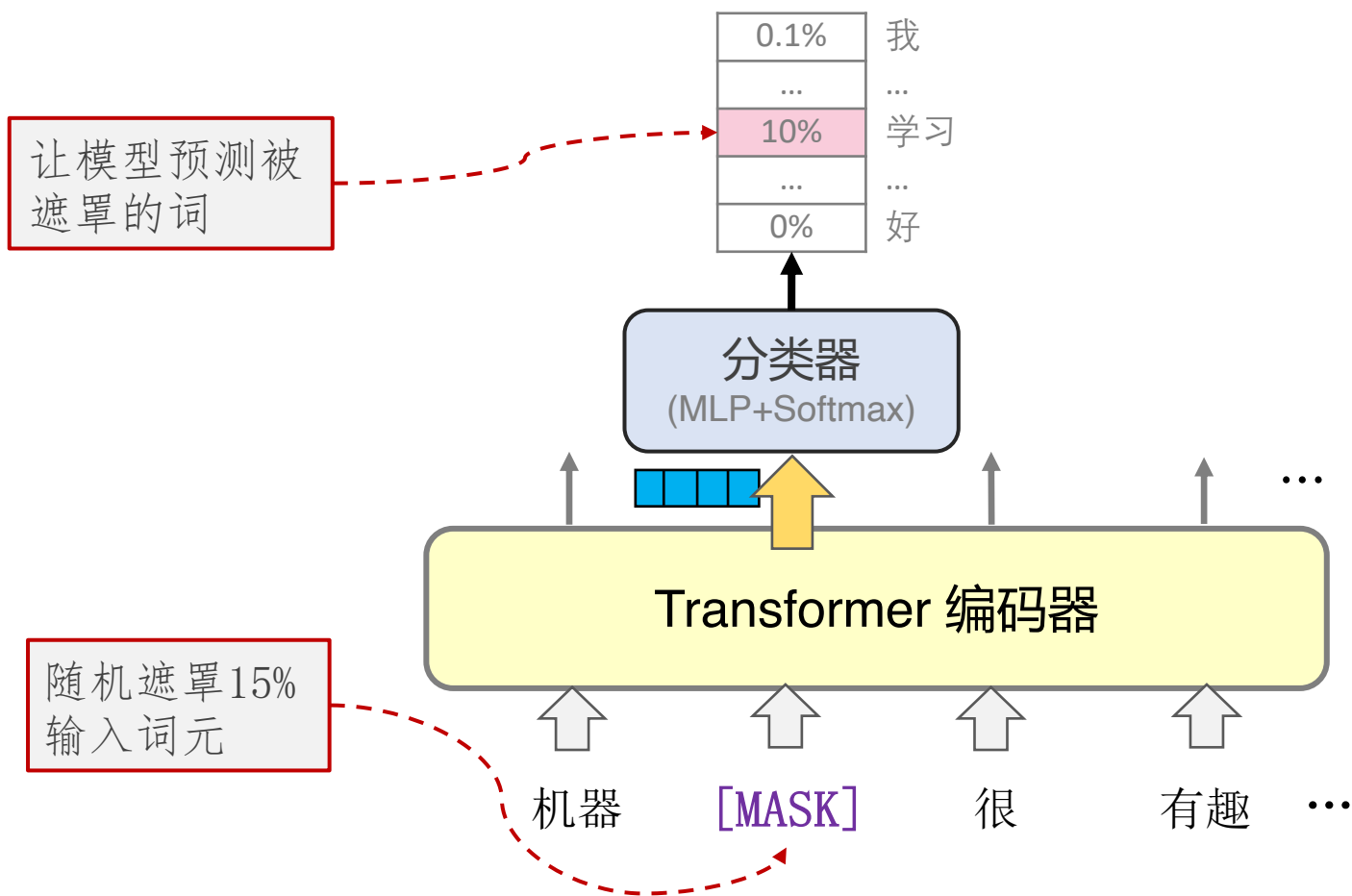
BERT = Transformer Encoder

- 和Transformer编码器一样, BERT 输入一个单词序列。
- 首字符用一个特殊符号[CLS] 表示该位置用于句子分类。



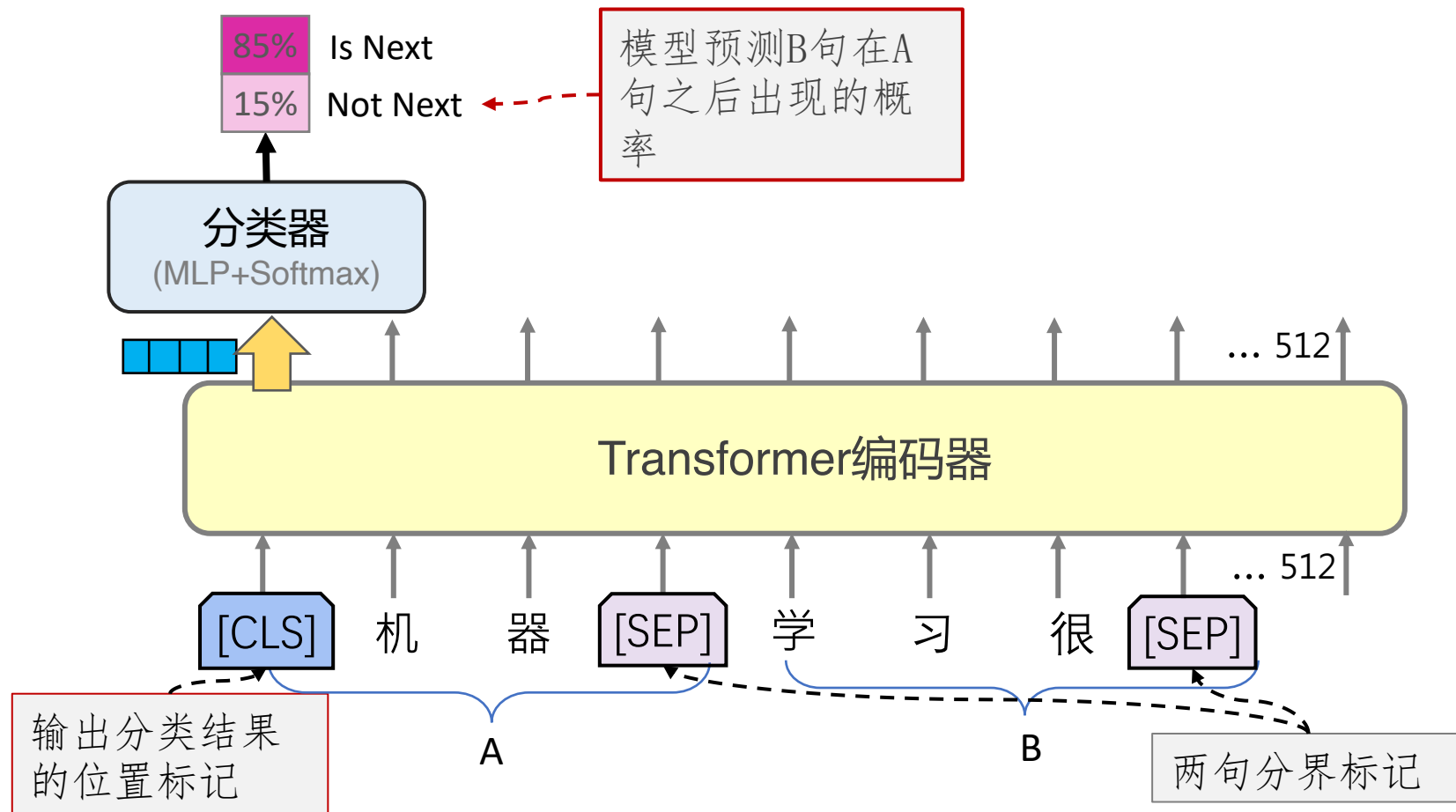
预训练

任务1: 遮罩语言模型



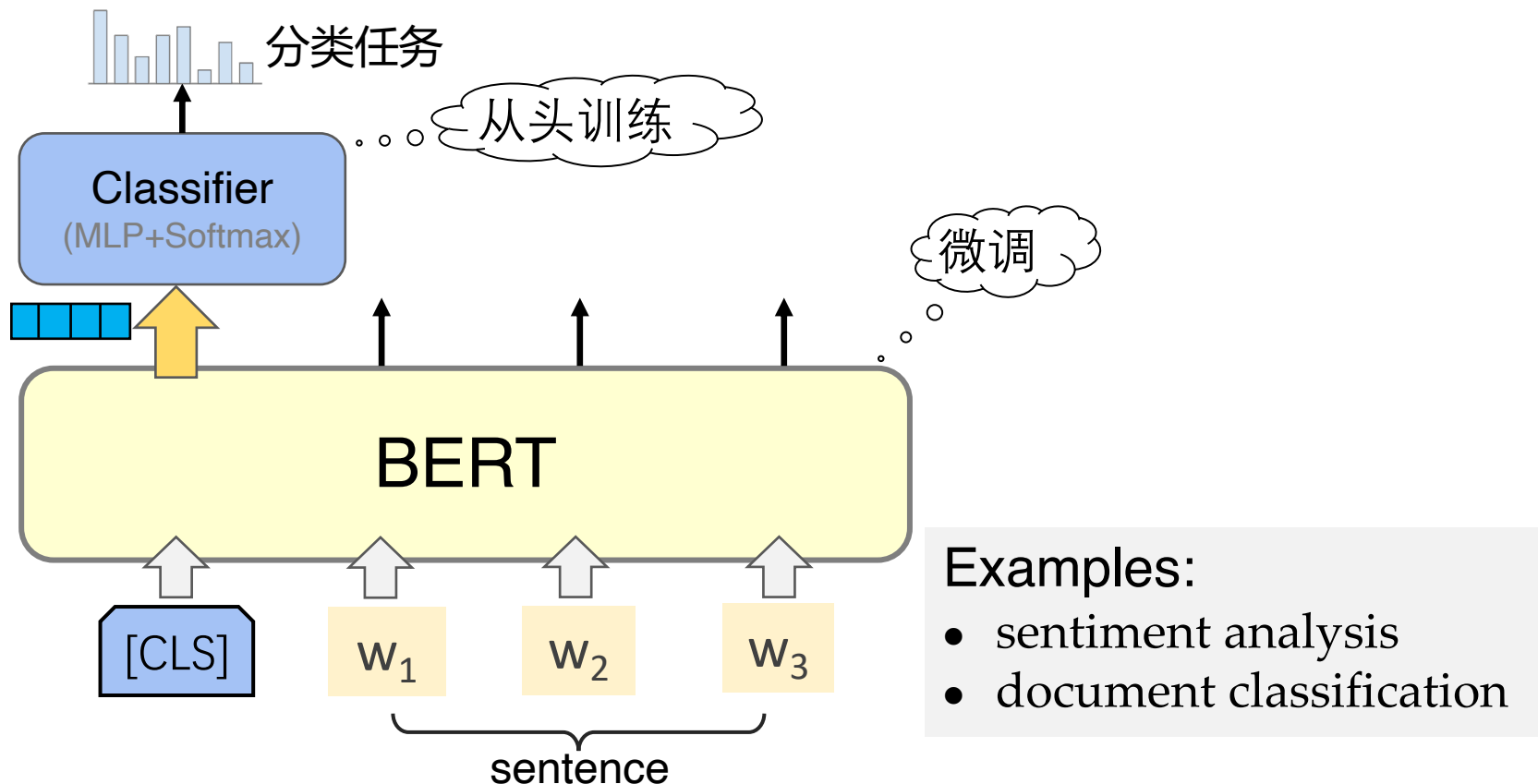
预训练

任务2: 相邻句预测



微调任务一：句子分类

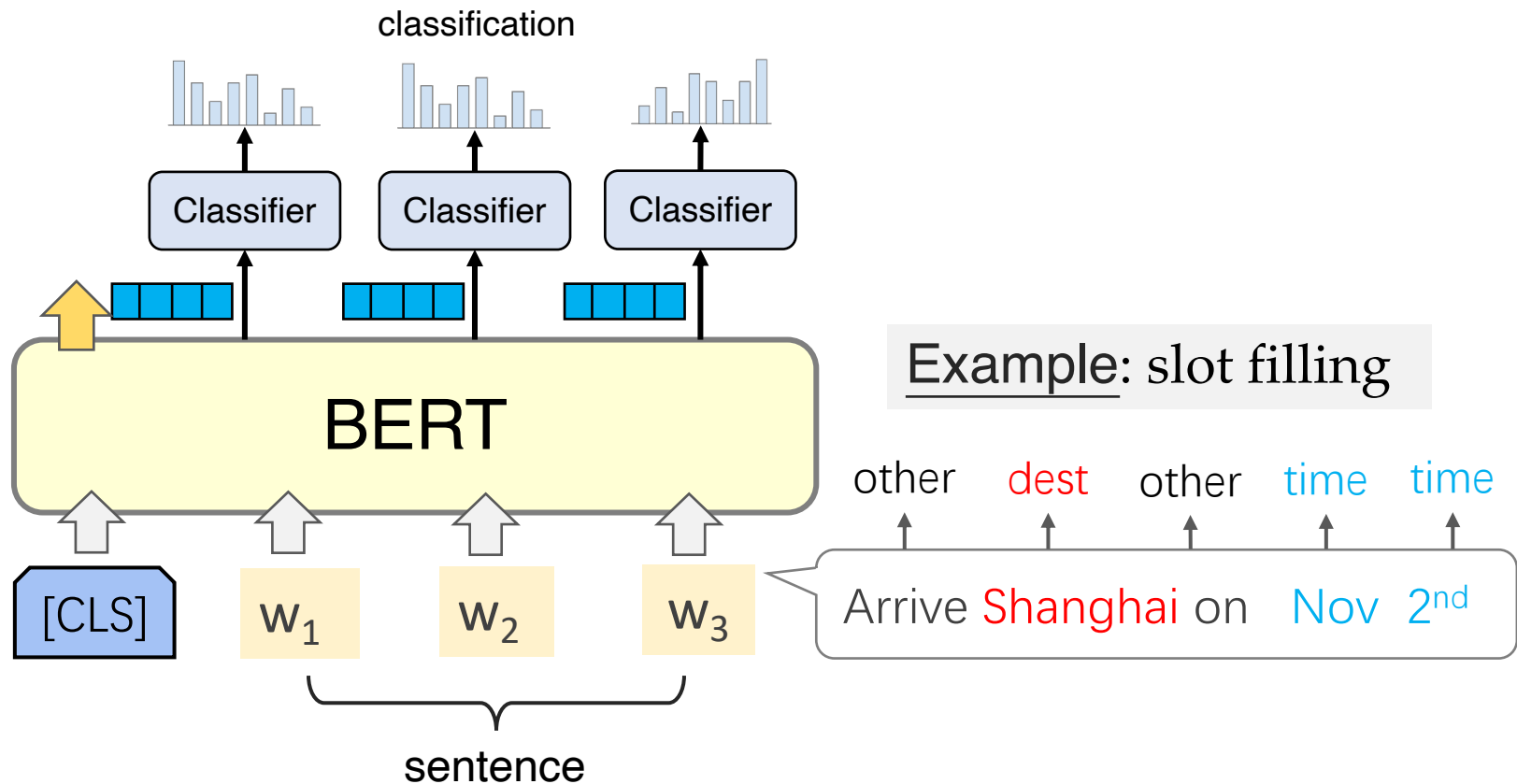
- **Input:** a single sentence, **Output:** sentence class



微调任务二：单词标注

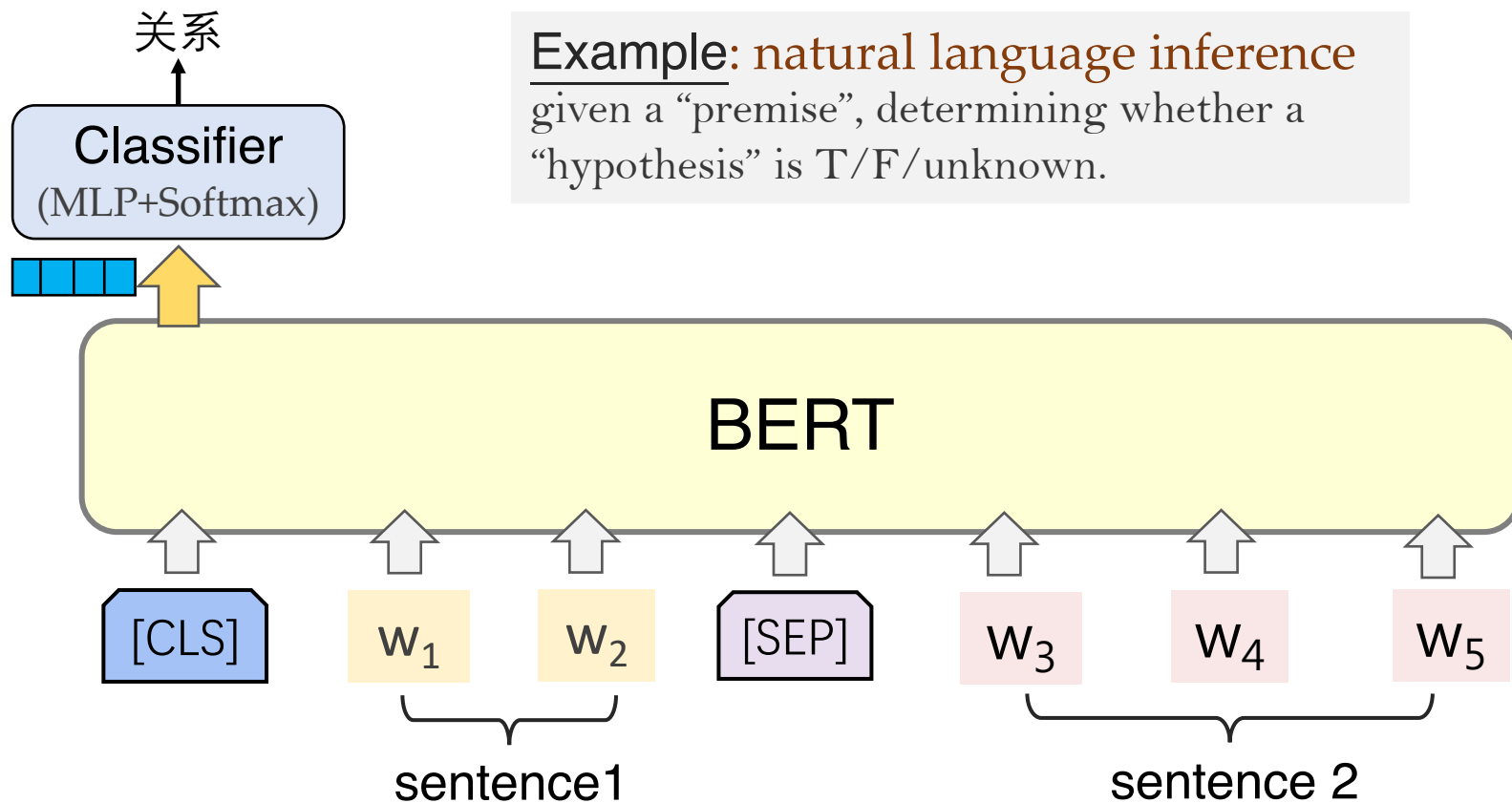
Input: a single sentence

Output: classification of each word



微调任务三：句子对关系分类

input: two sentences output: relationship



Generative Pre-Trained Transformer (GPT)

- 基于Transformer**解码器**的预训练语言模型

GPT-3 (175B)



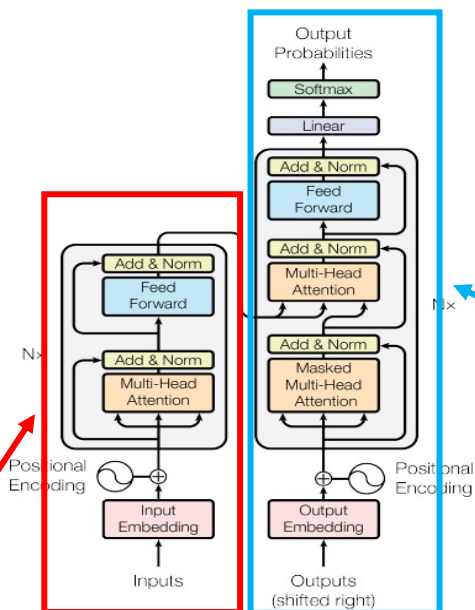
[OpenAI 2020]

GPT-2
(1542M)

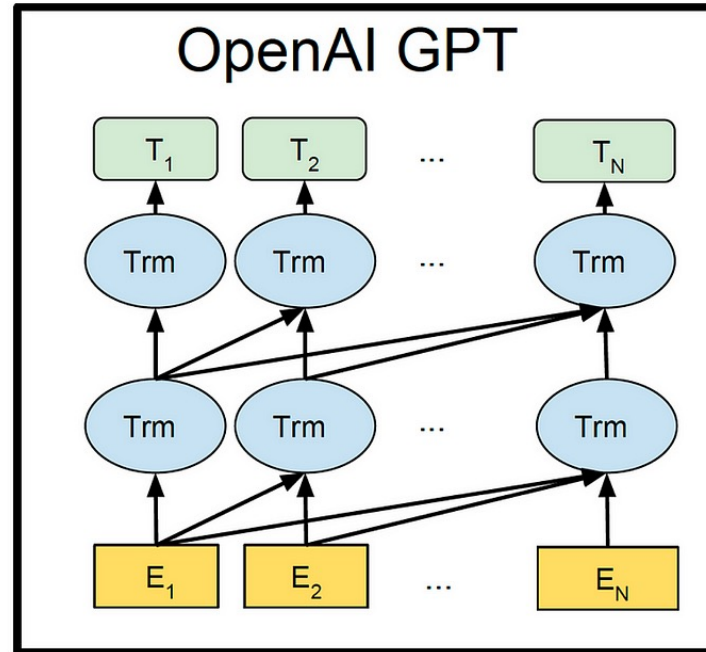
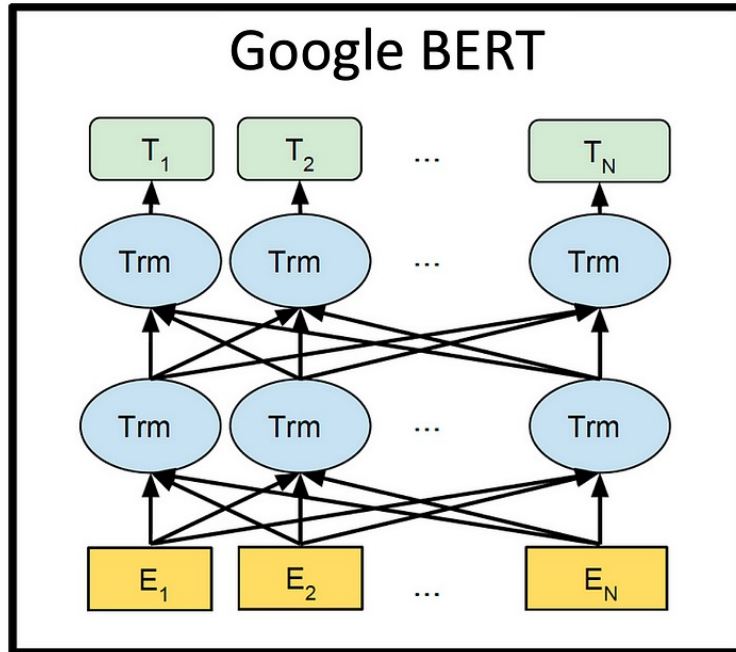


[OpenAI 2019]

BERT
(340M)



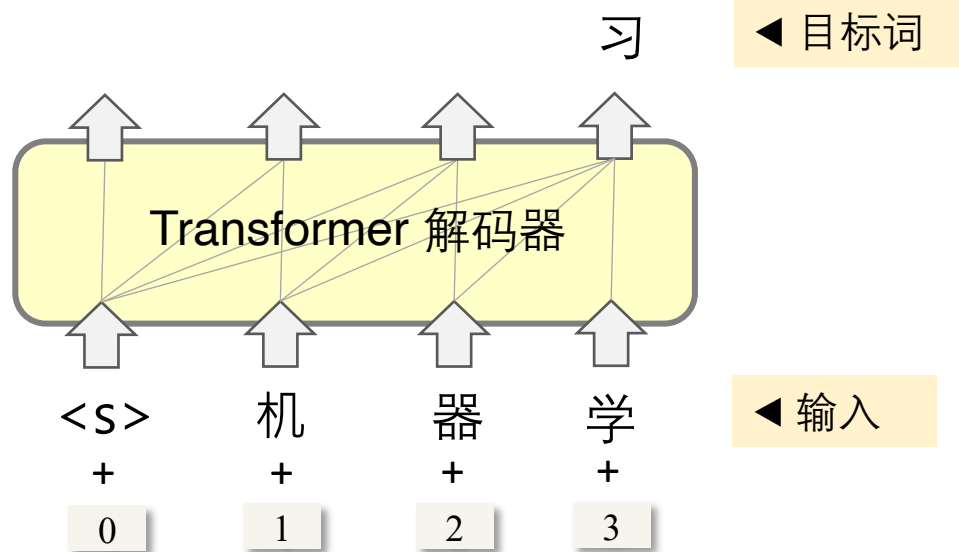
BERT vs GPT



GPT预训练

预训练任务: 就是简单的语言模型

— 输入前 $n-1$ 个单词, 让模型预测第 n 个单词.

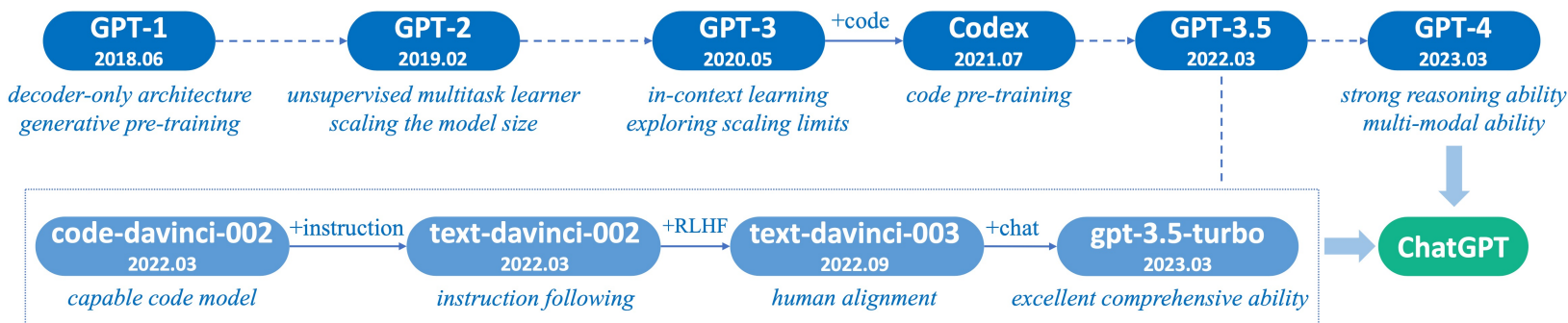


训练目标函数：

$$l(\theta) = - \sum_{t=1}^T \log p_{\theta}(x_t | x_1, \dots, x_{t-1})$$

<https://github.com/huggingface/transformers/tree/main/src/transformers/models/gpt2>

GPT 发展史

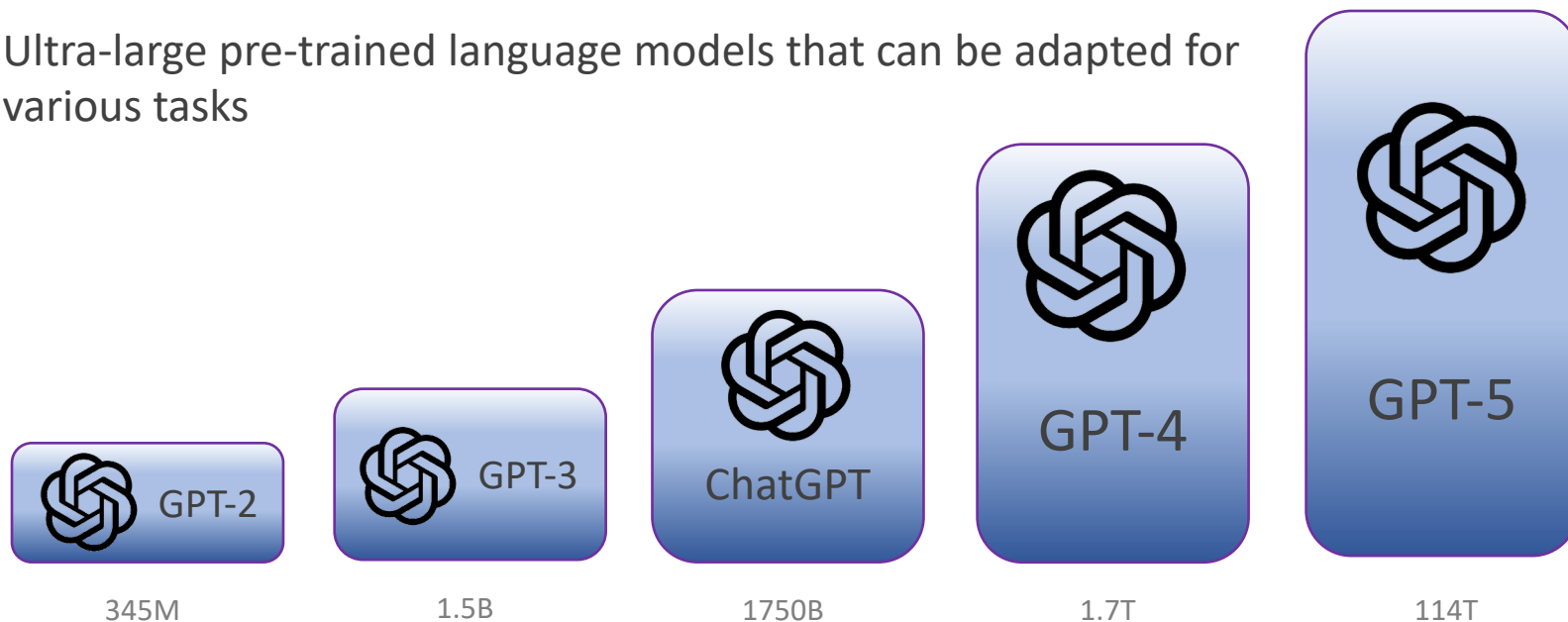


Simulated exams	GPT-4 estimated percentile	GPT-4 (no vision) estimated percentile	GPT-3.5 estimated percentile
Uniform Bar Exam (MBE+MEE+MPT) ¹	298 / 400 ~90th	298 / 400 ~90th	213 / 400 ~10th
LSAT	163 ~88th	161 ~83rd	149 ~40th
SAT Evidence-Based Reading & Writing	710 / 800 ~93rd	710 / 800 ~93rd	670 / 800 ~87th
SAT Math	700 / 800 ~89th	690 / 800 ~89th	590 / 800 ~70th
Graduate Record Examination (GRE) Quantitative	163 / 170 ~80th	157 / 170 ~62nd	147 / 170 ~25th
Graduate Record Examination (GRE) Verbal	169 / 170 ~99th	165 / 170 ~96th	154 / 170 ~63rd
Graduate Record Examination (GRE) Writing	4 / 6 ~54th	4 / 6 ~54th	4 / 6 ~54th
USABO Semifinal Exam 2020	87 / 150 99th - 100th	87 / 150 99th - 100th	43 / 150 31st - 33rd
USNCO Local Section Exam 2022	36 / 60	38 / 60	24 / 60
Medical Knowledge Self-Assessment Program	75 %	75 %	53 %
Codeforces Rating	392 below 5th	392 below 5th	260 below 5th

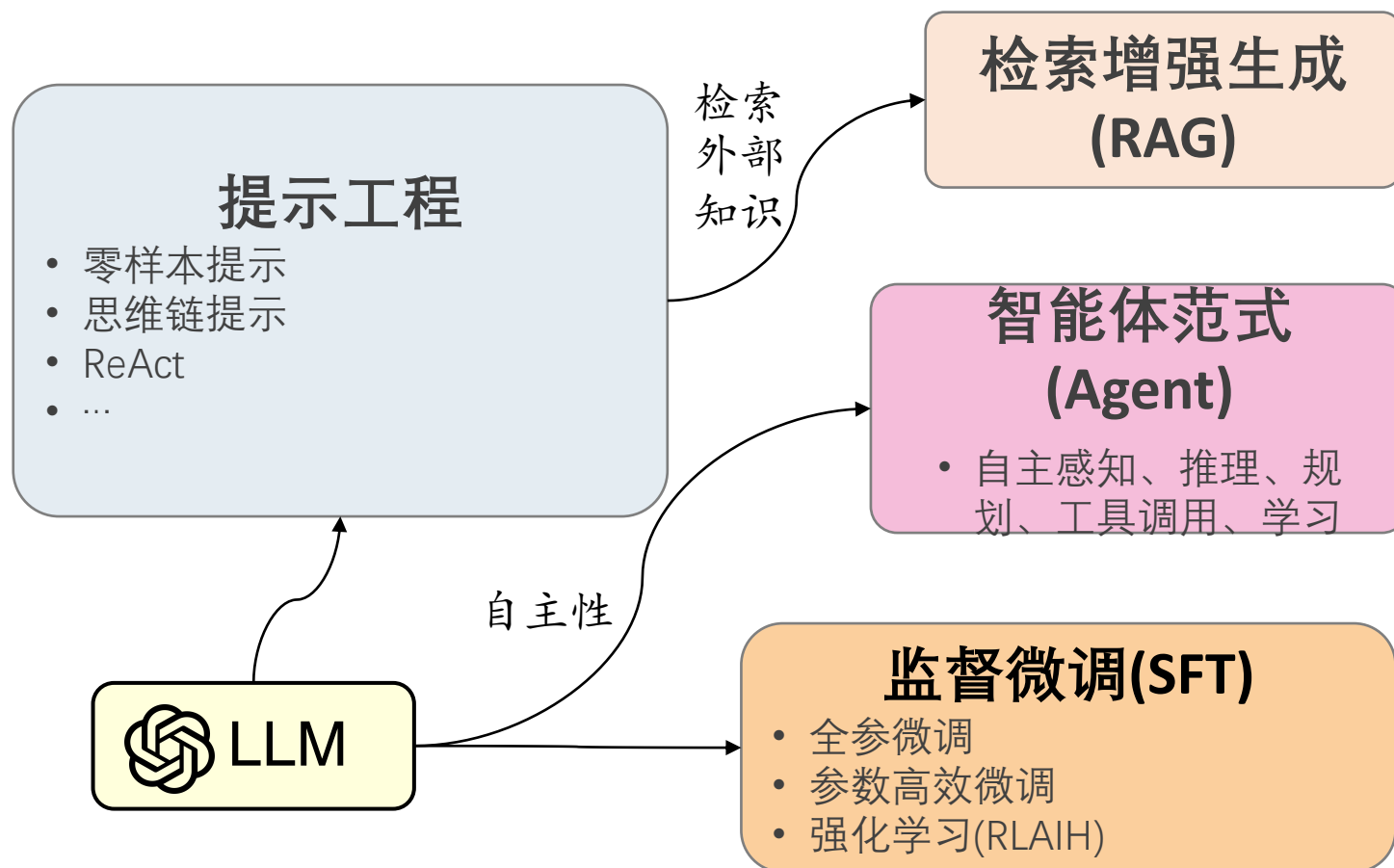
A Survey of Large Language Models <https://arxiv.org/abs/2303.18223>

大规模语言模型(LLM)

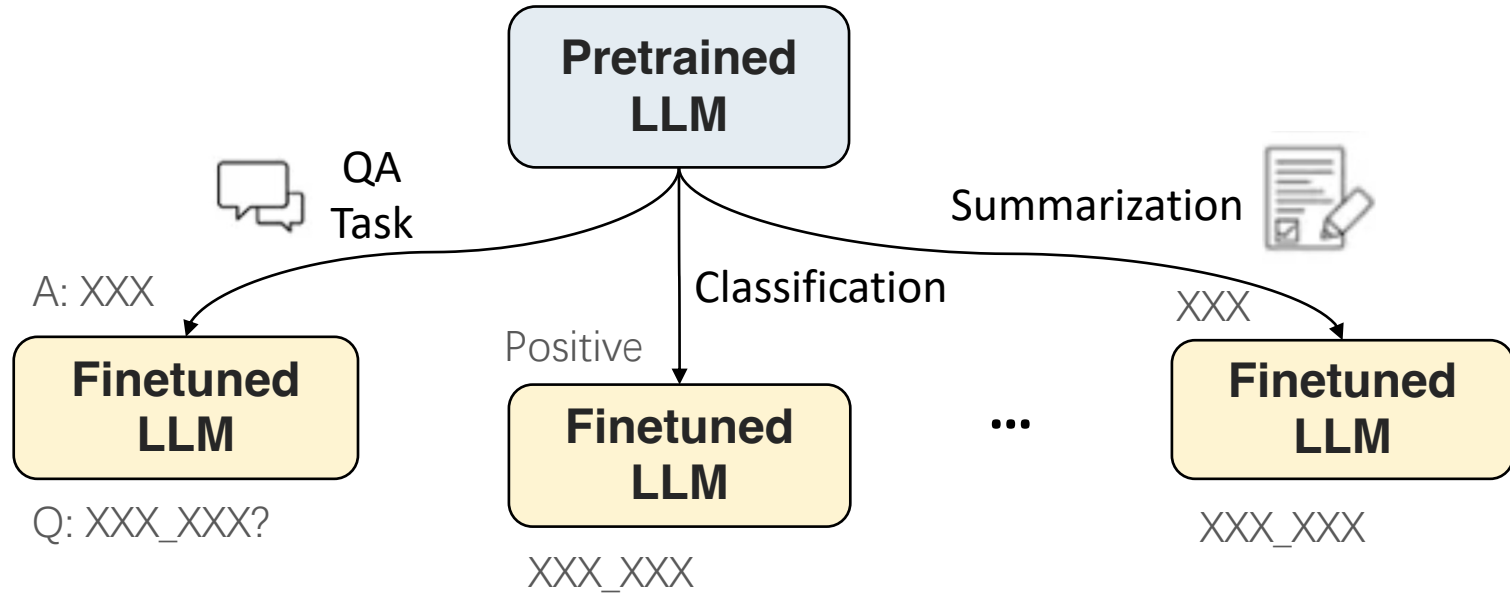
Ultra-large pre-trained language models that can be adapted for various tasks



LLM的使用方式



方式一：微调



Problems with full finetuning:

1. Computationally heavy
2. Overfitting

方式一：微调

在Huggingface网站下载模型checkpoint

The screenshot illustrates the process of finding and downloading a model checkpoint from Hugging Face. On the left, the Hugging Face logo and navigation links like 'Text Generation' and 'Inference Endpoints' are visible. A search bar contains the text 'gpt2', and a dropdown menu lists several model identifiers, with 'openai-community/gpt2' highlighted. On the right, the 'Files and versions' tab for the 'gpt2' model is shown. It displays a list of files and folders, including 'onnx', '.gitattributes', '64-8bits.tflite', '64-fp16.tflite', '64.tflite', and 'README.md'. The '64.tflite' file is highlighted, and a 'Safe' badge is visible next to it.

方式一：微调

将模型文件装在指定位置，使用以下方式微调模型

```
from transformers import GPT2LMHeadModel, AutoTokenizer, AutoModel
from transformers import TextDataset, DataCollatorForLanguageModeling
from transformers import Trainer, TrainingArguments

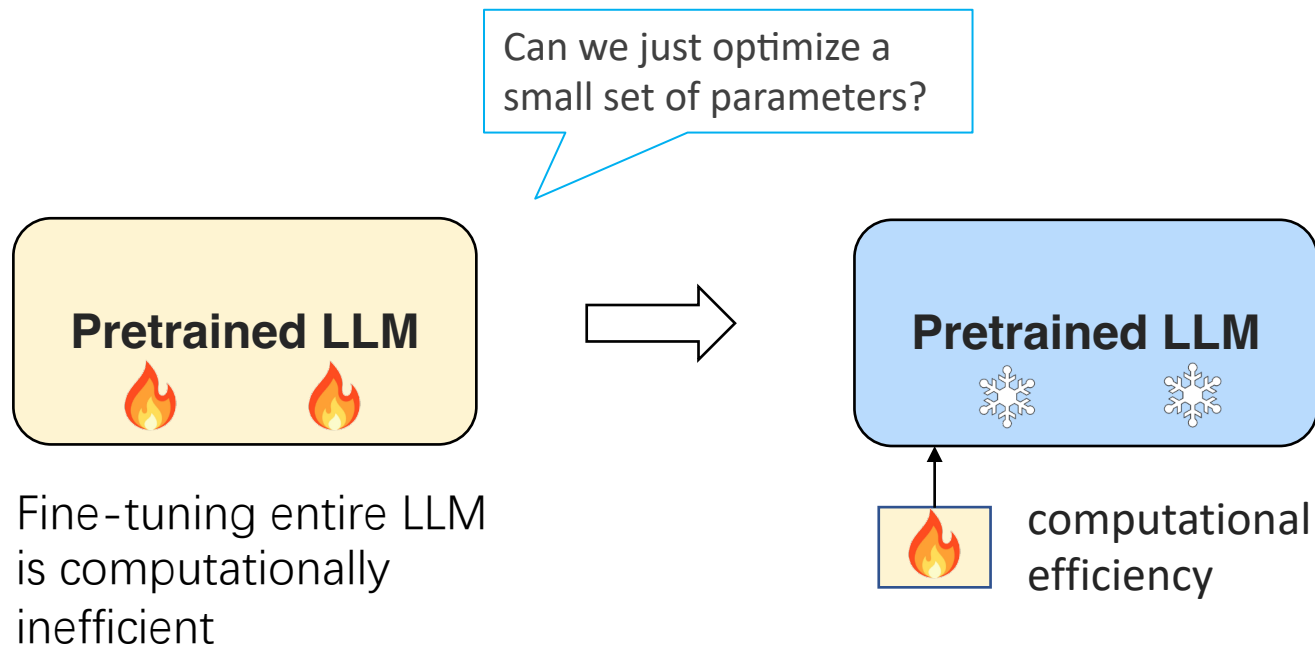
# 指定中文GPT模型和标记器
model_name = "uer/gpt2-large-chinese-cluecorpussmall"
model = GPT2LMHeadModel.from_pretrained('./model_cache/gpt2-chinese-cluecorpussmall/')
tokenizer = AutoTokenizer.from_pretrained('./model_cache/gpt2-chinese-cluecorpussmall/')

# 读取你的文本数据集
dataset_path = "./data/train.txt"

# 使用标记器编码数据集
tokenized_datasets = TextDataset(
    tokenizer=tokenizer,
    file_path=dataset_path,
    block_size=200, # 适当调整块大小
)

# 准备数据收集器
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False # 在文本补全中不需要遮蔽语言模型的训练
)
```

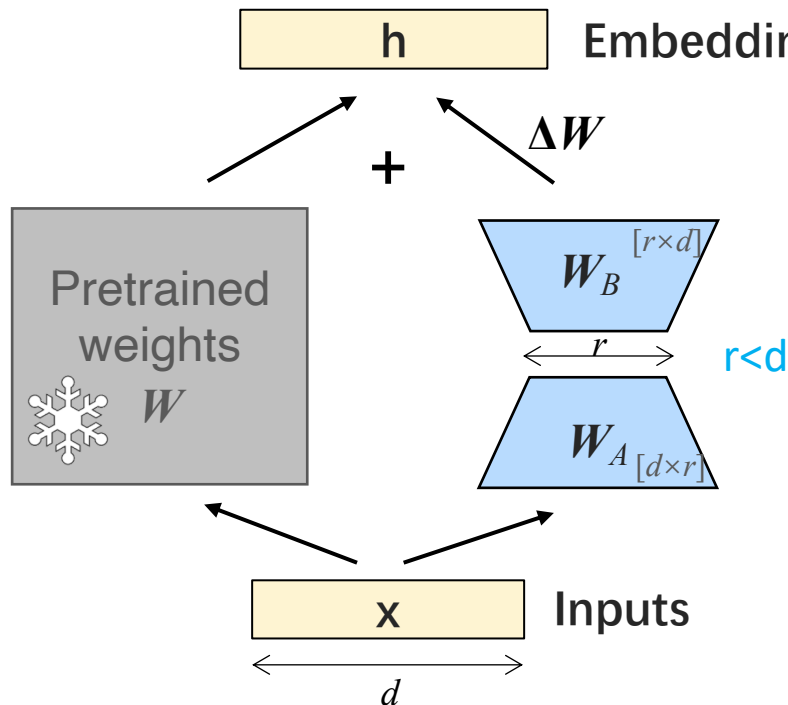
参数高效微调(PEFT)



LoRA

Low-Rank Adaptation of Large Language Models

- Represent weight updates using two smaller matrices through low-rank decomposition



TIME for Coding



Tutorial: AI Diagnosis By Fine-tuning LLMs

```
WORKSPACE [SSH: LABSERVER]  + 代码 + Markdown | 全部运行 重启 清除所有输出 | Jup

> 调制器数据整理
> bom_ocr
> codestyle
✓ diagnosisGPT
  > data
  > gpt2-finetuned
  > model_cache
  diagnoseLLM_API.ipynb
  diagnoseLLM_FT.ipynb
  diagnoseRAG.ipynb
  diagnoseRNN.ipynb
  diagnoseTransformer.ipynb
> dialogPrompt
> DSP_AI
> guxd.github.io
> job_rec
> nncircuit
> RepoQA
> Seq2Seq4TestcaseGen
> svm
> temp_field
> TemperatureFFA

diagnosisGPT > diagnoseLLM_FT.ipynb > M+ Version2: take special process

from transformers import GPT2LMHeadModel, AutoTokenizer, A
from transformers import TextDataset, DataCollatorForLang
from transformers import Trainer, TrainingArguments

# 指定中文GPT模型和标记器
model_name = "uer/gpt2-large-chinese-cluecorpussmall"
model = GPT2LMHeadModel.from_pretrained('./model_cache/gpt
tokenizer = AutoTokenizer.from_pretrained('./model_cache/g

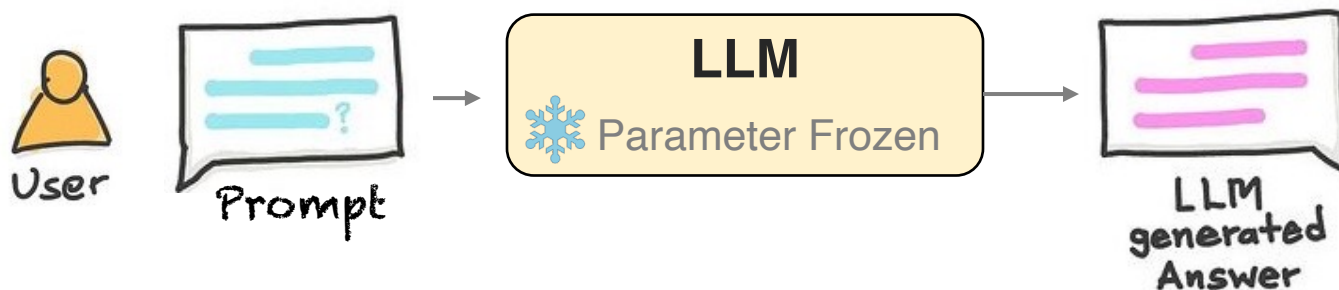
# 读取你的文本数据集
dataset_path = "./data/train.txt"

# 使用标记器编码数据集
tokenized_datasets = TextDataset(
    tokenizer=tokenizer,
    file_path=dataset_path,
    block_size=200, # 适当调整块大小
)

# 准备数据收集器
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False # 在文本补全中不需要遮蔽语言模型的训练
```

方式二：提示工程

- Prompt LLMs with task instructions, no parameter update.



Role

你是一名市场营销专家，具有丰富的品牌推广经验

Context

我们的公司即将发布一款新产品，具有创新的功能，可以显著提升用户体验。

Instruction

请你撰写一份产品发布材料，需要能够引起受众的兴趣，并清晰传达产品的核心卖点。

思维链提示

- **逐步**提示问题的思考过程引导最终的答案。

模型输入

Q: 一个花园里有5行玫瑰花，每行有4棵。又种了3行向日葵，每行有6棵。花园里一共有多少棵植物？

A: 让我们一步步思考
(Let's think step by step)。



LLM



模型输出

首先，计算玫瑰花的总数。有5行，每行4棵，所以 $5 * 4 = 20$ 棵玫瑰花。然后，计算向日葵的总数。有3行，每行6棵，所以 $3 * 6 = 18$ 棵向日葵。最后，把玫瑰花和向日葵加起来得到总数： $20 + 18 = 38$ 。因此，花园里一共有38棵植物。

ReAct (推理-操作)

Reasoning + Action:大模型交错输出推理(Reasoning)步骤和特定任务的动作(Action).

(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: `Search[Apple Remote]`
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: `Search[Front Row]`
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: `Search[Front Row (software)]`
Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: `Finish[keyboard function keys]`

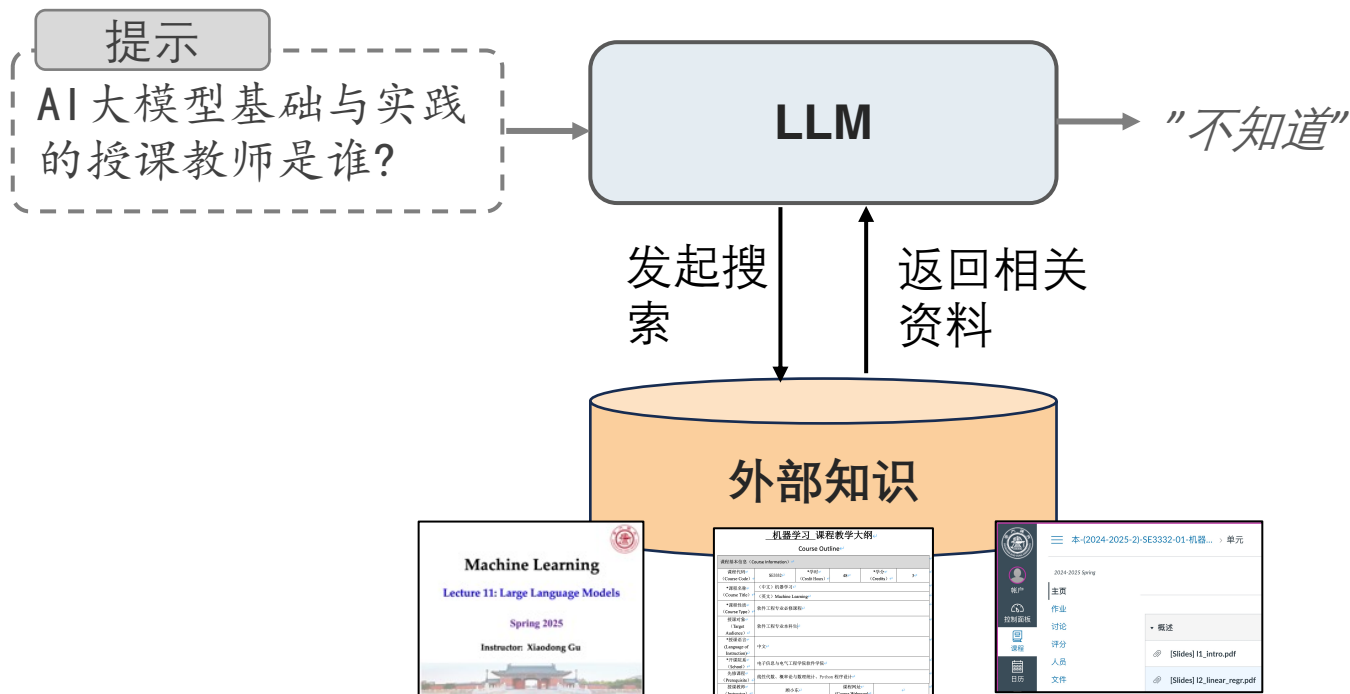
The ReAct framework allows LLMs to interact with external tools to retrieve additional information that leads to more reliable and factual responses.

<https://arxiv.org/abs/2210.03629>

检索增强生成 (RAG)

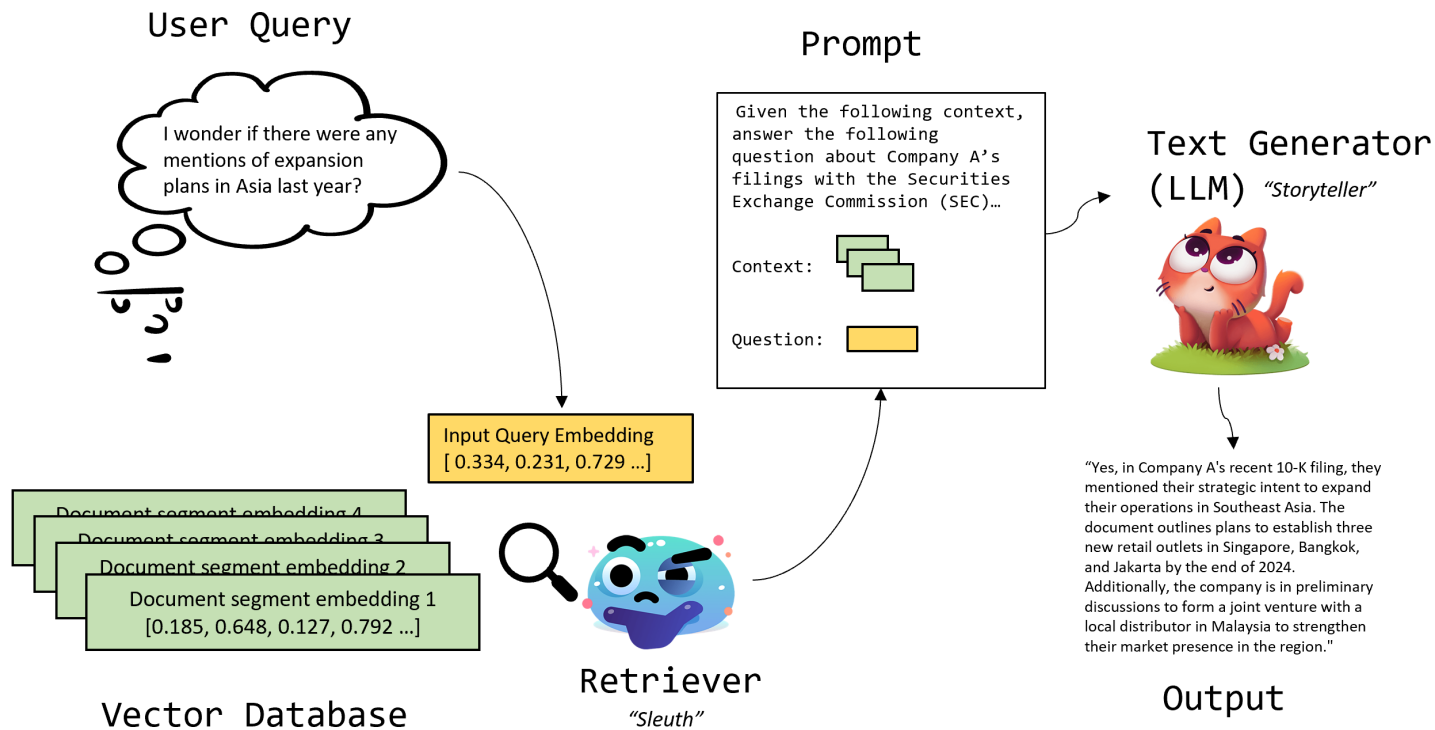
为什么要用RAG?

LLM可能缺乏最新的以及专用领域的知识。



RAG – 总体概览

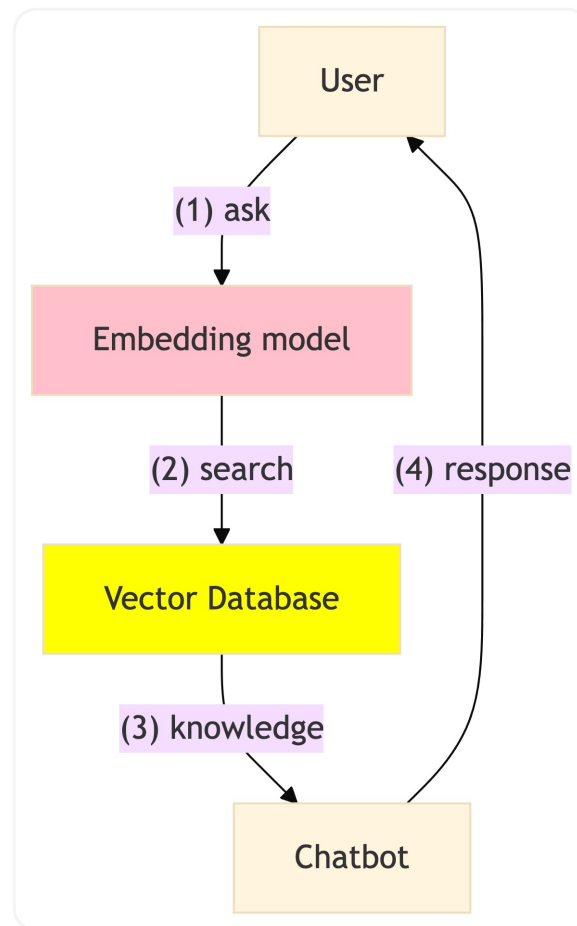
通过检索外部文档对大模型进行知识增强



RAG – 基本框架

RAG通常包括三大核心组件：

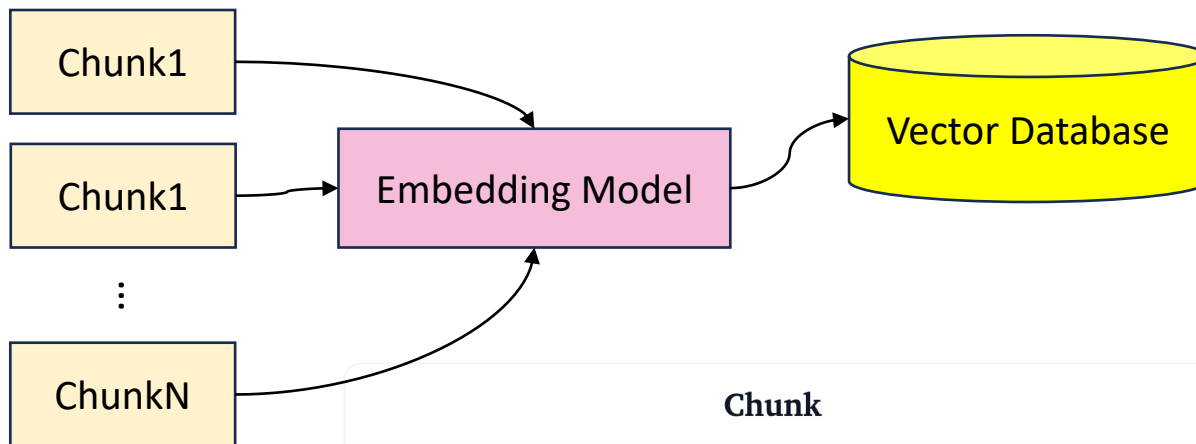
1. **向量嵌入模型**：将输入文档转换为向量，以进行快速检索；
2. **向量数据库**：将知识以对应向量形式存储起来并建立向量索引
3. **大模型**：根据检索的知识生成答案



<https://huggingface.co/blog/ngxson/make-your-own-rag>

RAG – 基本流程

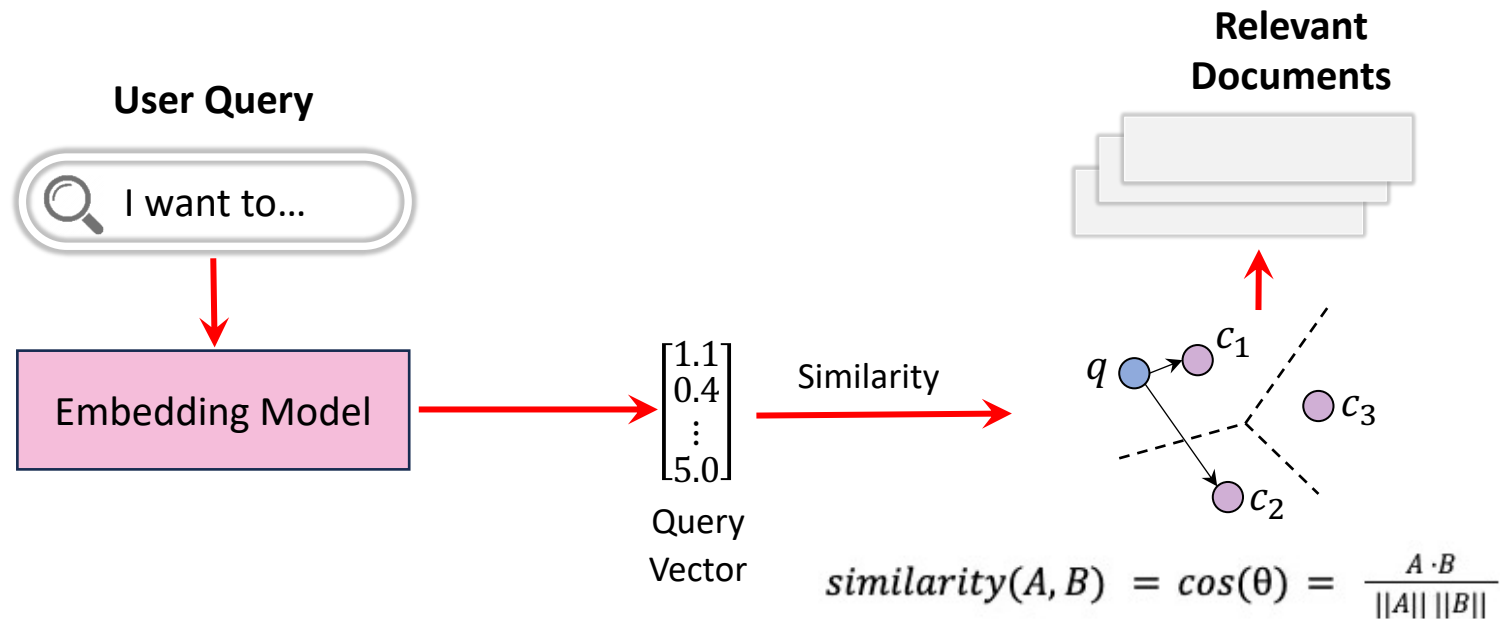
索引阶段: breaking documents into small **chunks** and embedding them into vectors that can be efficiently searched during generation.



Chunk	Embedding Vector
Italy and France produce over 40% of all wine in the world.	\[0.1, 0.04, -0.34, 0.21, ...]
The Taj Mahal in India is made entirely out of marble.	\[-0.12, 0.03, 0.9, -0.1, ...]
90% of the world's fresh water is in Antarctica.	\[-0.02, 0.6, -0.54, 0.03, ...]
...	...

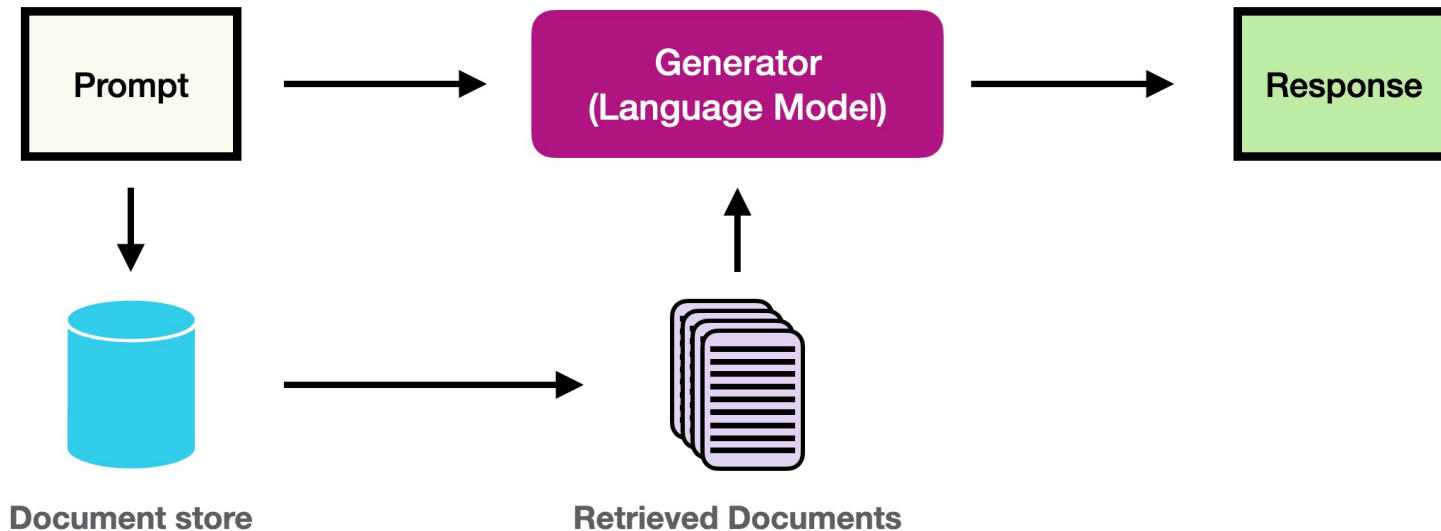
RAG – 基本流程

检索阶段: For a new query, calculate the query vector, and compare it against the vectors in the database to find the top-K relevant chunks.



RAG – 基本流程

生成阶段: Finally, the retrieved chunks will be augmented to the original prompt to the LLM to generate a response.



TIME for Coding



Tutorial: AI Diagnosis Using RAG

WORKSPACE [SSH: LABSERVER]

> 调制器数据整理

> bom_ocr

> codestyle

> diagnosisGPT

data

gpt2-finetuned

model_cache

diagnoseLLM_API.ipynb

diagnoseLLM_FT.ipynb

diagnoseRAG.ipynb

diagnoseRNN.ipynb

diagnoseTransformer.ipynb

> dialogPrompt

> DSP_AI

> guxd.github.io

> job_rec

> nncircuit

> RepoQA

> Seq2Seq4TestcaseGen

> svm

> temp_field

> TemperatureFFA

diagnosisGPT > diagnoseRAG.ipynb > Implement the retrieval function

+ 代码 + Markdown | ▶ 全部运行 ↺ 重启 ≡ 清除所有输出

Loading the dataset

Create a Python script and load the dataset into memory. The dataset is a list of strings.

Here is an example code to load the dataset:

```
dataset = []
with open('data/train.txt', 'r') as file:
    dataset = file.readlines()
print(f'Loaded {len(dataset)} entries')
```

[3] ✓ 0.0s

... Loaded 6882 entries

Implement the vector database

Now, let's implement the vector database.

We will use the embedding model from ollama to convert each c

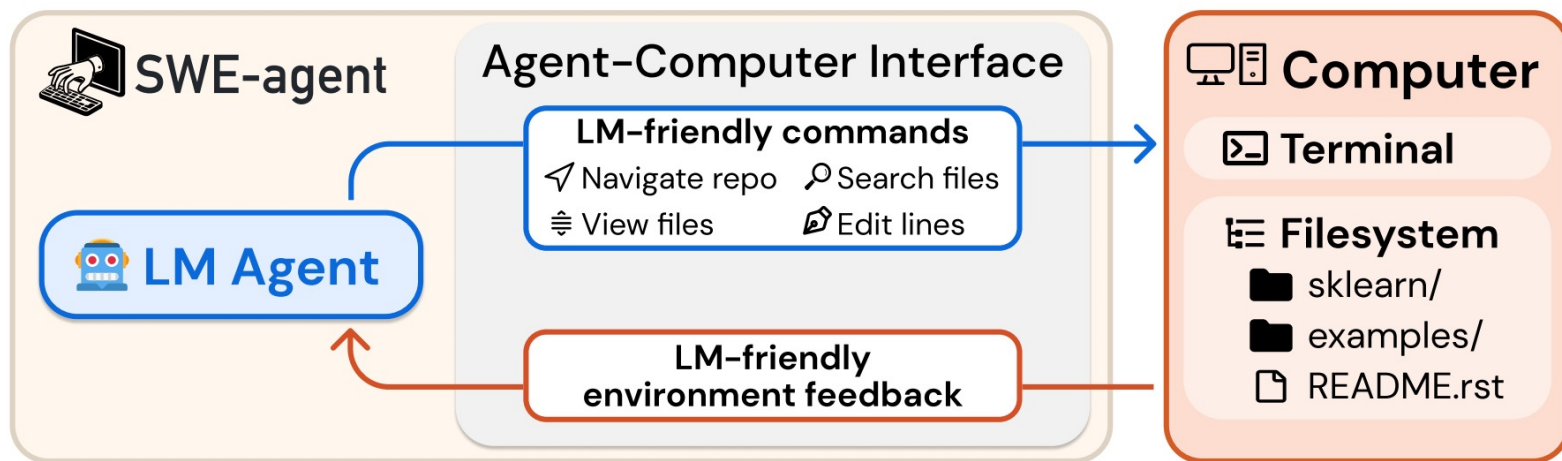
LLM Agents

<https://www.promptingguide.ai/research/llm-agents>

LLM Agents

以大模型为内核，具有自主思考、决策和执行任务能力的AI系统。

- **主动交互能力**: 不同于大模型的被动回答问题，智能体能够主动理解用户的高层目标并采取行动。
- **任务分解与规划**：将复杂问题分解为可执行的子任务，规划求解路径。
- **调用工具**: 灵活调用外部工具而不仅仅依赖底层大模型的思考能力。
- **反思与进化**: 评估自身行为效果并进行调整和进化



Agent核心组件

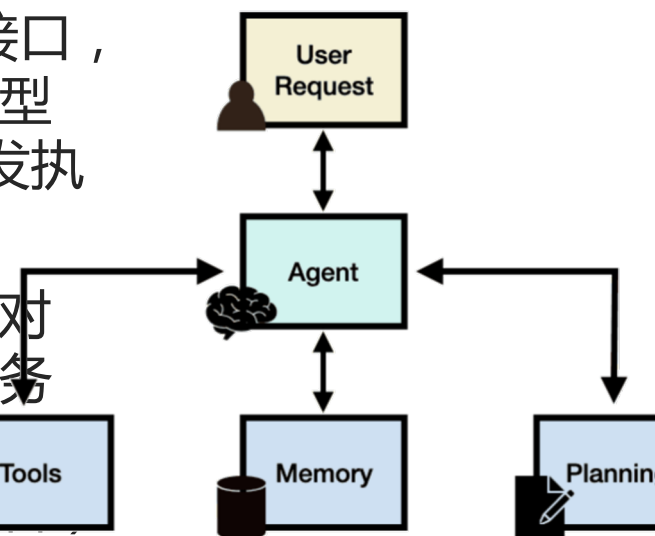
1.大模型：作为智能体的“大脑”，负责理解指令、进行推理、生成决策。

2.工具集 (Tools)：智能体与外部世界交互的接口，如检索工具、向量数据库、数学计算器等。大模型通过生成结构化指令（如函数调用、JSON）触发执行。

3.记忆 (Memory)：用于保存中间结果、历史对话或长期知识，包括短期记忆（如当前对话和任务执行上下文）以及长期记忆（如历史经验、用好等）。记忆内容通常表示为一个索引、 条目，用向量数据库实现。

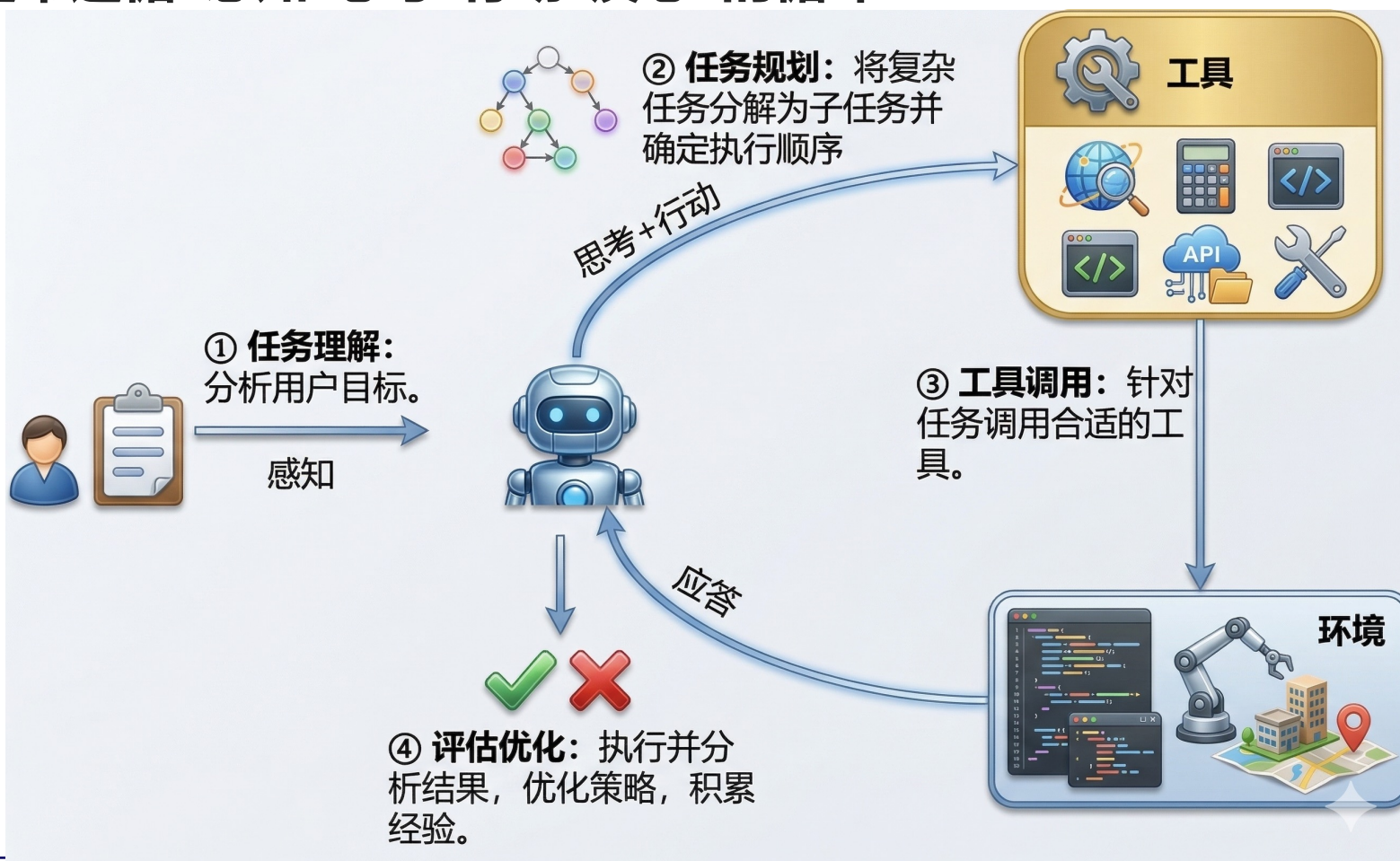
4.控制循环：智能体通常以“感知 → 决策 → 行动 → 反馈”的方式不断循环，直到任务完成或达到终止条件。

<https://aitutor.liduos.com/07-agents/07-1.html>



Agent工作流程

通常遵循"感知-思考-行动-反思"的循环



Agent工作流程

通常遵循"感知-思考-行动-反思"的循环

1. **任务理解 (Perception)**：用户给出一个高层次目标，而非明确步骤。大模型分析用户请求，识别任务类型和复杂度。
 2. **任务规划 (Planning)**：将复杂任务目标拆解为若干子任务，并确定执行顺序。例如："帮我分析公司上季度销售数据" → 分解为"获取数据"、"数据清洗"、"统计分析"、"生成报告"等子任务。
 3. **工具调用 (Action)**：针对当前子任务选择合适的工具，生成工具调用的具体参数，然后调用工具并获取执行结果，并将结果反馈给模型。例如：选择"数据查询工具"查询销售数据库。
 4. **评估优化 (Evaluation)**：分析工具执行结果是否满足要求。如果不满足，调整策略或选择其他工具重新执行。积累执行经验，优化未来决策。
 5. **循环与终止 (Iteration)**：根据新状态决定是否继续执行，直至任务完成。当任务完成时，整合所有子任务的结果，并生成符合用户需求的最终结果。
-

What's Next?

WHAT'S
NEXT?

Convolutional Neural Networks

The world of computer vision!



- A new type of neural networks that can understand images
- Computer vision tasks: classification, detection, segmentation, ..

