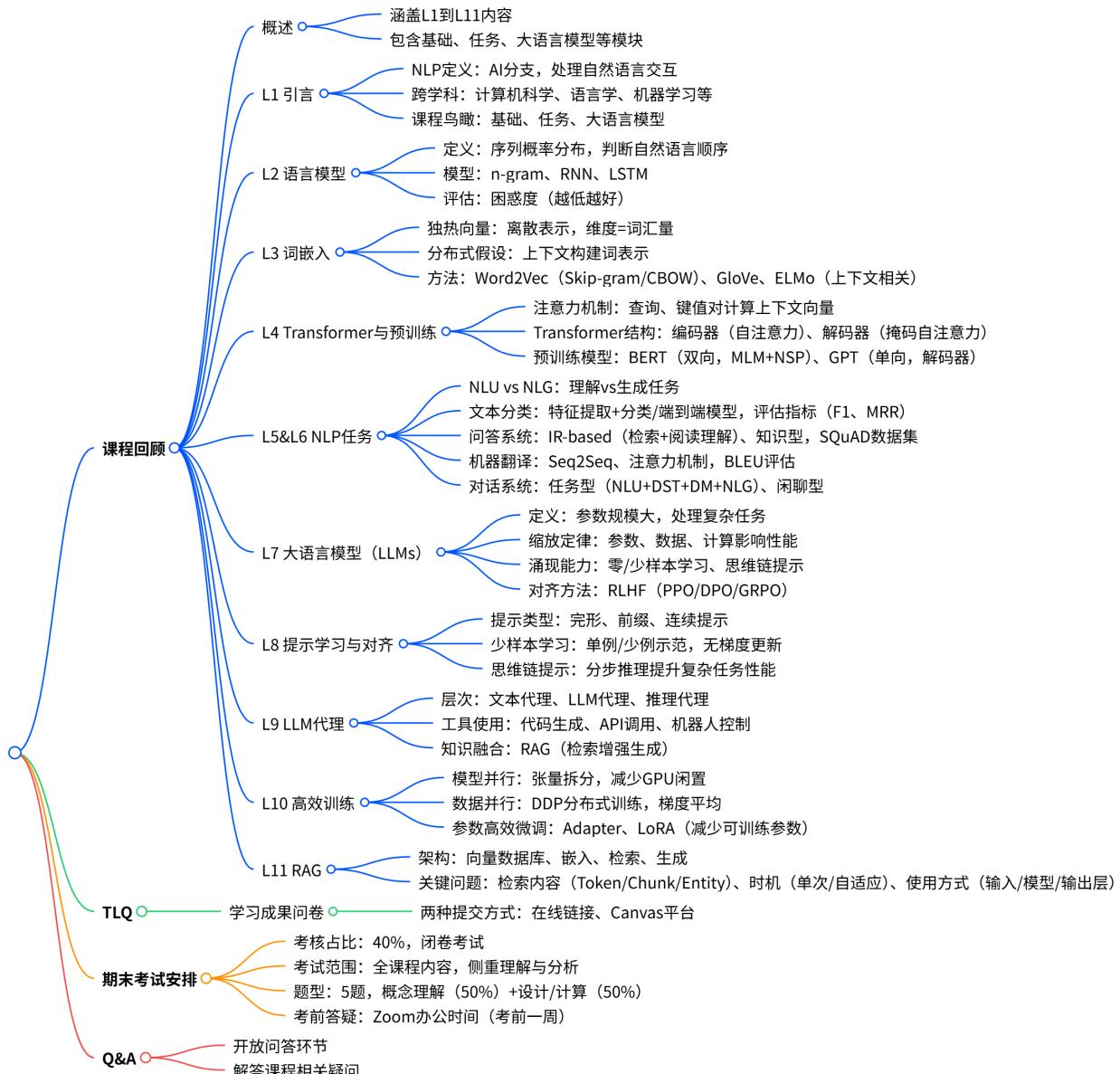


NLP-Review



豆包

你的 AI 助手，助力每日工作学习

Lecture 1: Introduction

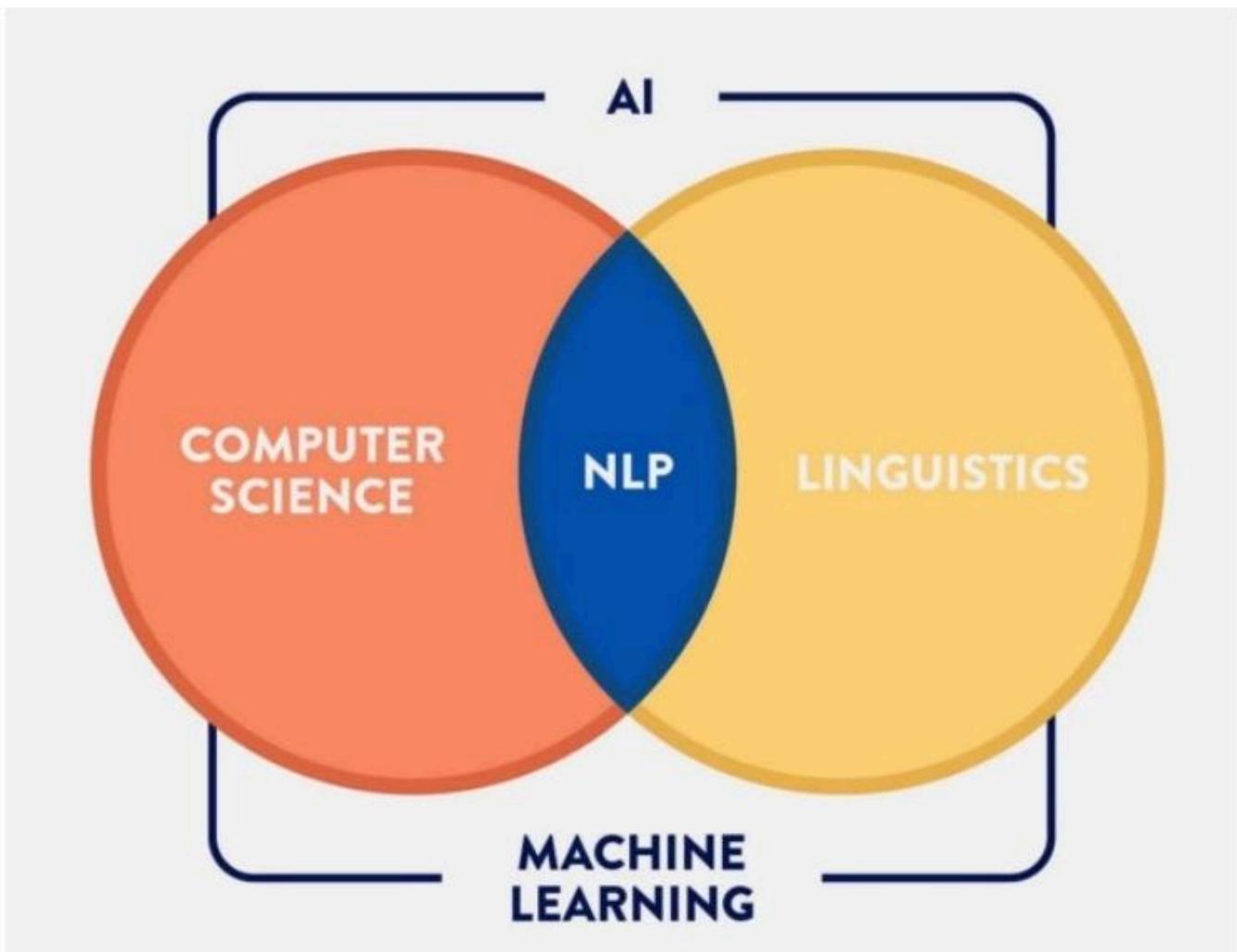


豆包

你的 AI 助手，助力每日工作学习

Introduction

NLP自然语言处理是人工智能的一大分支，核心目标：让计算机“理解”和“操作”自然语言



Course Content

该课程主要覆盖：

NLP基础：语言学 Linguistics, 语言模型 language models, 词嵌入 word embeddings

NLP任务：机器翻译 Machine translation, 问答 question answering, 对话 dialogue, 文本分类 text classification, 知识图谱 knowledge graph

大语言模型：Transformers, 预训练 pretraining (e.g., BERT, GPT), 提示词 prompting, 对齐技术 alignment, 高效微调 efficient finetuning, LLM代理 LLM agents

Challenges

NLP的挑战性：

- Ambiguity 歧义性：Similar strings mean different things, different strings mean the same thing
- Context 上下文关联性：同个词在不同位置有不同含义
- Commonsense Knowledge：需理解常识才可做准确判断

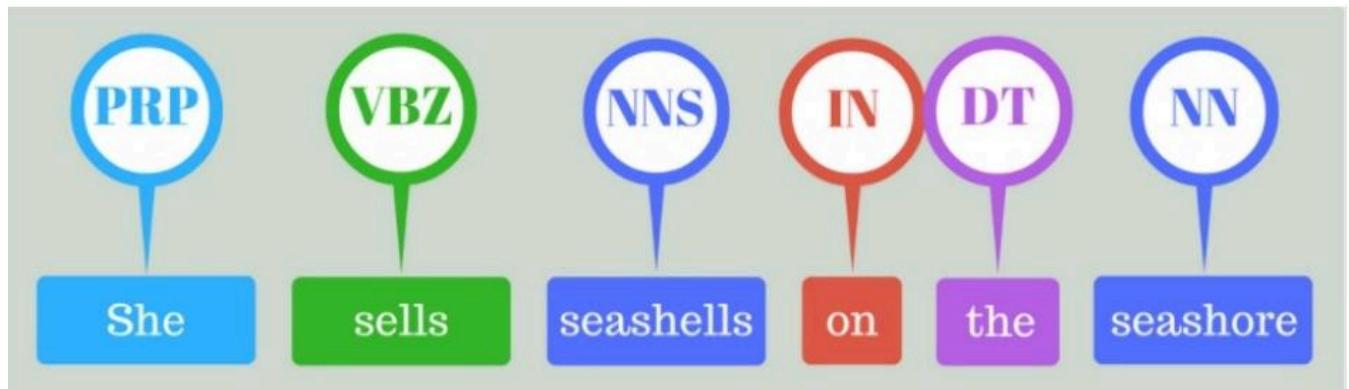
Processing of data

文本预处理 Text Preprocess

- **Tokenization**: 将文本拆分成最小单元例如词，字符等
 - **停用词移除 (Stop Words Removal)**: 过滤高频但无实际意义的词 (如 "the""is""的")
 - **词干提取 (Stemming)**: 将词简化为词根，不保证词根是有效词。例如"troubled"→"troubl"
 - **词形还原 (Lemmatization)**: 基于词典将词还原为词典中的标准形式，保证有效。"troubles"→"trouble"

	original_word	stemmed_word		original_word	lemmatized_word
0	trouble	troubl	0	trouble	trouble
1	troubled	troubl	1	troubling	trouble
2	troubles	troubl	2	troubled	trouble
3	troublesome	troublesom	3	troubles	trouble

- **词性标注 (Part-of-Speech Tagging, POS Tagging)**: 每个词标注语法类别 (名词、动词、形容词等)



- **N-grams 生成**: 保留词序信息, 捕捉相邻词的语义关联 (如 “ice cream” 是一个二元组 bigram), N-grams的数量满足:

$$N_{grams} = X - (N - 1)$$

其中X是句子中总词汇数，N是元数

文本向量化 Vectorization: 将字符串转换为计算机可处理的数值向量

1. 词袋模型 (Bag of Words, BOW): 忽略词序, 统计每个词在文本中的出现次数

Review 1: This movie is very scary and long
Review 2: This movie is not scary and is slow
Review 3: This movie is spooky and good

列表统计：

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

根据以上表格可知三个Reviews分别对应向量为

Vector of Review 1: [1 1 1 1 1 1 1 0 0 0 0]

Vector of Review 2: [1 1 2 0 0 1 1 0 1 0 0]

Vector of Review 3: [1 1 1 0 0 0 1 0 0 1 1]

该方法简单但：

- 向量过长（等同词典长度）
- 高稀疏性（大量 0 值）
- 丢失词序和语法信息

2. **TF-IDF (Term Frequency-Inverse Document Frequency)**: 评估词在文档中的重要性，结合词频 (TF) 和逆文档频率 (IDF)

- TF: 词在当前文档中的出现频率（如“恐怖”在某评论中出现 2 次，TF=2 / 文档词数）

$$t_{f_{t,d}} = \frac{n_{t,d}}{\text{该文档总词数}}$$

Term	Review 1	Review 2	Review 3	TF (Review 1)	TF (Review 2)	TF (Review 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

- IDF: 词在整个语料库中的稀有性 (如“的”在所有文档中频繁出现, IDF 接近 0)

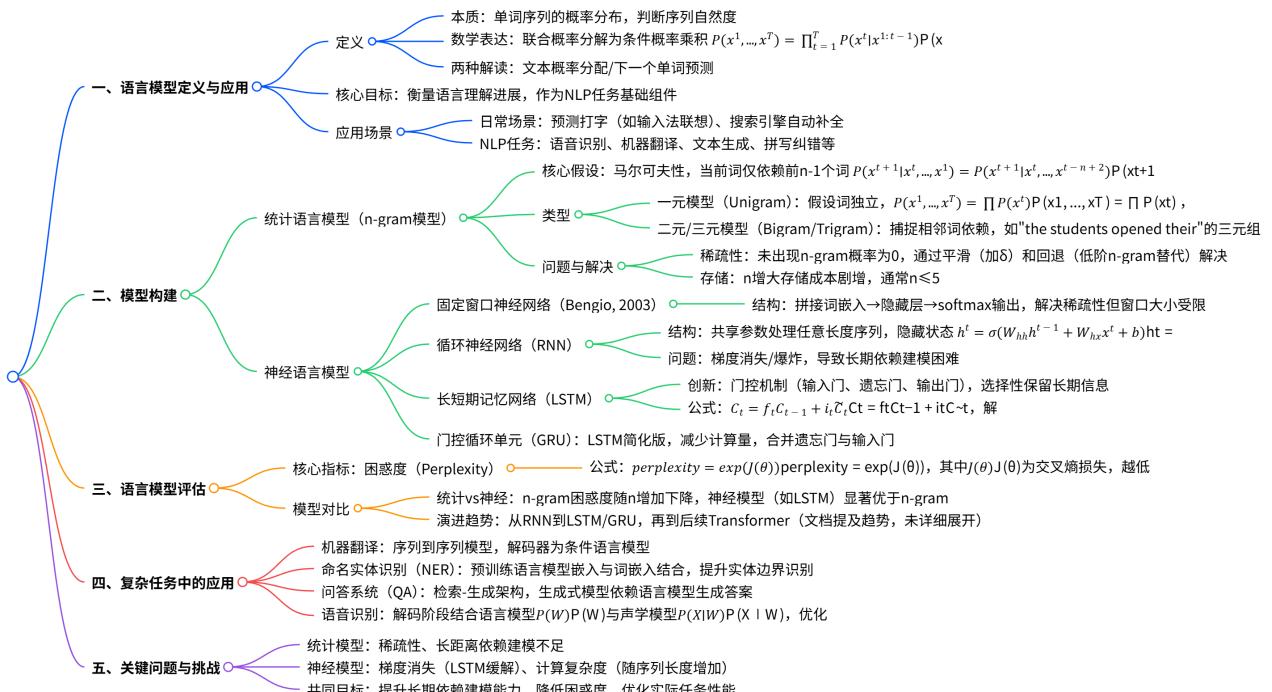
$$id_{f_t} = \log \frac{\text{文档总数}}{\text{包含 } t \text{ 的文档数}}$$

最终计算tf-idf值

$$(tf_idf)_{t,d} = tf_{t,d} * idf_t$$

用以突出在当前文档频繁但在整体语料稀有词 (如专业术语)

Lecture 2: Language Model



豆包
你的 AI 助手, 助力每日工作学习

What is a language model?

本质是对单词序列的概率分布 probability distribution over sequences of words

It can assign a probability $P(x^{(1)}, x^{(2)}, \dots, x^{(T)})$ to a whole sequence. 对整个序列分配一个概率

也可以说语言模型的任务是对一串文本 $x^{(1)}, x^{(2)}, \dots, x^{(T)}$ 分配一个概率, 从而量化“一句话是否合理”

并能根据概率判断下一个出现的词

n-gram model

基于马尔可夫假设 (Markov assumption)，下一个词 $x^{(t+1)}$ 仅依赖于前面的 ($n - 1$) 个词：

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)}) = P(x^{(t+1)} | x^{(t)}, \dots, x^{(t-n+2)})$$

这意味着在预测下一个词时，无需考虑所有历史词，只需关注最近的 ($n - 1$) 个词，简化了模型对语言序列依赖关系的建模

对于模型本身，词汇表 (Σ) 中的每个词，其概率由固定长度 ($(n - 1)$) 的历史词决定。例如，二元模型 ($(n = 2)$) 中，每个词的概率仅依赖于前一个词

n - gram 是 n 个连续词组成的片段 n-gram is a chunk of n consecutive words.

$$P(w|\text{students opened their}) = \frac{\text{count(students opened their } w\text{)}}{\text{count(students opened their)}}$$

the students opened their _____

unigrams: “the”, “students”, “opened”, “their”

bigrams: “the students”, “students opened”, “opened their”

trigrams: “the students opened”, “students opened their”

4-grams: “the students opened their”

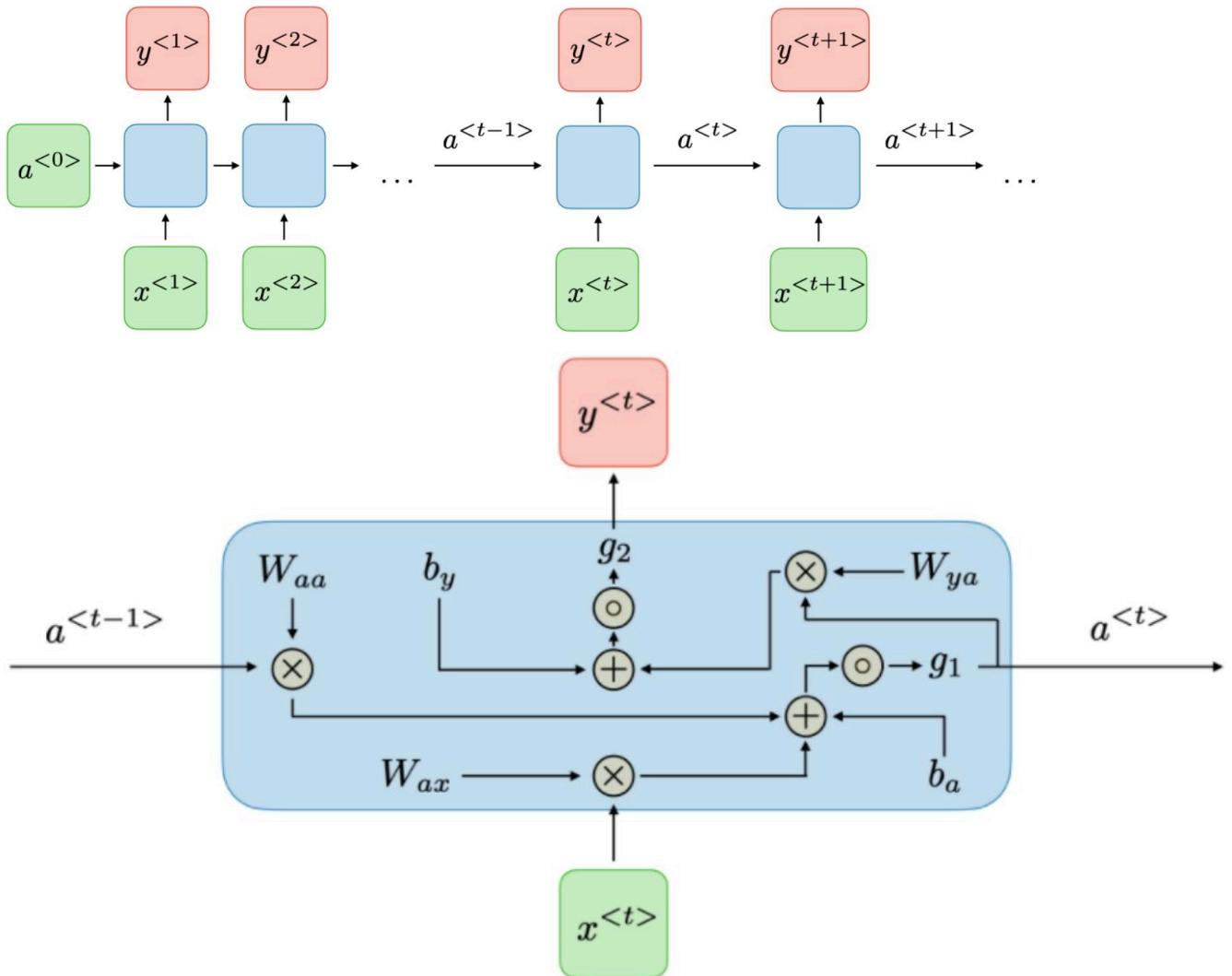
n-grams的问题：

- Sparsity 稀疏：
 - 对于从未出现的序列（分子部分），其概率会被误认为0。解决：添加一个很小值来平滑smoothing处理
 - 对于从未出现的前序列（分母部分），公式无法计算。解决：回退backoff到n-1-1来替代，例如students opened their替换成opened their
- Storage 存储：对于大的n值，存储成本大大增加。解决：通常使用不大于5的n

n-gram模型是统计模型statistic model的代表

RNN-based language model

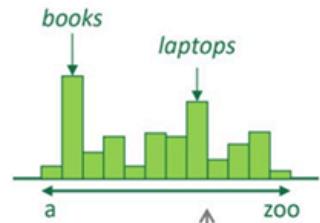
循环神经网络（RNN）中各个单元共享结构structure和参数parameters，输入 / 输出长度灵活，具备自连接 self connection 或与前层单元的连接 connect to previous layers，具有短期记忆能力 short term memory



基于RNN的语言模型过程如下：

1. 输入语言序列
2. 词嵌入将离散单词转换成连续低维向量
3. 隐藏状态 Hidden state 更新来传递历史信息
4. 生成下一个词的概率

$$\hat{y}^{(4)} = P(\mathbf{x}^{(5)} | \text{the students opened their})$$



output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}h^{(t)} + b_2) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma(\mathbf{W}_h h^{(t-1)} + \mathbf{W}_e e^{(t)} + b_1)$$

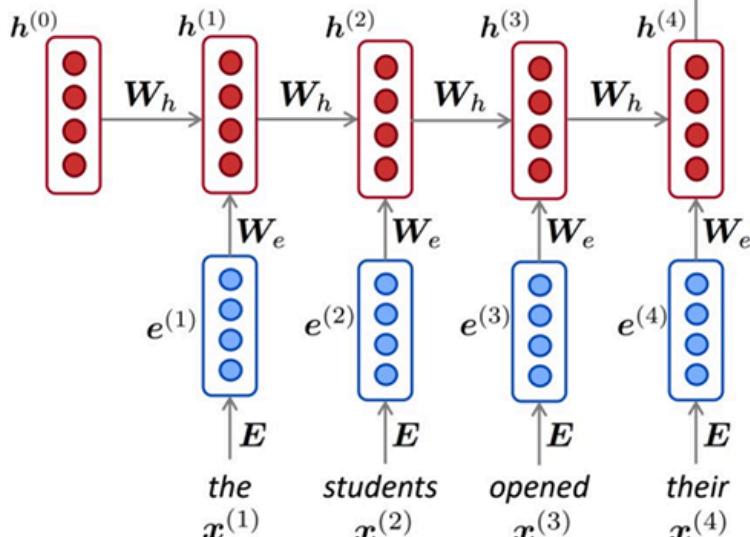
$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = \mathbf{E}x^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



Note: this input sequence could be much longer, but this slide doesn't have space!

优点：

- 处理任意长度输入
- 可利用多步前信息 information from steps back
- 模型大小不随输入增长
- 权重相同所以输入处理对称symmetry

缺点：更新权重时可能出现梯度消失 vanishing / 爆炸 exploding问题（因为反向传播时涉及指数运算 exponential operation when backpropagation）

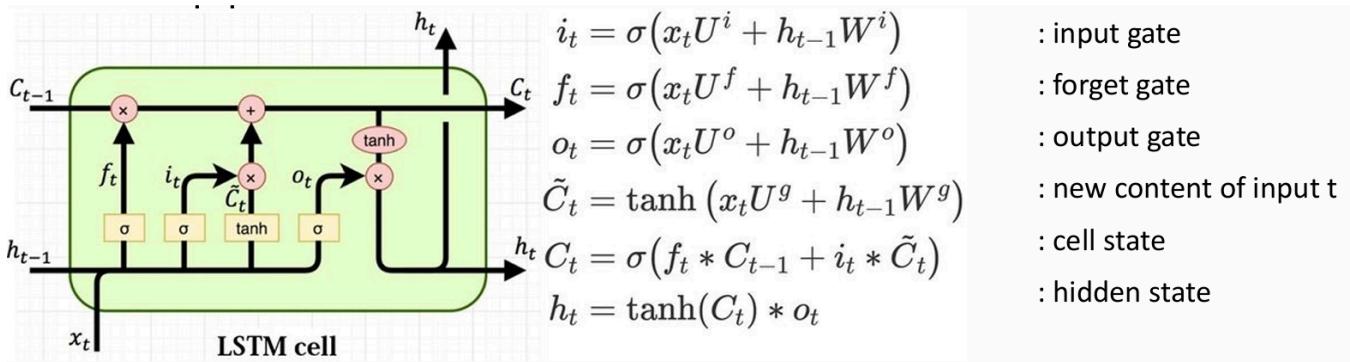
LSTM-based LM

在自然语言处理中，梯度消失/爆炸问题可能阻碍长距离的依赖关系学习（例如该tickets距离过长）

LM task: When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____.

LSTM没有像RNN那样简单累加历史信息，而是采用门控机制 **gating mechanism** 来更新细胞状态**cell state**，有效缓解了传统 RNN 中梯度消失的问题

包含丢弃信息，存储信息，更新信息等过程



由于门gates的存在，LSTM能更好控制信息流动，使梯度能更稳定地传播，避免因梯度消失而丢失长期信息

RNN-based LM和LSTM-based LM都是神经语言模型 **neural language model** 的代表

Evaluating language models

通过困惑度 (perplexity) 评估语言模型的预测能力。数值越低，模型的预测能力越强

$$\text{perplexity} = \prod_{t=1}^T \left(\underbrace{\frac{1}{P_{\text{LM}}(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})}}_{\text{Inverse probability of corpus, according to Language Model}} \right)^{1/T}$$

Normalized by
number of words

可以理解为模型对文本“平均不确定性”的量化。低困惑度表示模型对文本的预测更加准确（不确定性低），高困惑度则表示模型预测能力较差（不确定性高）

Lecture 3: Word Embedding



豆包

你的AI助手，助力每日工作学习

如何将词语交由语言模型处理？

传统的词典**WordNet**是通过同义词和上位词构建的词汇数据库，但：

- **Subjective 主观性**
- **Cannot compute accurate word similarity 无法精确计算词语相似度**

WordNet Search - 3.1

- [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss)

Verb

- S: (v) [mix](#), [mingle](#), [commix](#), [unify](#), **amalgamate** (to bring or combine together or with something else) Example url:

Adjective

<http://wordnetweb.princeton.edu/perl/webwn>

- S: (adj) **amalgamate**, [amalgamated](#), [coalesced](#), [consolidated](#), [fused](#) (joined together into a whole)

One-hot vector

将词语表示成离散符号 discrete symbol, 向量中只有一个1, 其余为0.例如

```
hotel = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
motel = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

一方面信息非常稀疏 sparse , 二方面无法表示相似性 similarity (hotel和motel的相似度不应为0)

如何更恰当表示相似性? **分布式假设 (Distributional hypothesis)**

Distributional hypothesis

核心思想:

- 当一个词 w 出现在文本中时, 其上下文 (context) 是在固定大小窗口内邻近出现的一组词
- 利用词 w 的多个不同上下文来构建该词的表示, 即词的意义由其所处的上下文决定

例如:

“...government debt problems turning into banking crises as happened in 2009...” 中

“banking”与“government debt problems (政府债务问题)”“crises (危机)”等词共现, 体现“银行业危机”的语境

word2vec基于分布式假设的思想

Word2vec

Word-embedding 词嵌入（又称（神经）词表示 ((neural) word representation) 的目标是：

- 为每个词构建一个**密集向量 (dense vector)**，区别于像 One - hot 这样的稀疏表示，密集向量维度固定且元素多为非零值
- 词的向量应与出现在相似上下文中的词的向量相似，即上下文相似的词在语义和向量空间上均相近
- 是一种分布式表示 (distributed representation)，每个维度共同参与语义表示，而非局部表示（如 One - hot 中单一维度代表语义）

$$\text{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word2vec就是一种实现word-embedding的框架，用于根据词义构建向量

输入：大规模文本语料（如大量句子或文档）

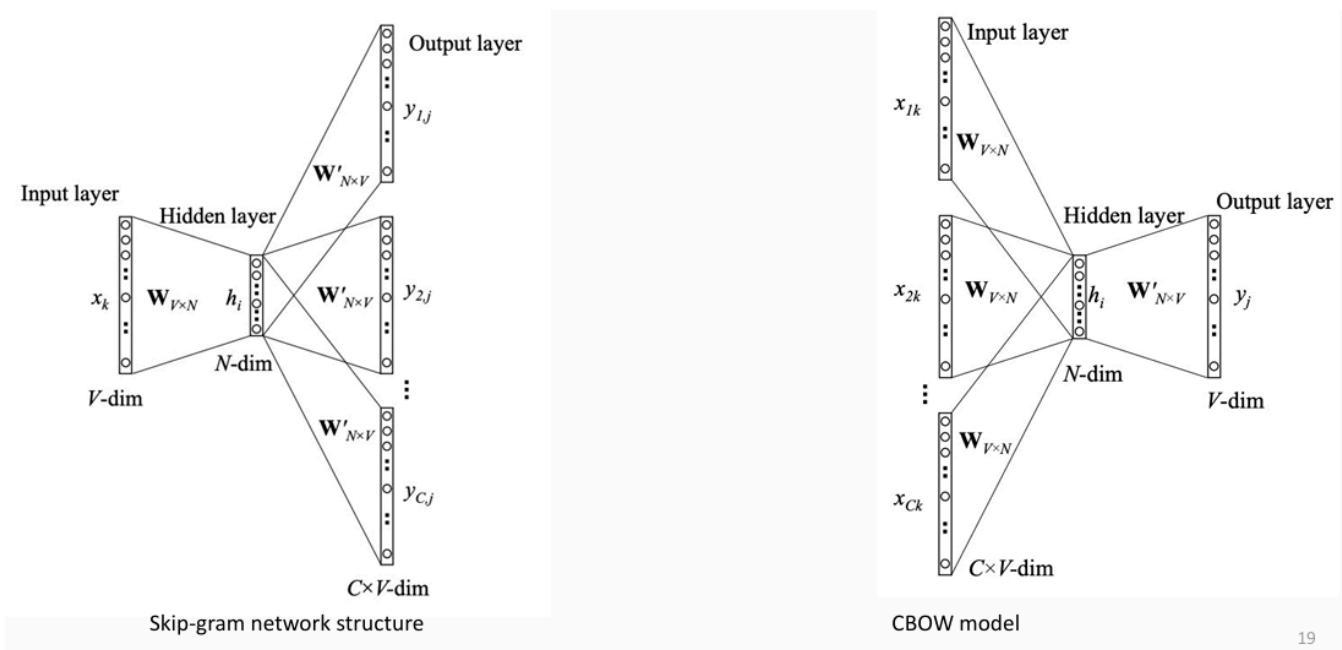
处理：遍历文本中每个位置 t ，该位置有目标词（中心词） w_t 和若干上下文词 w_c

输出：每个词的向量表示

两种类型：

- **Skip - gram**：给定目标词 target word w_t ，计算上下文词 context word w_c 的概率
- **连续词袋模型 (CBOW)**：给定上下文词 w_c ，计算目标词 w_t 的概率

优化目标：不断调整词向量，以最大化上述概率，从而学习到有效的词向量表示



19

Skip-gram's optimization problem

Skip-gram的优化目标是最大化数据似然性 (Data Likelihood) $L(\theta)$

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ is all variables to be optimized

但该公式涉及连乘难以计算，因此使用平均负对数似然损失 average negative log likelihood 作为目标函数（损失函数） $J(\theta)$

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

目标是最小化 $J(\theta)$ ，使得预测准确率predictive accuracy 最高

Improve the efficiency in training

由于词汇表 V 的规模非常大，计算耗时很高，因此网络的训练时长极大

考虑两种优化方法：

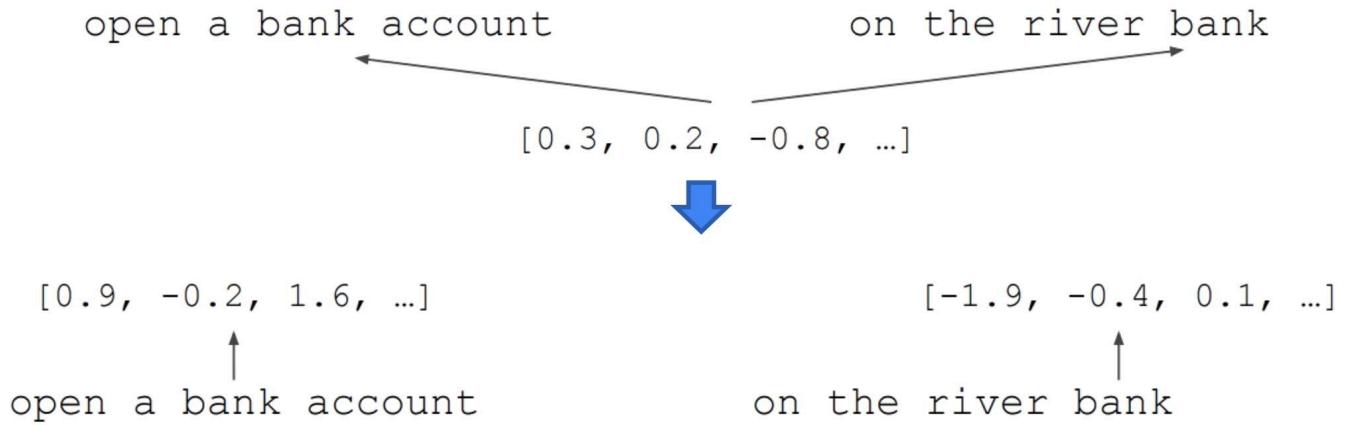
- **负采样 (Negative sampling)**: 通过采样少量负例来近似计算，减少计算量。该方法对高频词（常见词）和低维向量效果更优（就像“挑几个错误答案练练手”，快速高效，但可能漏掉一些细节（低频词））
- **层次化 Softmax (Hierarchical Softmax)**: 构建树形结构，将多分类问题转化为一系列二分类问题，降低计算复杂度。对低频词（不常见的词）效果更好（像“走迷宫找目标词”，通过一步步二分类决策找到答案，适合大规模词表）

Contextualized embeddings

但是，word2vec属于(non - contextual) embeddings 非上下文嵌入，这种处理方式在处理**多义词 Polysemous words** 时效果不佳，例如：

- “He sticks a stamp on the envelope.” (“stick” 意为“粘贴”）。
 - “The old lady leant on her stick as she talked.” (“stick” 意为“拐杖”）。
- 非上下文嵌入用固定向量表示“stick”，无法区分其在不同语境下的语义差异

ELMo实现了上下文相关嵌入的进步

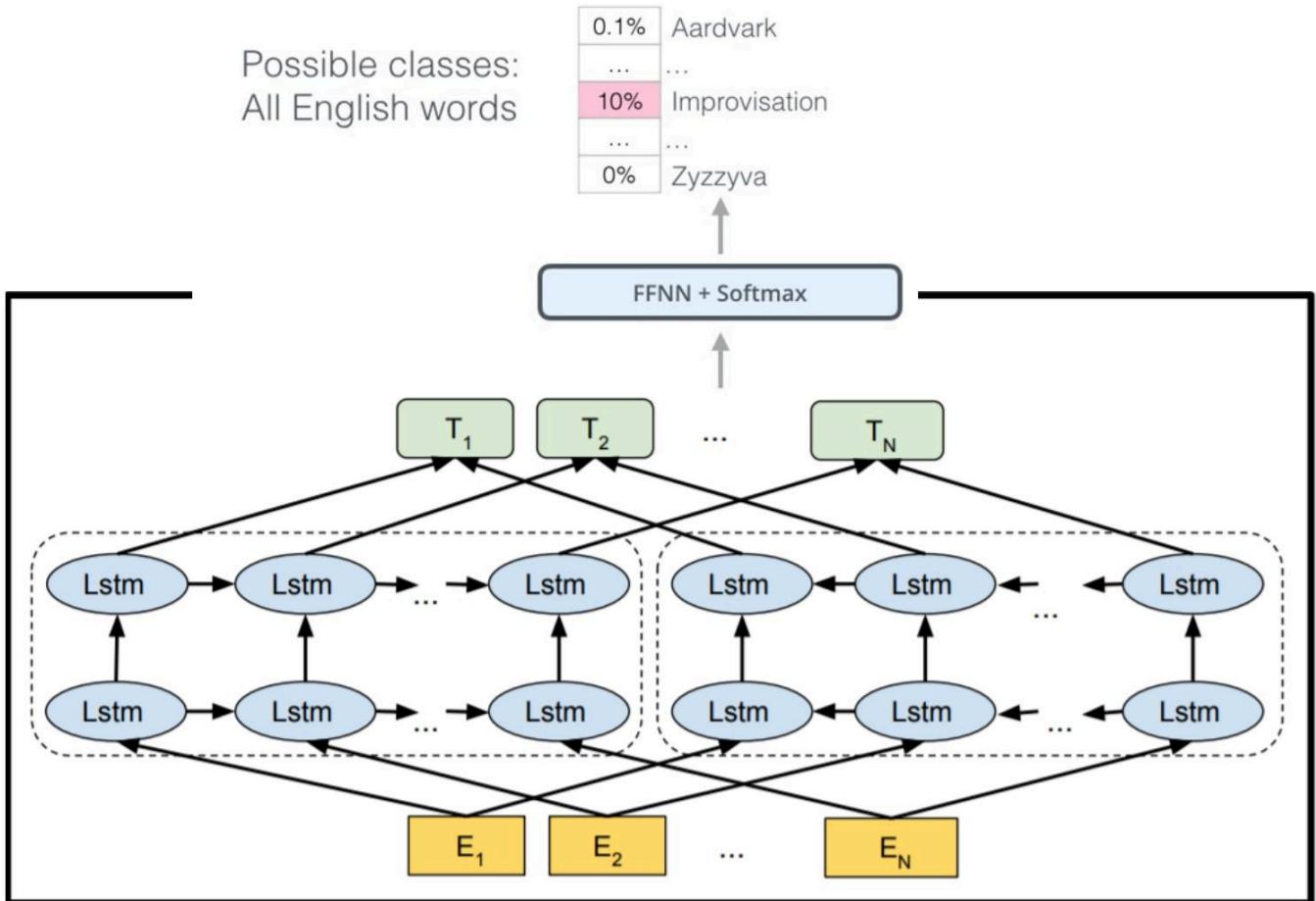


如果只询问单独的词，ELMo可能给出多个嵌入答案

如果询问独立的句子，ELMo根据上下文返回特定的嵌入

ELMo

ELMo 使用双向 LSTM 预训练语言模型 (ELMo uses a **bi - directional LSTM** to pre - train the language model)



训练过程中，ELMo：

1. 通过多层双向LSTM语言模型来分别捕捉上文和下文信息
2. 通过前馈神经网络（FFNN）和 Softmax 函数，预测下一个词

Lecture 4: Transformers and pretraining-finetuning



豆包
你的AI助手，助力每日工作学习

RNNs/LSTMs 存在长期依赖问题 (long - term dependency problems)，处理长序列有信息丢失，顺序处理导致效率不高

双向 RNNs/LSTMs相比起Transformer, 特征融合 (feature fusion) 和表示能力 (representation ability) 较弱

注意力机制能有效解决以上问题

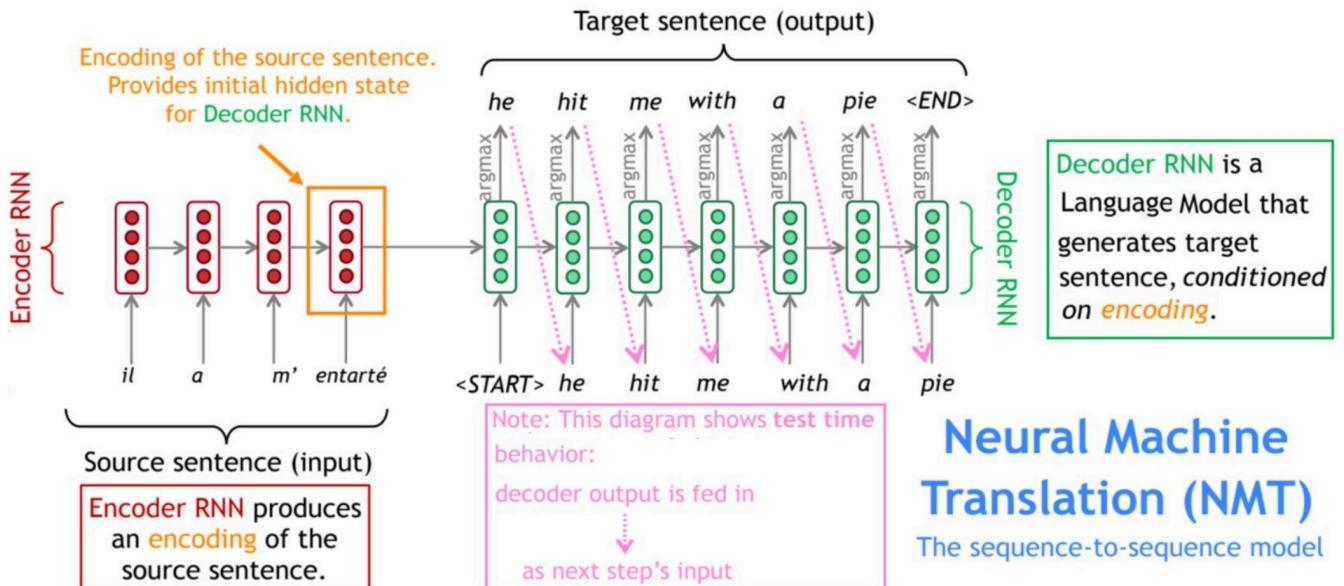
Attention

注意力机制类似于人类大脑，会将更多关注放在更重要的信息上

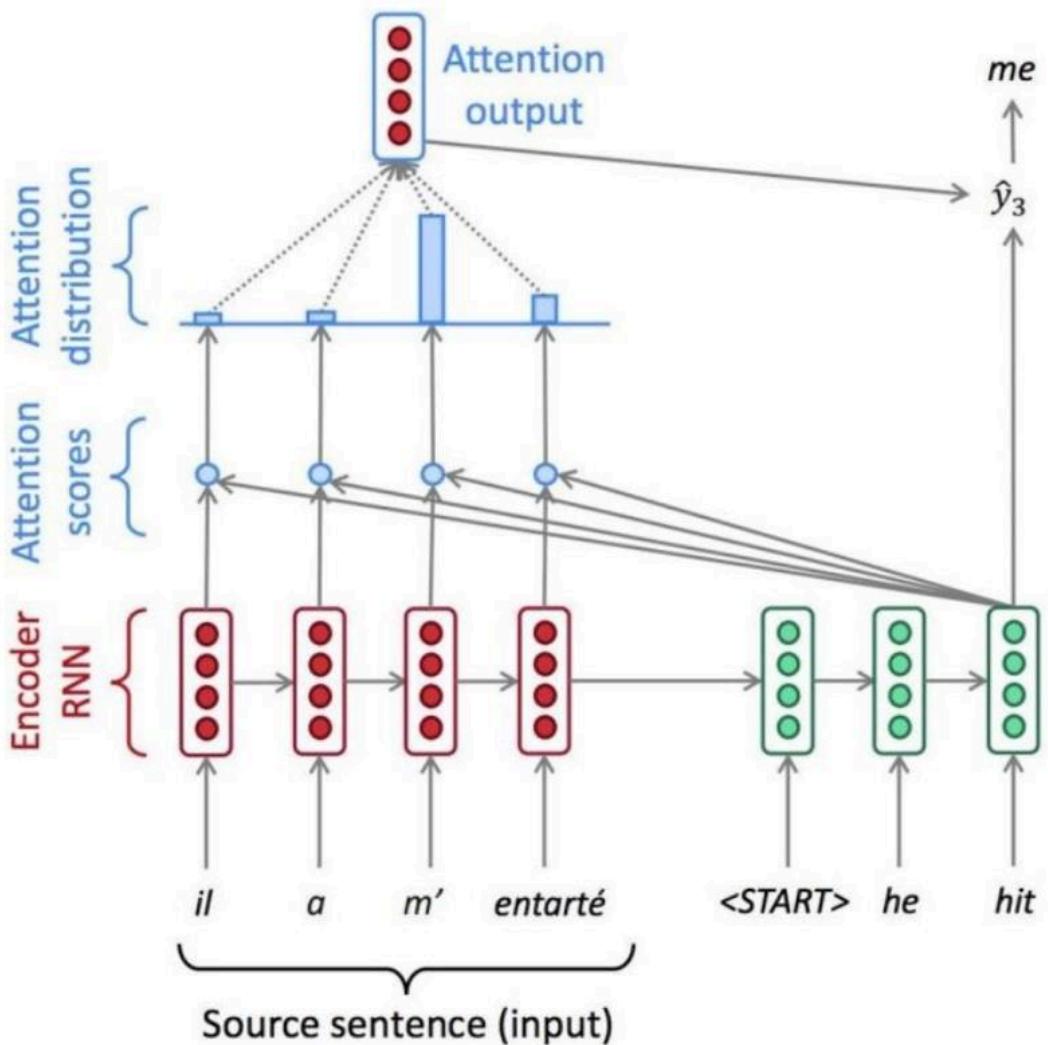
其将一个查询 (query) 和一组键值对 (key - value pairs) 映射到一个输出，其中查询、键、值和输出均为向量

输出是值 (values) 的加权和，每个值的权重通过查询与对应键 (key) 的兼容函数计算得出，即根据查询与键的关联程度确定值的重要性权重，也就是在一堆信息中，先找出与当前任务最相关的部分，再重点处理这些部分

下图为基于RNN的语言模型，其顺序处理信息，可能导致信息丢失



相比之下，注意力机制允许模型动态在全局中寻找最相关的信息



具体地，注意力机制包含以下过程：

1. 计算相似度：计算查询 (q) 与每个键 (k_i) 之间的相似度

$$s_i = \text{similarity}(q, k_i).$$

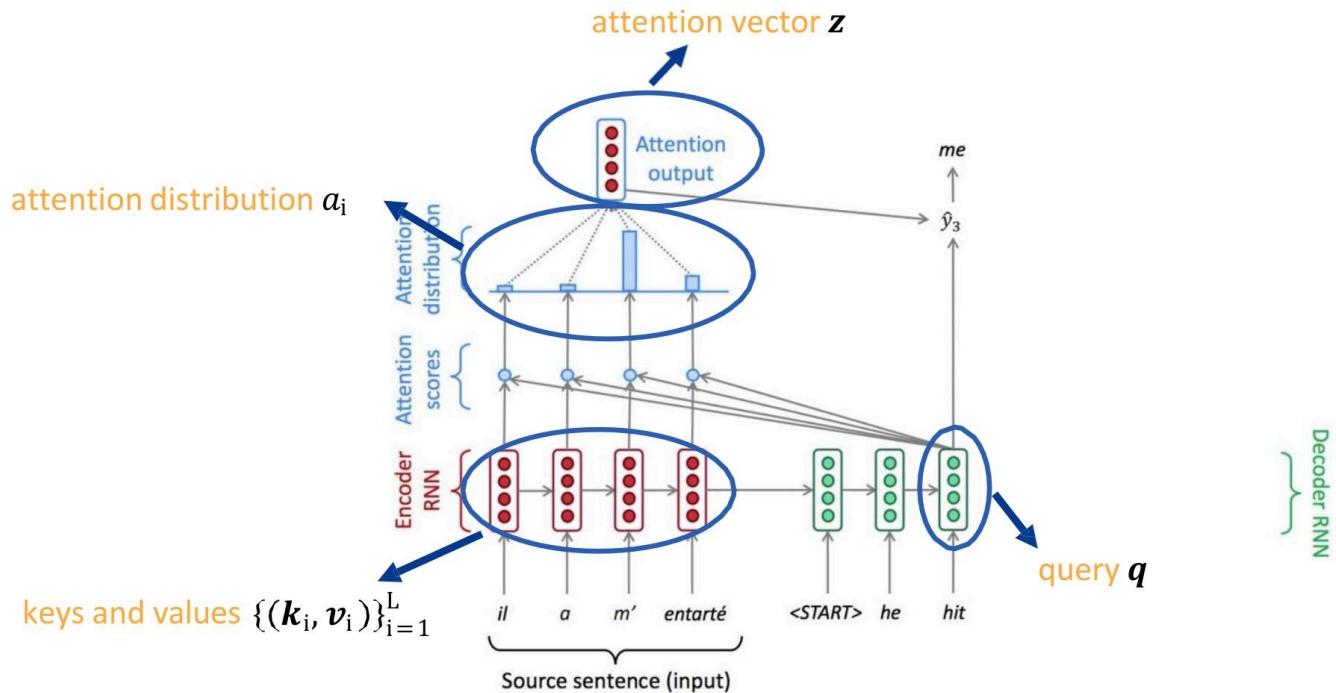
2. 注意力分布 (归一化 Normalize)：将相似度分数归一化到 $([0, 1])$ 区间且总和为 1，形成注意力分布。 (a_i) 表示每个值 (v_i) 在加权求和时的权重

$$a_i = \text{softmax}(s_i) = \frac{\exp(s_i)}{\sum_{j=1}^L \exp(s_j)}.$$

3. 注意力 / 上下文向量计算：得到最终值(z)，融合了各个值的信息

$$\mathbf{z} = \sum_{i=1}^L a_i \mathbf{v}_i.$$

键 (\mathbf{k}_i) 和值 (\mathbf{v}_i) 不一定相同，例如在机器翻译中，它们可以是不同的向量，体现了注意力机制的灵活性

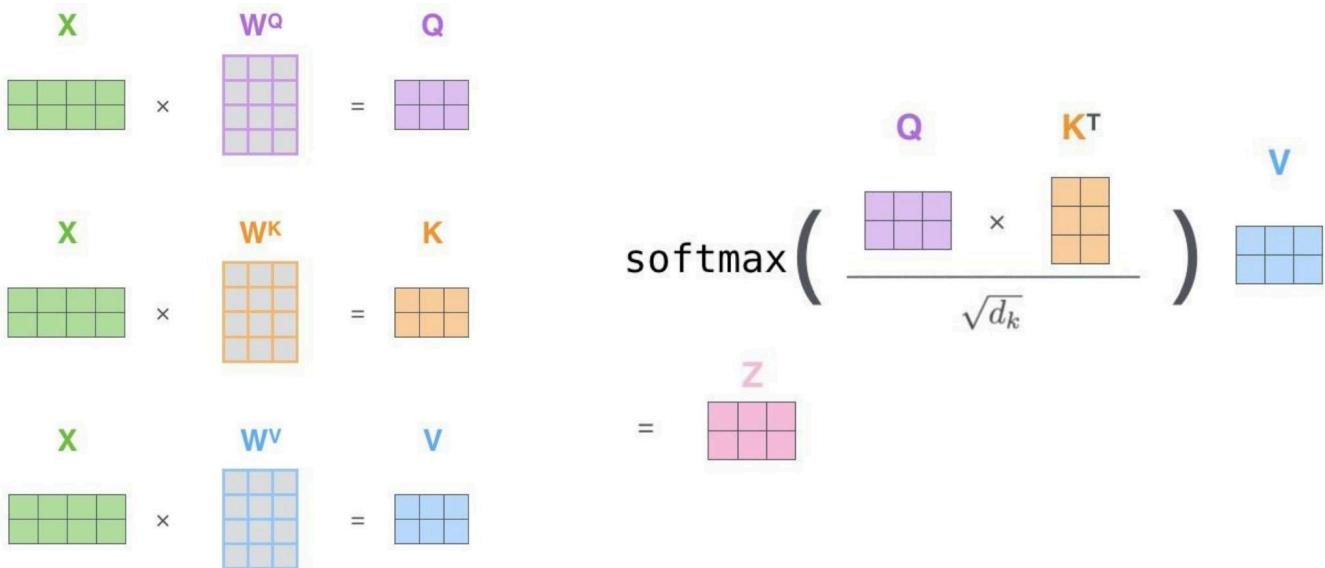


自注意力 (self-attention)：对于输入单词序列，在每个单词与其他所有单词（包括自身）间应用注意力机制，从而理解上下文关系

例如：“小明在公园里看到了一只狗，它正在追自己的尾巴。”

应用自注意力机制后，有：

- 看到“它”时，“它”和“狗”关系很紧密 → 重点看“狗”
- 看到“公园”时，“公园”和“追尾巴”关系较远 → 稍微忽略“追尾巴”



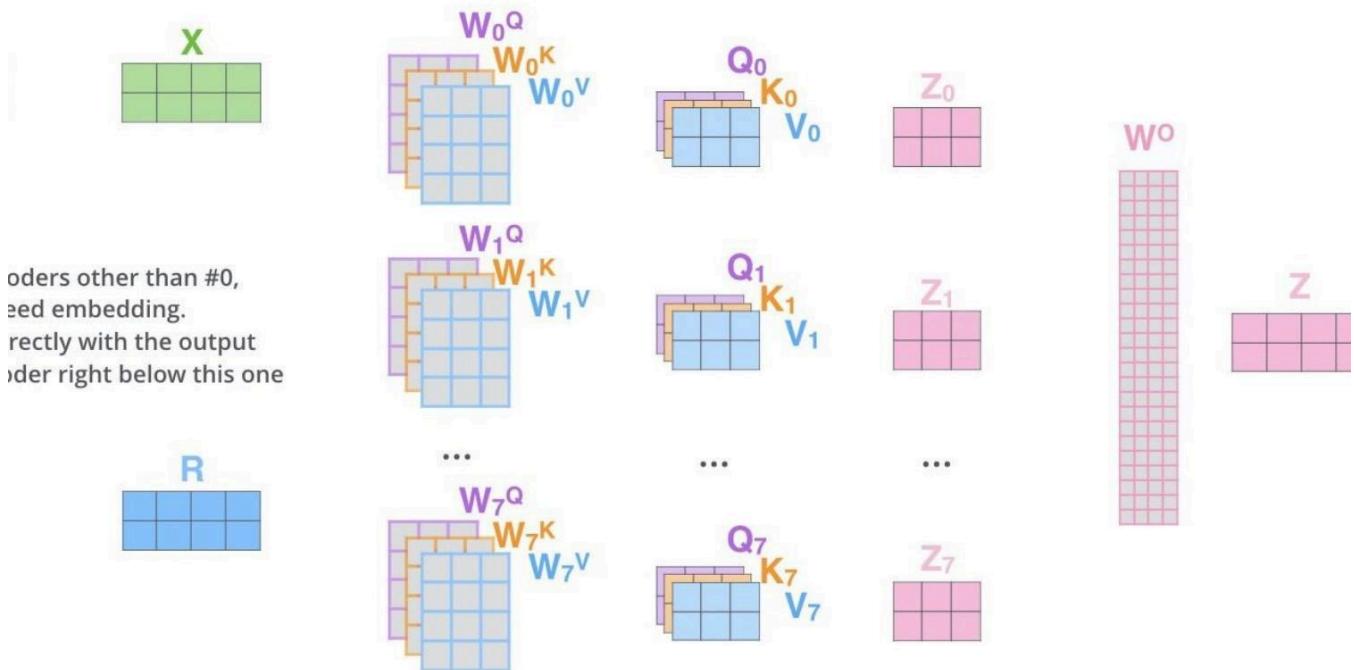
多头注意力 (Multi-Head Attention): 并行运行多个自注意力机制，每个机制使用不同的参数矩阵，捕捉输入的不同特征。每个头从不同角度分析输入数据。

例如：“拿破仑在1805年发动了乌尔姆战役，击败了奥地利军队。”

应用自注意力机制后就像请了三个专家来分析：

1. **专家A** (人际专家)：专注“拿破仑”和“奥地利军队”的关系 → 判断这是战争中的敌我关系。
2. **专家B** (历史专家)：关注“1805年”和“乌尔姆战役”的时间线 → 判断这是拿破仑早期的重要战役。
3. **专家C** (政治专家)：分析“击败”和“战役”的结果 → 推断这场胜利对欧洲局势的影响。

每个专家独立计算注意力权重，最后把他们的结论合并起来，形成一个更全面的理解。



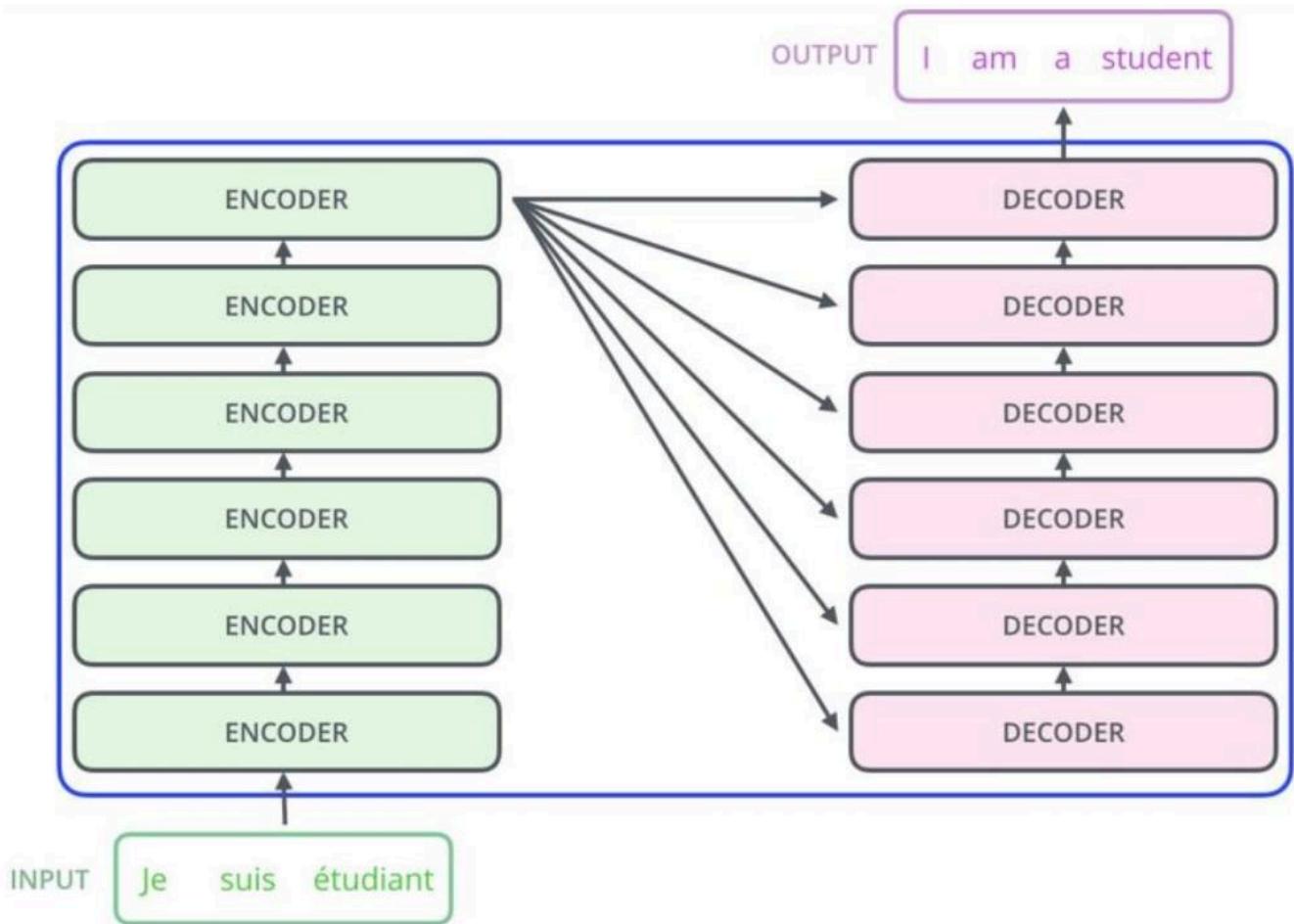
Transformer

一种新颖的架构，旨在解决序列到序列（sequence - to - sequence）任务，同时轻松处理长距离依赖（long - range dependencies）。核心为**多头自注意力机制（multi - head self - attention mechanism）**

其包含了两部分：

- **编码器（Encoder）**：将输入序列（如法文句子） (x_1, x_2, \dots, x_n) 编码为包含上下文信息的向量表示 $z = (z_1, z_2, \dots, z_n)$ ，通过这一过程进行特征提取
- **解码器（Decoder）**：基于编码器的输出(z)和已生成的部分结果，逐步生成目标序列（如翻译后的英文句子） (y_1, y_2, \dots, y_m) 这一过程具有自回归（auto - regressive）特性，即生成当前元素时，会将之前生成的符号作为额外输入，确保输出序列的连贯性

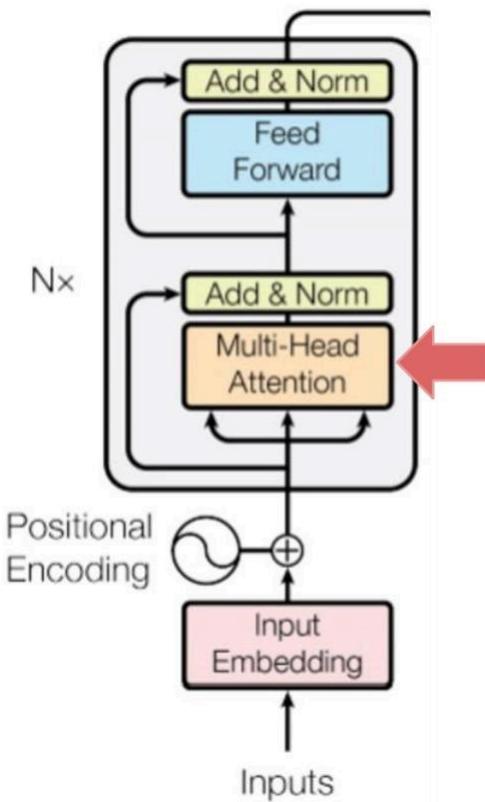
编码器和解码器均由堆叠的**自注意力（self - attention）层**和**逐点全连接层（point - wise, fully connected layers）**构成。自注意力机制用于捕捉序列内部的依赖关系，逐点全连接层对每个位置的特征进行独立变换



Attention in Encoder

在编码器的自注意力层中，所有的键（keys）、值（values）和查询（queries）均来自前一层的输出，或编码器中具有隐藏维度 (d_{model}) 的原始输入嵌入（raw input embedding）。表明自注意力机制在处理时，利用同一层或前一层的信息生成查询、键和值，以计算注意力权重

编码器中的每个位置都可以关注前一层的所有位置，体现了自注意力机制能够**捕捉全局依赖 global dependencies**的特性，打破了传统循环神经网络（RNN）的**顺序处理 sequential process**限制



Attention in Decoder

编码器处理输入序列，由于其处理得到的向量无固定顺序，因此可以自由访问所有位置

解码器与编码器不同，其生成的结果语句必须按顺序得出，需通过**掩码机制**限制只能关注当前位置及之前的信息，确保自回归特性（每次生成依赖已生成内容）。具体通过将未来位置（标记）的注意力分数设为 $(-\infty)$ ，经 Softmax 后这些位置权重趋近于 0

- **编码器**: 就像你读一篇完整的文章，可以随时回顾前后内容，理解整体含义。
- **解码器**: 就像你写一篇文章，每次只能看到已经写好的部分，不能提前知道后面的内容。

编码器 - 解码器注意力 (Encoder - decoder attention) 下图中第一个箭头：

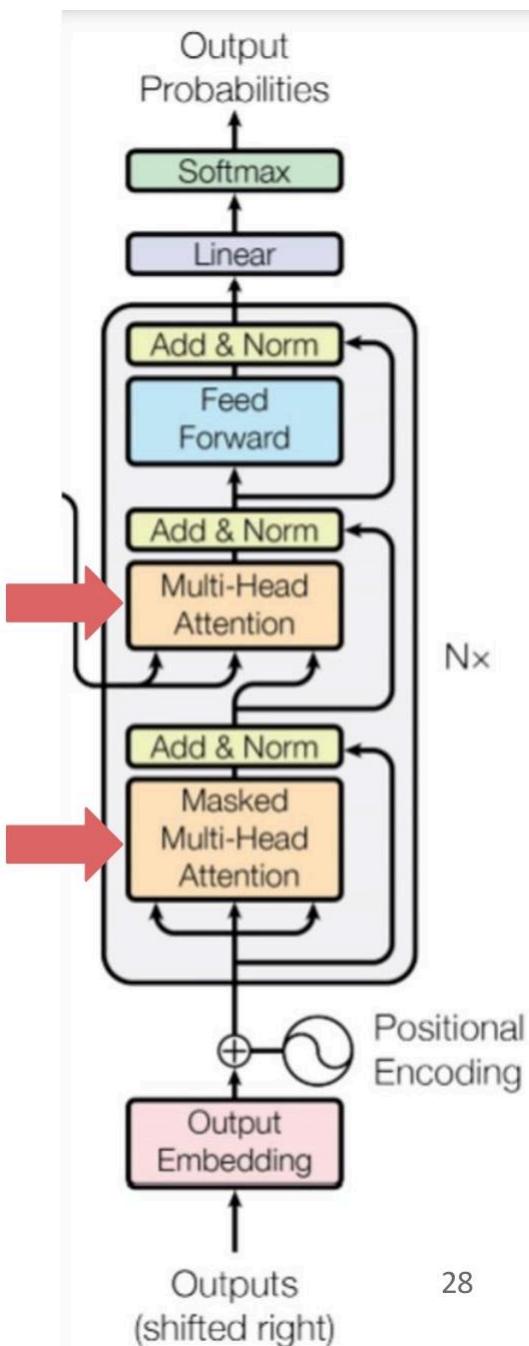
查询 (queries): 来自解码器中掩码多头注意力的输出（隐藏维度为 (d_{model}) ），使解码器在生成目标序列时能动态聚焦编码器输出的相关部分

键 (K) 和 值 (V) : 来自编码器的输出

就像在翻译时，一边参考原文（编码器的输出），一边结合自己已经翻译好的部分（解码器前一层的输出）

掩码自注意力 (Masked self - attention) 下图中第二个箭头：键、值和查询都来自解码器前一层的输出或原始输出嵌入（隐藏维度为 (d_{model}) ）

就像在写文章时，回顾自己已经写好的内容（解码器前一层的输出），确保逻辑连贯



28

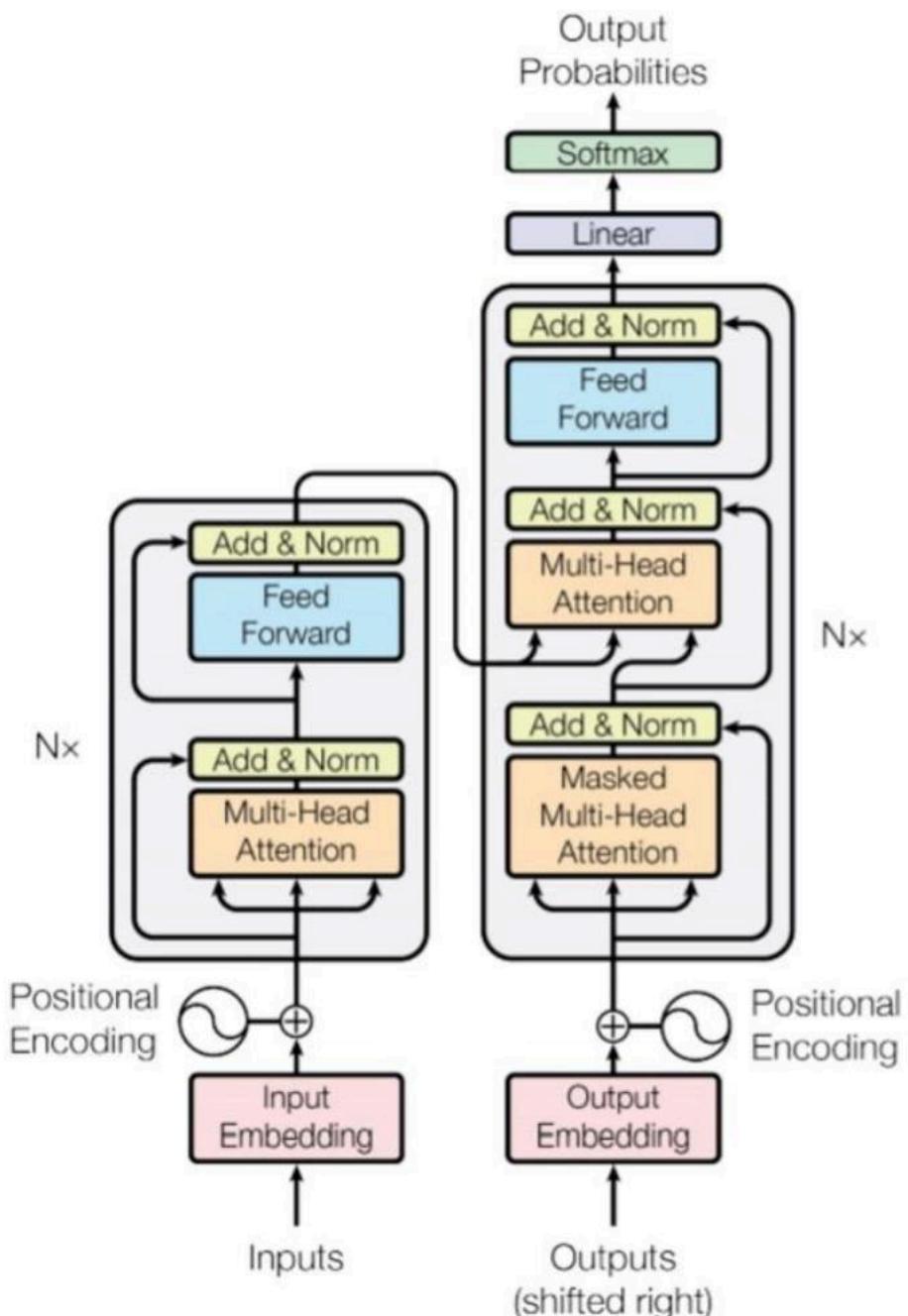
其他细节

全连接前馈网络 (FFN): 编码器和解码器的每一层都包含一个全连接前馈网络 (FFN)，该网络对每个位置position单独且相同地进行非线性变换，提取更复杂的特征

位置编码 (Positional Encodings): Transformer 本身不关心顺序，需要位置编码告诉模型序列中每个词的位置

残差连接 (Residual Connection): 其核心是让模型学习输入与子层输出之间的“变化部分”，让模型在深层网络中保留原始信息，防止信息丢失

学习嵌入 (Learned Embeddings): 发挥类似词嵌入的作用，使模型能捕捉词语的语义关系



BERT

BERT (Bidirectional Encoder Representations from Transformers, 来自 Transformer 的双向编码器表示)

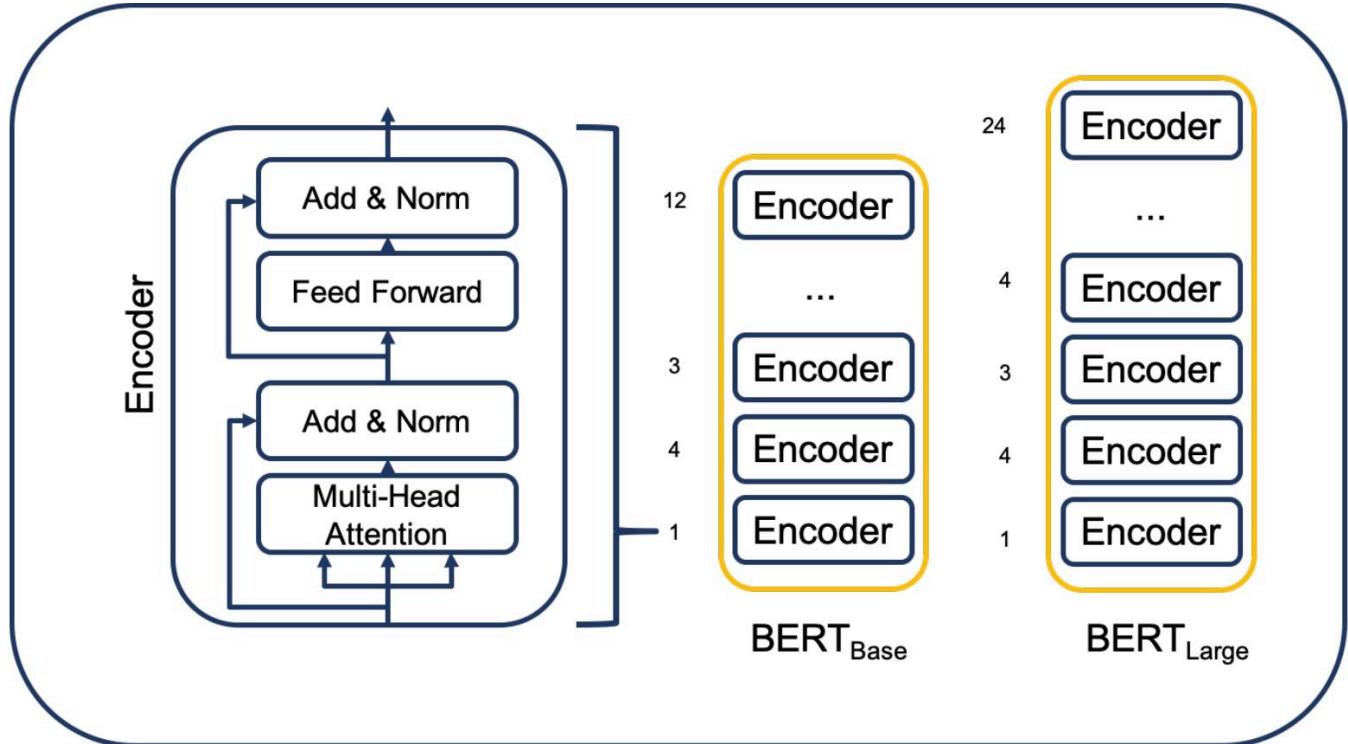
BERT 架构由多层双向 Transformer 编码器构成，这种设计使其能够在所有层中联合利用左右上下文信息，从而学习到深度双向的语言表示

BERT 用于捕捉更全面的语义信息，其输出为动态词向量（而不是自然语言）

两种不同的BERT模型：

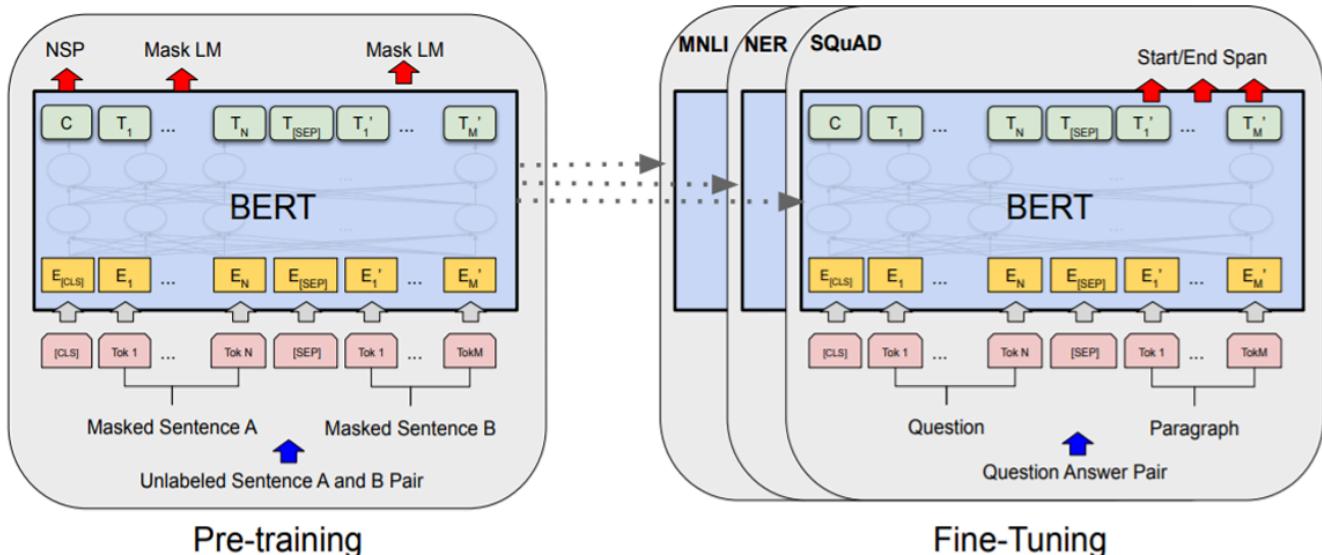
- **BERT_BASE**
- **BERT_LARGE**

两者在层数，隐藏层大小，自注意力头数量，参数数量等有差异



BERT: 预训练 (pretraining) 和微调 (finetuning)

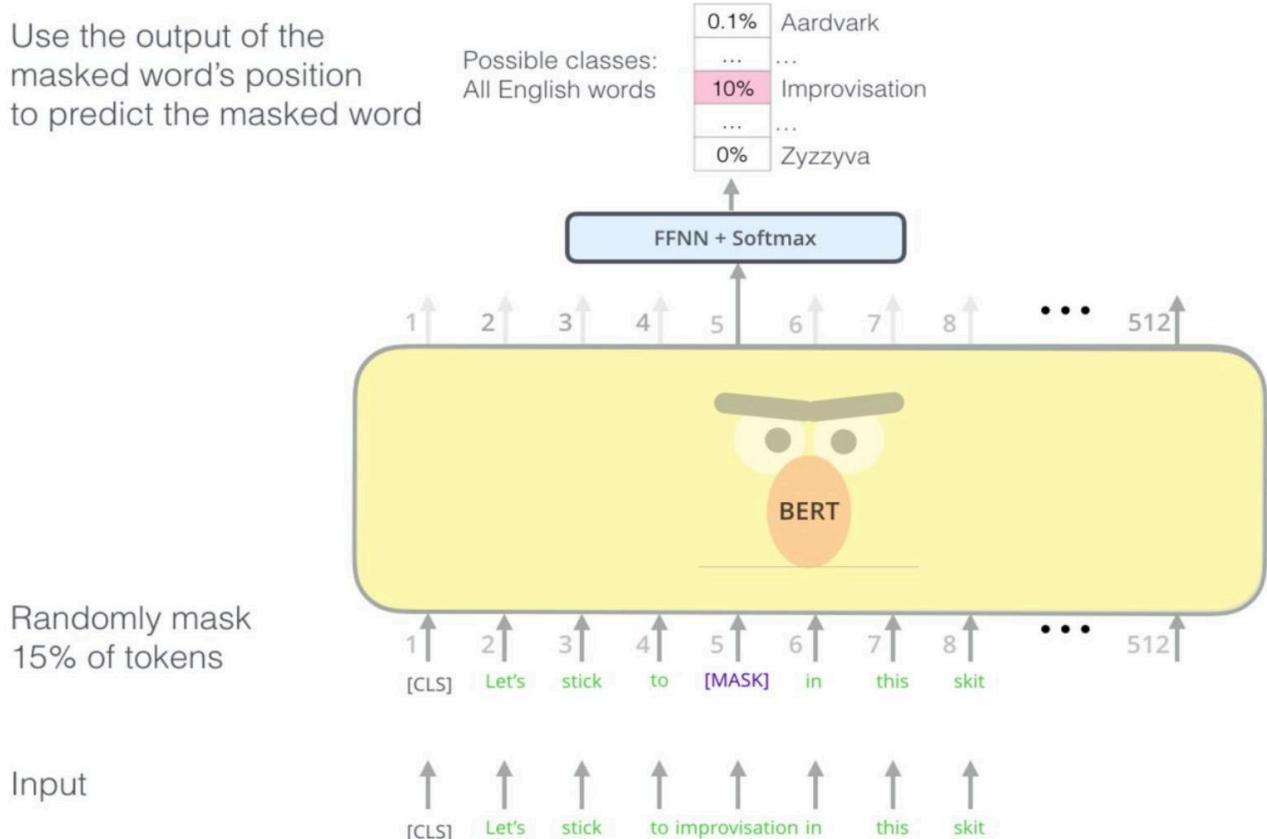
- 架构一致性：**输出层外，预训练与微调采用相同的架构，仅根据任务需求调整输出层，确保了模型结构的稳定性和通用性
- 参数初始化：**相同的预训练模型参数用于初始化不同下游任务的模型。体现了 BERT 的迁移学习特性，一个预训练模型可作为多种下游任务的起点，减少了从头训练的成本
- 全参数微调：**在微调阶段，所有参数都会进行调整



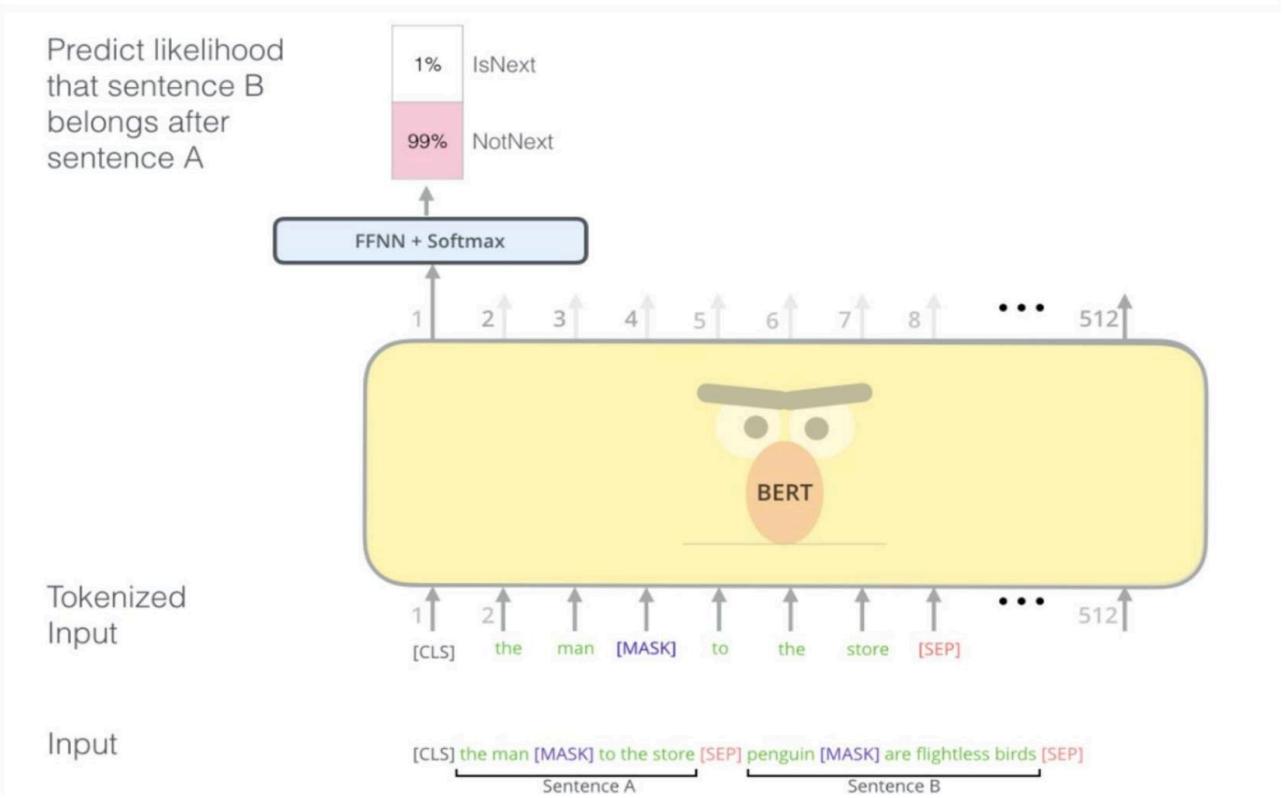
BERT 成功的关键因素

- 模型预训练方式的创新：**采用两个自监督任务进行预训练，而非传统单向语言模型

- 遮蔽语言模型（Masked Language Model, MLM）：随机遮蔽部分词汇，模型预测被遮蔽的词，强制其利用双向上下文信息，学习深层语义表征



- 下一句预测（Next Sentence Prediction, NSP）：判断句子 B 是否为句子 A 的下一句，捕捉句子间的逻辑关系与连贯性



- 训练损失为 MLM 似然均值 mean likelihood 与 NSP 似然均值 mean likelihood 之和，通过这两个任务，BERT 能更全面地理解语言结构与语义

- **对多种任务的兼容性：**只需针对特定任务微调 BERT 模型，即可在该任务上达到领先性能，无需为每个任务设计全新架构，体现了强大的通用性与迁移性。
 - BERT 在 11 个自然语言处理（NLP）任务上提升了当前技术水平

Generative Pre-training (GPT)

GPT的核心是半监督学习，包含两个阶段：

- **阶段 1 预训练：**在词级或短语级进行**无监督学习**（如词嵌入），此阶段较少依赖具体任务，更多与自然语言理解（NLU）相关。
- **阶段 2 微调：**利用这些词级特征进行**有监督训练**，但模型会因不同任务产生较大差异。

预训练通过增强语言理解能力，使模型在不同任务中具备更好的基础表现，减少对特定任务的过度依赖，提升泛化性

GPT架构

采用**单向多层 Transformer 解码器**，通过**多头自注意力机制**和**前馈网络**捕捉上下文依赖关系

预训练：自监督学习，类似 word2vec，给定标记 U ，训练目标是最大化上下文的概率 $L1(U)$

监督微调：在预训练模型的基础上，添加任务相关的输出层（如分类头）。给定输入 X 和标签 y ，最大化任务损失 $L2(C)$

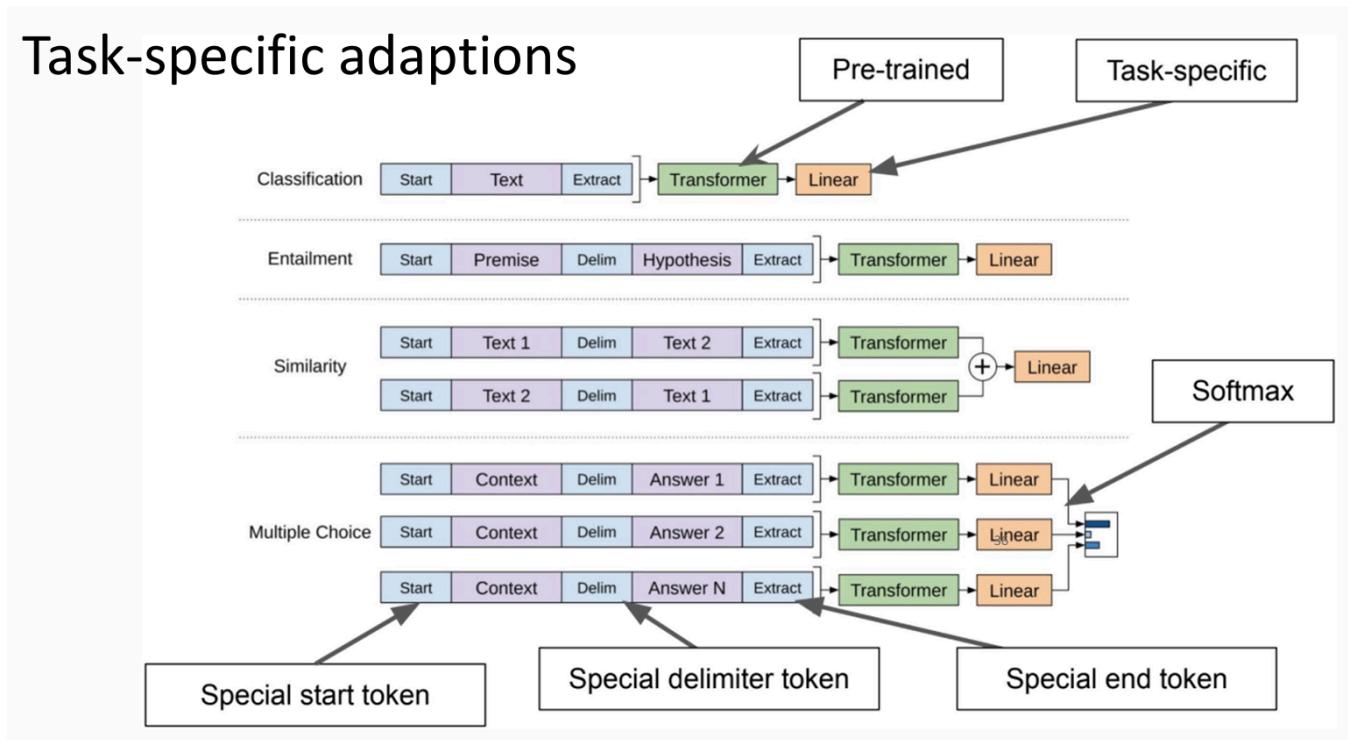
联合训练目标：结合预训练和微调目标，定义总损失 $L3(C)$ ：

$$L3(C) = L2(C) + \lambda * L1(C)$$

Task - specific adaptations

添加特殊标记适配不同任务

Task-specific adaptions



Comparison

1. ELMo

- 架构：双向 LSTM 语言模型，通过 LSTM 捕捉文本的上下文信息。
- 预训练：在无监督语料库上学习，同时提取词和语言上下文特征，这些特征用于支持下游任务。
- 下游任务：适用于基于特征的任务（如将预训练特征输入到特定任务模型中），需依赖任务特定的模型设计。

2. GPT

- 架构：单向 Transformer 解码器，采用自左向右的单向注意力机制。
- 预训练：在无监督语料库上进行，针对每个特定任务需进行判别式微调，以适配任务需求。
- 下游任务：适用于基于模型的任务，采用与任务无关的模型（task - agnostic models），通过微调统一架构处理不同任务。

3. BERT

- 架构：双向 Transformer 编码器，通过双向注意力机制充分利用上下文信息。
- 预训练：在无监督语料库上进行，设计了两个无监督任务（遮蔽语言模型 MLM 和下一句预测 NSP），增强对语义和语境的理解，同样需通过判别式微调适配具体任务。
- 下游任务：与 GPT 类似，支持基于模型的任务和与任务无关的模型，通过微调统一架构处理多种任务，灵活性强。

	Architecture	Pre-training	Downstream tasks
ELMo	Bi-directional LSTM language model	Unsupervised corpus. Learn both words and linguistic context features that support downstream tasks.	Feature-based tasks and task-specific models.
GPT	Uni-directional transformer decoder	Unsupervised corpus. Each specific task requires discriminative fine-tuning .	Model-based tasks and task-agnostic models.
BERT	Bi-directional transformer encoder	Unsupervised corpus. 2 unsupervised tasks: MLM and NSP. Each specific task requires discriminative fine-tuning .	Model-based tasks and task-agnostic models.

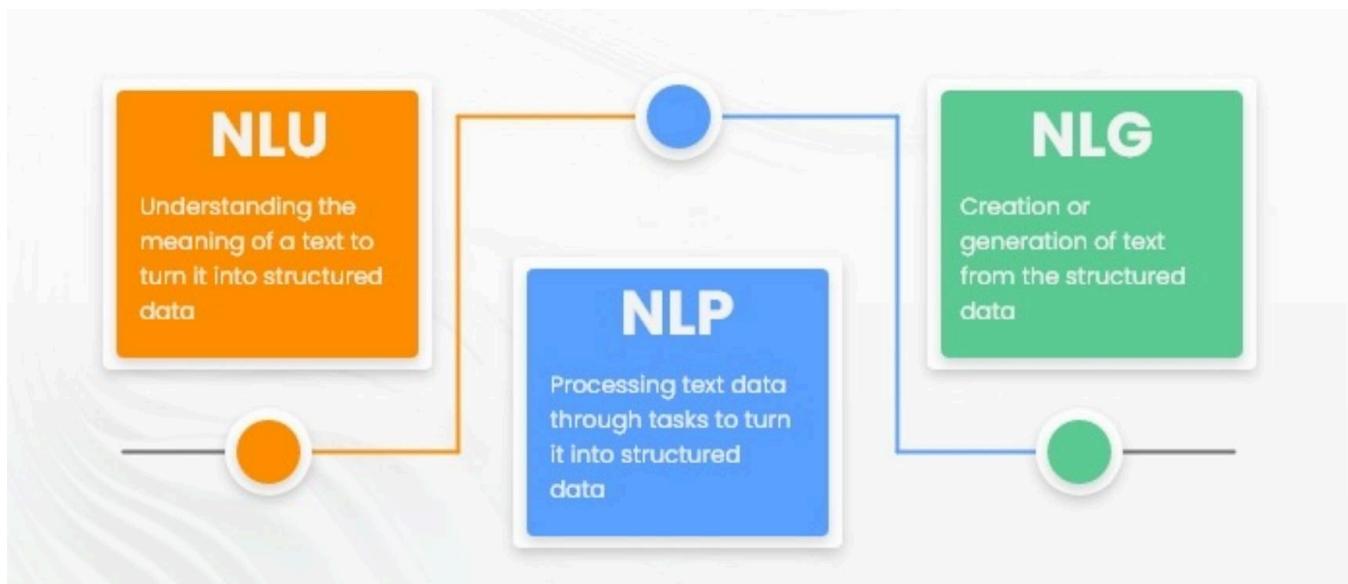
Lecture 5: NLP tasks (1) Understanding tasks



豆包

你的AI助手, 助力每日工作学习

NLU vs. NLG



NLU: (理解与分析), 包括文本分类 text classification 和问答系统 question answering (reading comprehension) 等

NLG: (生成与创造), 包括机器翻译 machine translation 和对话生成 dialogue generation

Text Classification

自然语言处理 (NLP) 中的经典问题。其核心目标是为句子、查询、段落和文档等文本单元分配标签 (labels) 或标记 (tags)

$$h : \mathcal{D} \rightarrow C$$

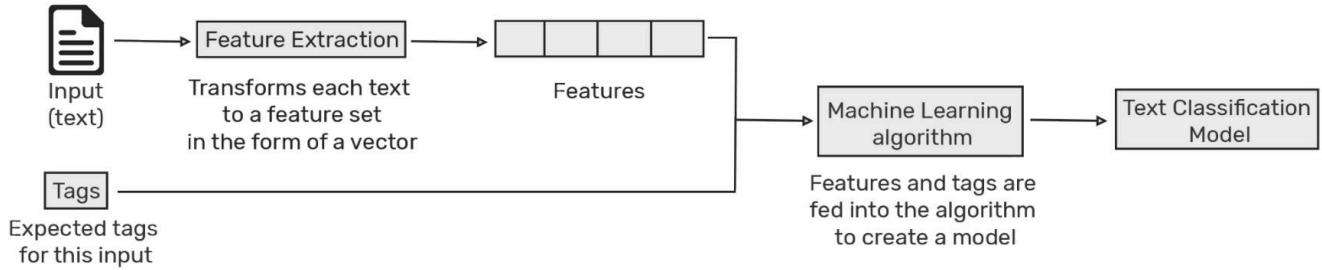
其中, $\mathcal{D} = \{x_1, x_2, \dots\}$ 代表文本数据项的集合 (如句子、文档等), $C = \{c_1, c_2, \dots, c_n\}$ 是有限的类别集合 (如“积极”“消极”“体育”“科技”等标签), 分类器 h 起映射作用

文本数据可来自多种渠道, 如网页数据、电子邮件、聊天记录、社交媒体内容、票据、保险索赔、用户评论, 以及客户服务中的问答内容等

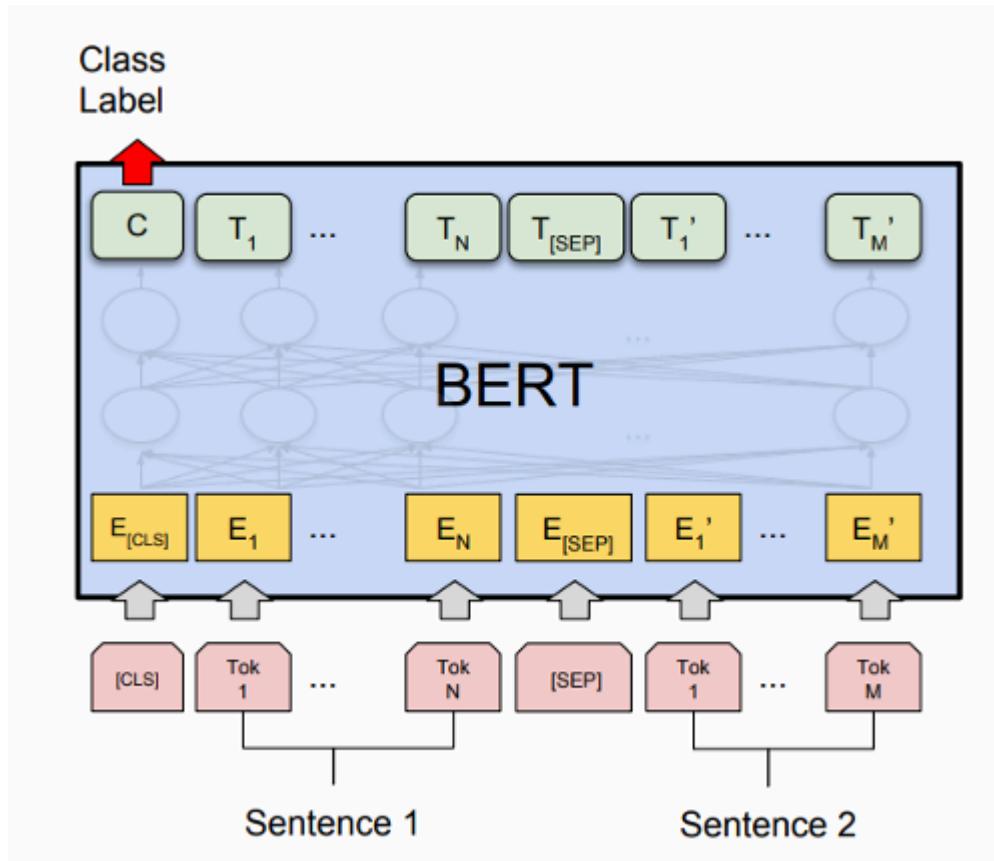
任务示例 (Task examples): 垃圾邮件检测 spam detection, 情感分析 sentiment analysis, 新闻分类 news categorization

2 Methods:

- **特征提取 + 分类 (Feature extraction + classification):** 输入文本 (Input text) 经过**特征提取 (Feature Extraction)**, 将文本转换为向量形式的特征集 (Features)。这些特征与预期标签 (Tags) 一同输入**机器学习算法 (Machine Learning algorithm)**, 训练得到**文本分类模型 (Text Classification Model)**



- **端到端模型 (End - to - end model)**: 以 BERT 为例, 直接输入文本, 模型内部自动处理 (如利用自注意力机制捕捉语义), 最终输出类别 (Class), 无需手动提取特征



文本分类中**网络选择 network selection**的相关策略：

- **选择预训练语言模型 (PLM)**: 直接选用如 BERT、GPT 等预训练语言模型，利用其已学习到的通用语言知识
 - **领域自适应 (Domain adaptation)**: 通过在领域内数据上对选定的通用领域 PLM 进行持续预训练，使其适应特定领域
 - **特定任务模型设计 (Task - specific model design)**: 在预训练模型顶部添加一个或多个特定于任务的层，以生成目标任务的最终输出，使模型更贴合具体任务需求
 - **特定任务微调 (Task - specific fine - tuning)**: 特定任务层可单独训练（固定 PLM）或与 PLM 一起训练，有时多任务训练也是不错的选择，以优化模型在特定任务上的性能
 - **模型压缩 (Model compression)**: 由于 PLM 部署成本较高，常通过知识蒸馏等方法对其进行压缩，以满足实际应用中的延迟和容量限制

文本分类任务中的评估 evaluation 指标

- **精确率 (Precision)**: 精确率衡量预测为正例的样本中实际为正例的比例，即“预测的正例有多准”

- **召回率 (Recall)**: 召回率衡量实际正例中被正确预测为正例的比例，即“实际的正例被找全了多少”
- **F1 分数 (F1 - score)**: 精确率和召回率的调和平均数

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F1\text{-score} = \frac{2 \times \text{Prec} \times \text{Rec}}{\text{Prec} + \text{Rec}}$$

Question answering

QA的核心理念是从材料（文档、对话、在线搜索等）中提取信息，给出**简短精确 (short and concise)** 的答案，满足用户的信息需求

分为：

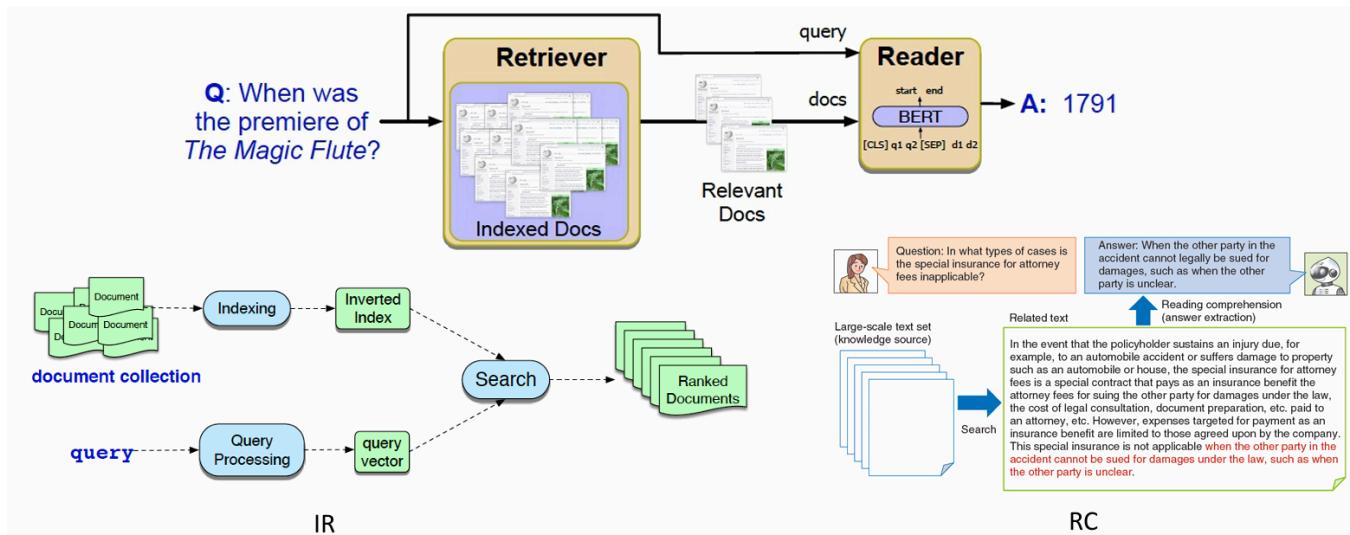
- **Factoid (事实型)**: 最简单且常见，例如询问“氧化汞的符号是什么？”
- **Mathematical (数学型)**: 涉及数学计算，如“ $2 + 3 = ?$ ”

Factoid QA

可分为两种思路：

- **基于信息检索 (IR) 的问答 (开放域问答) Information-retrieval (IR) based QA (open domain question QA)**: 当收到用户问题时，首先利用信息检索技术从网络上大量资料中查找相关段落。然后，通过神经阅读理解算法 neural reading comprehension 读取这些检索到的段落，并直接从文本片段中提取答案
- **基于知识的问答 Knowledge-based QA**: 先将查询构建为语义表示 (逻辑查询)，再用这种表示查询事实数据库

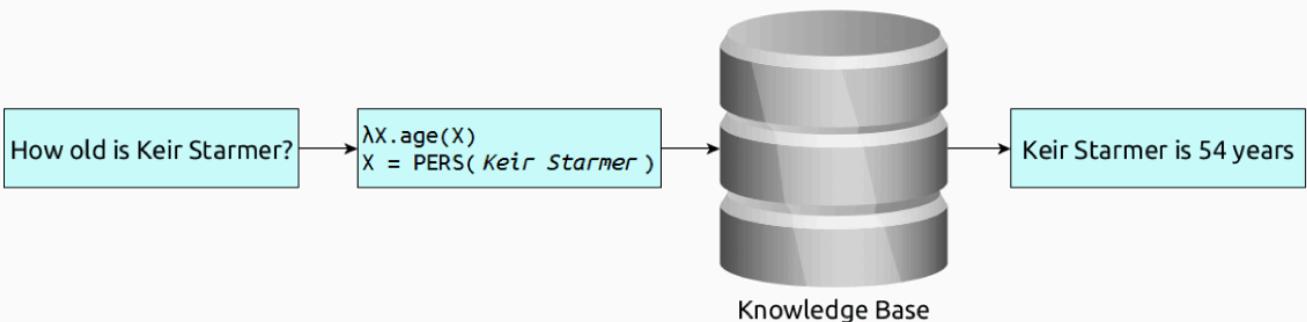
IR-based QA



IR: 对文档生成索引 Index，然后将用户的问题生成查询向量 query vector，根据查询向量找到相关文档 relevant document

RC: 将IR得到的文档输入阅读器网络中，阅读器分析得到答案

Knowledge-based QA



Question → Logical Form → KB Query → Answer

问题 → 逻辑形式 → 知识库查询 → 答案

阅读理解 (Reading Comprehension, RC)

阅读并理解非结构化文本，然后回答相关问题的能力

- **输入：**包含上下文（一段文本）和查询（问题）。
- **输出：**答案，分为两种类型：
 - **摘要型 (abstractive)：**自由形式的答案，更强调生成内容（例如，对文本内容进行概括总结后回答）。
 - **提取型 (extractive)：**直接提取文本中的子串作为答案，更侧重对文本的理解（例如，从文本中直接找出符合问题的具体语句）

RC在QA中扮演一种可能的解决方法 a possible solution approach

Stanford question answering dataset (SQuAD)

SQuAD 1.1

高质量的**提取型 Extractive** QA数据库，人工创造

形式为“问题 - 上下文 - 答案” “question - context - answer” 三元组

有助于训练和评估模型的文本理解与答案提取能力

Question: Which team won Super Bowl 50?

Passage

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.

Computational complexity theory is a branch of the theory of computation in theoretical computer science that focuses on classifying computational problems according to their inherent difficulty, and relating those classes to each other. A computational problem is understood to be a task that is in principle amenable to being solved by a computer, which is equivalent to stating that the problem may be solved by mechanical application of mathematical steps, such as an algorithm.

By what main attribute are computational problems classified utilizing computational complexity theory?
 Ground Truth Answers: inherent difficulty their inherent difficulty inherent difficulty
 Prediction: inherent difficulty

3 gold answers are collected for each answer

随后有了进阶版本SQuAD 2.0

其中包含了根据已有上下文无法回答的问题 unanswerable questions

这要求系统不仅需要在有答案时正确回答问题，还必须能够判断段落中有没有支持答案的内容

Genghis Khan united the Mongol and Turkic tribes of the steppes and became Great Khan in 1206. He and his successors expanded the Mongol empire across Asia. Under the reign of Genghis' third son, Ögedei Khan, the Mongols destroyed the weakened Jin dynasty in 1234, conquering most of northern China. Ögedei offered his nephew Kublai a position in Xingzhou, Hebei. Kublai was unable to read Chinese but had several Han Chinese teachers attached to him since his early years by his mother Sorghaghtani. He sought the counsel of Chinese Buddhist and Confucian advisers. Möngke Khan succeeded Ögedei's son, Güyük, as Great Khan in 1251. He

When did Genghis Khan kill Great Khan?

Gold Answers: <No Answer>

Prediction: 1234 [from Microsoft nInet]

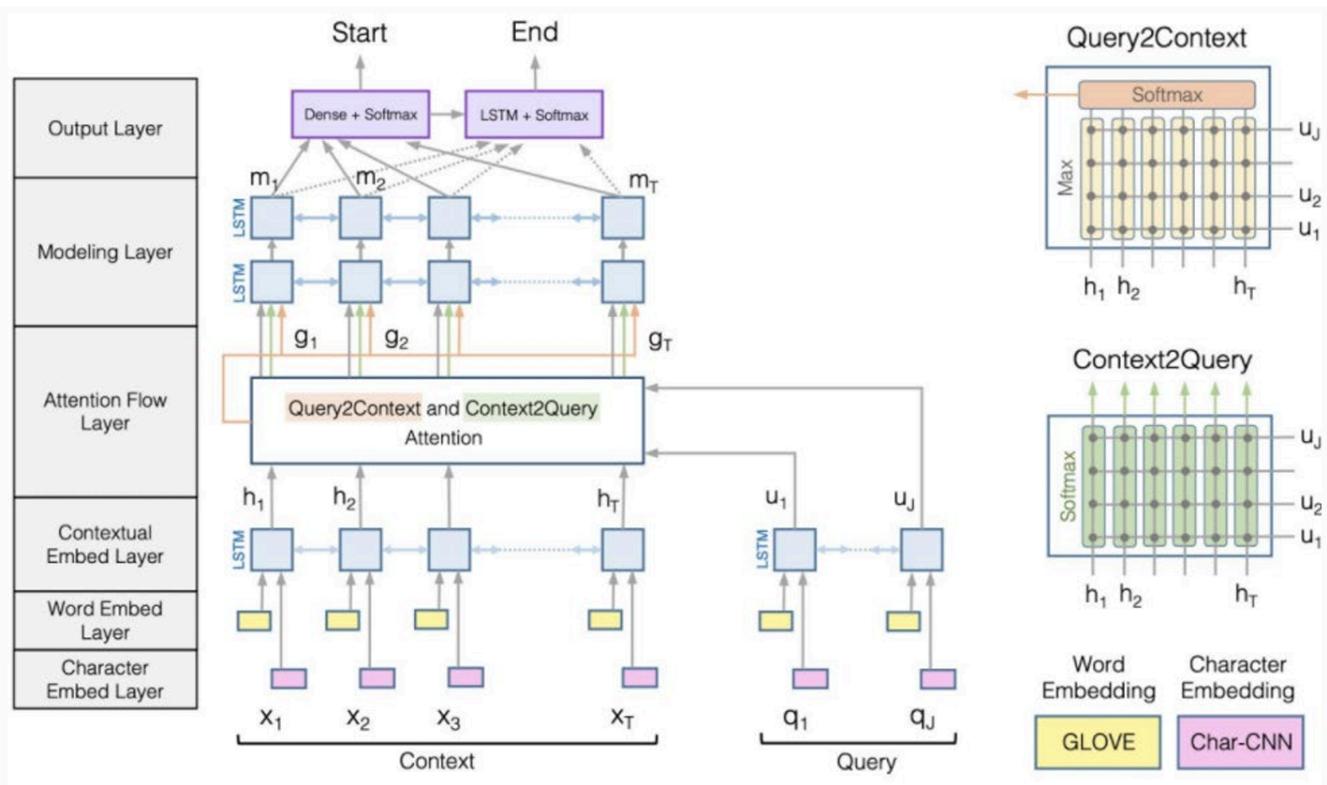
BiDAF

BiDAF (Bidirectional Attention Flow for Machine Comprehension, 机器理解双向注意力流)

一个模型，旨在更好地将注意力机制融入问答（QA）系统

特点：

- **双向注意力：**包含“查询到上下文（query - to - context）”和“上下文到查询（context - to - query）”的双向注意力机制
- **多粒度嵌入：**整合了字符级 character-level、词级 word-level 和上下文 contextual 嵌入，丰富输入表示



- 嵌入层 (Embedding Layers)**: 包含词嵌入层 (Word Embedded Layer), 字符嵌入层 (Character Embedded Layer) 和上下文嵌入层 (Contextual Embedded Layer)
- 注意力流层 (Attention Flow Layer)**: Query2Context 注意力以及Context2Query 注意力, 输出注意力向量
- 建模层 (Modeling Layer)**: 将注意力层传来的注意力向量用以建模长距离依赖关系, 生成最终的上下文表示
- 输出层 (Output Layer)**: 基于上下文表示预测答案的起始位置和终止位置

Performance metrics

阅读理解模型的性能指标

训练时：采用对数似然 log likelihood 函数来评估

$$L(\theta) = -\frac{1}{N} \sum_i [\log(p_{y_i}^1) + \log(p_{y_i}^2)]$$

其中 N 是样本数量, $p_{y_i}^1$ 和 $p_{y_i}^2$ 分别表示正确答案起始和结束位置的预测概率

测试时：选择起始和结束索引对 $(i, j) (i < j)$, 使 $p1(i) \times p2(j)$ 最大化, 其中 $p1(i)$ 和 $p2(j)$ 分别是位置 i 作为起始点、j 作为结束点的概率

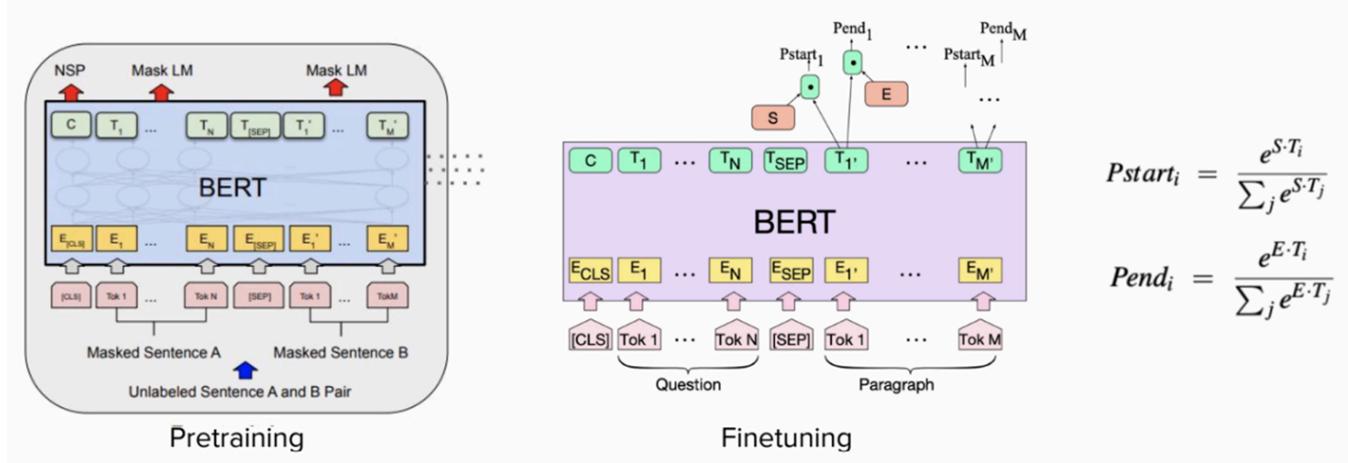
- 完全匹配 (Exact Match, EM)**: 预测的起始和结束索引与某一标准答案的索引完全一致
- F1 分数**: 将预测答案和标准答案视为词袋, 计算精确率 (Precision) 和召回率 (Recall) 的调和均值

$$\text{Precision} = \frac{\text{正确预测的词数}}{\text{预测的总词数}}$$

$$\text{Recall} = \frac{\text{正确预测的词数}}{\text{标准答案的总词数}}$$

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Current SOTA: pre-trained models



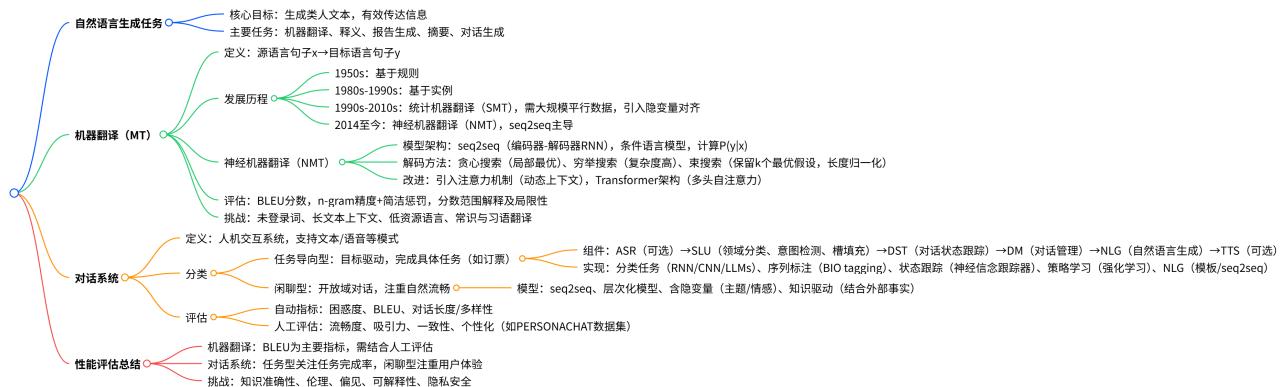
微调BERT：一个先进的预训练后模型

为QA任务微调时，引入两个向量 S （起始向量）和 E （结束向量）作为微调参数，用于计算每个词作为答案起始位置（start）和结束位置（end）的概率

$$P_{\text{start}_i} = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$

$$P_{\text{end}_i} = \frac{e^{E \cdot T_i}}{\sum_j e^{E \cdot T_j}}$$

Lecture 6: NLP tasks (2) Generation tasks



豆包
你的AI助手，助力每日工作学习

NLG

NLG 核心：聚焦于生成类似人类的文本，以有效传达convey信息或实现有效沟通communication
例如机器翻译machine translation和对话生成dialogue generation等

Machine Translation

Introduction

机器翻译任务：将一个句子x从一种语言（源语言）翻译成另一种语言（目标语言）的句子y

x: *L'homme est né libre, et partout il est dans les fers*



y: *Man is born free, but everywhere he is in chains*

- Rousseau

翻译模型 $P(x|y)$ 的学习需要大量的**并行数据**（即两种语言的人工翻译句子对，如德语 - 英语句子对）

随后引入**隐变量 a**，将模型表示为 $P(x, a|y)$ 。其中 a 代表**对齐 alignment**，即源语言与目标语言词汇间的映射规则

然而对齐并不是简单地一一对应，不同语言之间存在类型差异 Typological Difference，例如法语的 “Le” 在英语原句中无直接对应词

因此对齐在机器翻译中很重要

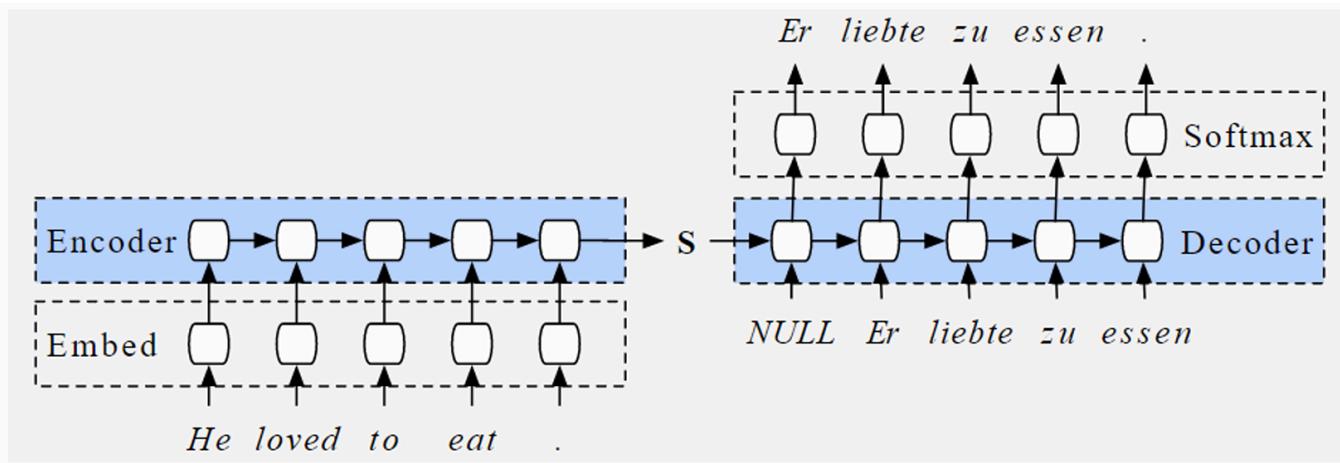
NMT

神经机器翻译 Neural Machine Translation，一种基于**深度学习**的机器翻译方法

序列到序列 (seq2seq)

主要由编码器（Encoder RNN）和解码器（Decoder RNN）两部分构成

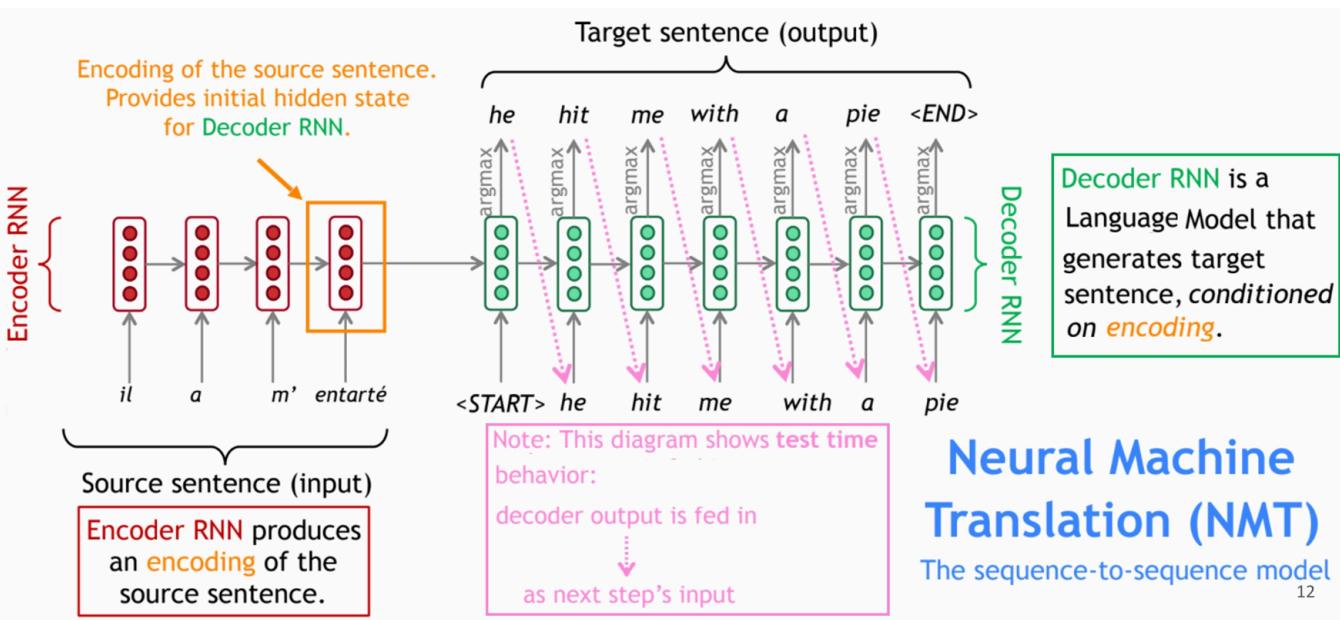
编码器负责捕捉源句子的语义信息，解码器负责基于编码器输出来生成目标句子



基于seq2seq模型的NMT结构如下

Encoder RNN基于源句子的语义生成编码

Decoder RNN在编码器输出的编码基础上生成目标句子



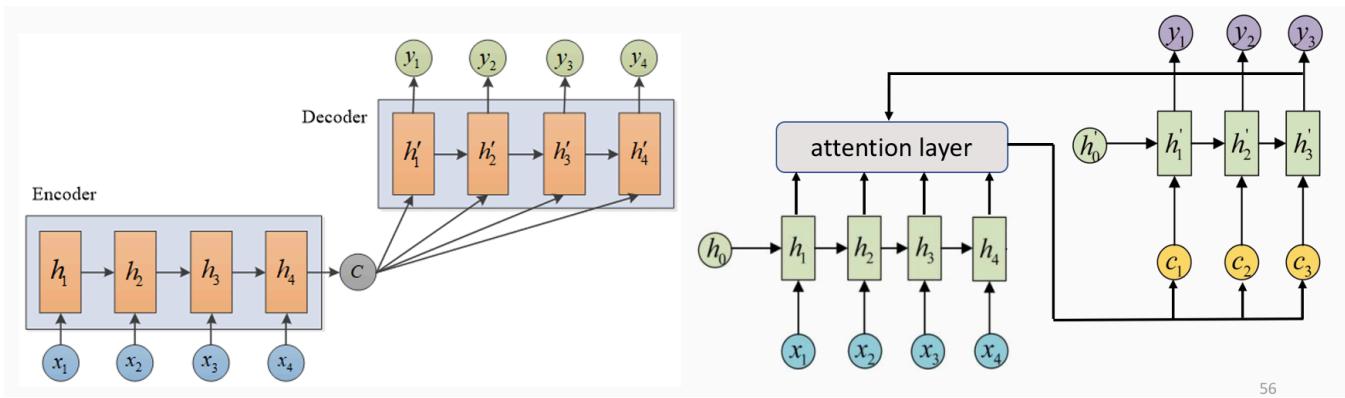
本质上，seq2seq是条件语言模型的一个实例，也就是说其遵循语言模型的基本功能，即根据已有条件（源句子和已生成序列）预测概率

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x)\dots P(y_T|y_1, \dots, y_{T-1}, x)$$

seq2seq中的注意力机制

无注意力机制：编码器生成一个固定的上下文向量 C ，解码器仅依赖于该 C 生成输出。由于上下文信息不变，解码器难以动态聚焦输入序列的不同部分。

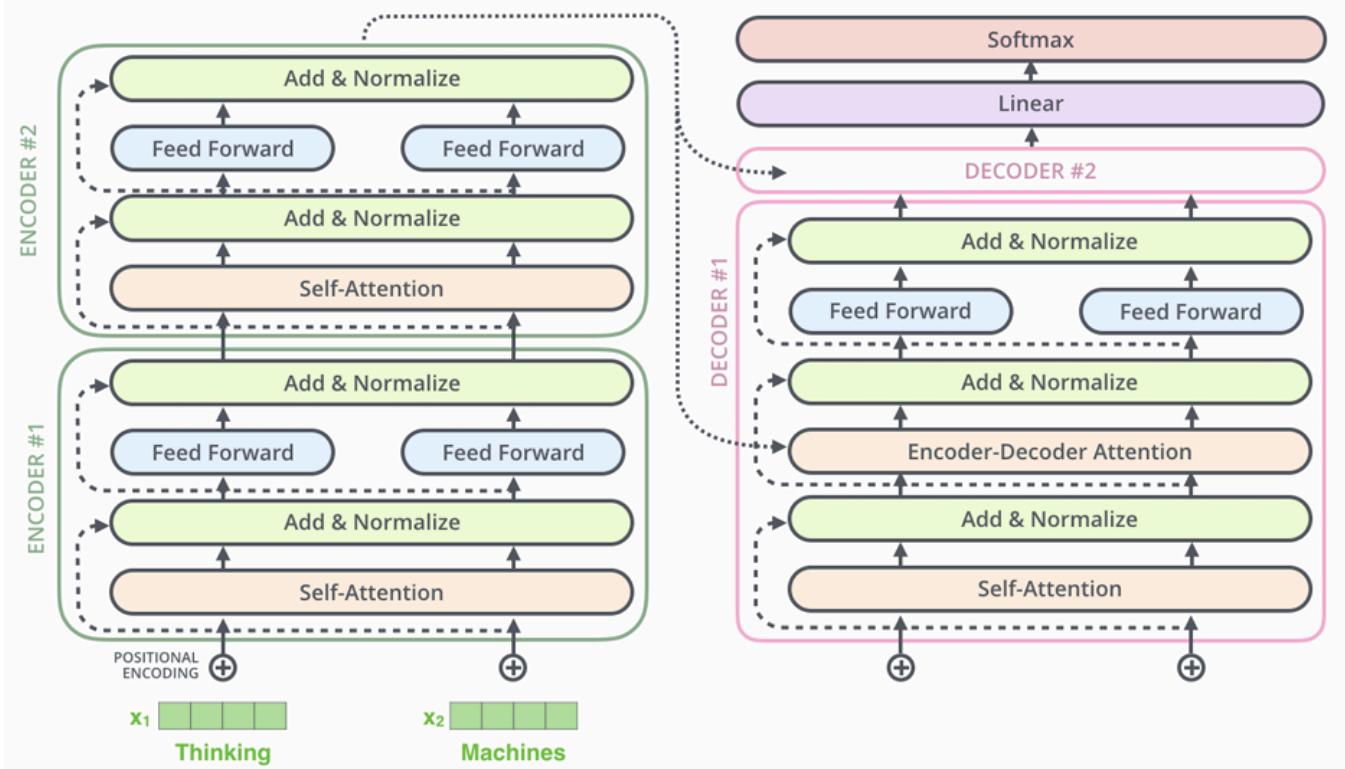
有注意力机制：编码器处理后的输出通过注意力层动态计算不同的上下文向量 c_1, c_2, c_3 ，使解码器在生成输出时能灵活关注输入序列的不同部分



56

Transformer-based MT

编码器 - 解码器 (Encoder - Decoder) 结构，解码方式与序列到序列 (seq2seq) 模型类似，但内部机制依赖 Transformer 的核心组件**多头自注意力机制 (multi-head self attention mechanism)**



Decoding

在生成概率后，需要decoding，即将模型的概率输出转换为连贯coherent且有意义meaningful的文本序列

贪婪解码 (Greedy decoding)

在解码器的每一步，通过取 **argmax** (选择概率最大的词) 来生成目标句子，即每一步都选择当前最可能的词

问题：不可逆性 no way to undo，发现错误后无法回溯修改

是一种局部最优的解码策略，每一步只考虑当前最佳选择，但缺乏全局调整的能力，可能导致后续翻译效果不佳

穷举搜索解码（Exhaustive search decoding）

不同于每一步单独最大化概率，目标是寻求整体概率最优的翻译序列

问题：计算量极大，因为需要将所有可能的序列都罗列并计算概率，实际应用中难以实现

集束搜索解码（Beam search decoding）

在解码的每一步，保留 k 个最可能的部分翻译（k 个假设），k 称为集束大小，实际应用中通常取 5 到 10。这种方式平衡了贪婪解码（仅选当前最优）和穷举搜索（计算所有可能）的缺陷，在计算效率与搜索质量间取得折中

其中包含了对假设评分以及长度归一化的过程，避免翻译长度影响评分

BLEU

BLEU (Bilingual Evaluation Understudy, 双语评估辅助工具)，专门用于评估MT模型

评估方式：BLEU 将机器生成的翻译（候选句子）与一个或多个手工翻译（参考句子）进行比较，基于以下两方面计算相似度分数：

- **n - gram 精度 (p_n)**：通常计算 1 - gram、2 - gram、3 - gram 和 4 - gram 的精度，衡量机器翻译中 n 元组在参考译文中出现的比例
- **简洁惩罚 (Brevity Penalty, BP)**：用于惩罚过短的机器翻译

最终分数计算： $\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right)$ ，其中 w_n 是各 n - gram 精度的权重

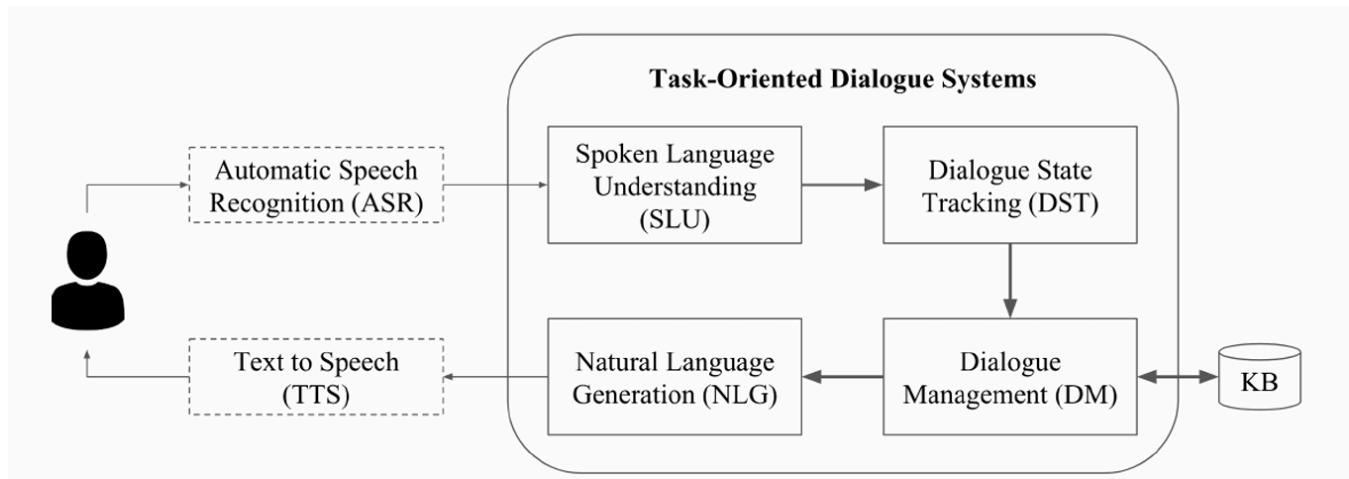
Dialog Systems

对话系统是旨在与人类进行对话的计算机系统

主要包含：

- **任务导向型 (Task - oriented)**
 - 涵盖开放或封闭领域domain。
 - 目标是识别用户的任务，并执行相应操作以完成目标。
 - 示例包括预订餐厅、订购电影票、查询账户余额等。
- **闲聊型 (Chitchat)**
 - 属于开放领域。
 - 旨在以对话的方式回应用户输入，侧重于社交互动性的交流。
 - 例如进行日常社交性质的对话。

Task-oriented



其中Task-oriented dialogue systems主要包括：SLU (NLU) -> DST -> DM (<-> knowledge base) -> NLG

NLU (Natural Language Understanding): 理解用户请求，例如餐厅推荐

DST (Dialogue State Tracking): 通过状态机state machines跟踪对话状态

DM (Dialogue Management): 决定智能体应采取的行动，例如是否需要请求额外信息

NLG (Natural Language Generation): 将系统决策转化为自然语言或 GUI 输出

ChitChat

从不同角度（语料学习、隐变量建模、知识整合）实现闲聊对话生成

Evaluation

人工评估的局限性：理想情况下，人工评估能准确衡量对话系统质量，但成本高昂

启发式评估方法 Heuristics：采用自动评估指标，包括困惑度 perplexity、BLEU、对话长度 / 多样性等

开放性问题：对话系统评估仍是一个未完全解决的开放问题

Lecture 7: Large Language Models



豆包

你的 AI 助手，助力每日工作学习

Large language model definition

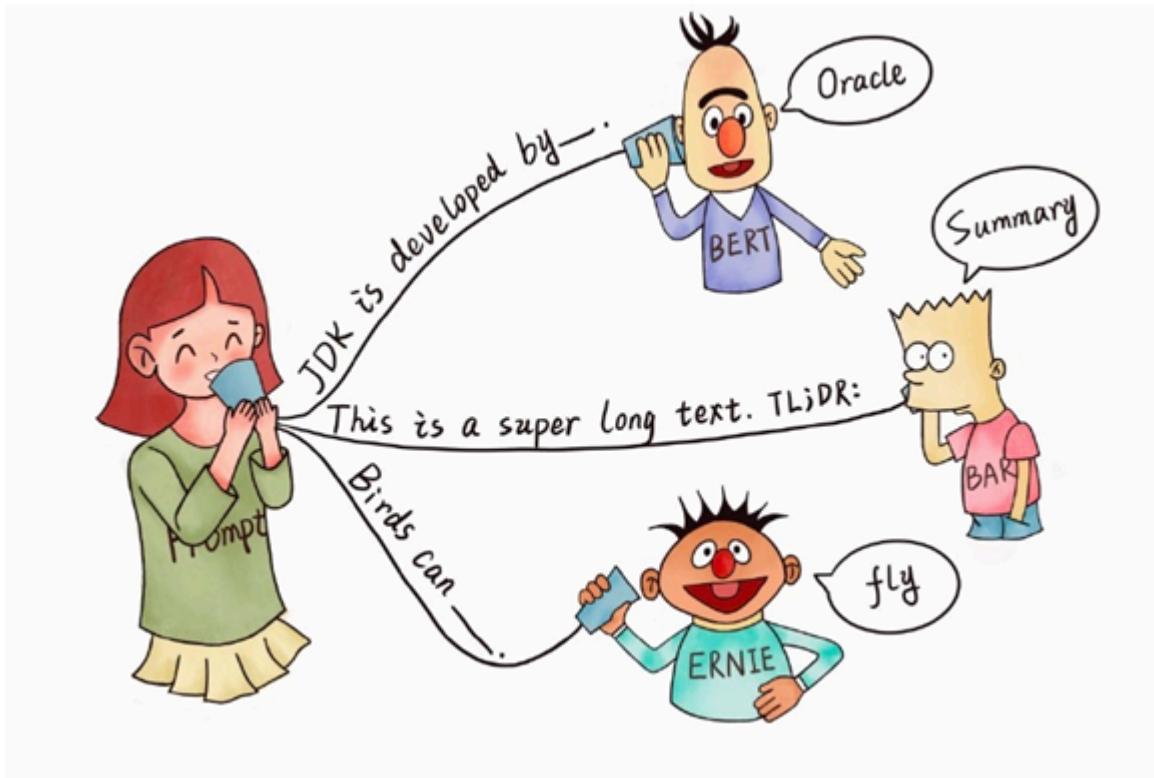
大型语言模型（LLMs）具有强大的文本处理能力，可以进行故事编写，编程，音乐创作等多种应用

相比起小语言模型SLMs，LLMs的差别在于：

- **规模（Size）**：小语言模型参数可能少于 1500 万，而大语言模型拥有数百亿参数。
- **计算需求（Computational Requirements）**：小语言模型可使用移动设备处理器，大语言模型则需要数百个 GPU 处理器。
- **性能（Performance）**：小语言模型处理简单任务，大语言模型能应对复杂、多样的任务。
- **部署（Deployment）**：小语言模型在资源受限环境中更易部署，大语言模型部署通常需要大量基础设施。
- **训练（Training）**：小语言模型训练周期约一周，大语言模型训练可能需数月。

Prompt engineering

提示工程（Prompt engineering）：提示（Prompt）是插入输入示例中的一段文本，其作用是将原始任务转化为（掩码）语言建模问题，以便模型更好地理解和执行任务

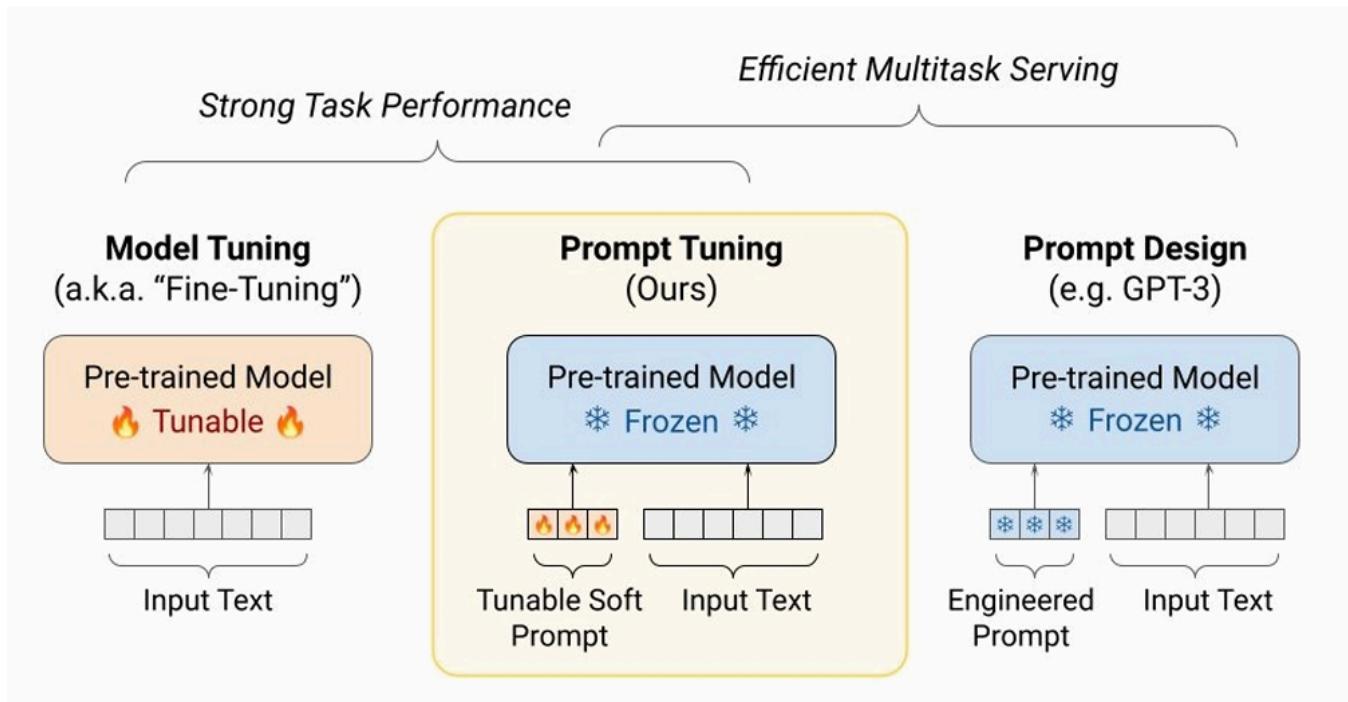


Paradigm shift

LLM实现了从“预训练 - 微调”(pretraining - finetuning) 转向“提示”(prompting) 的新模式

在传统LM中，预训练模型是可调整的（Tunable），通过微调使其适应下游任务

而到了LLM中，预训练模型通常被冻结（Frozen），通过添加输入前缀（即提示）让模型执行不同任务，无需修改模型本身



Scaling law

Laws

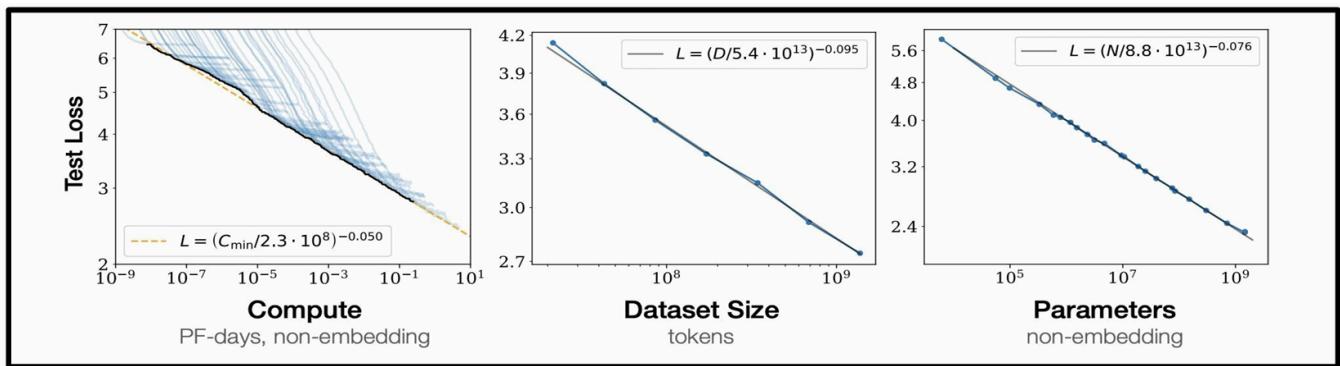
缩放定律 Scaling law是用于预测模型性能的简单规则

通过经验观察发现，扩大模型规模能可靠地提升模型的困惑度 perplexity

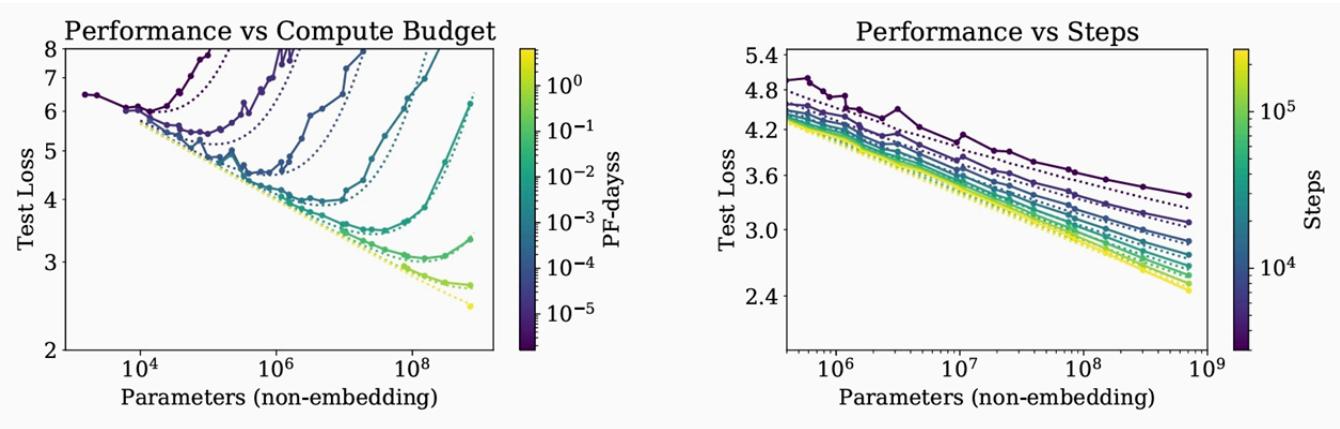
- **模型性能 (L)**: 通常用交叉熵损失 cross entropy loss 衡量。
- **模型参数数量 (N)**: 模型参数 model parameter 数量 (如GPT-3的1750亿参数)。
- **数据量 (D)**: 训练数据的token数量 (如GPT-3使用约5700亿token)。
- **计算资源 (C)**: 训练过程中所需的浮点运算次数 (FLOPs)。

其中：

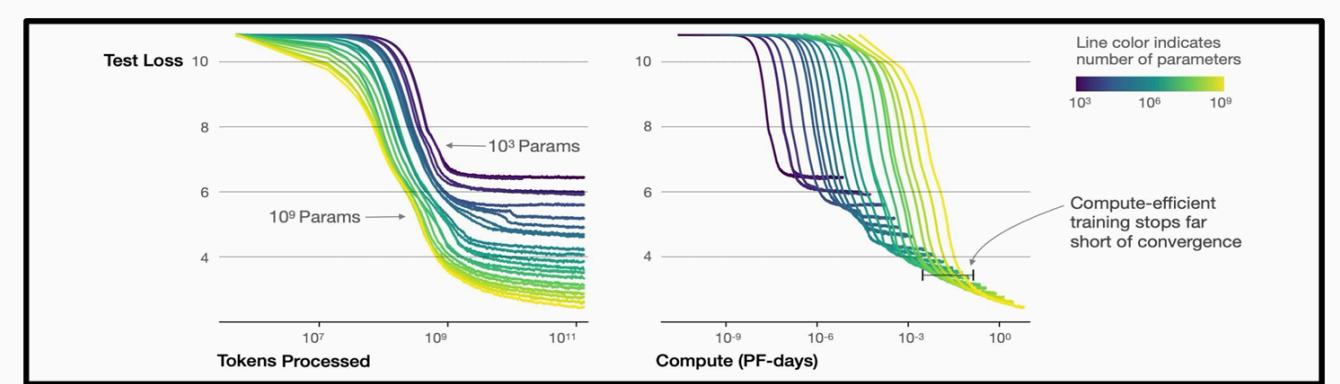
- **性能与规模的强依赖关系 Performance strongly depends on scale, weakly on model shape**：模型性能 (Performance) 强烈依赖于“规模 (Scale)”，规模由三个因素构成：模型参数数量 N (非嵌入部分)、数据集大小 D、训练所用计算量 C。在合理范围内，性能对模型架构 model shape 的依赖很弱
- **平滑的幂律关系 Smooth Power Laws**：当不受其他两个因素限制时，性能与 N、D、C 三个规模因素分别呈平滑的幂律关系



- **过拟合的普遍性 (Universality of overfitting)**: 只要同时扩大模型参数 N 和数据集 D，模型性能会可预测地提升；但如果固定 N 或 D 中的一个，仅增加另一个，收益就会降低
- **训练的普遍性 (Universality of training)**: 训练曲线遵循可预测的幂律，其参数大致与模型大小无关



- **样本效率 (Sample efficiency)**: 指出大模型比小模型具有更高的样本效率，即大模型能够以更少的优化步骤和数据点达到与小模型相同的性能水平



Emergent abilities

zero-shot/few-shot learning

GPT - 3拥有大量参数N以及训练数据D，仅训练语言模型，却意外获得了处理多种任务的能力，如文本生成、机器翻译、阅读理解等

zero-shot: 无实际数据下学习

Zero-shot

1

Translate English to French: ←

2

cheese => ←

Few-shot: 只有少量数据下学习

Few-shot

1

Translate English to French: ←

2

sea otter => loutre de mer ←

3

peppermint => menthe poivrée ←

4

plush girafe => girafe peluche ←

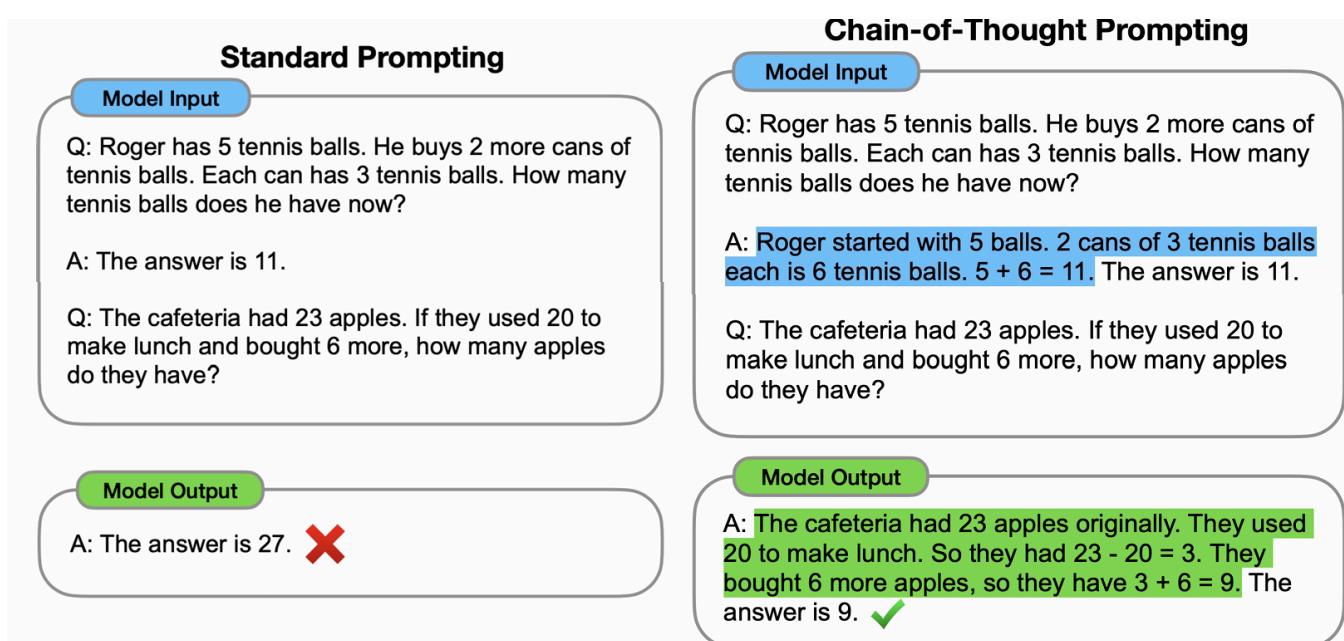
5

cheese => ←

Chain-of-thought prompting

大语言模型（LLM）在生成最终答案前，显式输出中间推理步骤的提示技术

将复杂问题分解为多个逻辑步骤，从而提升模型的推理能力、准确性和可解释性



Alignment

Language modeling ≠ assisting users

传统语言模型在理解和对齐用户真实意图方面存在不足，仅依据自身语言模式生成内容，未必能真正协助用户达成特定目标

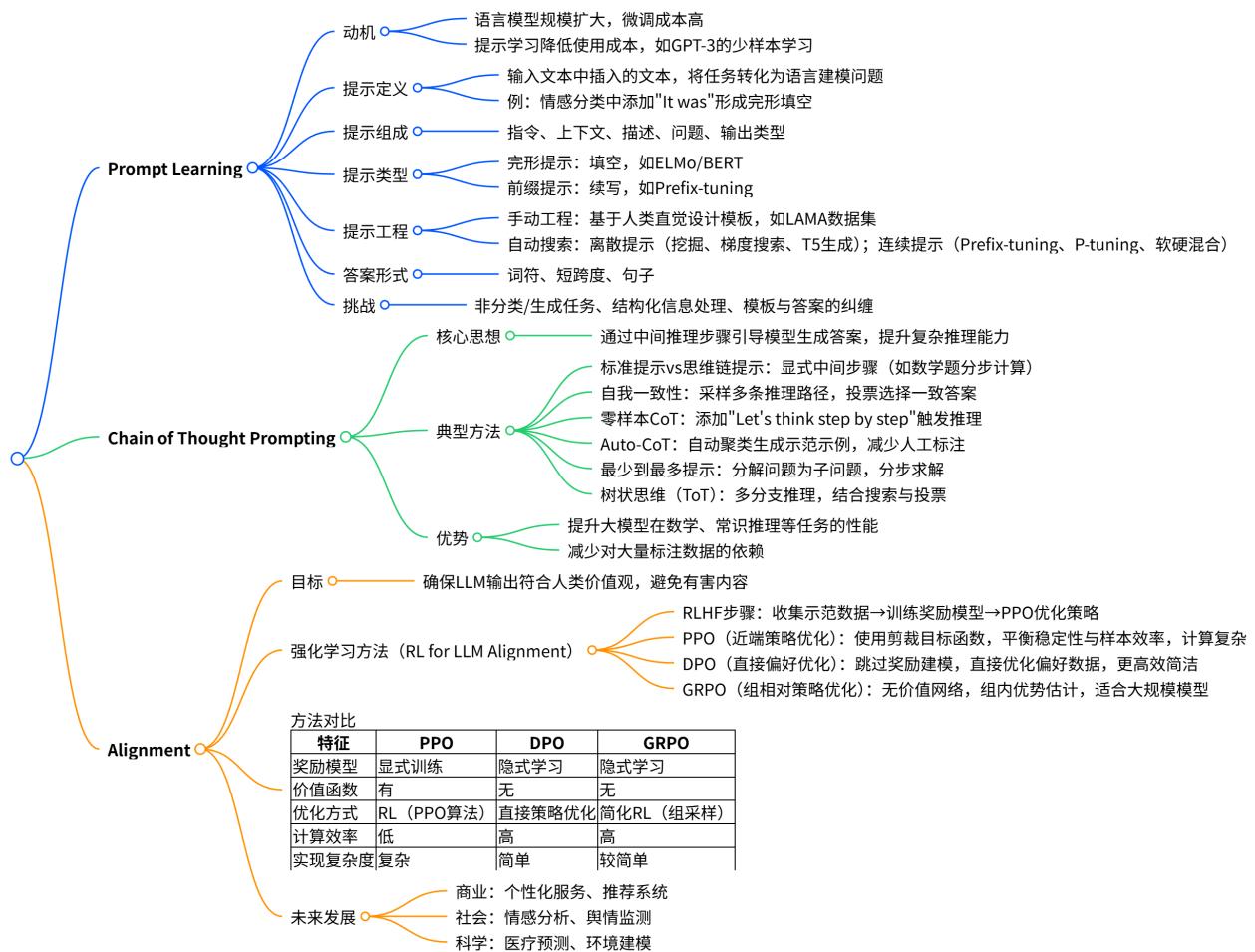
因此需要微调使得语言模型更好地对齐用户意图

例如通过收集多种（指令，输出）(instruction, output) 对来对语言模型进行微调，使其更好地理解和执行用户指令

具体地，通过人类反馈强化学习 Reinforcement Learning from Human Feedback (RLHF) 来微调

1. 收集示范数据，训练监督策略 Collect demonstration data, and train a supervised policy.
2. 收集比较数据，训练奖励模型 Collect comparison data, and train a reward model.
3. 基于奖励模型优化策略 Optimize a policy against the reward model using reinforcement learning.

Lecture 8: LLM Prompting and Alignment



豆包

你的AI助手，助力每日工作学习

LLM Prompting

使用prompt来进行低成本地调整模型，避免对LLM进行高成本的调整

提示（prompt）是插入输入示例中的一段文本，其作用是将原始任务转化为（掩码）语言建模问题

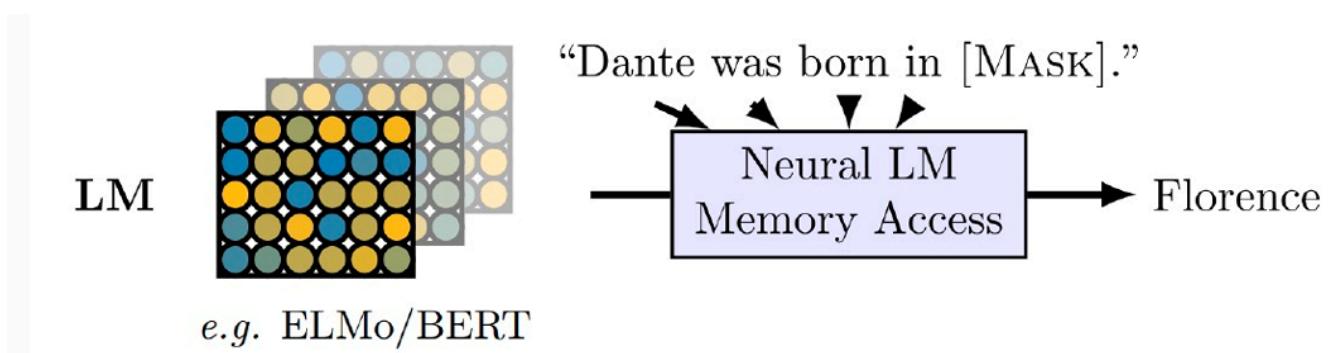
例如：“No reason to watch. It was .”，通过这个prompt，LLM完成情感分类任务，而不需要进行微调

prompt learning

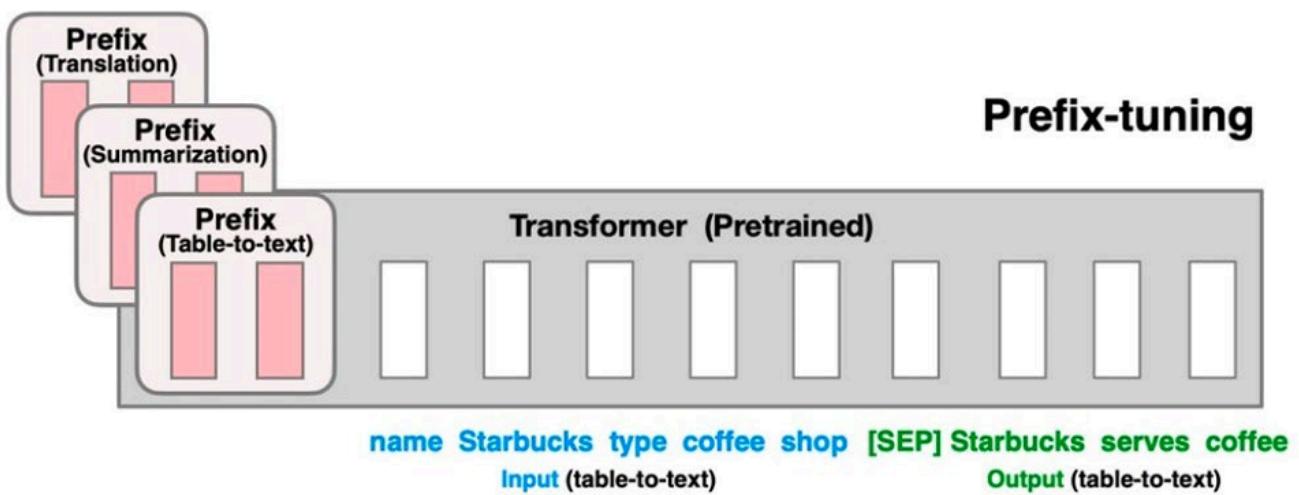
学习高效利用提示词

Prompt shapes

Cloze prompts (完形提示)：在文本字符串中填空



Prefix prompts (前缀提示)：延续一个字符串的前缀。通过给定的前缀引导模型生成符合任务需求的后续内容，例如"translate"前缀对应机器翻译任务，生成对应内容



Prompt engineering

如何合理引导LLM完成任务

手动提示工程 (Manual prompt engineering)：人类手工创造，但耗时大且难以发现最优解

自动化提示搜索 (Automated prompt searching)

自动完成提示工程，包含两种思路：

- **离散提示** (Discrete prompts, 又称硬提示 hard prompt)：在离散空间中描述的模板，通常对应自然语言短语
- **连续提示** (Continuous prompts, 又称软提示 soft prompt)：并非人类可理解的自然语言形式，直接在模型的嵌入空间 (embedding space) 中生成嵌入向量(embedding vector)，更灵活

离散提示 (Discrete prompts)

Mining - based approach: 提取主语subject 和宾语object 之间的中间词或短语作为提示

Prompt paraphrasing: 利用往返翻译round-trip translation（先将提示翻译成其他语言，再译回）来提高词汇多样性

Gradient - based search: 对实际 token 进行基于梯度的搜索，寻找能生成目标序列的prompt

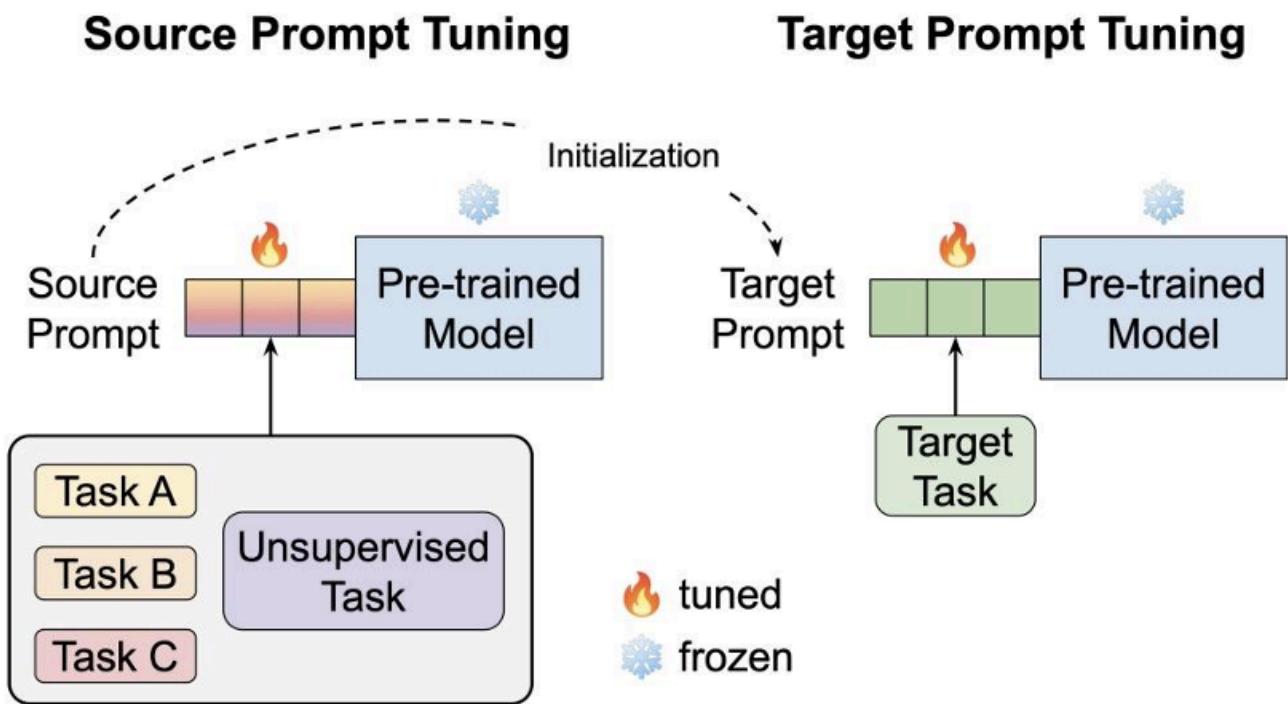
Prompt generation: 将 seq2seq引入搜索中，生成对应模板

连续提示 (Continuous prompts)

连续提示移除了自然语言形式限制以及预训练语言模型（LM）参数化的限制

Prefix tuning: 在输入前添加连续的任务特定向量序列，同时保持预训练语言模型的参数冻结，仅优化提示向量，从而高效适配不同任务

如图，在一个或多个源任务上学习一个通用的源提示，随后对目标提示进行调优，实现任务适配

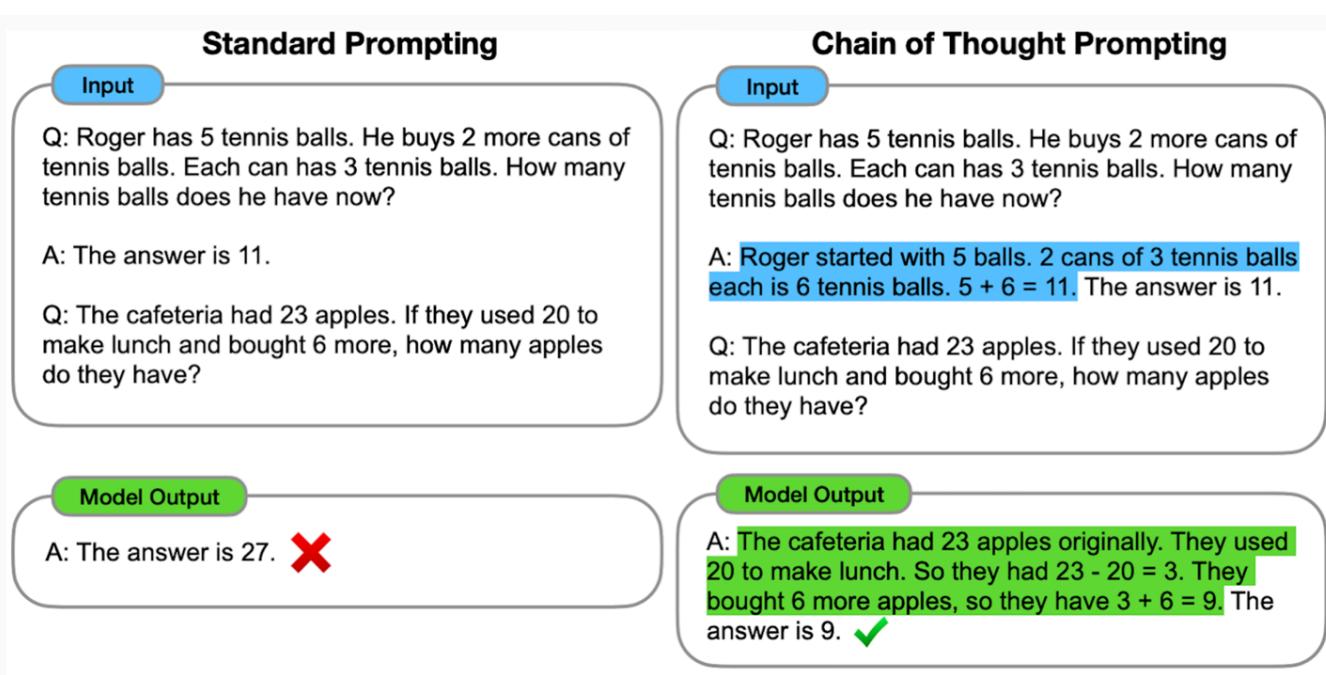


Tuning initialized with discrete prompts: 利用已通过离散提示搜索方法创建或发现的提示，初始化连续提示的搜索

Hard - soft prompt hybrid tuning: 在离散提示模板中引入可学习的连续向量（软提示）

Chain of Thought Prompting

通过在prompt中加入自然语言推理步骤natural language reasoning steps，提升回答准确性



自我一致性 (Self-consistency)

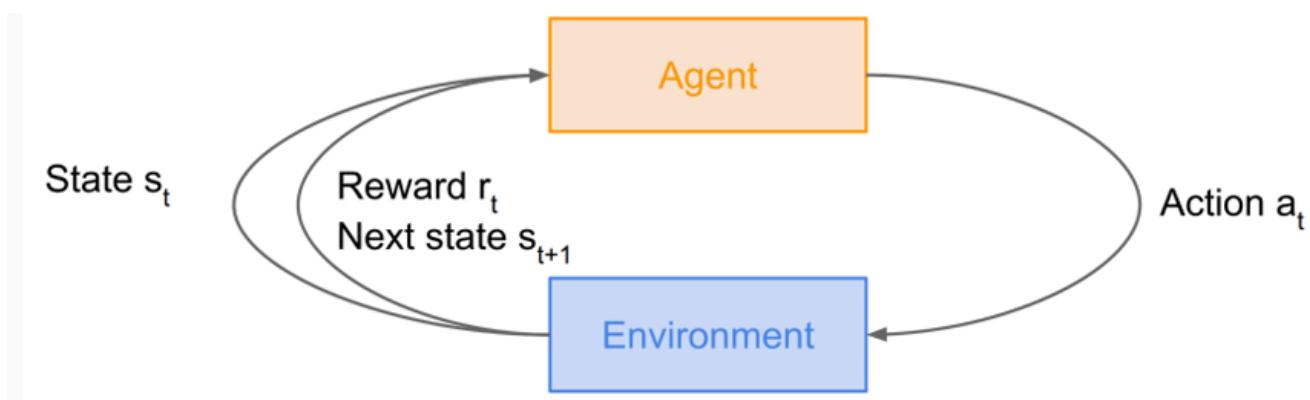
一种改进CoT的策略，通过生成多样化的推理路径并选择最一致的答案，提升语言模型在复杂推理任务中的准确性

传统CoT使用 **贪婪解码 Greedy Decoding**，只选择概率最高的输出，生成唯一一条推理路径

而**自我一致性 Self-consistency** 生成多个不同的推理链，从所有推理路径中提取最终答案，选择出现频率最高的答案

Alignment

通过RLHF改进LLM的语言理解和用户意图之间的对齐



Agent根据策略采取行动与环境交互，获得Reward，据此调整策略

我们的目标是找到最优策略 π^* ，最大化总奖励

在LLM中，即通过RL找到最优策略使得LLM可以最好地对齐用户意图

由于使用Q-Learning寻找最优策略太过复杂，考虑直接学习策略，从策略集合中寻找最优策略

这个过程涉及对策略参数进行梯度上升 Gradient ascent on policy parameters，即“**策略梯度 Policy Gradient**”

三种实现方法：

PPO (Proximal Policy Optimization, 近端策略优化)

DPO (Direct Preference Optimization, 直接偏好优化)

GRPO (Group Relative Policy Optimization, 组相对策略优化)

PPO

核心是 **剪切机制 (Clipping)**，避免策略更新过大导致训练不稳定

应用于RL中时，用于策略优化阶段

优点在于其有效性effectiveness

缺点在于高计算成本

DPO

直接从人类偏好数据（如成对比较）优化策略，**无需显式训练奖励模型**

相较于PPO简化了奖励模型训练的过程，效率更高

步骤：

1. **收集成对比较数据**：收集模型输出的成对比较 paired comparison，即针对同一输入获取人类标注的偏好 preferred与非偏好输出non-preferred
2. **直接优化模型 Directly optimize model**：使用基于偏好的监督损失函数直接优化大语言模型（LLM），无需显式构建奖励模型或进行强化学习

优点：简洁高效

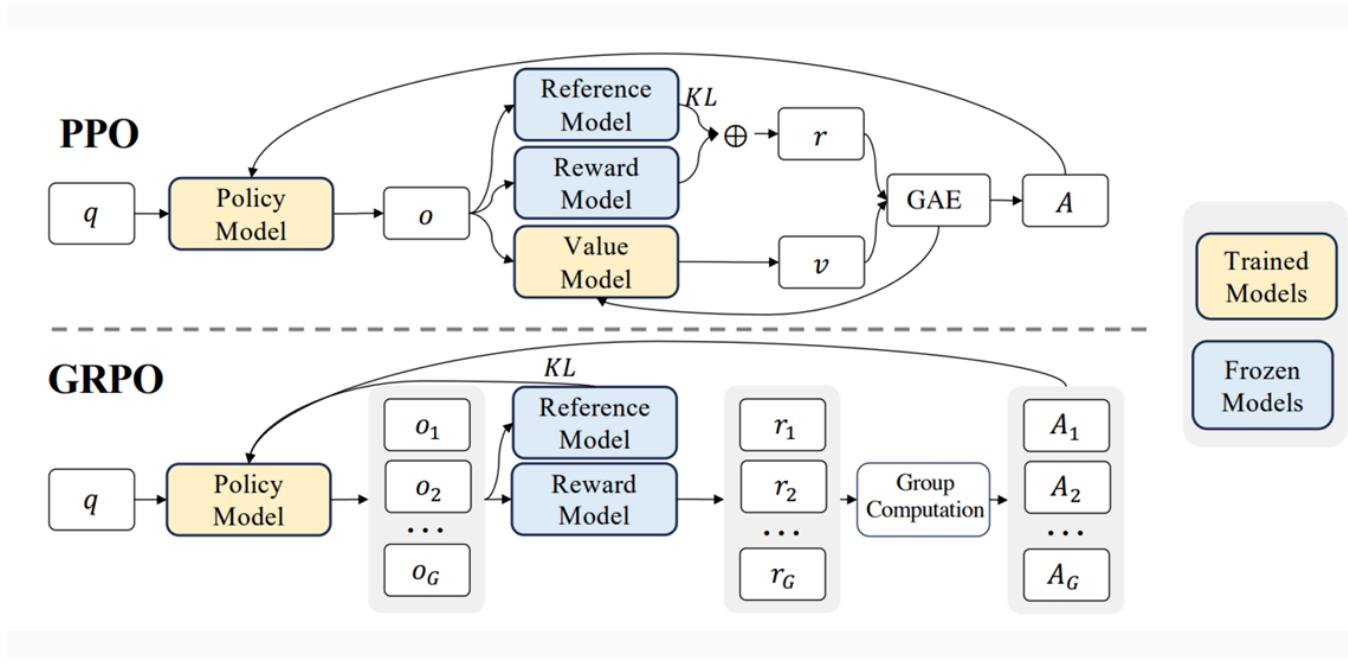
缺点：依赖高质量数据，泛化能力弱

GRPO

通过**组内相对比较** 优化策略，无需显式奖励模型或价值网络 critic network

GRPO在继承PPO的鲁棒性的同时避免了其计算开销

如下图，特点是是没有Value Model，以及包含Group-Computation



Lecture 9: LLM Agents

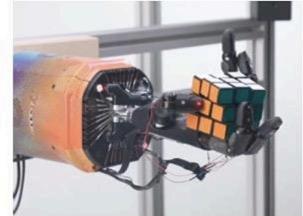


豆包
你的 AI 助手, 助力每日工作学习

Introduction

语言模型（Language Models）仍有许多不足，例如幻觉Hallucination等

考虑让LLM结合内部推理 / 规划 internal reasoning/planning 和外部工具 / 知识 external tools/knowledge 来扩展自身能力，提高鲁棒性

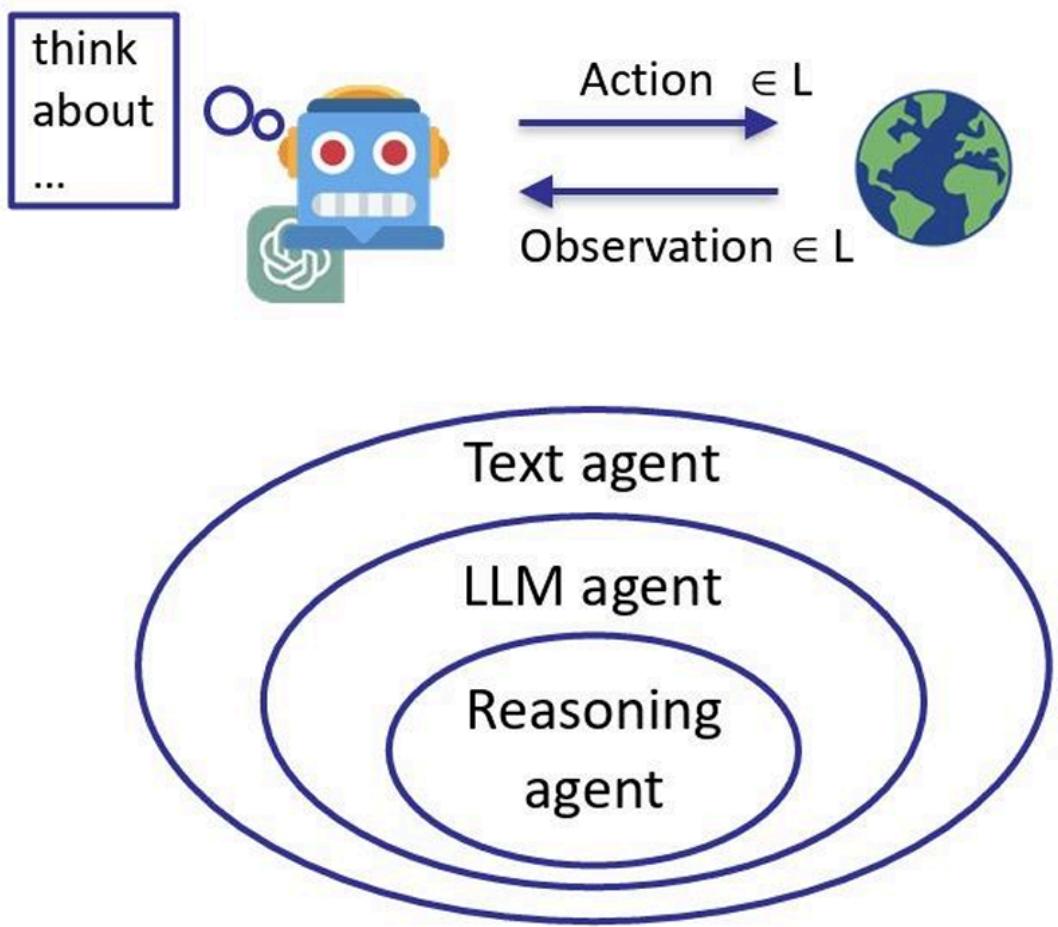


Agent: 与某种 "环境 environment" 交互的 “智能 intelligence”系统

环境不仅指物理环境，也包含数字环境，人类等

例如：物理环境——自动驾驶汽车，数字环境——Siri，人类环境——ChatGPT

LLM Agent:



Level 1: Text agent (文本代理): 使用文本进行动作和观察，例如ELIZA

Level 2: LLM agent (大语言模型代理): 直接利用大语言模型执行动作。例如：SayCan

Level 3: Reasoning agent (推理代理) 重点: 借助大语言模型进行推理，再基于推理结果执行动作。例如：ReAct

Reasoning

其中**CoT** (Chain of Thought) 对推理至关重要，其引导LLM进行思考推理以及调用外部工具

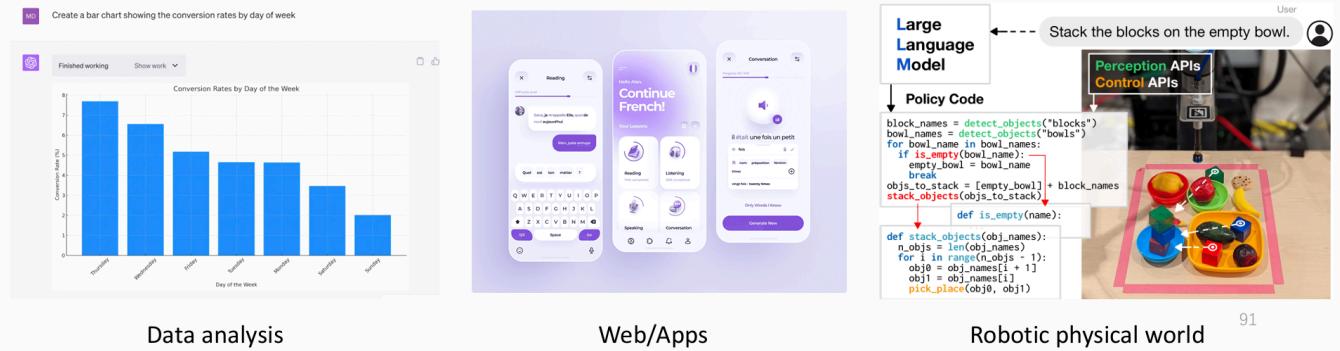
Tool

人类使用工具来改进自身局限性，LLM也是如此

通过使用工具，语言模型得以与**可执行动作**相关联 Ground language models into executable actions

通过将自然语言指令**映射为可在多种环境中执行的代码或动作**来使用工具 Mapping natural language instructions into code or actions executable within various environments

LM (planning and reasoning) + actions



输入：

- **语言 (Language)**: 用户的问题或请求
- **工具包 (Toolkit)**: 包含代码、搜索引擎 API、自定义函数、专家模型等
- **环境 (Environment)**: 涉及数据库、集成开发环境 (IDE)、网页 / 应用、视觉与机器人物理世界等

输出：

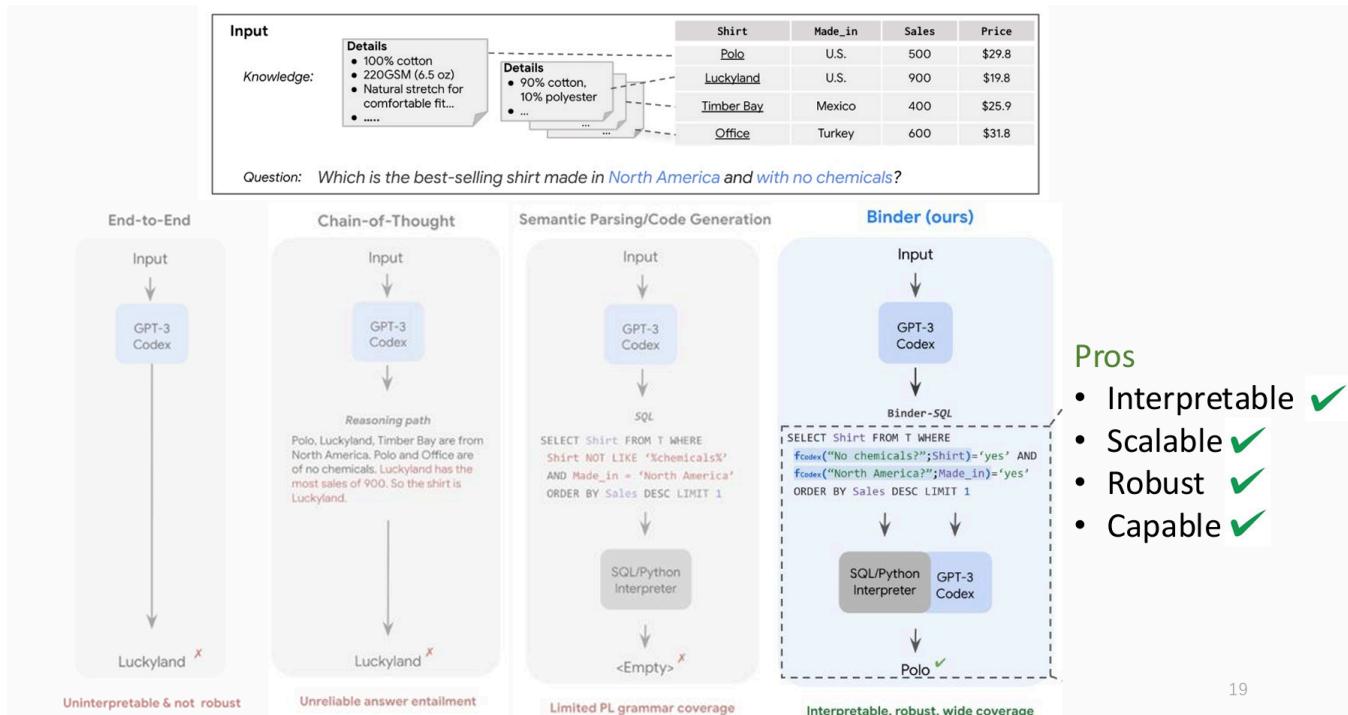
- **可执行的基础推理代码 / 动作序列**: 生成能在对应环境中执行的代码或动作
- **工具的选择与使用**: 明确选择哪些工具，以及何时、如何使用这些工具

LLM + code and NLP expert function APIs

例如：对一个框架Binder，其允许：

1. 将 GPT - 3 Codex 的 API 调用绑定到 SQL/Python 中
2. 通过 SQL/Python 解释器 + GPT - 3 Codex 执行以得出答案

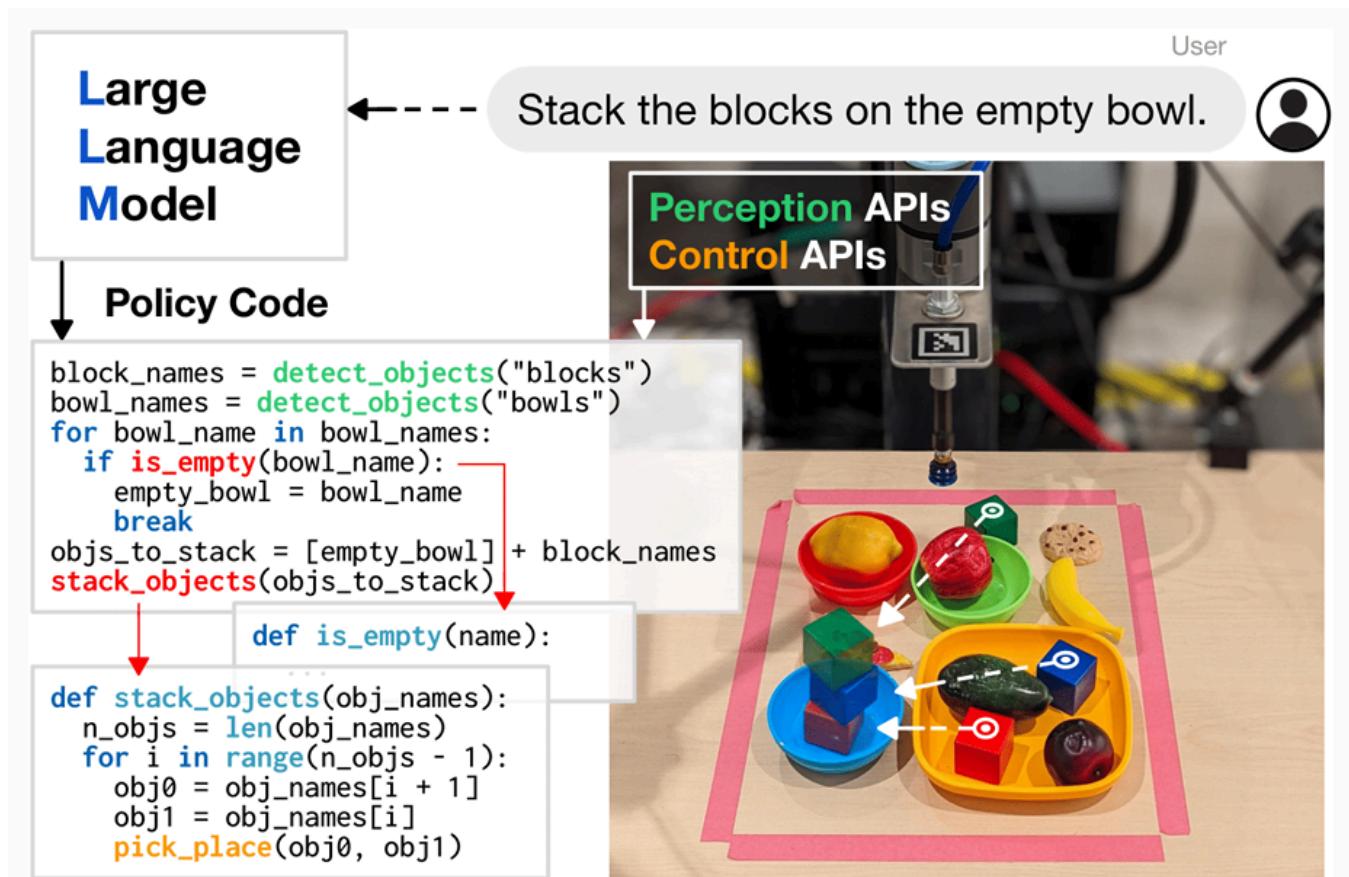
通过使用 Binder，我们可以有效解决问题：



其通过GPT - 3 Codex将自然语言问题解析成Binder程序和代码，再调用相应的API后获取返回的结果

LLM + code, robotic arm, expert models: Code as Policies

使用 LLM 根据自然语言指令编写机器人策略代码 policy codes

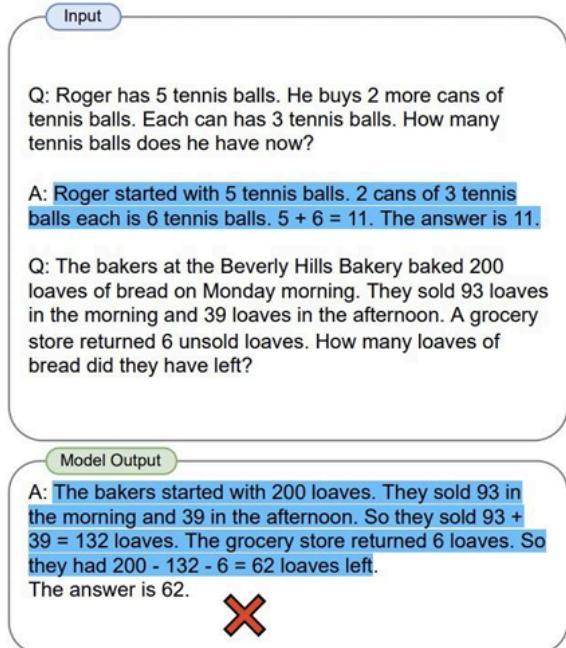


Tool Learning

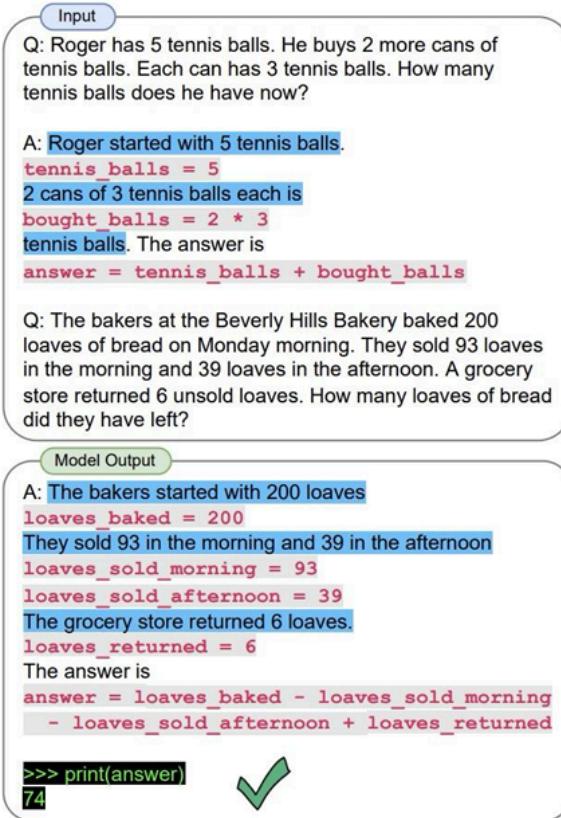
在CoT的基础上，通过在prompt中加入其余工具的使用可进一步加强推理准确度

例如下图加入编程

Chain-of-Thought (Wei et al., 2022)



Program-aided Language models (this work)



LLM finetuning/pretraining for tool use: TALM

迭代自玩技术 iterative “self-play” technique，逐步提升工具使用能力与任务处理性能。使用预定义的工具

Language Model

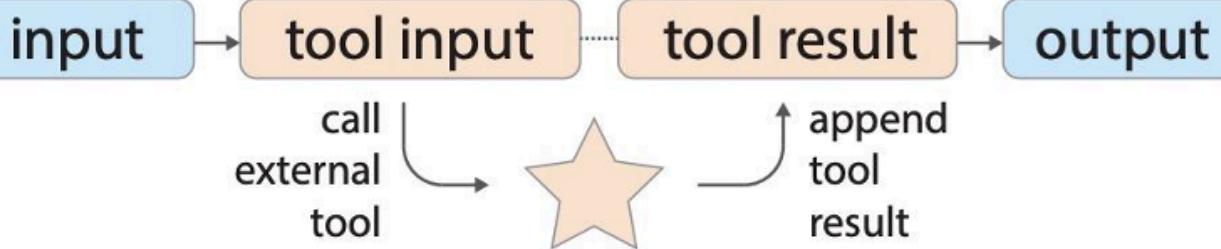


```

graph LR
    input1[input] --> output1[output]

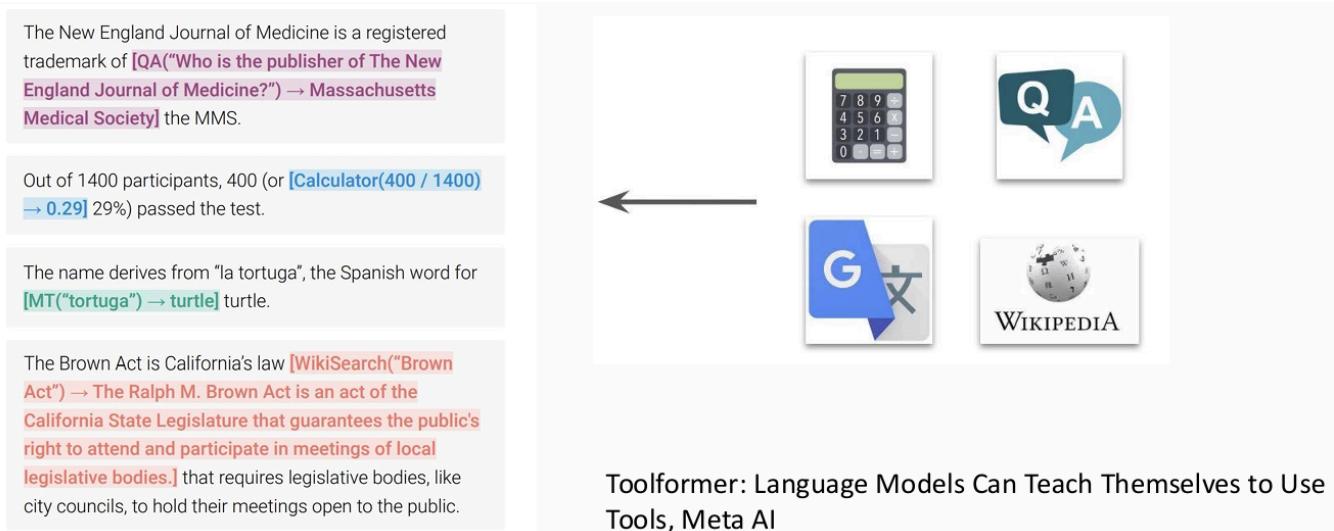
```

Tool Augmented Language Model



LLM finetuning/pretraining for tool use: Toolformer

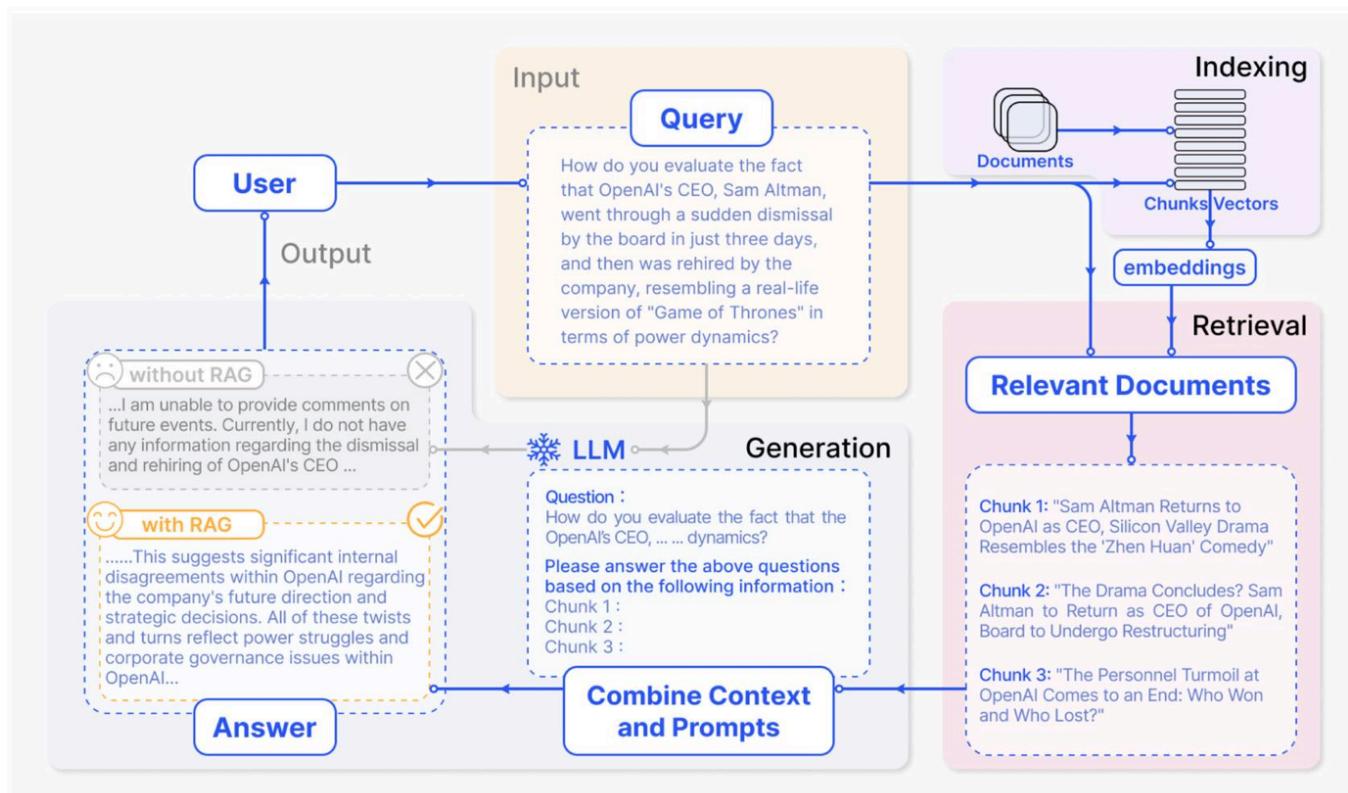
通过自监督方式 self-supervised 训练模型决定 API 调用（如选择工具、时机、参数等），并整合结果到预测中。自学生成API标记在文本中，外部调用工具执行，检索token



Other recent works

Incorporating external knowledge-RAG

使用检索增强生成（RAG）技术，整合外部知识以提升大语言模型（LLM）的回答准确性



向量数据库（Vector Database）：

- **嵌入（Embedding）：**将文档转换为向量表示。
- **索引（Indexing）：**对文档向量建立索引，便于快速检索

根据用户查询 **Query** 在数据库中检索 **Retrieval**, 找到相关文档

根据文档内容后处理 **Post-Process**

输入生成器 (**Generator**) 中, 利用LLM结合检索到的外部知识生成回答

Lecture 10: Efficient Training of LLMs



豆包

你的 AI 助手, 助力每日工作学习

Background

模型权重 (Model Weights)

模型在训练过程中学习到的参数, optimizer通过损失函数loss function 梯度 gradient对权重进行调整

例如transformer中, 自注意力 (self - attention)、位置前馈网络 (position - wise feed - forward networks) 和层归一化 (layer normalization) 中的参数都是权重

然而, 权重并非内存的主要占用, 以 16 位精度 16-bit precision 为例, 一个拥有 15 亿 (1.5B) 参数的 GPT - 2 模型, 其模型权重需占用 3GB 内存, 但即使使用32G内存的GPU也常常会out of memory, 因此除模型权重外, 训练过程中存在其他内存开销

混合精度训练 (mixed precision training)

在混合精度训练中，前向传播与反向传播forward and backward propagation 均采用 fp16 进行，借此发挥 fp16 计算速度快的优势，加速训练过程

但仍保存fp32的副本copy和fp32的其他优化器状态 **all the other optimizer states**，保留精度accuracy

优化器状态 (optimizer states)

优化器在训练过程中存储的内部数据，用于高效更新模型权重

随着模型规模增大，优化器状态所消耗的内存可能成为内存占用的主导因素

注：Adam优化器需为每个参数维护一个或两个状态

Memory consumption

假设模型具有 Ψ 个参数

在半精度 (fp16) 训练时，存储模型权重需 2Ψ 字节，存储对应梯度也需 2Ψ 字节

而在Adam优化器中：

- 参数的单精度 (fp32) 副本： 4Ψ 字节
- 动量 (momentum) 的 fp32 副本： 4Ψ 字节
- 方差 (variance) 的 fp32 副本： 4Ψ 字节

总共 16Ψ 字节

总结：weight+gradient+copy+momentum+variance

Activation (intermediate results)

激活值，在前向传播forward pass 过程中存储的中间结果，用于支持后续的反向传播backward pass计算

基于 Transformer 的模型，其激活内存与以下因素成正比：

$$\text{transformer layers} \times \text{hidden dimensions} \times \text{sequence length} \times \text{batch size}$$

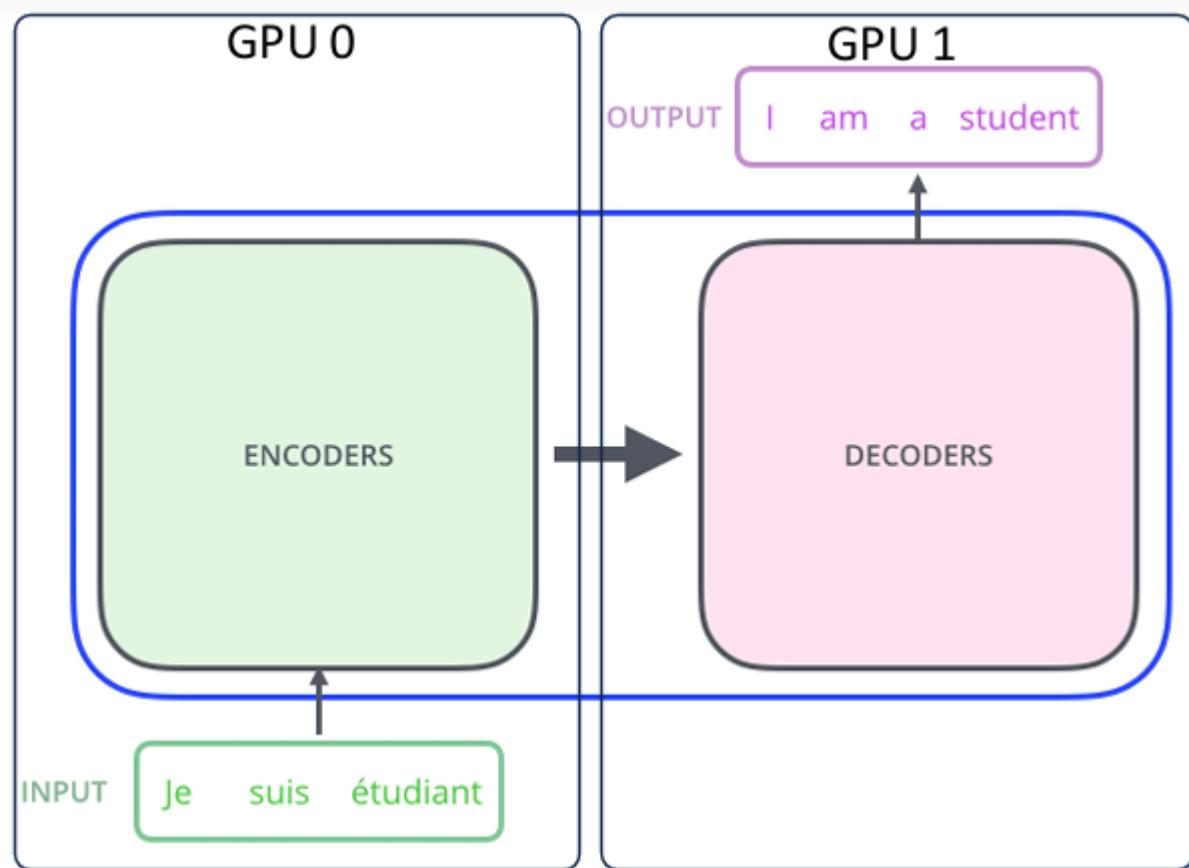
例如：对于具有 15 亿 (1.5B) 参数的 GPT - 2 模型，当序列长度为 1000 (1K)、批量大小为 32 时，训练所需的激活内存约为 60GB

Model Parallelism

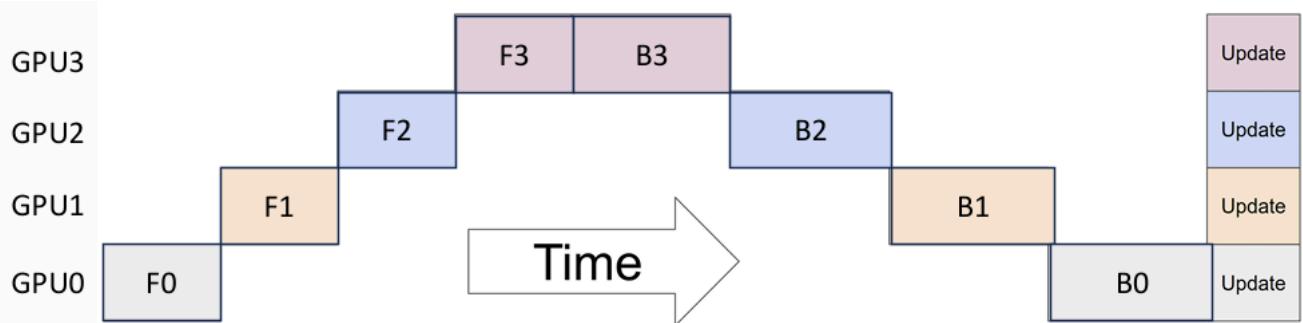
考虑模型并行以解决单设备内存或计算能力不足的问题

Naïve Model Parallelism

其核心是垂直划分 **vertical spreads**, 即将模型的层 layer 和组 group 分布到多个 GPU 上, 以实现并行处理
数据在 encoder 内部时正常流动, 但要去到解码器时引入通信开销



Simple Case:

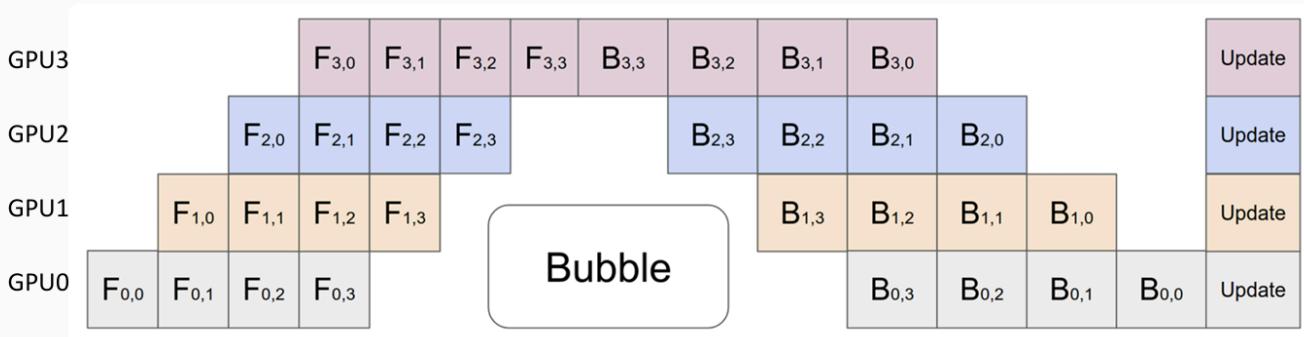


如上图模式

优点: 解决资源受限问题

缺点: GPU 出现空闲时间 (即“气泡”问题)

GPipe



将一批训练样本分割成更小的 **微批次 (micro - batches)**，然后在各个 **单元 (cells)** 上以流水线方式执行

优点：减少气泡

缺点：需实验确定微批次大小以高效利用GPU

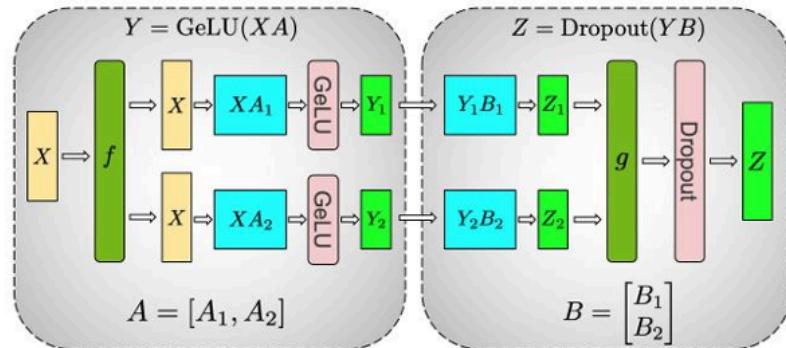
Tensor Parallelism

Tensor (张量)：深度学习框架中的核心数据结构，用于表示数据、模型参数和中间结果

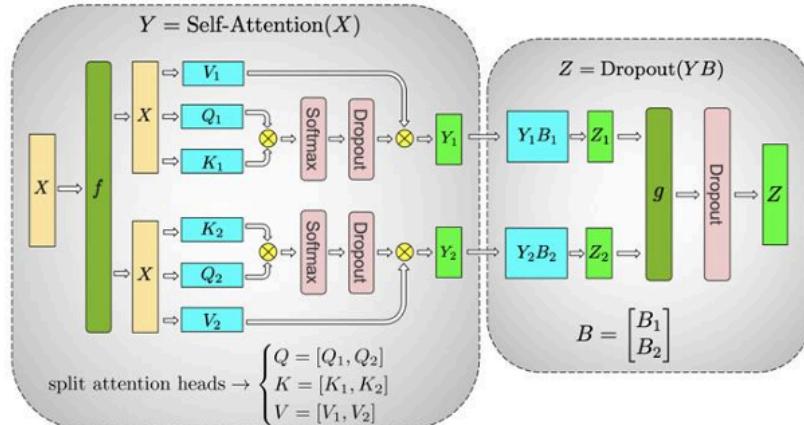
张量并行：每个 GPU 仅处理张量的一个切片 slice，仅在需要完整张量的操作（如聚合 aggregate）时才组合完整张量。也就是将张量相关运算给拆分了

通过按列column-wise或按行row-wise的方式对张量进行并行化parallelize处理

矩阵乘法 Matrix multiplication 可拆分为张量并行提供了理论依据



(a) MLP



(b) Self-Attention

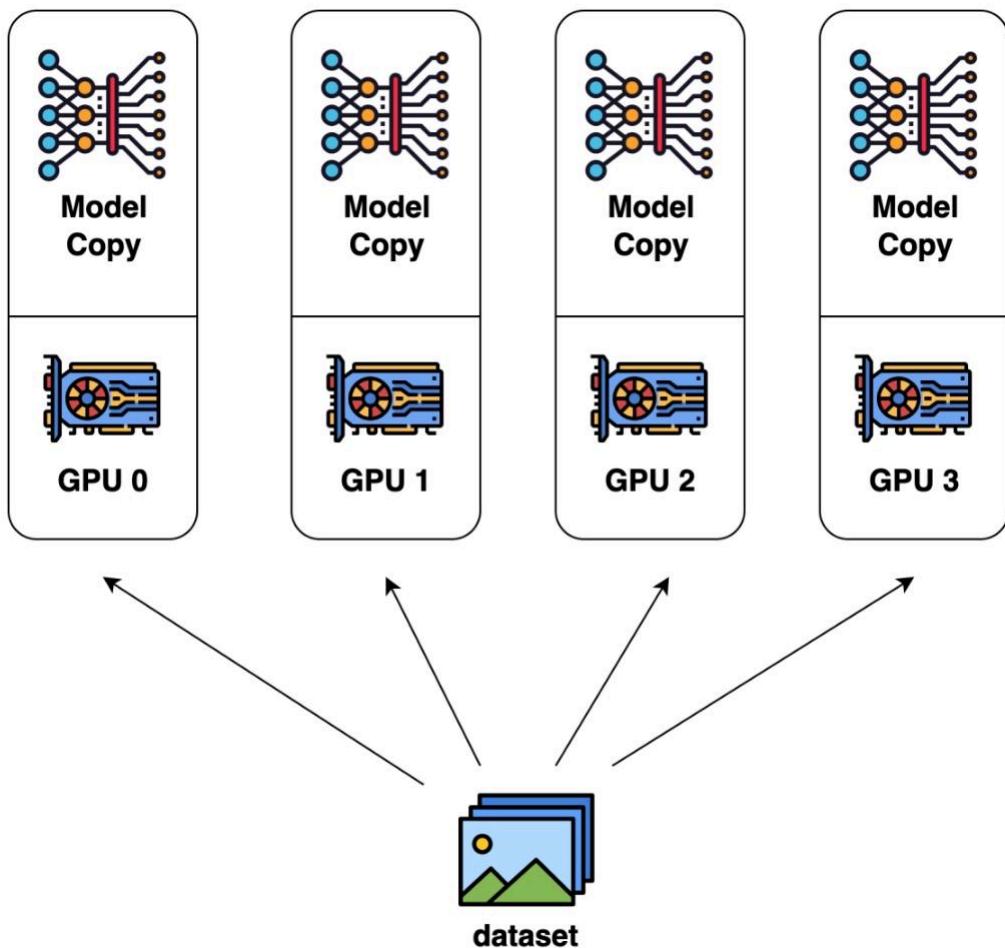
Data Parallelism

Naïve DP

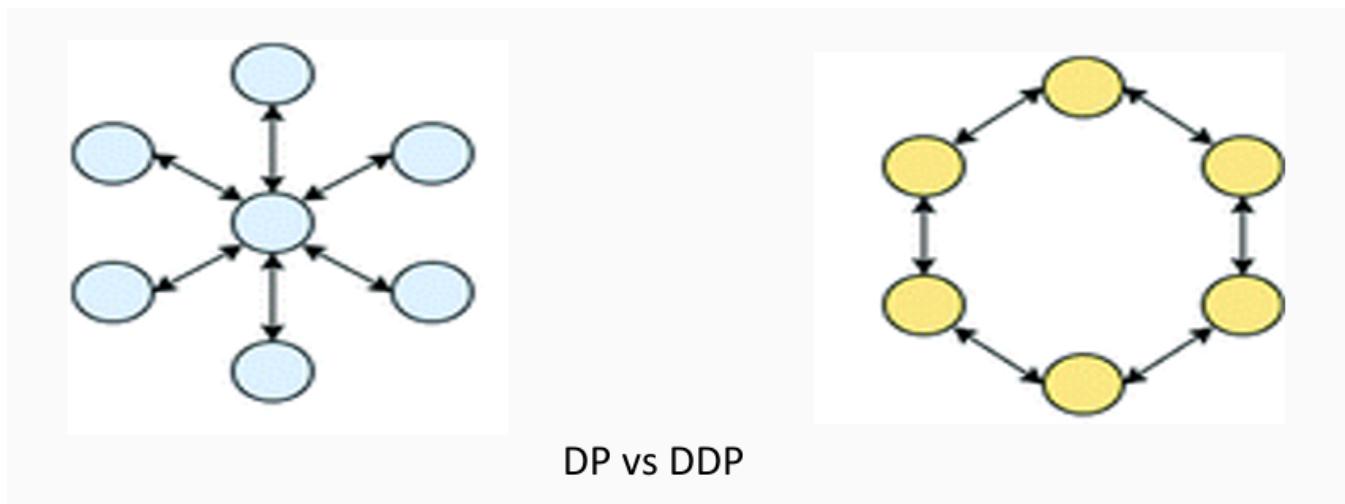
在数据并行 **data parallel training** 训练中，数据集被划分为多个分片，每个分片分配给一个设备

这一过程中，每个设备都会持有完整的模型副本（**Model Copy**），包含模型参数、梯度和优化器状态，并对分配到的数据集进行训练

因此DP无法缓解模型参数导致的OOM问题



Naïve DP vs DDP



Naïve DP是单进程single-process 多线程 multi-thread 模型，由于其基于线程，只支持单机多卡
而且最中心的GPU0既需要读取完整数据，又需要整合最终结果，负载大，容易出现OOM。而其他GPU则利用率低

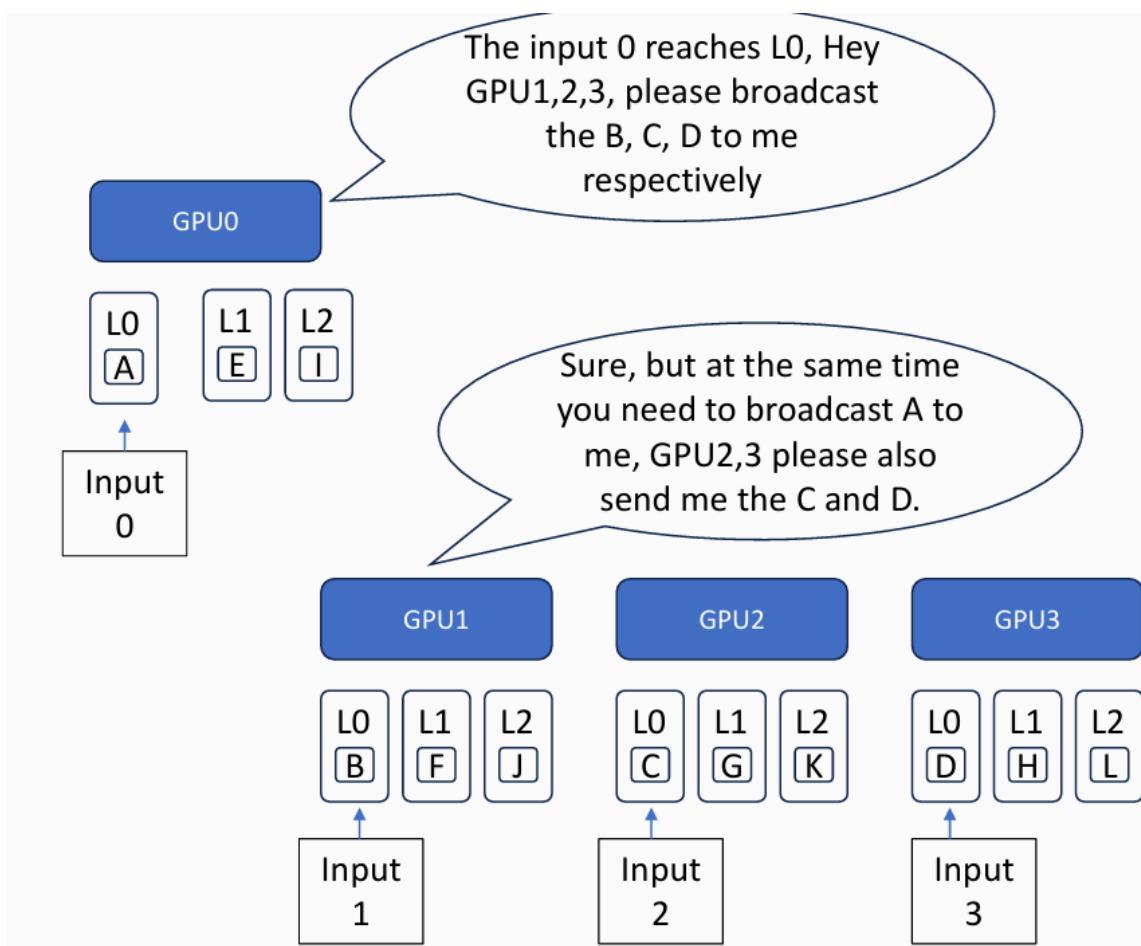
而**DDP Distributed Data Parallel**基于多进程，支持单机和多机训练,更灵活
每个GPU只完成自己的任务，去中心化decentralize，避免OOM

DeepSpeed ZeRO

针对数据并行中的完整模型导致的OOM问题，用DeepSpeed ZeRO优化

与传统数据并行（DP）中每个 GPU 复制完整的模型参数、梯度和优化器状态不同，DeepSpeed ZeRO 让每个 GPU 仅存储这些内容的一个切片

在运行过程中，当某一层需要完整的参数时，所有 GPU 会进行 **同步synchronize**，互相提供彼此缺失的部分
而输入数据遵循DP模式，分割后输入不同GPU



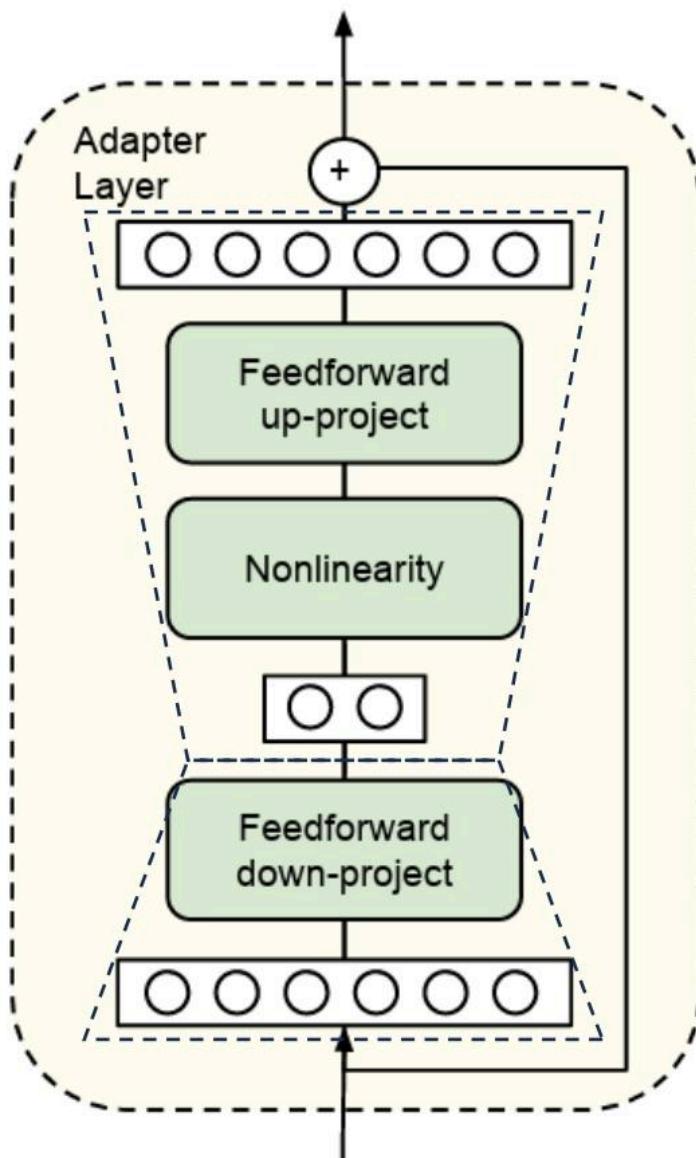
使用完同步获得的参数后，将其删除以节约内存

Parameter-Efficient Fine-Tuning

PEFT是一种在大规模预训练模型（如大语言模型、视觉模型）中，仅通过调整少量参数即可适应下游任务的技术
核心是冻结预训练模型的大部分参数，仅在模型中添加少量可学习的模块或参数

Adapter

基于适配器的方法在冻结的预训练模型的注意力层和全连接层之后，添加额外的可训练参数 **extra trainable parameters**。这种方式减少了内存使用并加速了训练，因为预训练模型的大部分参数保持不变，仅适配器部分更新
适配器通常体积很小，但性能可与完全微调的模型相媲美。它使得在资源有限的情况下，仍能有效训练更大规模的模型



核心是瓶颈Bottleneck结构

但adapter会引入推理阶段的时延latency

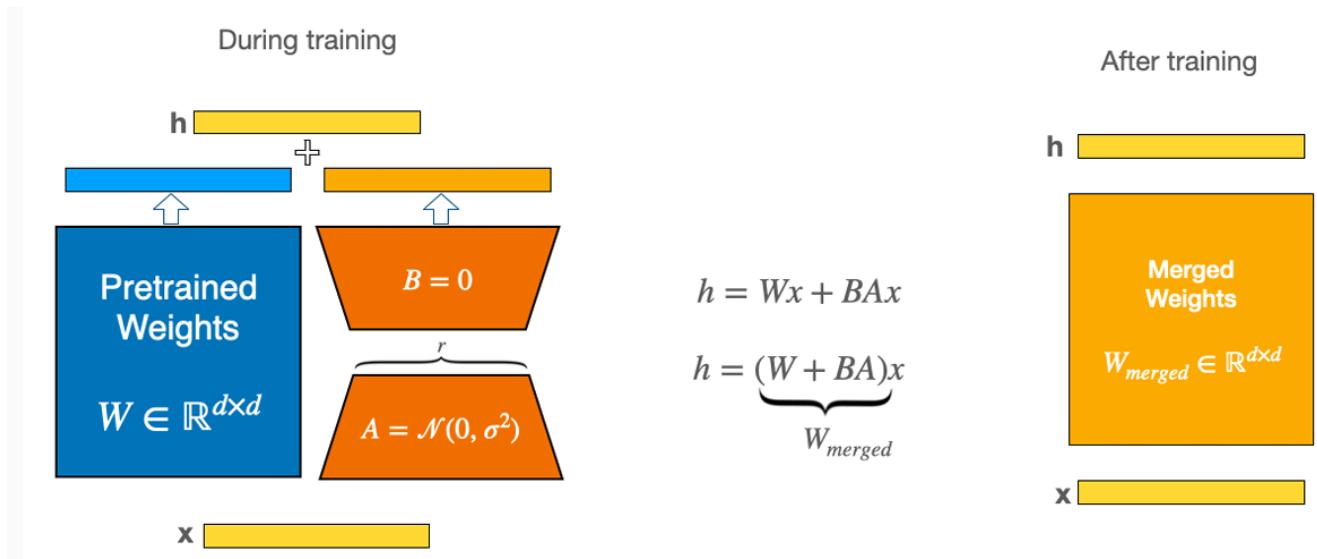
LoRA

Low - Rank Adaptation

考虑一个 $d \times d$ 大小的预训练权重矩阵 \mathbf{W}_0 ，在训练过程中保持不变。若直接更新 $d \times d$ 大小的权重 $\Delta\mathbf{W}$ ，会消耗大量内存和时间

采用顺序adapter训练会增加推理延迟

而LoRA引入了新参数 r ($r \ll d$) 以减小矩阵规模，定义两个更小的矩阵 \mathbf{A} ($r \times d$) 和 \mathbf{B} ($d \times r$)，形成“瓶颈 Bottleneck”结构



与adapter不同，LoRA以并行方式将可训练参数添加到预训练权重中，因此不引入推理时延

Lecture 11:RAG

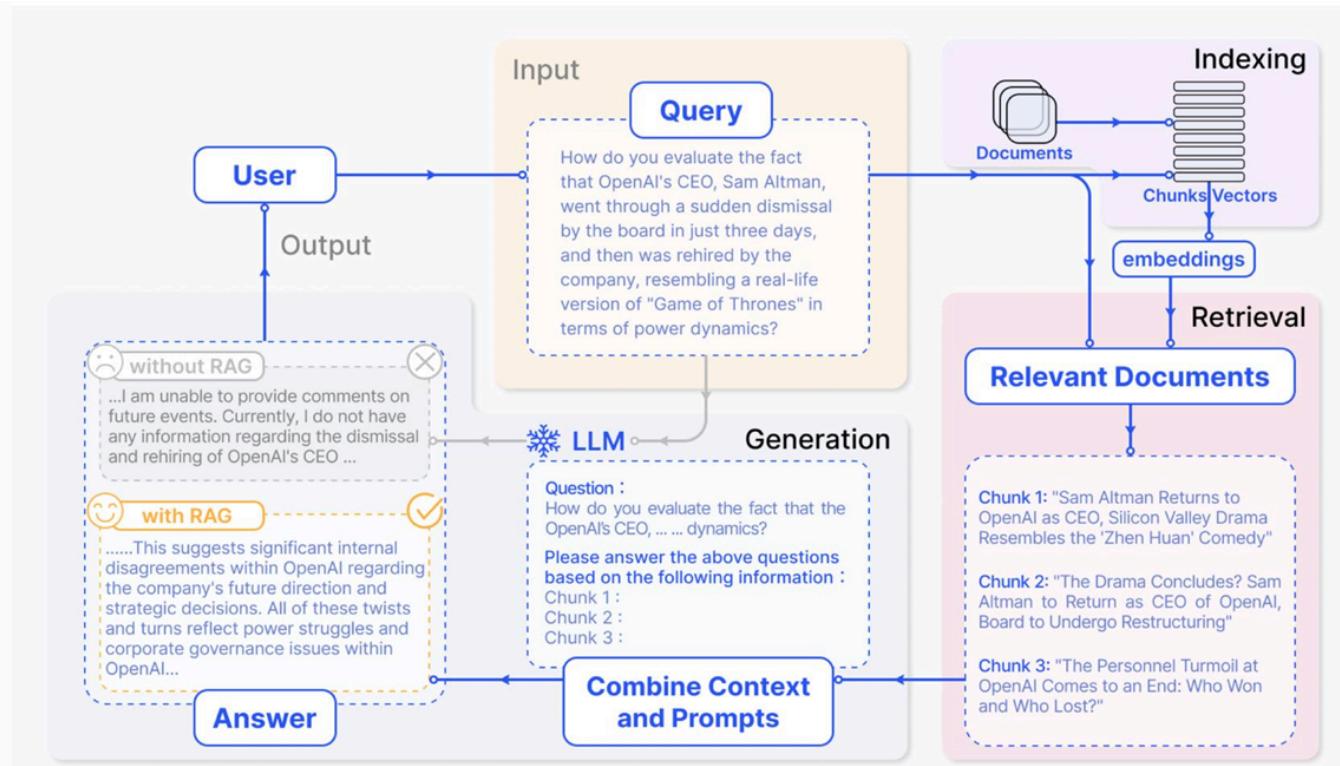


豆包

你的 AI 助手，助力每日工作学习

What's RAG

RAG 是一种通过集成外部知识库来增强LLM的范式paradigm，结合了信息检索机制 information retrieval mechanisms 与生成技术 generation techniques



构建向量数据库 (Vector Database):

- **嵌入 (Embedding)**: 将文档转换为向量表示。
- **索引 (Indexing)**: 对文档向量建立索引，便于快速检索

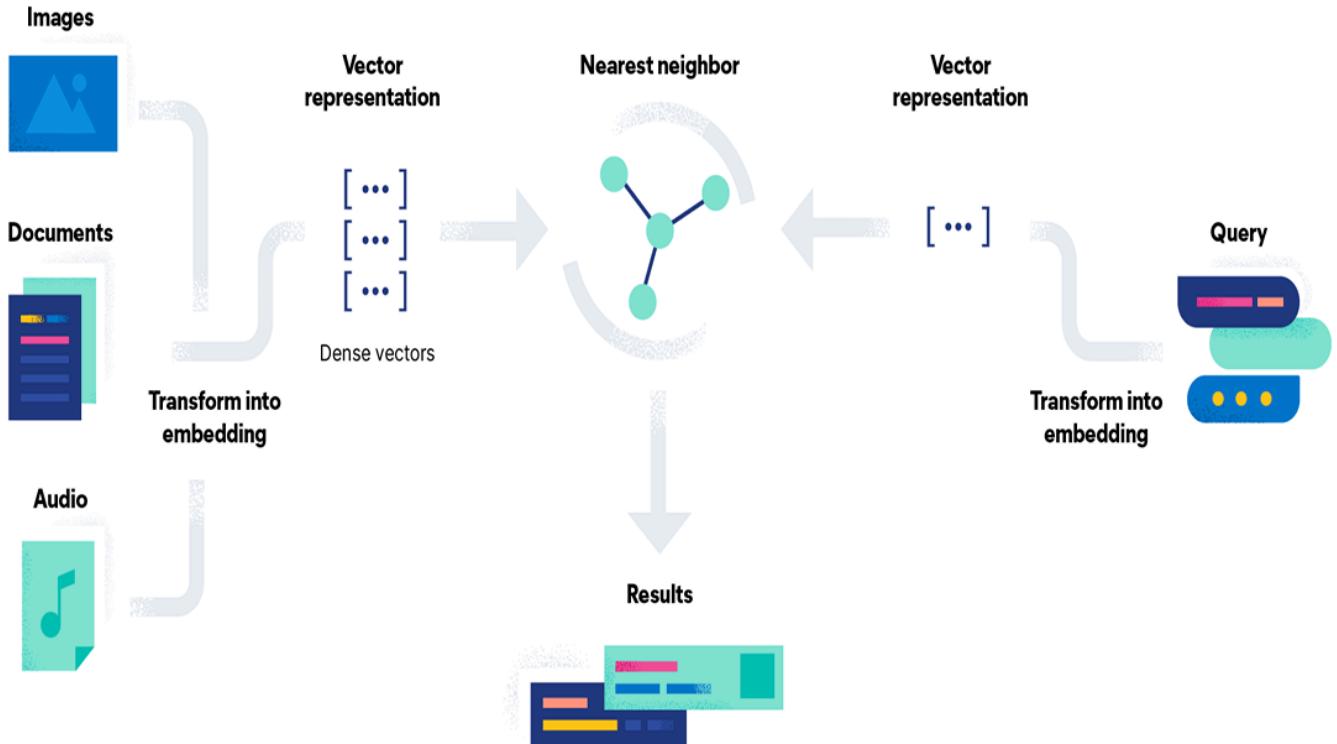
根据用户查询 **Query** 在数据库中检索 **Retrieval**，找到相关文档

根据文档内容后处理 **Post-Process**

输入生成器 (Generator) 中，利用LLM结合检索到的外部知识生成回答

Vector Database

将信息存储为向量（即数据对象的数值表示，也称为**向量嵌入 vector embeddings**）的数据库

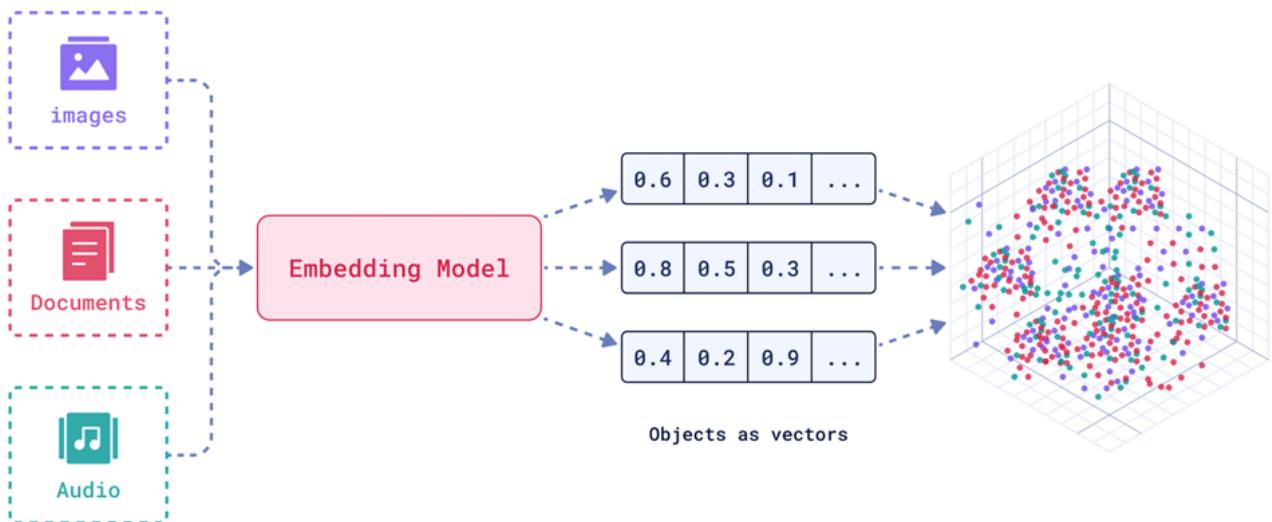
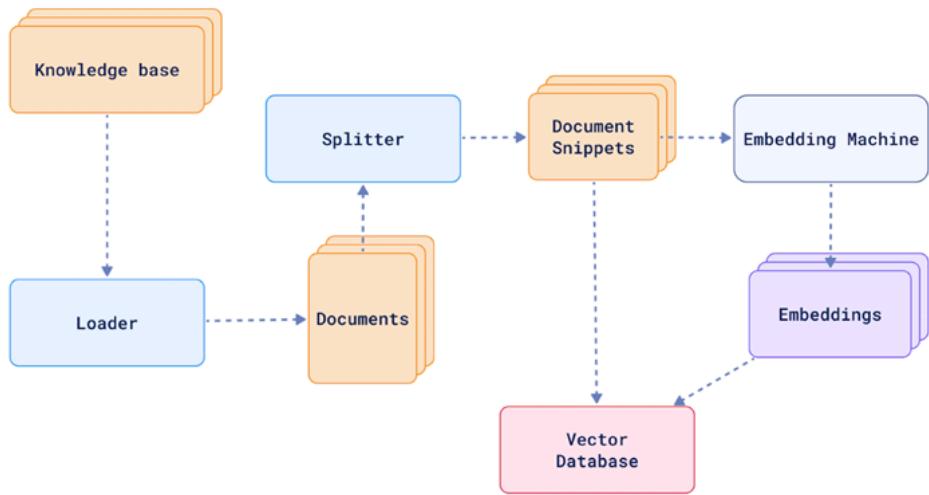


Embedding

包含三个步骤：

1. 加载文档：使用**加载器 (loader)** 收集包含数据的文档
2. 分割文档：通过**分割器 (splitter)** 将文档划分为更小的文本块 chunks（如句子或段落），称为**文档片段 (document snippets)**
3. 生成向量嵌入：将每个文本块输入到**嵌入机器 (embedding machine)**，如 TF - IDF、BM25、BERT、GPT 等。嵌入机器通过复杂算法将文本转换为**向量嵌入 (vector embeddings)**

Indexing



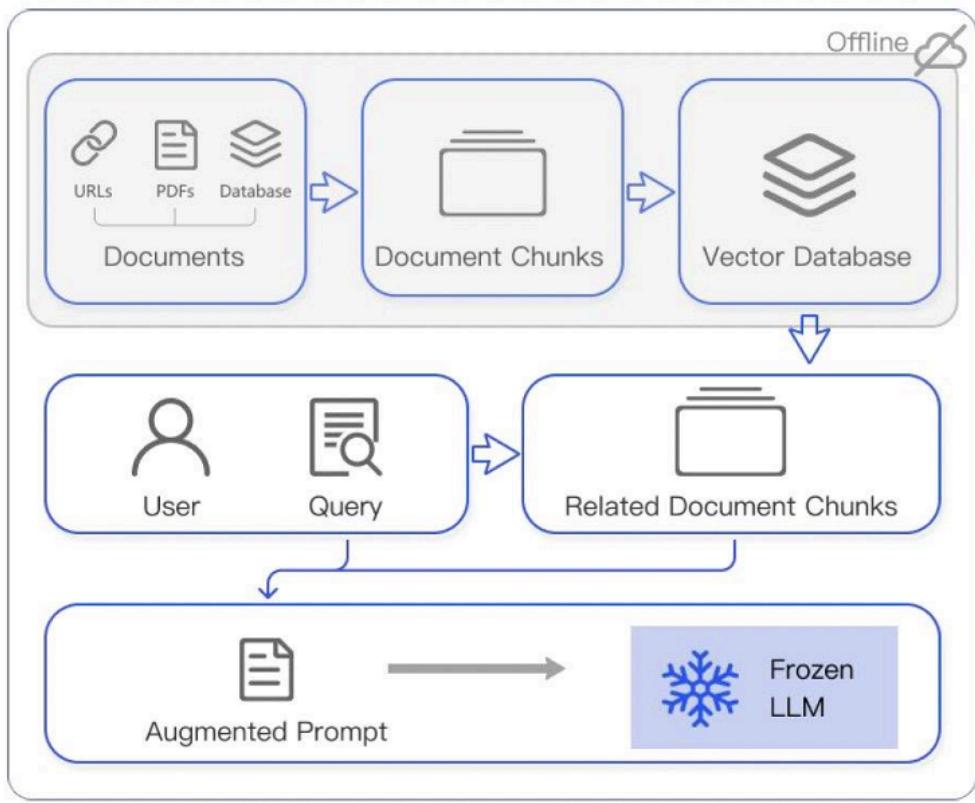
Indexing

为数据库建立索引，以便高效检索

- 哈希 (Hashing)
- 量化 (Quantization)
- 基于图 (Graph - based)

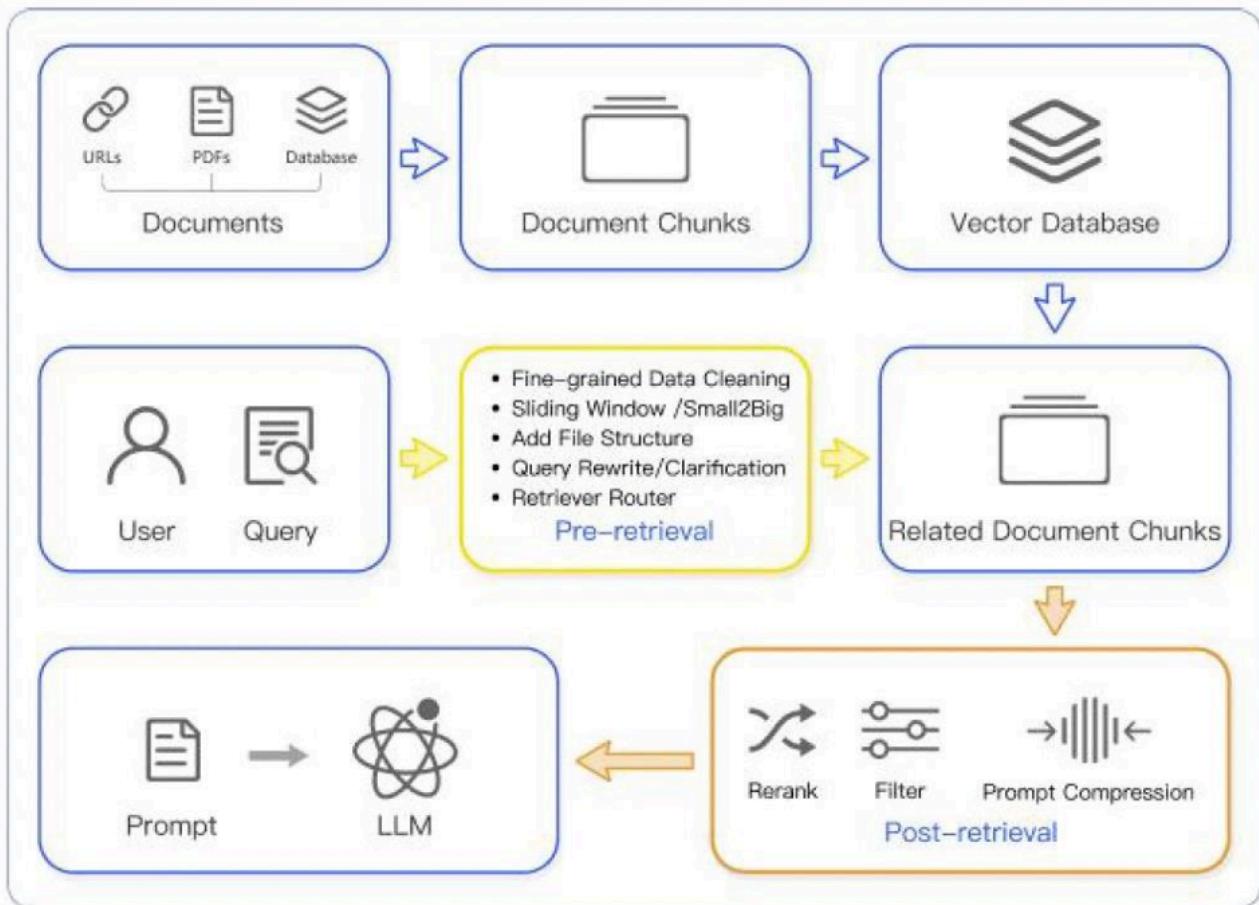
Naïve RAG

最基础的RAG，先索引 Indexing，再检索 Retrieval，最后生成 Generation



Advanced RAG

更高级的RAG，其核心步骤为：索引优化 → 预检索处理 → 检索 → 检索后处理 → 生成



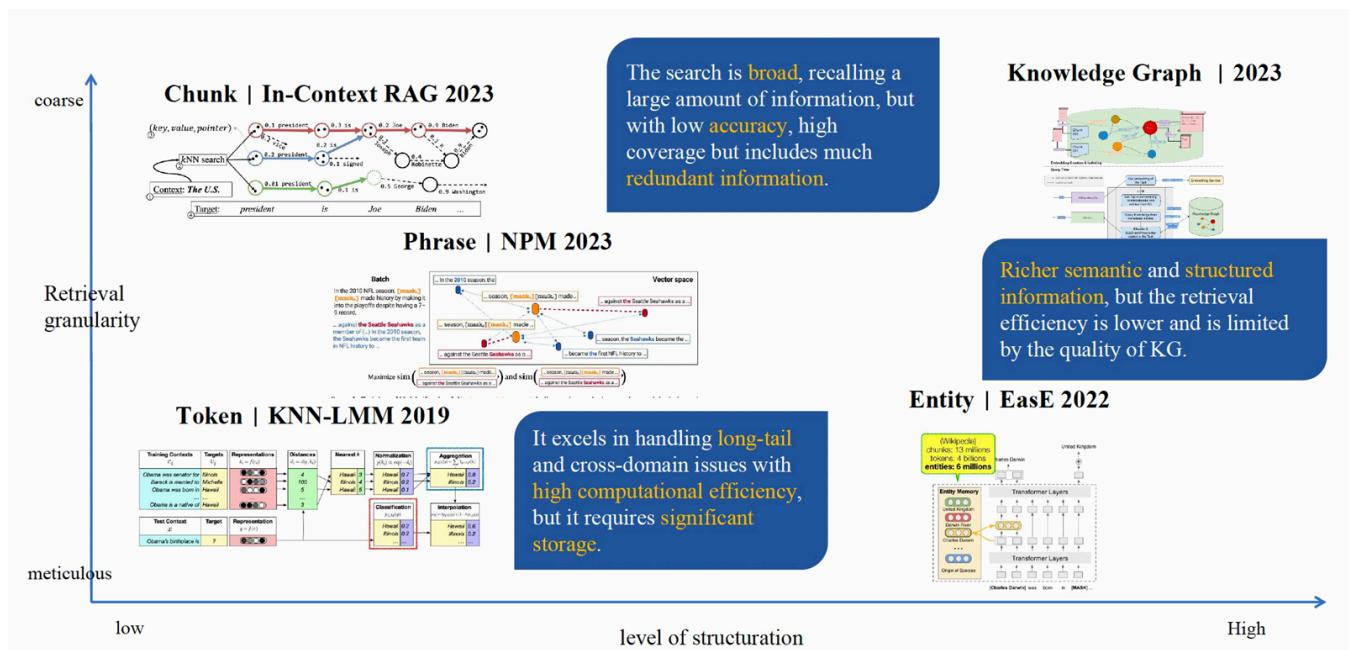
其改进在于：

- 索引优化 (Index Optimization): 通过滑动窗口 sliding window等方式优化数据索引过程
- 预检索处理 (Pre - Retrieval Process): 在正式检索前, 对用户查询进行处理
- 检索后处理 (Post - Retrieval Process): 对检索到的内容进行筛选与优化

What / When / How to retrieve

三大RAG核心问题

What



越细 **meticulous**则越高效计算，但占用空间越高

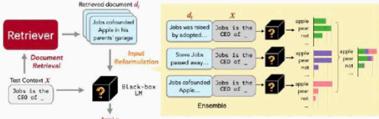
越粗 **coarse**则召回信息量越大，但准确率低冗余多

结构化程度越高**high**则语义和结构化信息越多

When

High efficiency, but low relevance of the retrieved documents

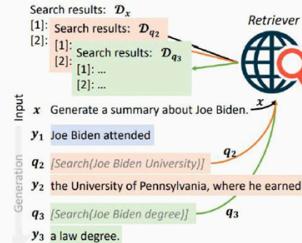
Once | Replug 2023



Conducting once search during the reasoning process.

Balancing efficiency and information might not yield the optimal solution

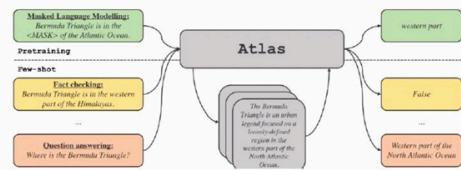
Adaptive | Flare 2023



Adaptively conduct the search.

A large amount of information with low efficiency and redundant information.

Every N Tokens | Atlas 2023



Retrieve once for every N tokens generated.

Low

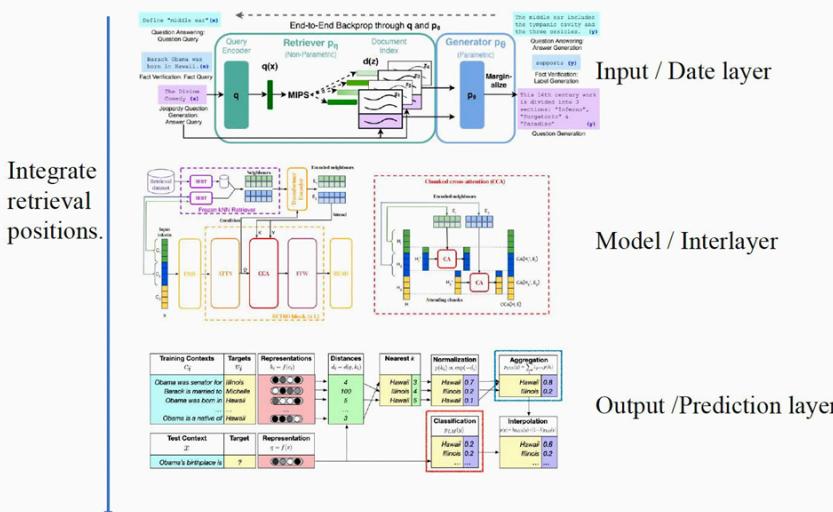
Retrieval frequency

High

检索频率 frequency 越低 low 则效率越高，但检索到的相关性 relevance 越低

检索频率越高 high 则效率越低，相关性高但可能冗余 redundant

How



Integrate retrieval positions.

Using simple, but unable to support the retrieval of more knowledge blocks, and the optimization space is limited.

Supports the retrieval of more knowledge blocks, but introduces additional complexity and must be trained.

Ensuring the output results are highly relevant to the retrieval content, but the efficiency is low.

在推理过程中如何将检索到的信息整合到生成模型的不同层次

Input/Data layer (输入 / 数据层): 将检索信息和原始查询组合输入。简单但不支持更多知识块knowledge blocks的检索（模型输入长度限制），优化空间 optimization space 有限

Model/Interlayer (模型 / 中间层): 在模型中间层融入检索信息。支持检索更多知识块但引入额外训练复杂性

Output/Prediction layer (输出 / 预测层): 在生成输出后，依据检索信息进行调整。确保输出结果与检索内容高度相关highly relevant，但需额外处理步骤，效率较低

Example Questions

Recently, the Hong Kong government has launched a system to let patients self-report their COVID-19 test results. However, people need to fill out forms to report their COVID-19 cases. Now we would like to design a task-oriented dialogue system (a Chatbot) to interact with humans such that people can report their COVID-19 test results automatically to the Chatbot. We term this Chatbot the COVID-19 Reporting roBot (CRB).

1.1 To design such a task-oriented dialogue system, we need to first perform natural language understanding tasks, including domain classification, intent detection, and slot filling. Explain in this CRB system, what is the domain classification problem, what is the intent detection problem, and what is the slot filling problem. You can design 5 reasonable slots to indicate this CRB example. [9 marks]

1.2 What are the next-step tasks/modules after the natural language understanding tasks to finish this closed-loop system? Explain how these next-step tasks/modules work using this CRB example. [9 marks]

1.3 Can you provide at least two metrics to evaluate this CRB system performance? Explain their meanings. [4 marks]

A:

1.1:

Domain Classification: 将用户输入文本归类到特定领域

Intent Detection: 识别用户的具体意图

Slot Filling: 提取关键信息并结构化存储

1. **Test_Result:** 阳性或阴性结果。
2. **Date_of_Test:** 测试日期。
3. **Location_of_Test:** 测试地点。
4. **Symptoms:** 症状描述。
5. **Contact_Info:** 联系方式 (电话/邮箱)

1.2:

Dialogue State Tracking: 跟踪对话状态，了解当前信息和首要任务

Dialogue Management: 根据对话状态做出决策，决定下一步采取什么行动

Natural Language Generation: 生成对用户问题的回答

1.3:

准确率 (Accuracy): 系统正确处理用户请求的比例

用户满意度 (User Satisfaction): 评估用户体验

2.1 What is the distributional hypothesis in word representations? What is the key problem that the contextualized word representation (like ELMo) solves, compared with context-independent word representations (like word2vec or GloVe)? [4 marks]

2.2 If we select the skip-gram model as our training strategy on a larger corpus, and then start the training process. However, we found that the training speed is very slow. Why does this problem occur and how to improve the efficiency of training? Give at least one of these possible solutions and explain it. [6 marks]

2.3 After obtaining a well-trained language model, we incorporate it with a multi-layer recurrent neural network (RNN) to address the named entity recognition task. A major challenge of the target dataset is that the text length of different cases in the dataset varies greatly, ranging from several words to hundreds of words. Is this network a good choice for the task? Give your answer and explanation. If not a good choice, how to improve the model? [5 marks]

2.1:

Distributional Hypothesis: 词的意义由其出现的上下文决定

Problem: Process polysemous words

2.2

原因：词汇表vocabulary过大

使用hierarchical softmax, it turns the vocabulary into tree-shape, to accelerate training efficiency.

2.3

No, it's not because when dealing with entity recognition task, the text length is quite long and there might be gradient problems occur due to long dependencies, A good way is to use attention mechanism.

Read the following program code snippet to define a sentiment classification model in PyTorch. Does this code snippet work for the sentiment classification task? If not, please clarify the bugs. [5 marks]

```
import torch.nn as nn
from transformers import BertModel

class SentimentClassificationBERT(nn.Module):
    def __init__(self, bert_version, num_class):
        super().__init__()
        self.bert = BertModel.from_pretrained(bert_version)
        hidden_size = self.bert.config.hidden_size
        self.linear_decoder = nn.Linear(hidden_size, num_class)

    def forward(self, input_ids, attention_mask):
        sequence_output, pooled_output = self.bert(input_ids=input_ids,
                                                    attention_mask=attention_mask)
        logits = self.linear_decoder(sequence_output)
        return logits
```

```
import torch.nn as nn
from transformers import BertModel

class SentimentClassificationBERT(nn.Module):
    def __init__(self, bert_version, num_class):
        super().__init__()
        self.bert = BertModel.from_pretrained(bert_version)
        hidden_size = self.bert.config.hidden_size
        self.linear_decoder = nn.Linear(hidden_size, num_class)

    def forward(self, input_ids, attention_mask):
        sequence_output, pooled_output = self.bert(input_ids=input_ids,
                                                    attention_mask=attention_mask)
        logits = self.linear_decoder(sequence_output)
        return logits
```

应使用 `pooled_output` 作为 `linear_decoder` 的输入，因为线性层期望的是二维输入 `(batch_size, hidden_size)`，而 `sequence_output` 的形状是 `(batch_size, sequence_length, hidden_size)`

```
def forward(self, input_ids, attention_mask):
    _, pooled_output = self.bert(input_ids=input_ids, attention_mask=attention_mask)
    logits = self.linear_decoder(pooled_output)
    return logits
```