

大模型RAG学习-从线性回归到RAG

前置学习

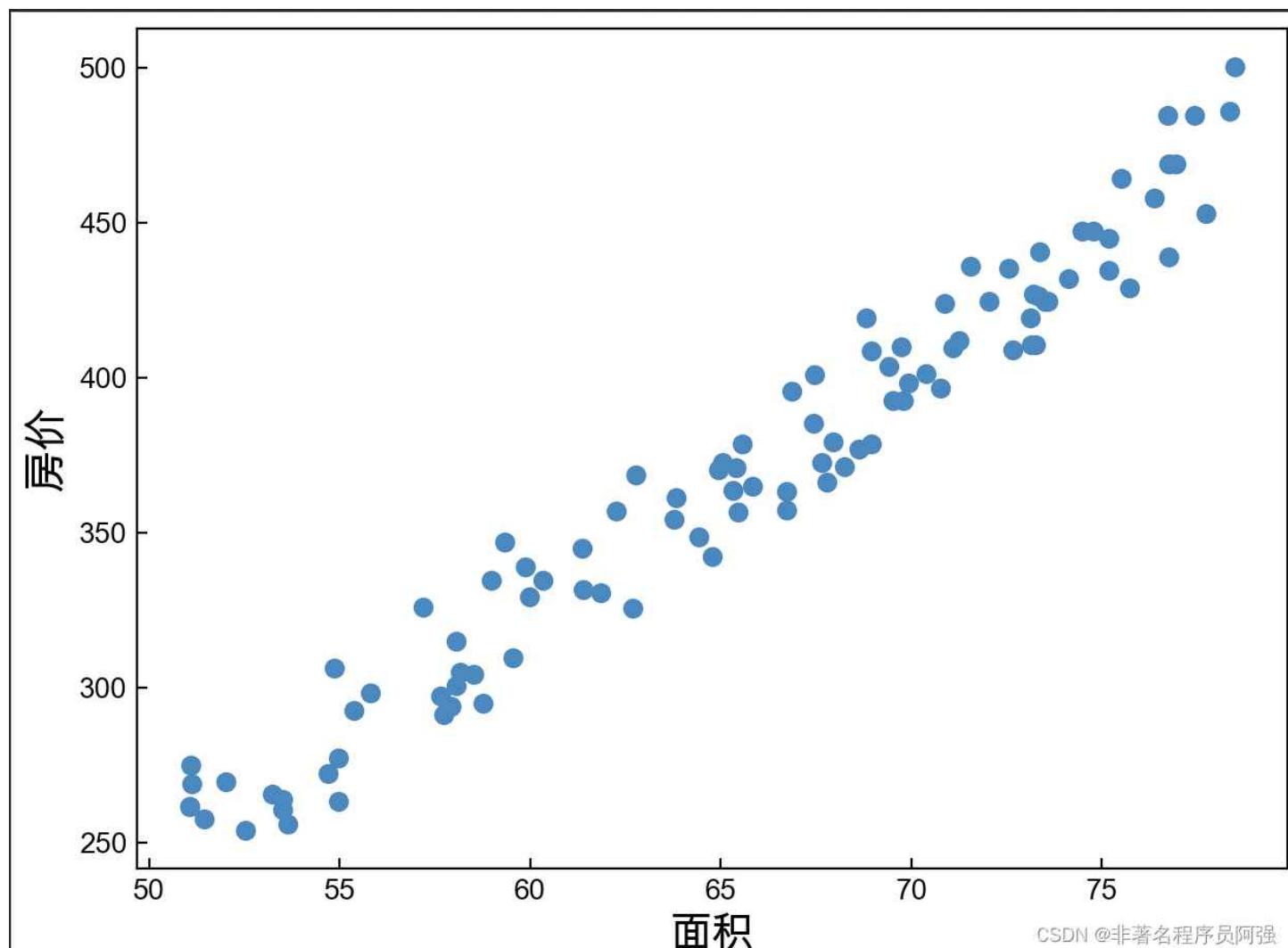
神经网络和全连接层

线性回归

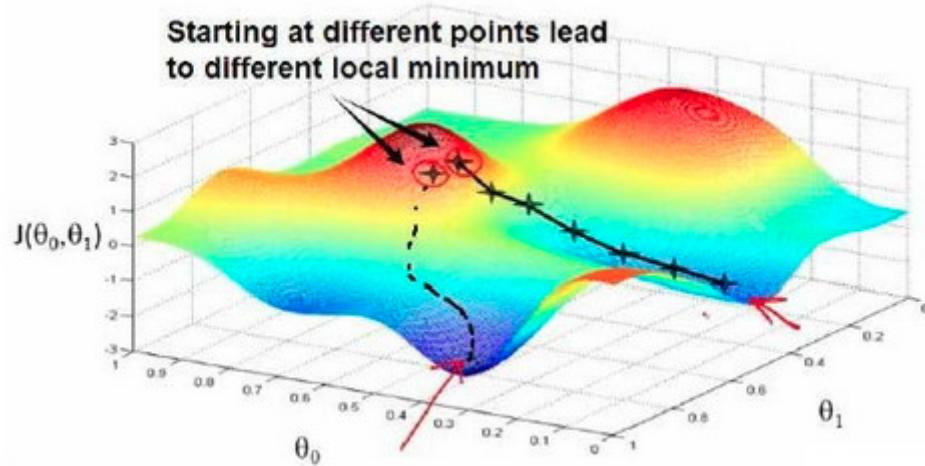
对房价和房屋面积进行数据的统计，并绘制图，可以发现，两者存在一定的关联关系，我们需要一个数学表达式来说明他们的关联性，先假设他们是一维线性的，即呈现一条直线， $\text{房价}(Y) = W * \text{X(面积)} + B$

这里W和B是未知的，我们可以调整他们的值来拟合曲线达到最适配的结果

拟合的过程就是一个学习修正的过程，我们把这些点的面积当做输入，房价当做输出，最开始不知道W和B，默认为0



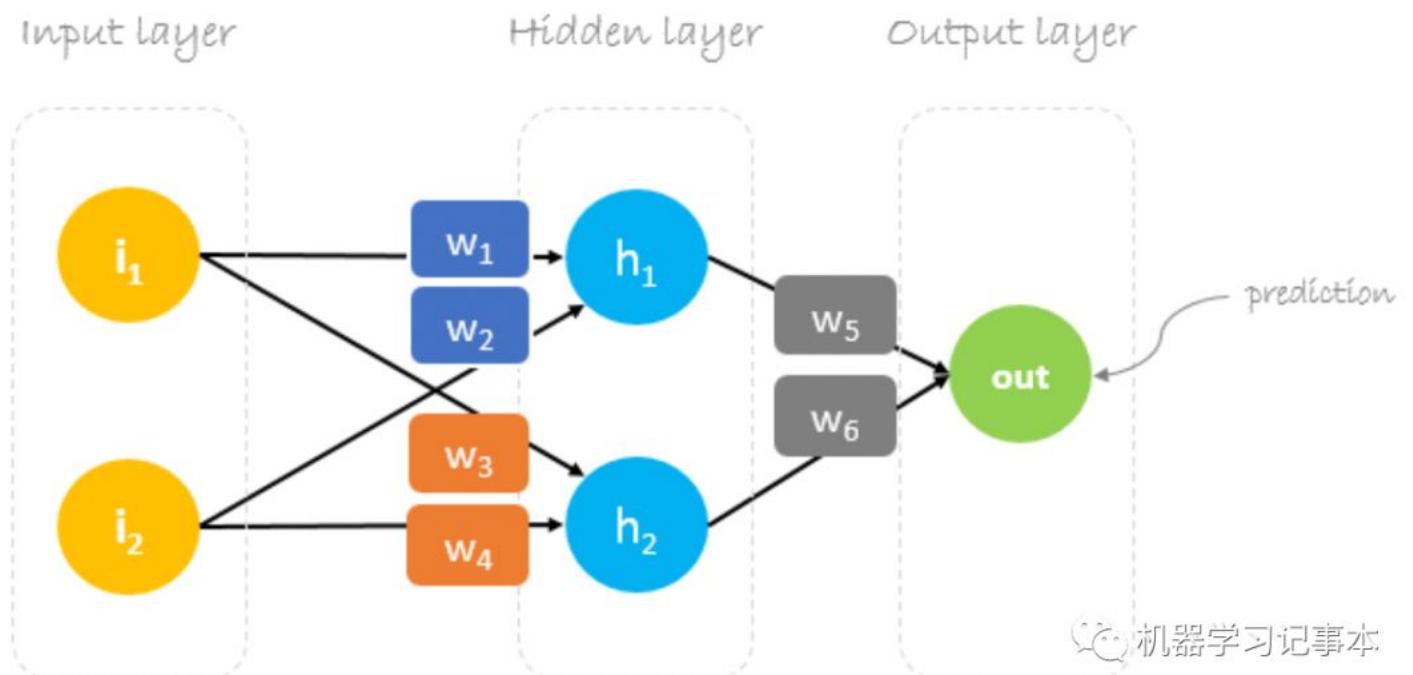
定义损失函数来衡量估计值和真实值之间的差异: $f = W^*X + B - Y$, 把W,B看做变量, 损失函数就是关于这两个值的一个二维函数, 对W和B求偏导数, 得到(W',B')就是梯度方向, 也是f变换最快的方向, 所以 $W = W - a * W'$, $B = B - a * B'$, 带入x,y的真实值尝试性地去移动, 最终会使得误差最小(a太大会错过底部, 但是太小训练时间太长)



同时, 为了更精准预测, 可能建模时会采用多项式形式, 比如 $y = w_1*x + w_2 *x(2次方) + \dots + w_n*x(n次方)$, 损失函数和求偏导的方式还是不变

神经网络结构

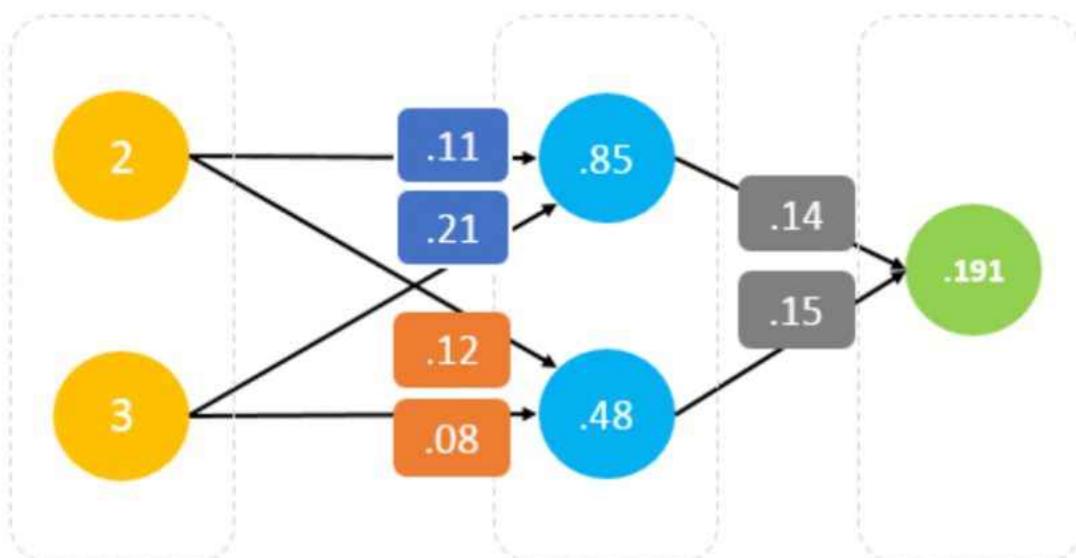
跟线性回归一样, 只是维度变成了二维, 输入是一个二维向量了, 所以同时有w1,w2...来支持向量的乘法



Input layer

Hidden layer

Output layer



→ Forward Pass

$$[2 \ 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \ 0.48] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = [0.191]$$

$$2 \times 0.11 + 3 \times 0.21 = 0.85$$

$$0.85 \times 0.14 + 0.48 \times 0.15 = 0.191$$

$$2 \times 0.12 + 3 \times 0.08 = 0.48$$

Matrix multiplication

Details

机器学习记事本

Input layer

Hidden layer

Output layer



Error = 0, if prediction = actual

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

Error is always positive because of the square

$\frac{1}{2}$ is added to ease the calculation of the derivative

$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

机器学习记事本

最靠近神经网络尾部的直接求导一次，中间的需要把偏导传播回去，就是反向传播

$$*W_x = W_x - a \left(\frac{\partial \text{Error}}{\partial W_x} \right)$$

Derivative of Error
with respect to weight

Old weight ↓
 ↓
 Learning rate

↑ New weight

* $W_6 = W_6 - a (h_2 \cdot \Delta)$
 * $W_5 = W_5 - a (h_1 \cdot \Delta)$
 * $W_4 = W_4 - a (i_2 \cdot \Delta W_6)$
 * $W_3 = W_3 - a (i_1 \cdot \Delta W_6)$
 * $W_2 = W_2 - a (i_2 \cdot \Delta W_5)$
 * $W_1 = W_1 - a (i_1 \cdot \Delta W_5)$

机器学习记事本

$$\frac{\partial \text{Error}}{\partial W_6} = \frac{\partial \text{Error}}{\partial \text{prediction}} * \frac{\partial \text{prediction}}{\partial W_6}$$

chain rule

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

$$\frac{\partial \text{Error}}{\partial W_6} = \frac{1}{2}(\text{predictoin} - \text{actula})^2 * \frac{\partial (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\text{prediction} = (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$$

$$\frac{\partial \text{Error}}{\partial W_6} = 2 * \frac{1}{2}(\text{predictoin} - \text{actula}) \frac{\partial (\text{predictoin} - \text{actula})}{\partial \text{predicton}} * (i_1 w_3 + i_2 w_4)$$

$$h_2 = i_1 w_3 + i_2 w_4$$

$$\frac{\partial \text{Error}}{\partial W_6} = (\text{predictoin} - \text{actula}) * (h_2)$$

$$\Delta = \text{prediction} - \text{actual}$$

← delta

$$\frac{\partial \text{Error}}{\partial W_6} = \Delta h_2$$

机器学习记事本

$$\frac{\partial \text{Error}}{\partial W_1} = \frac{\partial \text{Error}}{\partial \text{prediction}} * \frac{\partial \text{prediction}}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$

chain rule

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

$$\frac{\partial \text{Error}}{\partial W_1} = \frac{1}{2}(\text{predictoin} - \text{actula})^2 * \frac{\partial (h_1) w_5 + (h_2) w_6}{\partial h_1} * \frac{\partial i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\text{prediction} = (h_1) w_5 + (h_2) w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

$$\frac{\partial \text{Error}}{\partial W_1} = 2 * \frac{1}{2}(\text{predictoin} - \text{actula}) \frac{\partial (\text{predictoin} - \text{actula})}{\partial \text{predicton}} * (w_5) * (i_1)$$

$$\Delta = \text{prediction} - \text{actual}$$

← delta

$$\frac{\partial \text{Error}}{\partial W_1} = (\text{predictoin} - \text{actula}) * (w_5 i_1)$$

机器学习记事本

自然语言理解

自然语言即我们日常说的语言，理解就是理解该语句在特定场景下的语义，包含情感倾向，语句意图，语句核心词，比如对“future”的理解，普通人认为是“未来”，但对于金融相关的专业可能就会认为是“期货”，而“期货”对金融专业人士来说是一个语境。

比如一句「你开心就好」，可以在不同的场景下传达鄙视和祝愿等多种意思。

分词和向量化

分词难点

- 一词多义、和切词歧义是问题是中文下的常见的一个问题，如经典的【武汉市长江大桥】和【无线电法国别研究】案例。前者可以毫无违和地切为[武汉市 / 长江大桥]和[武汉市长 / 江大桥]，后者则可以切分为[无线电 / 法国 / 别 / 研究]和[无线电法 / 国别 / 研究]截然不同的词块。
- 未登录词,比如“老六”、“卷王”、“栓Q”、“摆烂”等词。

分词技术[搜索引擎（二）](#) | 主流分词技术概览

- FMM

正向最长匹配FMM，是从句子开头查找字典，切分得到以当前字开头的最长单词

以经典示例“无线电法国别研究”为例，基于FMM的切词可得：【无线电法 / 国别 / 研究】，得到了一个合理的分词结果。

- RMM

RMM切分的方向与FMM相反，是从句子的结尾反过来寻找每个字往前找的最大词块。

- BMM

简单来说，BMM就是结合FMM和RMM两家之所长，来降低歧义词的分词错误率。一种比较常见的融合规则如下：

- (1) 同时执行正向和逆向最长匹配，若两者的词数不同，返回词数更少的那个。
- (2) 如果相同，返回两者中单字（单个字成词）更少的结果。
- (3) 当单字数也相同时，则优先返回逆向最长匹配的结果。

- N-gram语言模型

在各种切词集合中找出最合理的组合，即计算出分词概率最大路径。

公式化的来说，假设一个句子由若干词 \square 组成： $\square=[\square_0, \square_1, \square_2, \dots, \square_n, \square_{n+1}]$ ，此时该 \square 的概率如下：

$$\begin{aligned} p(W) &= p(w_0, w_1, w_2, \dots, w_n, w_{n+1}) = p(w_1|w_0) * p(w_2|w_0w_1) * \dots \\ &\quad * p(w_{n+1}|w_0w_1\dots w_n) = \prod_t^{n+1} p(w_t|w_0w_1\dots w_{t-1}) \end{aligned}$$

在实际计算中往往引入马尔可夫假设来简化语言模型，即假设分词的当前状态只与前一项相关，从而退化成容易求出的2-gram

- 基于大模型进行分词

以ELMo、BERT、XLNet、GPT为代表的大规模预训练模型，分词作为NLP任务中的序列标注问题

分块技术

1. 基于规则的分块: - 这种方法简单直接，根据预定义的规则（如字符数、单词数或句子数）将文档切成固定大小的块。例如，可以设定每个块包含一定数量的字符或单词，不考虑内容的连贯性。
2. 基于语义聚类的分块: - 在这种方法中，文档首先被分析以识别语义相关的段落或句子，然后这些段落被聚类成块。这可能涉及到使用主题建模或聚类算法，如K-means，以确保每个块内语义的一致性。
3. 基于机器学习模型的分块: - 这种方法可能使用预训练的模型，如BERT，来预测每个位置作为分块边界的概率，或者学习一个专门的模型来决定最佳的分块点。
4. 基于LLM (Large Language Model) 代理的分块: - 利用大型语言模型的生成能力，通过自动生成的分割点来划分文档。这可能涉及模型生成分割指示符，然后根据这些指示符进行分块。
5. 上下文感知的分块: - 考虑到RAG模型需要理解上下文，一些方法可能会尝试在保持语义连贯性的基础上进行分块，这可能涉及到使用上下文敏感的分词或分段算法。
6. 动态分块策略: - 根据模型的实时需求，动态地决定何时和如何分块，这可能涉及到在运行时评估模型的内存使用和计算效率，以确定最佳的分块策略。

向量化技术

主要目的是将分段自然语言文本翻译成向量，便于机器学习训练和向量数据库存储

1. Word2Vec:
 - a. - CBOW (Continuous Bag of Words): 在这个模型中，目标是预测一个词，给定它的上下文（通常是周围的几个词）。通过将上下文词的向量相加，然后通过一个softmax层来预测中心词。这种模型假设上下文词对中心词的预测是有帮助的，从而学习到有用的词向量。 -
 - b. Skip-gram: 相反，Skip-gram尝试预测上下文词，给定中心词。这样，每个词的向量都要负责预测它周围可能出现的词，这有助于学习更通用的词向量。
2. GloVe: 旨在通过捕获词汇的共现统计信息来学习词的分布式表示，结合了全局统计和局部上下文信息，同时通过优化损失函数来学习词向量，使得向量空间中的相似性反映了词汇在语料库中的共现模式。
3. ELMo使用LSTM (长短期记忆网络) 或更先进的Transformer模型，对文本序列进行前向和后向的预测。这意味着模型在处理每个词时，不仅考虑了前面的上下文，也考虑了后面的上下文，从而获得更丰富的语境信息。

4. BERT (Bidirectional Encoder Representations from Transformers): - BERT是基于Transformer的预训练模型，它利用了Transformer的自注意力机制，可以同时考虑词的前后文信息。在预训练阶段，BERT使用两个任务：掩码语言模型（Masked Language Modeling）和下一句预测。前者随机遮蔽一部分输入词，模型需要预测被遮蔽的词；后者则判断两个连续的句子是否真的是连续的。这两个任务使得BERT能学习到丰富的双向上下文信息。

RNN神经网络一文看懂循环神经网络 RNN（2种优化算法+5个实际应用）

基本结构

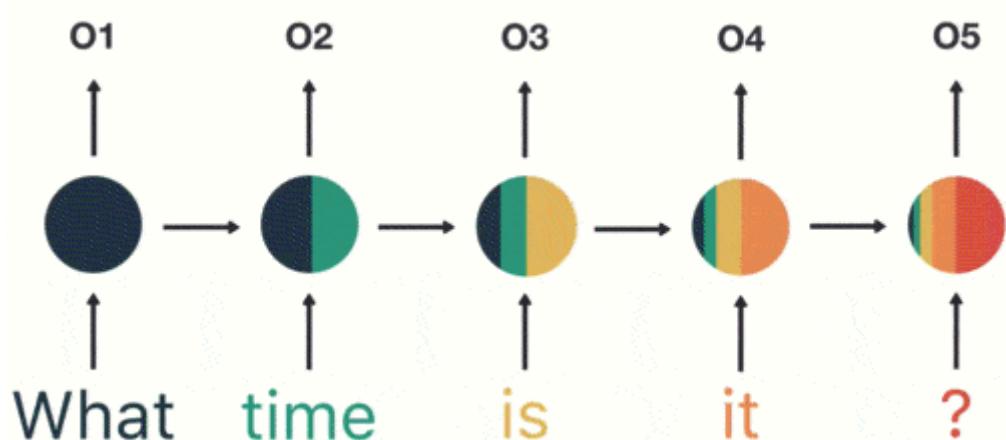
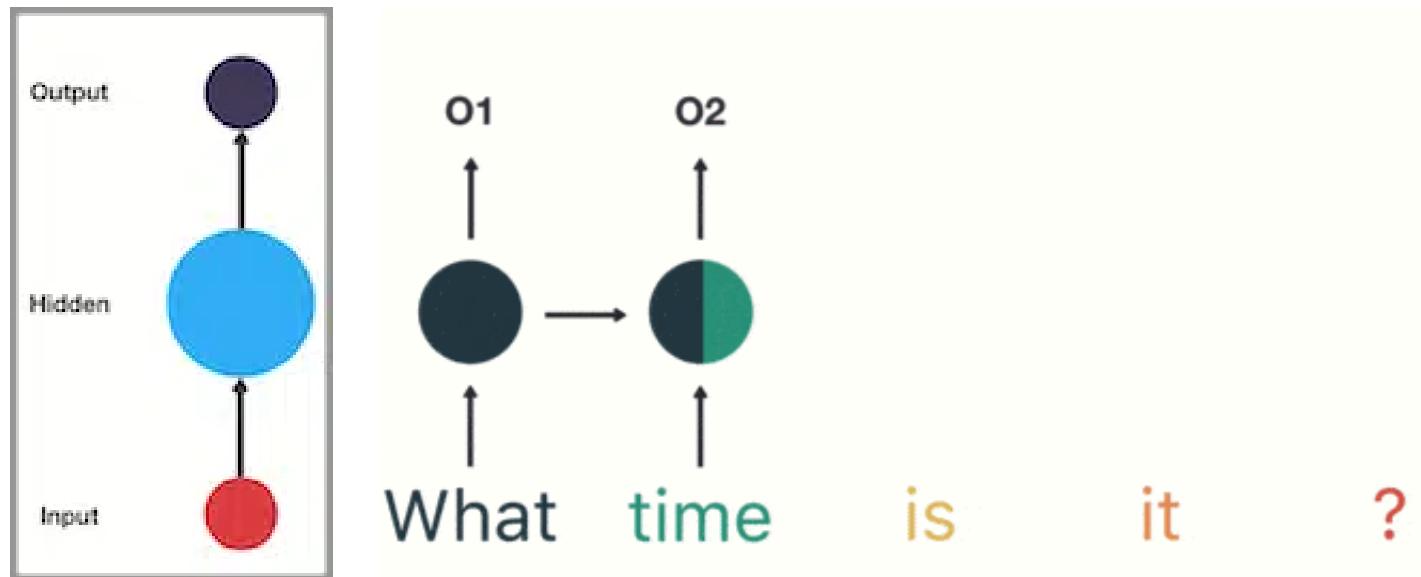
普通的算法大部分都是输入和输出的一一对应，也就是一个输入得到一个输出。不同的输入之间是没有联系的。



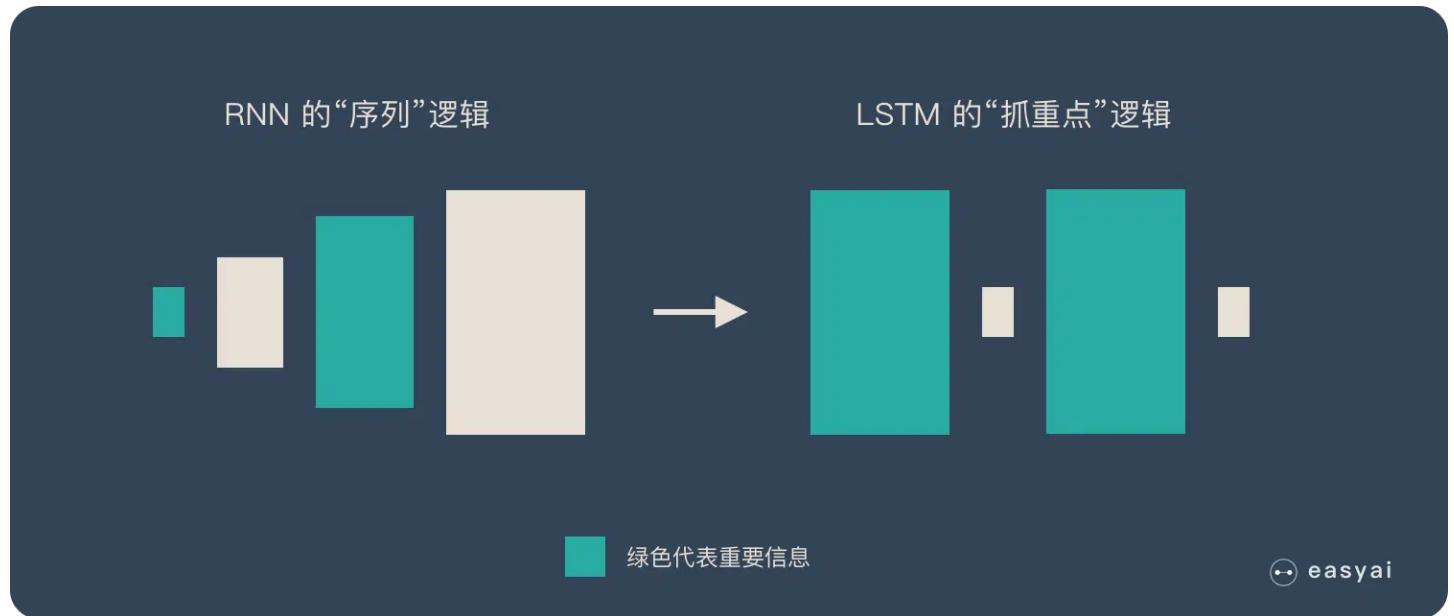
为了填好下面的空，取前面任何一个词都不合适，我们不但需要知道前面所有的词，还需要知道词之间的顺序。



可以看到，单个的神经元是一个输入对应一个输出，
而rnn神经网络中当time作为第二神经元输入的时候，它还受到了来自第一个输入what产生的一部分影响，也就是说，每个输入，会产生一个输出o，产生一个状态变量s,s会影响后续的输入。
我们把o当做本次的输出，把s当做本次迭代后保留的状态，因为这个状态存在，所以说整个神经网络有了记忆性，下一句话输入时，会受到前一句话保持的状态的影响，也会再更新这个状态



通过上面的例子，我们已经发现，短期的记忆影响较大（如橙色区域），但是长期的记忆影响就很小（如黑色和绿色区域），这就是 RNN 存在的短期记忆问题。



举个例子，我们先快速的阅读下面这段话：

此商品不支持7天无理由退货！此商品不支持7天无理由退货！此商品
不支持7天无理由退货！重要的事情说3次！
当天到货后用了一个小时，压的耳朵就非常疼，耳朵都**压红
了！！！跟客服联系，都说不能退货！只要拆了就不给退！！
而且有些机型不适配，我本来要送人，默认只能适配ios，部分安卓
手机就不支持，建议买之前千万想好，签收了就没法退了。而且说好
送的手环也没有，真是醉了

easyai

当我们快速阅读完之后，可能只会记住下面几个重点：

此商品不支持7天无理由退货！

压的耳朵非常疼

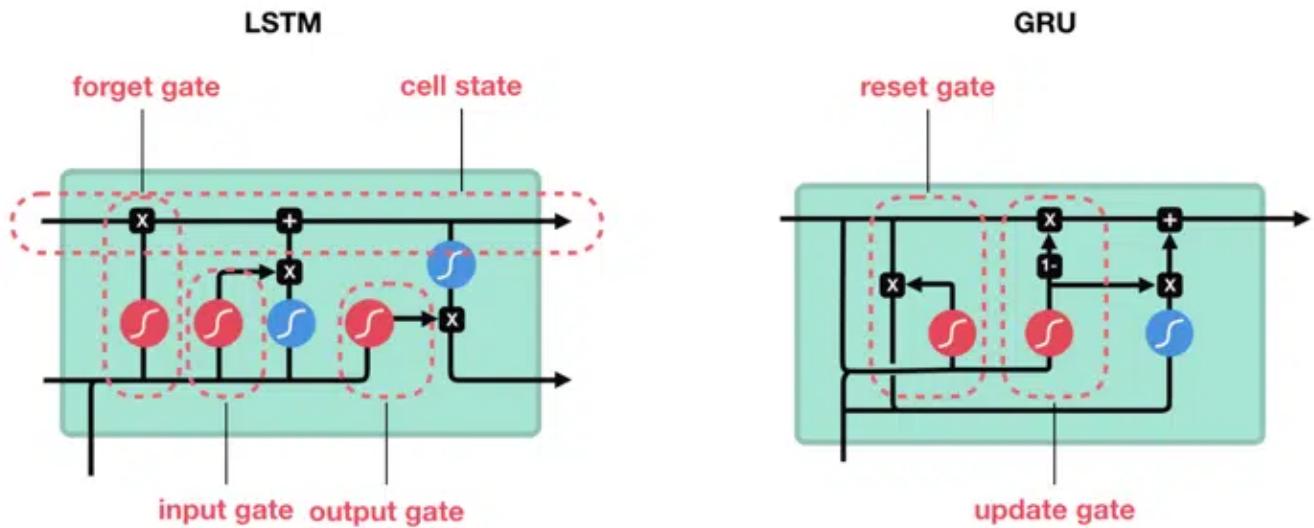
只要拆了就不给退

默认只能适配ios，部分安卓

手机就不支持

easyai

内部结构通过各种函数变换和门机制（类似逻辑与或非门，但是可以学习）来过滤不重要的信息，而GRU是减少了门的数量，提高了训练速度，同时准确度也有提升(结构简单到位了)



注意力机制

自然语言形成的文本中，常常有长距离的逻辑关联。这种逻辑关联，往往很难靠类似循环神经网络（RNN）这样的简单序列处理模型来发现和利用。

这儿离购物中心不远，打车也就十分钟，不堵车的话，其实也就两公里吧

基本原理

注意力机制解决了文本中词语之间的关联性，且自适应地参与模型

1 The animal didn't cross the street because it was too tired

1. 首先将查询和已有句子进行分词，并向量化，得到每个词的查询向量表示(C)

"The" -> [0.3, 0.1, -0.2]

"animal" -> [0.5, -0.7, 0.4]

"didn't" -> [-0.1, 0.2, 0.3]

"cross" -> [0.4, -0.5, 0.6]

"the" -> [0.2, -0.3, 0.1]

"street" -> [0.7, 0.1, -0.2]

"because" -> [-0.3, 0.2, 0.5]

"it" -> [0.1, -0.4, 0.3]

"was" -> [0.2, 0.1, -0.3]

"too" -> [-0.4, 0.5, 0.1]

"tired" -> [0.6, -0.3, 0.4]

2. 线性转换:

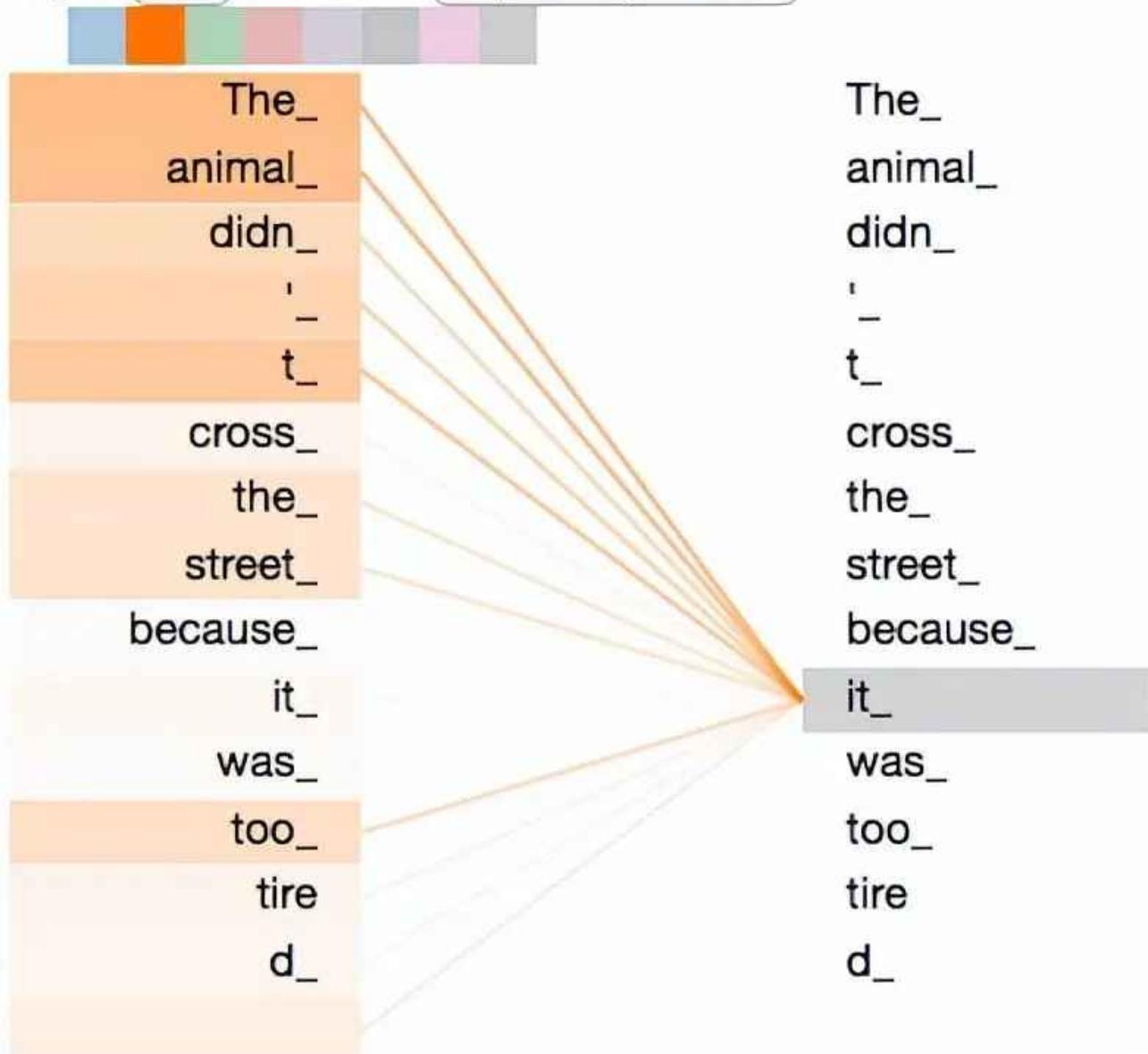
将每个词向量经过三个线性变换(线性变换就是 $W^*I + S$ 的结构)，得到Q,K,V三个向量

3. 相似度计算

使用查询向量(Q)（例如，"animal"的向量）与每个键向量(K)（例如 it）进行点积操作，以计算相似度。用函数将相似度转换为概率分布，作为注意力权重，这里得到的会是一个标量(例如 animal:0.4表示这个单词对整个句子的贡献和重要程度为0.4)

4. 加权聚合：根据计算出的注意力权重，对所有词的值向量 (V) 进行加权求和。如果 "animal" 和 "too tired" 的注意力权重大，那么他们在聚合后的向量中会有较大的权重，这将强调句子中动物疲惫的主题。

Layer: 5 ▲ Attention: Input - Input ▲

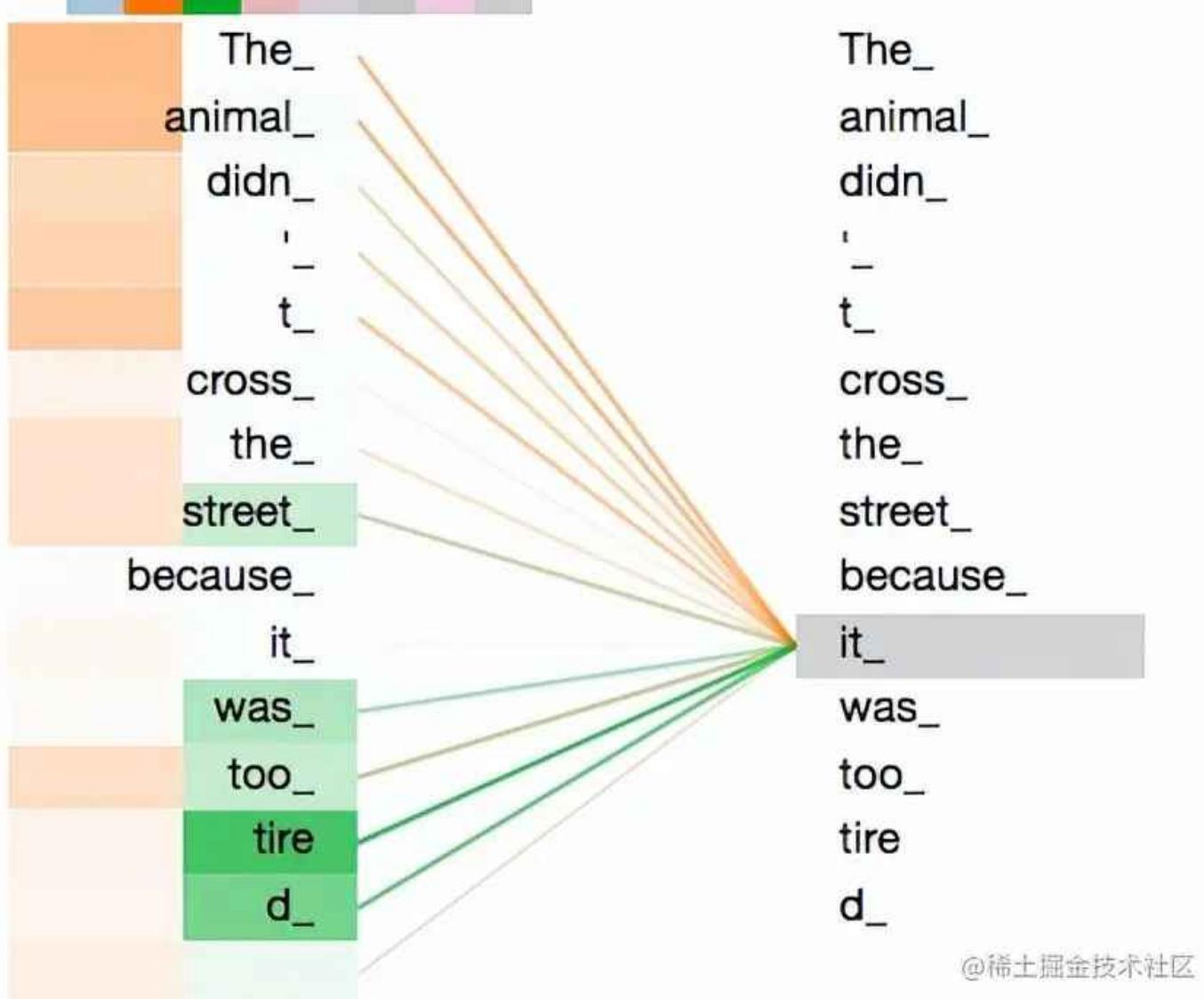


@稀土掘金技术社区

多头注意力机制

基于之前的背景，如果不仅考虑词与词的相似度，再加入词语在句子中的位置，词语的情感倾向等因素，分别构建多个Q,K,V的线性变换，计算加权的最终向量，那么得到的就是这个句子多个维度的词语之间的关系

Layer: 5 Attention: Input - Input



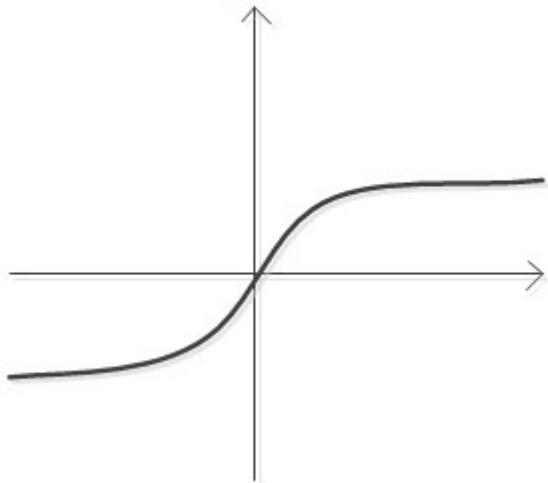
@稀土掘金技术社区

TransFormer架构

非线性激活函数

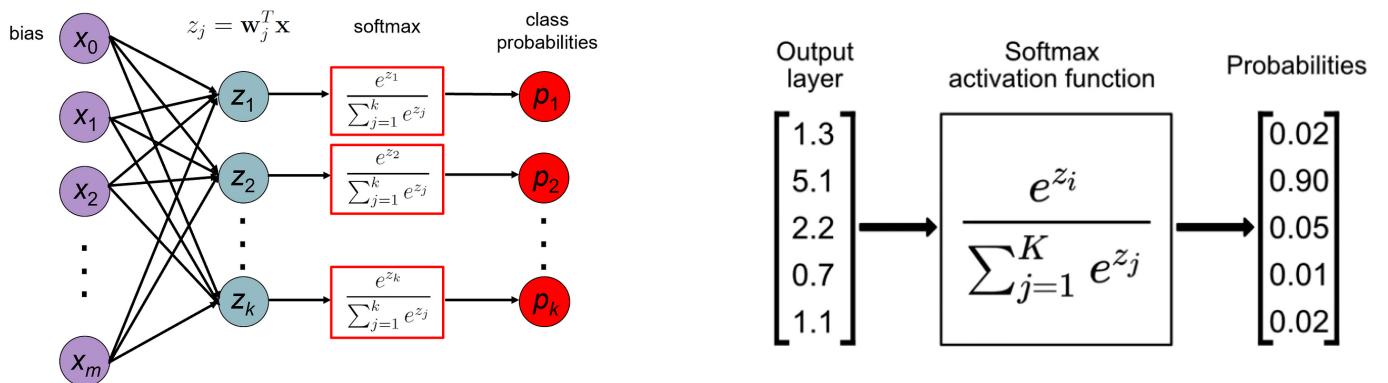
如果神经网络没有非线性函数，将永远是线性的可计算关系，引入非线性函数，让整体具备了非线性因素，同时对于计算梯度时的梯度消失问题有缓解

sigmoid

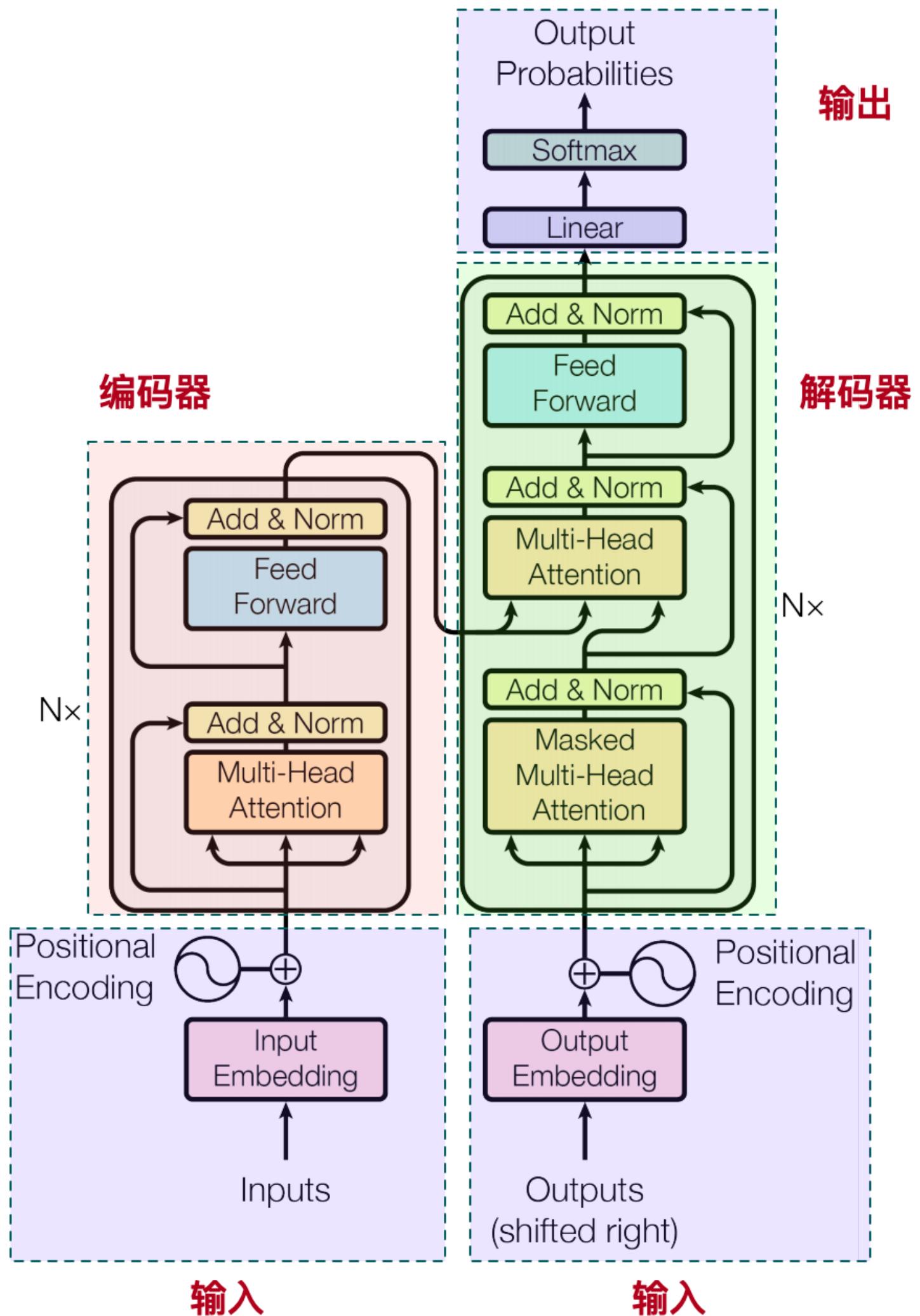


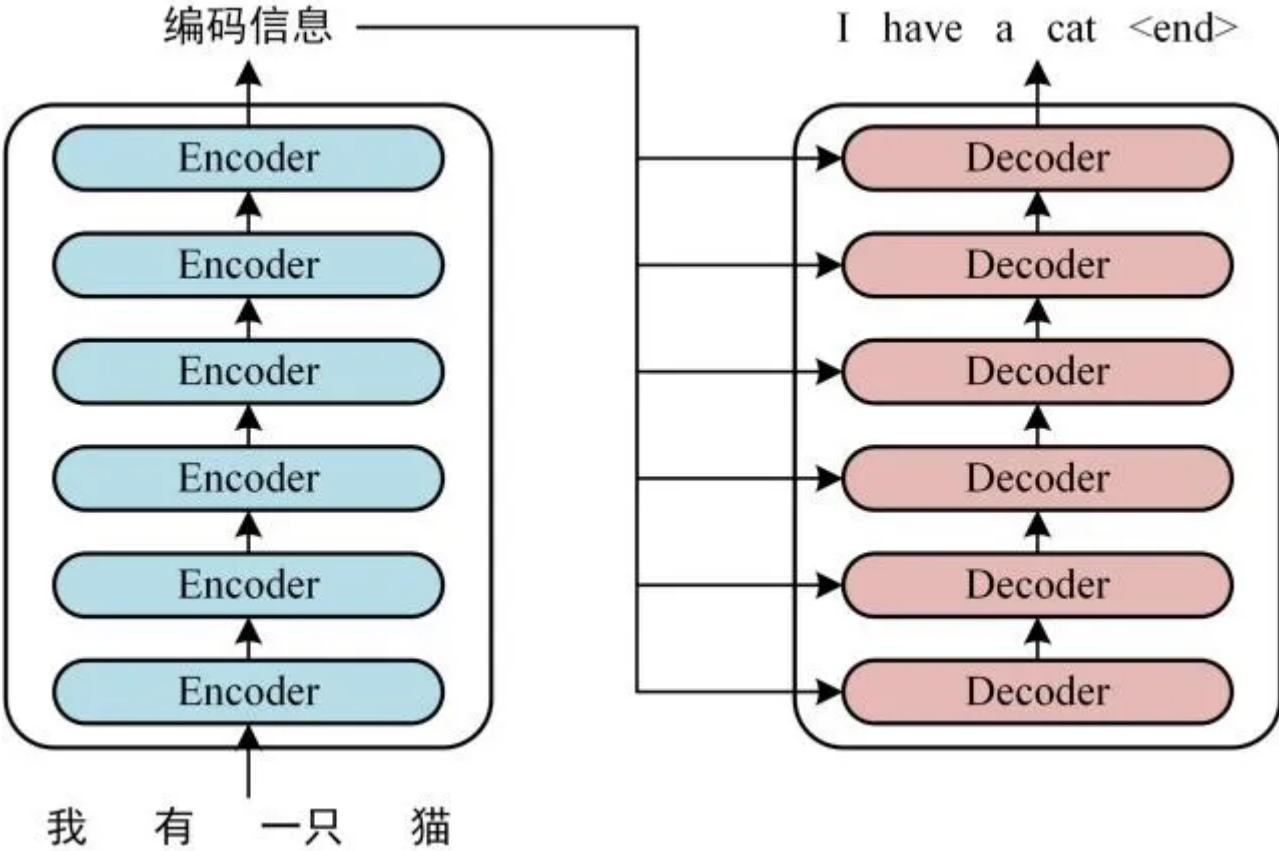
Softmax

经典的一种神经网络，将任何输出，都变成不超过1的概率向量的模式



拆开看编码器和解码器结构为下图，FeedForward和linear还有softmax都看作普通的神经网络层，把Add & Norm看作是非线性的函数，即对于输入x，输出 $y=f(x)+x$ ，这里的 $f(x)$ 就是非线性函数，同时可以看到，解码器还针对编码器的输出进行了多头的注意力模型导入





GPT背景

GPT模型与传统Transformer模型的区别，主要在于它的 Pre-trained 预训练，通过预训练-微调 (pre-training, fine-tuning) 的方法实现。

1. 预训练（无监督）：首先在大规模语料上进行预训练得到一个通用的模型，主要是获得语言理解和生成能力
2. 微调（有监督）：然后在特定任务上进行微调，以适应更贴合用户和实际特定场景的需求

本质上还是Transformer的架构，但是层级和参数非常大

GPT-1 (2018)

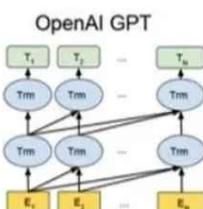
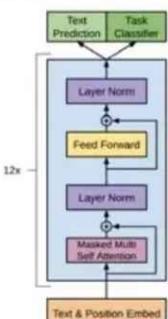
GPT-2 (2019)

GPT-3 (2020)

ChatGPT (2022)

陈巍谈芯

12层Transformer，每层12个注意头

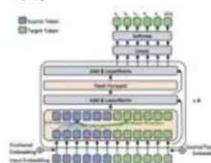


在GPT-1的基础上，GPT-2做了以下改进：

- ① GPT-2有48层，使用1600维向量进行词嵌入
- ② 将层归一化移到每个子块的输入，并在最终的自注意力块后增加一层归一化
- ③ 修改初始化的残差层权重，缩放为原来的 $1/\sqrt{N}$ 。其中，N是残差层的数量。
- ④ 特征向量维数从768扩展到1600，词表扩大到50257

在GPT-2的基础上，GPT-3做了以下优化：

- ① GPT-3有96层，每层有96个注意头
- ② GPT-3的单词嵌入大小从GPT-2的1600增加到12888
- ③ 上下文窗口大小从GPT-2的1024增加到GPT-3的2048
- ④ 采用交替密度和局部带状稀疏注意模式



ChatGPT基于GPT-3.5架构，并做以下优化：

- ① ChatGPT 使用来自人类反馈的强化学习进行训练
- ② 通过近端策略优化算法进行微调，为信任域策略优化算法带来了成本效益



@稀尘掘金技术社区

LLM大模型到RAG

RAG技术

基本概念

它是一种结合了检索和生成的方法，用于自然语言处理任务。它的思路很简单，既然LLM的语料存在时效性和数据源的问题，那么提供语料的这个操作就由用户传递过去。

基本路径就是将私域的数据向量化后存储，在提问时，查询到关联的向量对应的私域数据文本，和问题一起作为大模型的输入，也是prompt工程的应用

例如

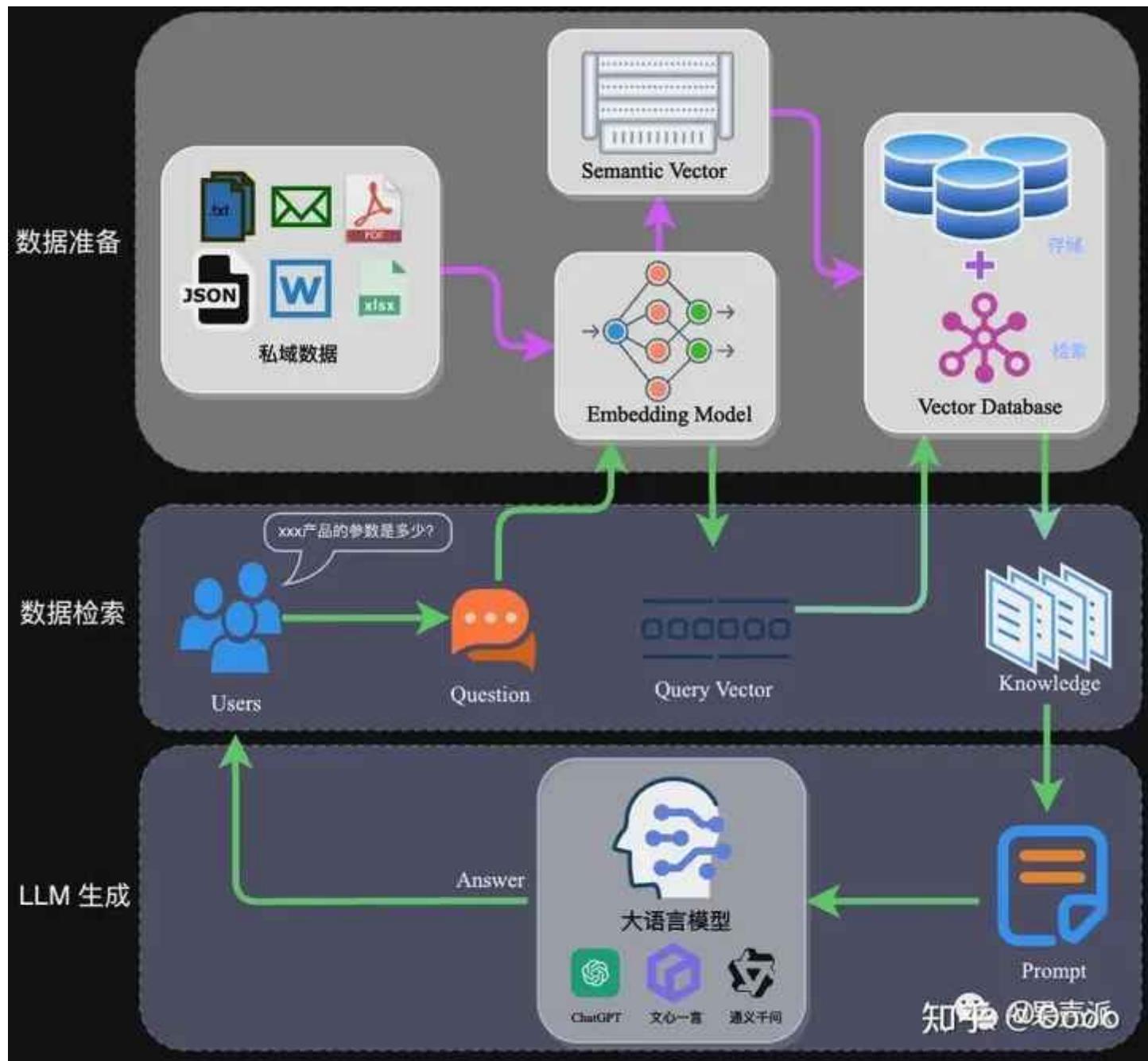
用户问题: "我们公司去年Q4的财务报表显示的净利润是多少？"

查询到的关联上下文:之前导入了去年的财务报表数据到向量数据库中，所以根据问题去检索得到关键数据：“2023年第四季度净利润为1.2亿人民币”。

作为大模型输入的prompt格式为

- 1 【任务描述】
- 2 假如你是一个专业的客服机器人，请参考【背景知识】，回
- 3 【背景知识】
- 4 “**2023年第四季度净利润为1.2亿人民币**”。 // 数据检索得到的相关文本
- 5 【问题】

6 我们公司去年Q4的财务报表显示的净利润是多少?



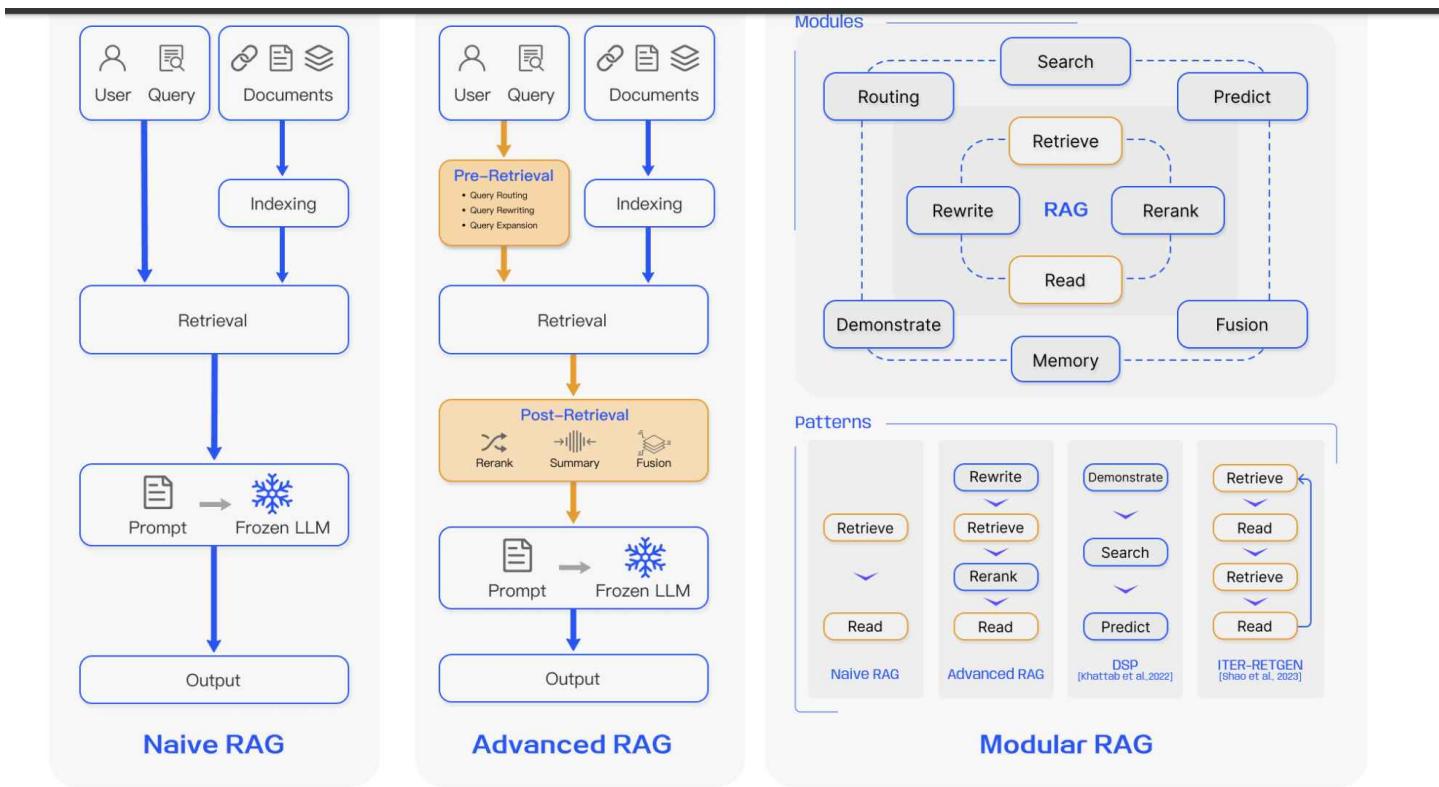
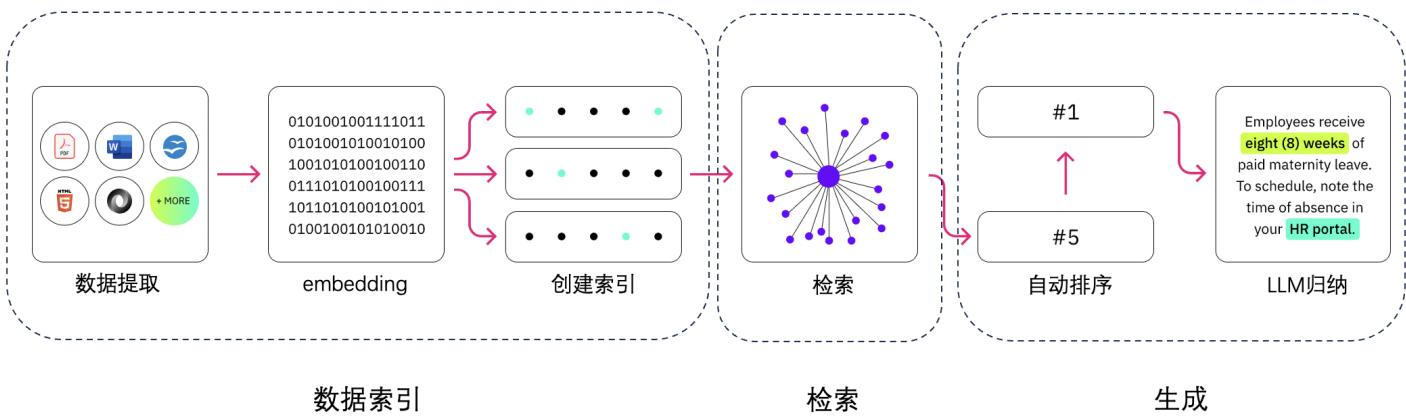
高级RAG

[检索增强llm编码模型开源库](#)

[\[论文阅读\]大型语言模型检索增强生成综述](#)

[高级RAG指南](#)

[meta公司的rag技术博客](#)



文档采集

因为要构建私有知识库，数据可能来源pdf文档，表格，图片，网页，可能出现图文表格混排等复杂场景，同时需要对导入前进行数据的清洗，以langchain框架举例

- **文档解析器【Document parser】**：文档解析器的核心作用是从不同的文档格式中提取出结构化的信息，特别注意格式的处理。这包括解析可能包含图片和表格的 PDF 等。
- **文档格式【Document formats】**：文档解析器必须能够处理多种文档格式，比如 PDF、Word、Excel 等，保证文档处理的灵活性。这涉及识别和管理嵌入的内容，比如超链接、多媒体元素或注释，以呈现文档的完整内容。
- **表格识别【Table recognition】**：从文档中的表格中识别和提取数据对于保持信息的结构性很重要，尤其是在报告或研究论文中。提取表格相关的元数据，包括标题、行和列信息，可以加深对文档组织结构的理解。[Table Transformer](#) 等模型可以帮助完成这项任务。
- **图像识别【Image recognition】**：光字符识别 (OCR) 应用于文档中的图像，以主动识别和提取文本，使其可以进行索引和后续检索。

- **元数据提取【Metadata extraction】**：元数据是指文档的附加信息，不属于其主要内容。它包括作者、创建日期、文档类型、关键词等细节。元数据提供了有用的上下文，有助于组织文档并提高搜索结果的相关性。可以使用 NLP/OCR 管道提取元数据，并将文档作为特殊字段进行索引。

文档分割

[RAG中的文档分割技术](#)

[从文档中分割图像/表格代码库](#)

文本分割器都根据 `chunk_size` (块大小)和 `chunk_overlap` (块与块之间的重叠大小)进行分割。

- `chunk_size` 指每个块包含的字符或 Token (如单词、句子等) 的数量
 - `chunk_overlap` 指两个块之间共享的字符数量，用于保持上下文的连贯性，避免分割丢失上下文信息
-
- `RecursiveCharacterTextSplitter()`: 按字符串分割文本，递归地尝试按不同的分隔符进行分割文本。,比如先按"\n"分割，发现分割后长度过长，再按","分割
 - `CharacterTextSplitter()`: 按字符来分割文本。
 - `MarkdownHeaderTextSplitter()`: 基于指定的标题来分割markdown 文件。
 - `TokenTextSplitter()`: 按token来分割文本。
 - `SentenceTransformersTokenTextSplitter()`: 按token来分割文本
 - `SimilarSentenceSplitter`: 根据句子之间的相似性分组，其实也是理解了上下文的语义，做到精准分块
 - `Language()`: 用于 CPP、Python、Ruby、Markdown 等。
 - `NLTKTextSplitter()`: 使用 NLTK (自然语言工具包) 按句子分割文本。
 - `SpacyTextSplitter()`: 使用 Spacy按句子的切割文本(依赖已经训练的模型，spacy是一个python的自然语言处理工具库)。

Chunking Techniques For RAG

Technique	UseCase	Pros	Cons
Character splitter	Text	Versatile: Handles various separators Flexible: Adapts to different languages Cost-Effective: Does not require a ML model	Performance: May have increased computational load Complexity: Requires parameter tuning Sentence Interruption: May cut sentences midway
Recursive character splitter	Text, code	Versatile: Handles various separators Flexible: Adapts to different languages Cost-Effective: Does not require a ML model	Performance: Recursive nature may increase computational load Complexity: Requires parameter tuning Sentence Interruption: May cut sentences midway
Sentence splitter	Text	Considers Sentence Boundaries: Avoids cutting sentences prematurely Customizable: Parameters for stride and overlap Cost-Effective: Works with light sentence segmenter	Lack of Versatility: Limited to sentence-based chunks Overlap Issues: May lead to redundancy
Semantic splitter	Text, Chat	Contextual Grouping: Organizes text based on semantic similarity Overcomes Challenges: Handles chunk size and overlap	Complexity: Requires similarity model and tuning Parameter Dependency: Relies on setting appropriate parameters Resource Intensive: Demands computational resources
Propositions	Text, Chat	Atomic Expression: Introduces novel retrieval unit (propositions) Distinct Factoids: Each proposition is self-contained Contextualization: Provides necessary context	Complexity: Requires LLM model Parameter Dependency: Relies on setting appropriate prompt Resource Intensive: Demands computational resources



编码器选择

检索增强llm编码模型开源库

编码器模型基准测试

- 稠密嵌入 (Dense Embeddings) :

这些嵌入通过将词汇映射到高维向量空间中的连续向量来捕获词语的全局语义。

Word2Vec和GloVe是常用的稠密嵌入方法，用于捕捉词汇之间的语义关联。

- 稀疏嵌入 (Sparse Embeddings) :

稀疏嵌入通常在维数较高但大部分元素为零的情况下使用，以强调特定特征或属性。

它们在处理高维稀疏数据（如文档中的关键词）时特别有用。

- 多向量嵌入 (Multi-Vector Embeddings) :

这种类型的嵌入允许不同部分的信息在后期处理阶段进行交互，例如在协同过滤中使用不同的向量表示用户和物品。LightGBM的Embedding Feature Column就是一个例子，它允许在模型训练中结合多个特征向量。

- 可变维度嵌入 (Variable-Dimension Embeddings) :

这些嵌入的维度可以根据输入的特性动态变化，适应不同的数据分布。

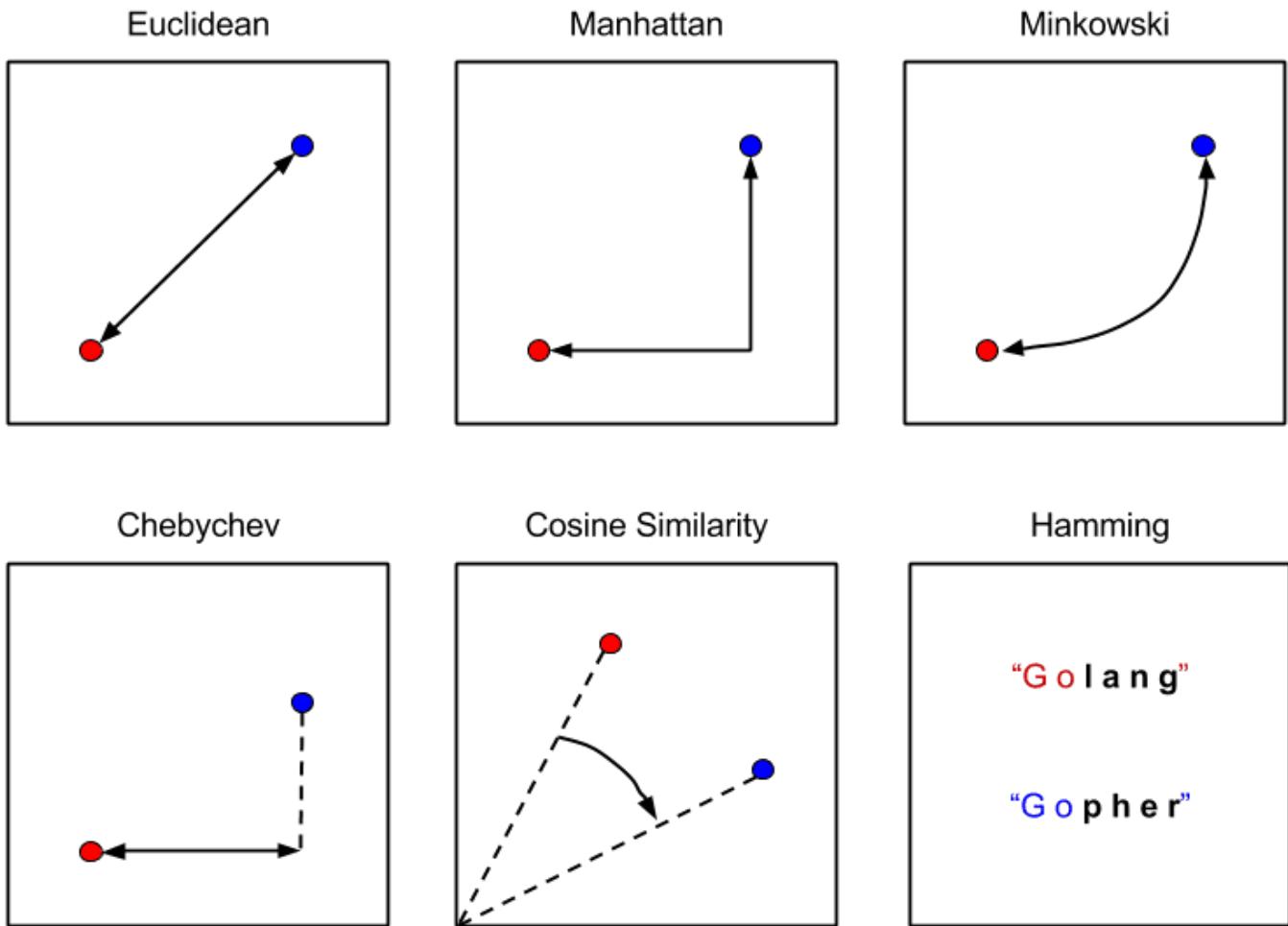
它们通常用于处理具有不同长度序列的场景，如变长的文本片段。

- 代码嵌入 (Code Embeddings) :

最近的研究引入了将代码视为自然语言的嵌入方法，如Code2Vec、Code2Seq等。

这些嵌入帮助理解代码的结构和逻辑，促进了代码的自动完成、重构和错误检测等任务。

词向量的相似度又多种表示，



向量检索优化

大型语言模型(LLMs)很容易被无关的上下文所分散注意力，而且拥有大量上下文（检索到的 topK 文档）可能会因为LLMs的注意力模式而错过某些上下文。因此，利用相关和多样化的文档来提高检索是至关重要的。

同时选择数据库要考虑分布式并发扩展，主备集群化的高可用和稳定性，持久化能力，读写效率等工程因素

- 预过滤：通过用户指定的元数据或关键词字段或使用语言模型抽取问题中的关键元数据信息，进行前置过滤筛选
- 图关系检索：如果可以将很多实体变成node，把它们之间的关系变成relation，就可以利用知识之间的关系做更准确的回答。特别是针对一些多跳问题，利用图数据索引会让检索的相关度变得更高；
- HyDE 技术：通过先让llm根据上下文和问题来生成响应，用来扩展问题，使得查询包含了语义理解的深度信息，以及之前的上下文信息，再进行向量库的查询。

- 稀疏向量 (Sparse Vectors) : 稀疏向量可以用于表示文档的关键词或主题，或者在检索阶段用于快速匹配查询。
- Hybrid search: 在精确搜索和近似搜索间找到性能和准确度平衡
 - 近似搜索: 如HNSW,通过构建有层级的索引，顶层的近似邻居多，底层的近似邻居少，搜索时逐层检索，找到满足要求的数量就返回，平衡了性能和准确性
- 查询路由: 基于元数据信息和自然语言理解后的语义知识，进行向量库和表的选择，最后汇总数据进行结果排序
- 信息压缩: 大语言模型 (LLM) 的处理上下文长度是有限的，如果检索到的文档中含有过多的噪声（比如：无关的信息），可能会导致响应质量下降。
- 最大边际相关性 (MMR): 查询的时候不仅考虑和问题的相似相关程度，也要考虑输出结果时语句的相似度，保证输出文档的多样性
- 结果重排: 根据初步数据库筛选出的文档进行第二次评估和问题的关联度，提高回答质量和查询效率
 - 交叉编码器: 根据输入的两个句子判断关联性，输入查询和返回的结果文档，得到关联性，再进行排序
 - ColBERT: 根据token级别（字/词级别）表示来计算查询和结果文档的关联程度进行排序，比交叉编码器效率高，且允许文档存储和问题查询使用不一样的向量编码器
 - RankGPT: 直接利用llm大模型来进行语义关联度排序
- 查询重写:
 - **基于历史记录重写【Rewrite based on history】**：这种方法中，系统利用用户的查询历史记录来理解对话的上下文并改进后续查询。
 - **创建子查询【Create subqueries】**：由于检索问题，复杂查询可能难以回答。为了简化任务，查询会被分解成更具体的子查询。这有助于检索生成答案所需的正确上下文。
 - **创建相似查询【Create similar queries】**：为了提高检索相关文档的可能性，我们会根据用户输入生成类似的查询。这可以克服检索在语义或词汇匹配方面的限制。如：*我想知道白金信用卡 -> 告诉我白金信用卡的优点*
- 迭代式检索: 根据本次检索的查询和回答进行提炼，进行再次检索，随着上下文对话进行，保留精炼的关键信息
- 融合检索器: 结合多个独立的检索结果,进行筛选和评估

向量数据库比较

Vendor	About					Search							
	OSS	License	Dev Lang	VSS Launch	Filters	Hybrid Search	Facets	Geo Search	Multi-Vector	Sparse	BM25		
Activeloo...		MPL 2.0	python c++	2023		-	-						
Anari AI		Proprietary	-	2023	-		-		-	-	-	-	
Apache C...		Apache-2.0	java	2023				-	-	-			-
Apache S...		Apache-2.0	java	2022						-	-		
ApertureDB	-	-	-	-	-	-	-	-	-	-	-	-	-
Azure AI S...		Proprietary	c++	2023									
Chroma		Apache-2.0	python	2022			-	-					
ClickHouse		Apache 2.0	c++	2022									
CrateDB		Apache 2.0	java	2023			-					-	
DataStax ...		Proprietary	java go	2023									
Elasticsea...		Elastic Lice...	java	2021									
Epsilla		GPL-3.0	c++	2023			-	-					-
GCP Verte...		-	-	2021			-	-					
KDB.AI		Proprietary	python	2023		-	-	-	-	-		-	-
LanceDB		Apache-2.0	rust	2023				-	-		-		
Marqo		Apache-2.0	python	2022				-	-				
Meilisearch			MIT	rust	2023								
Milvus		Apache-2.0	go c++	2019			-						
MongoDB...		GNU AGPL ...	c++ java	2023									
MyScale		Proprietary	c++	2023						-			

多模态检索

从文档中读取的信息可能包含文本，图像，表格，音频，视频等内容，我们可以将这些非文本资源进行文本化转义也存储在向量库，同时建立和原始数据的索引关联，使得在回答和检索时能对多种类型进行检索和回答

[Llmaindex+Llava实现多模态RAG](#)

prompt工程

prompt是提交信息给llm前的最后一步，需要给出输出的格式，甚至举出例子让llm模型按照步骤执行给出示例：

```

1 prompt = f"""
2 您的任务是以一致的风格回答问题。
3
4 <孩子>：请教我何为耐心。
5
6 <祖父母>：挖出最深峡谷的河流源于一处不起眼的泉眼；最宏伟的交响乐从单一的音符开始；最复杂的挂毯以一根孤独的线开始编织。
7
8 <孩子>：请教我何为韧性。
9 """

```

```
10 response = get_completion(prompt)print(response)Copy to clipboardErrorCopied
11 <祖父母>：韧性是一种坚持不懈的品质，就像一棵顽强的树在风雨中屹立不倒。它是面对困难和挑战时
    不屈不挠的精神，能够适应变化和克服逆境。韧性是一种内在的力量，让我们能够
```

规定格式：

```
1 prompt_2 = f"""
2 1-用一句话概括下面用<>括起来的文本。
3 2-将摘要翻译成英语。
4 3-在英语摘要中列出每个名称。
5 4-输出一个 JSON 对象，其中包含以下键：English_summary, num_names。
6
7 请使用以下格式：
8 文本：<要总结的文本>
9 摘要：<摘要>
10 翻译：<摘要的翻译>
11 名称：<英语摘要中的名称列表>
12 输出 JSON：<带有 English_summary 和 num_names 的 JSON>
13
14 Text: <{text}>
15 """
16 response = get_completion(prompt_2)
17 print("\nprompt 2:")
18 print(response)
```

给出答案的选择：

```
1 prompt = f"""
2 以下用三个反引号分隔的产品评论的情感是什么？
3
4 用一个单词回答：「正面」或「负面」。
5
6 评论文本：```{lamp_review}```
7 """
8 response = get_completion(prompt)
9 print(response)
```

流式交互

由于llm生成结果太慢，所以一般来说采用流式的输出方式给到用户

agent执行器

Todo

评估方式

rag系统的评估主要基于三个维度

- 答案的可信度: 指该模型必须保持给定上下文的真实性, 确保答案与上下文信息一致, 并且不会偏离或矛盾, 评估的这一方面对于解决大型模型中的幻觉至关重要。
- 答案相关性: 生成的答案需要与提出的问题直接相关。
- 上下文相关性: 检索的上下文信息尽可能准确和有针对性, 避免不相关的内容

RAGAS

- 在评估答案的可信度上, 使用LLM将答案分解为单独的陈述, 并验证每个陈述是否与上下文一致。最终, “可信度分数”是通过将支持的陈述数量与陈述总数进行比较来计算的;
- 评估答案相关性上, 使用LLM生成潜在问题并计算这些问题与原始问题之间的相似度。答案相关性分数是通过计算所有生成的问题与原始问题的平均相似度得出;
- 评估上下文相关性上, 使用LLM提取与问题直接相关的句子, 并使用这些句子与上下文中句子总数的比率作为上下文相关性得分。

ARES

首先使用语言模型从目标语料库中的文档生成综合问题和答案, 以创建正样本和负样本, 然后使用合成数据集对轻量级语言模型进行微调, 以训练它们评估上下文相关性、答案忠实性和答案相关性, 最后使用置信区间对RAG系统进行排名。

私域大模型应用



私域大模型的应用实践

企业级RAG实践

几种实现架构对比

实现框架	应用语言	方案优势	方案劣势	实现难度	
spring-ai,spring6,spring-boot3	java 17+	spring架构体系, 学习和构建成本低, 微服务治理组件能复用 背靠spring, 社区活跃度大, 后续更新和排查问题方便	项目需要java版本和spring版本与现有技术栈有差别 目前还在snapshot不够稳定, 生态和功能没	简单	springBoot3更新点 spring-ai+postgres+chatGpt实践 升级springBoot3问题 1、资源使用情况, 日志, 监控, 微服务完全匹配现有架构 2、用户认证和授权使用Oauth2+jwt适配现有架构 4、对于高并发、多模型的场景如何适配使用:

		集成了分词器，编码器模型适配，大模型适配，向量数据库读写支持	有langChain丰富，一些深度优化的轮子可能得自己造		<p>延用现有技术方案，选择高性能向量数据库+分层索引+缓存支持向量匹配效率</p> <p>使用缓存+上下文prompt压缩整理降低对话token量级</p> <p>使用国内或者内网部署的开源llm模型提高响应速度</p> <p>多模型场景在应用侧开发网关api层，或开发路由组件根据问答场景和参数不同进行模型切换，采用模板方法+策略+builder的设计</p> <p>比如评估回答结果场景，参考langchain评价指标+用户行为写切面或者包装模式进行埋点或metric点上报</p>
langChain	python 3	社区活跃，框架主流，功能最丰富 集成了分词器，编码器模型适配，大模型适配，向量数据库读写支持，集成了上下文缓存，上下文裁剪，数据返回效果，流式返回等组件	跨语言交互，适配各种微服务治理中间件和调用链 整体服务器吞吐不如java服务器 维护成本过高，相当于独立于java的一整套技术栈	难	<p>1.注册和发现可以直接用nacos，负载均衡依托k8s和内部请求时选择nacos中注册的其他机器实现，监控依托k8s和内部采集器上报到时序数据库，日志同理spring-cloud使用open-feign能联通python注册到nacos的服务</p> <p>2、用户认证和授权使用Oauth2+jwt适配现有架构</p> <p>3.对于高并发和多模型，依托langchain框架内置插件+多协程实现</p> <p>4.二次开发</p> <p>扩展langchain的流程和新构建组件来实现扩展</p>
langChain4j	java 17+	相比spring-ai功能更丰富，同时也是可以接入spring框架体系中 集成了分词器，编码器模型适配，大模型适配，向量数据库读写支持	项目需要java版本和spring版本与现有技术栈有差别，各个组件学习成本较高	一般	
springBoot+one-api	java 8,go	spring版本和java版本能适配现有技术栈，各	跨语言交互，适配各种微服务治	较难	<p>1.微服务治理相关组件java侧复用，go服务需要注册和发现可以直接用nacos，负载均衡依托k8s和内部请求时选择nacos</p>

	个组件自由选择构建，灵活度高 go服务器吞吐性能高于java,占用资源更少	理中间件和调用链成本高 各个组件学习成本和构建成本较高	中注册的其他机器实现，监控依托k8s和内部采集器上报到时序数据库，日志同理 2.用户认证和授权使用Oauth2+jwt适配现有架构
--	--	--------------------------------	--

应用场景

1. 导入售后相关数据到向量库，实现智能客服回答售后相关问题，做不同车型不同用户在数据结构的区分，便于多次查询和多级索引构建
2. 导入办公相关制度规章制度，项目需求开发文档，项目技术文档，财务记账信息等结合钉钉实现办公快速查询问答助手
3. 对接日志系统，zipkin全链路系统，监控平台，发布管理平台结合钉钉机器人，快速定位问题，提高研发效率降低维护成本
4. 对接反爬虫系统，根据用户行为和网络地址进行分析决策，进行安全风控
5. 对接输出敏感词系统，提高过滤和识别效率
6. 爬虫拉取各大平台的车评，反馈信息，利用大模型进行信息摘要，情感分析，图表生成等，帮助项目管理和产品经理分析决策
7. 爬虫拉取各大新闻网站车圈或赛力斯相关报道，进行信息分类和摘要总结，辅助项目管理和产品经理
8. 辅助UI根据知识库中存储的关键信息进行图片设计，交互式了解产品特点

向量数据库对比

[向量数据库对比](#)

[各个向量数据库近邻检索的基准测试](#)

向量编码模型对比

[编码模型各场景基准测试](#)