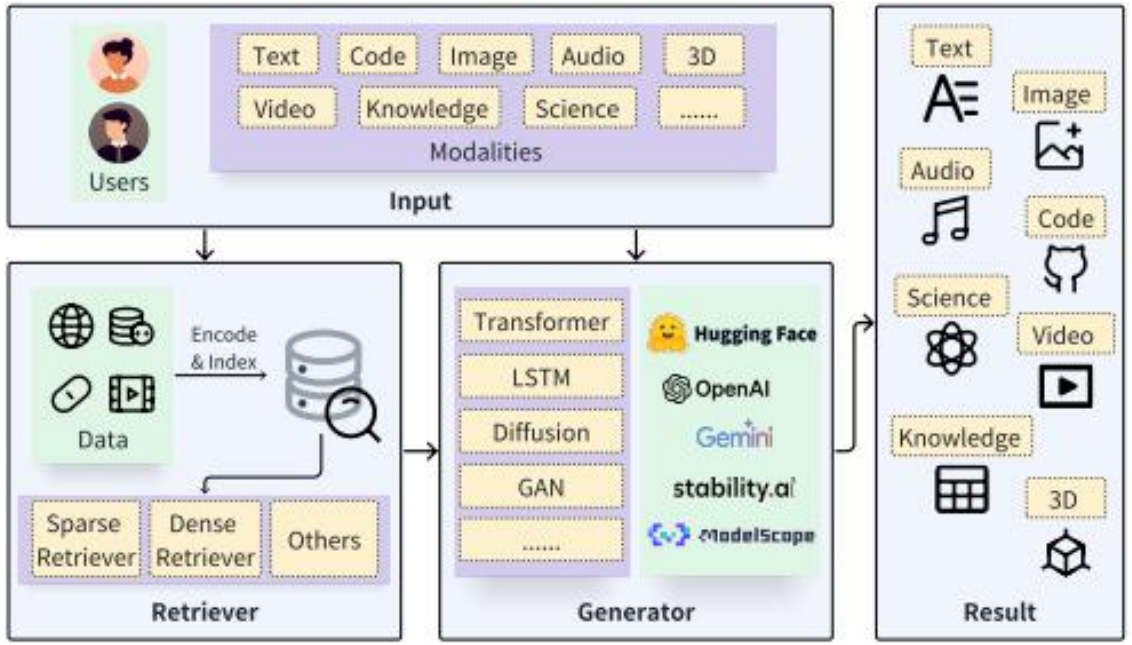
组会学习汇报

南京邮电大学

姓名：　　李文珂

目录

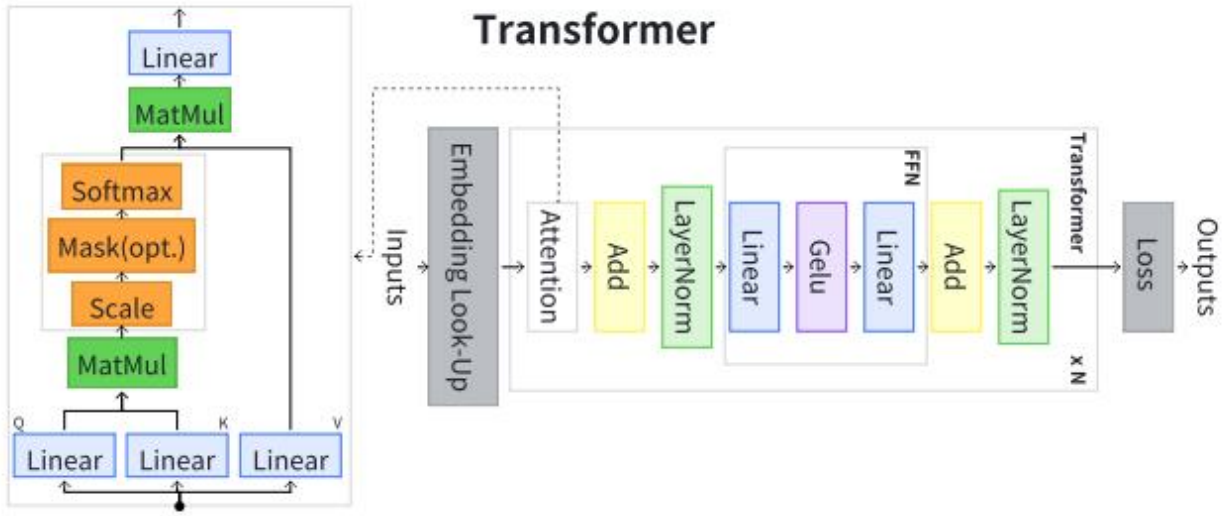
1.RAG基本结构



典型的RAG过程如图所示。给定一个输入查询，检索器定位并查找相关数据源，然后检索结果与生成器互动以增强整个生成过程。检索结果可以以不同方式与生成过程互动：它们可以作为生成器的增强输入；它们可以在生成的中间阶段作为潜在表示加入；它们可以以logits的形式对最终生成结果做出贡献；它们甚至可以影响或省略某些生成步骤。此外，在典型的基础RAG过程基础上，还提出了许多增强方法来提高整体质量。

1.RAG基本结构

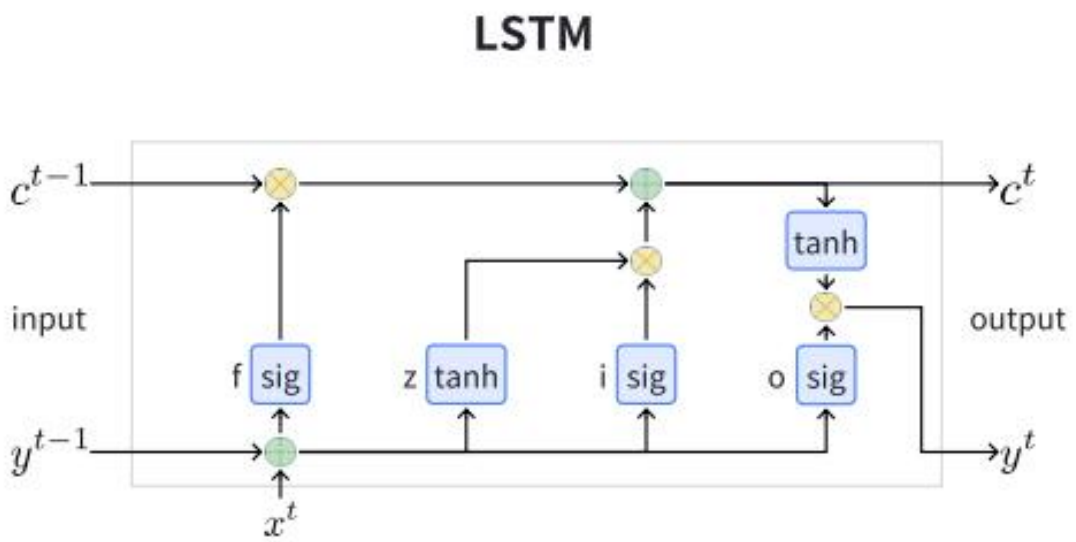
1.1生成器



Transformer模型是在自然语言处理领域中表现最优的模型之一，它包含了自注意力机制、前馈神经网络、层归一化模块以及残差网络。通过在每个步骤上对潜在表示进行词汇分类得到预测的概率分布。

1.RAG基本结构

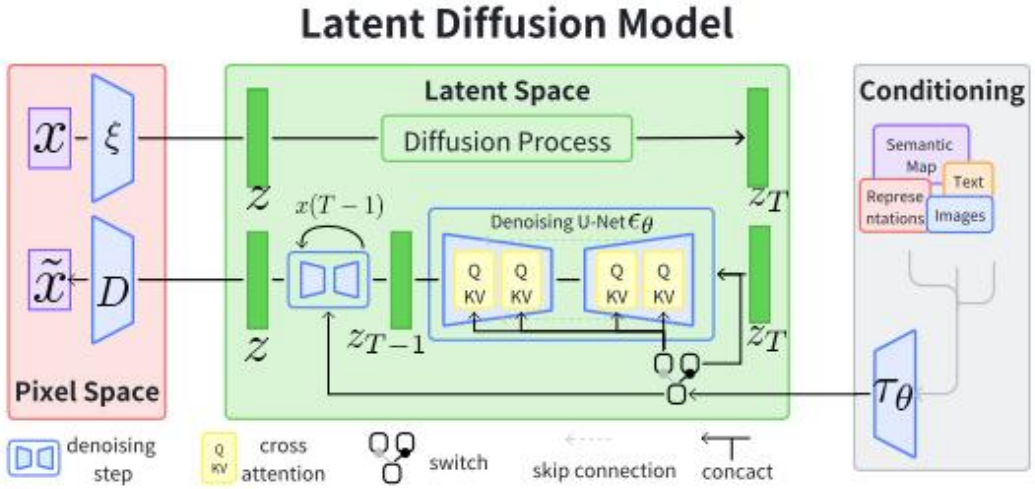
1.1生成器



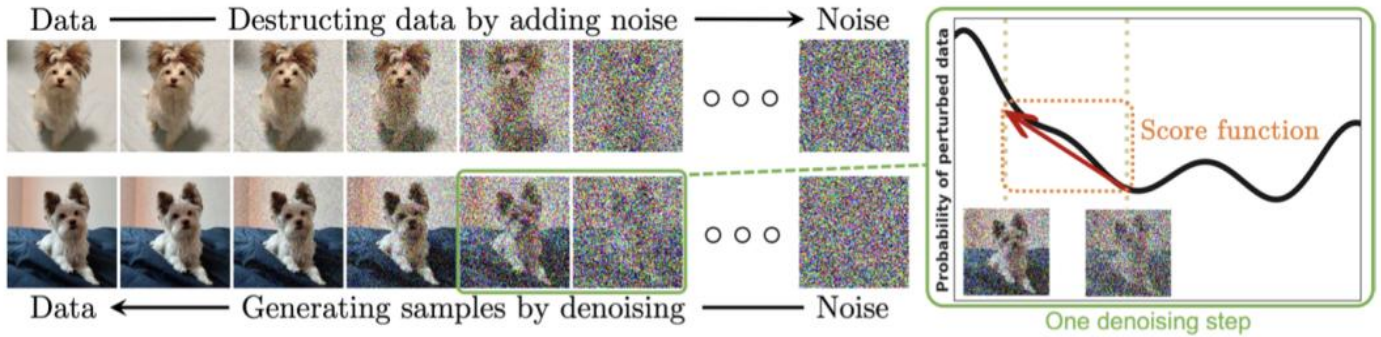
LSTM是一种特殊的循环神经网络，通过引入细胞状态和门控机制解决了在处理长期依赖信息时出现的梯度爆炸/消失问题。该模型包含三个门（输入门、遗忘门和输出门），它们起到信息过滤器的作用，以及一个核心模块——细胞状态，可以记忆和保持信息。

1.RAG基本结构

1.1生成器

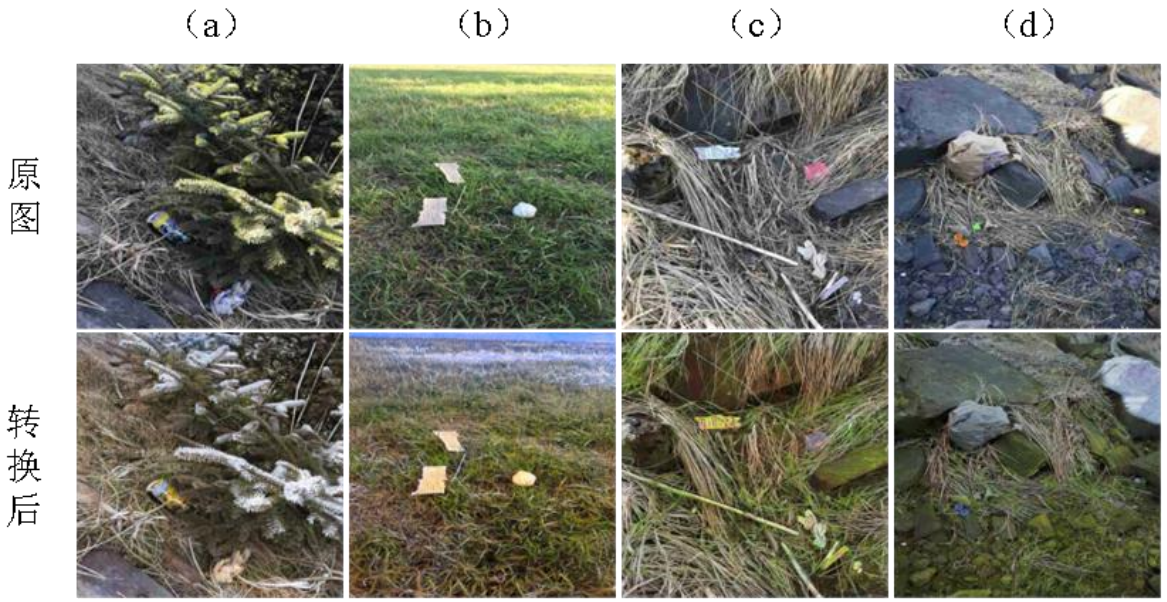


扩散模型是一类能够生成真实且多样化数据样本的深度生成模型，例如图像、文本、视频等。如图所示，扩散模型通过逐渐向数据添加噪声，直到变得随机，然后通过反向过程从噪声生成新数据。



1.RAG基本结构

1.1生成器



对抗网络可以生成逼真的图像和音频等数据，主要由生成器和判别器组成。图为使用CycleGAN将一些图片进行风格转换。

1.RAG基本结构

1.2检索器

检索器用于检索识别所需要的信息，将存储信息化为键值对。给定一个查询，检索器会使用相似度函数搜索最相似的前k个键，并获取配对的值。根据不同的相似度函数，检索器可以被分为不同类型。

1.稀疏检索

稀疏检索器常用于文本搜索，通常使用 TF-IDF，query likelihood和BM25等匹配方法。

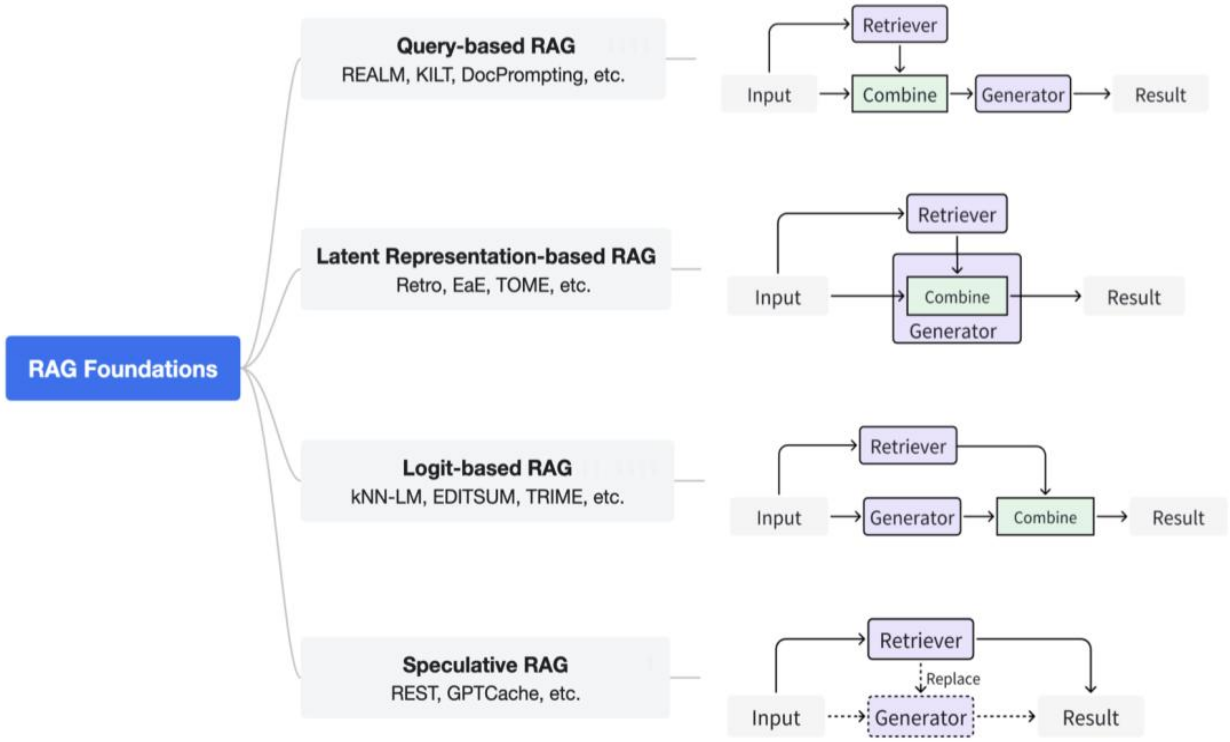
2. 密集检索

密集检索方法使用密集的嵌入向量来表示查询和键，并构建近似最近邻（ANN）索引来加速搜索过程。

3.其他

有些研究工作并不计算表示形式，而是直接使用自然语言文本之间的编辑距离，或者代码片段抽象语法树（AST）之间的差异。

2.RAG基本类型



1. 基于查询的RAG

将知识库查询结果直接融合到RAG中,提升问答系统的效果。

2. 基于潜在表示的RAG

在基于潜在表示的 RAG 框架中，生成模型与检索对象的潜在表示进行交互，从而增强模型的理解能力和生成内容的质量。

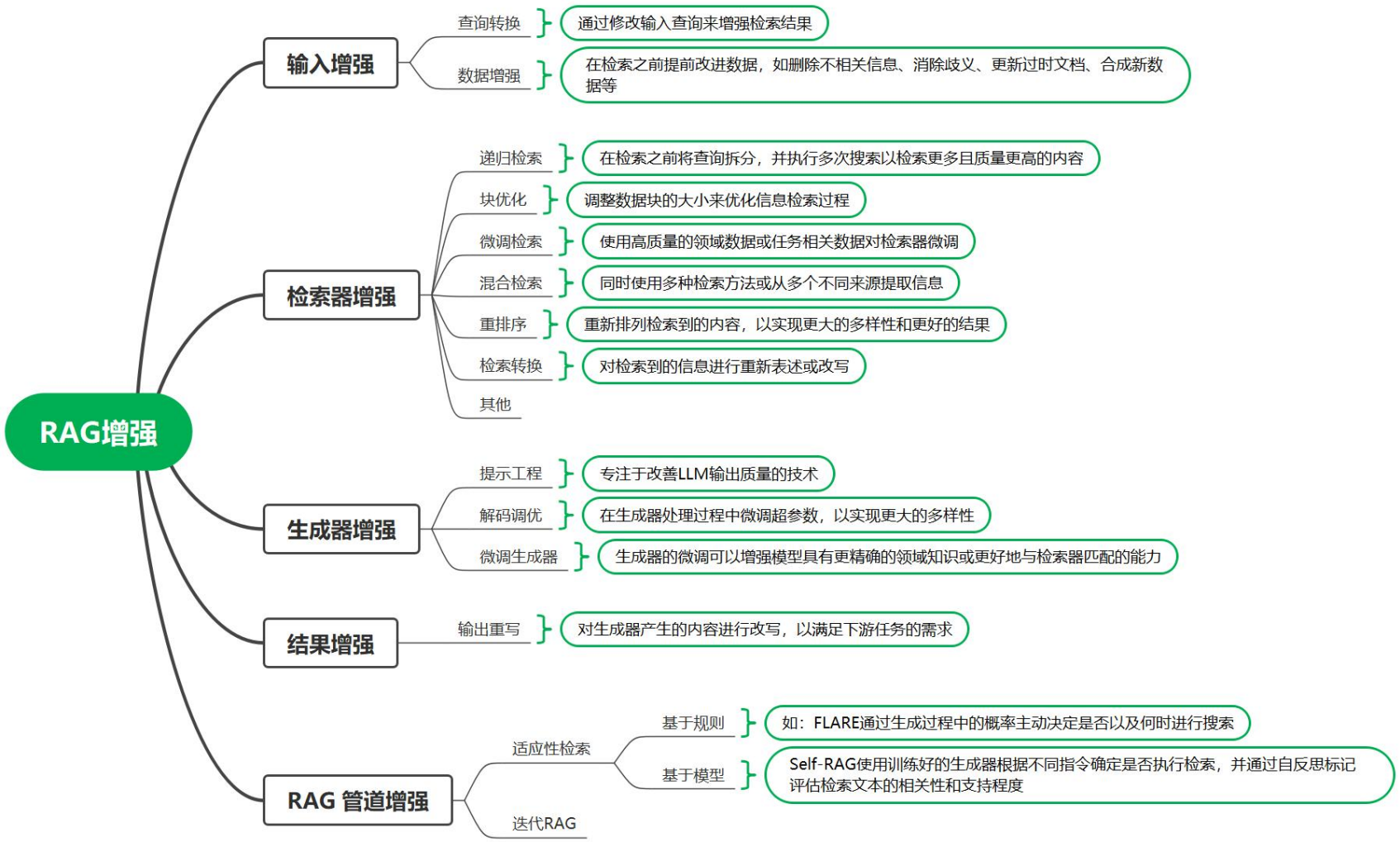
3. 基于对数几率的 RAG

在基于对数几率的RAG中，生成模型在解码过程中通过对数几率来整合检索信息。通常，对数几率通过简单的求和或是特定的模型来计算。

4. 推测性RAG

这种RAG寻找机会使用检索信息，而不是纯粹生成，以此节省资源。

3.RAG增强技术


RAG增强
输入增强
查询转换
通过修改输入查询来增强检索结果
数据增强
在检索之前提前改进数据，如删除不相关信息、消除歧义、更新过时文档、合成新数据等
检索器增强
递归检索
在检索之前将查询拆分，并执行多次搜索以检索更多且质量更高的内容
块优化
调整数据块的大小来优化信息检索过程
微调检索
使用高质量的领域数据或任务相关数据对检索器微调
混合检索
同时使用多种检索方法或从多个不同来源提取信息
重排序
重新排列检索到的内容，以实现更大的多样性和更好的结果
检索转换
对检索到的信息进行重新表述或改写
其他
生成器增强
提示工程
专注于改善LLM输出质量的技术
解码调优
在生成器处理过程中微调超参数，以实现更大的多样性
微调生成器
生成器的微调可以增强模型具有更精确的领域知识或更好地与检索器匹配的能力
结果增强
输出重写
对生成器产生的内容进行改写，以满足下游任务的需求
RAG 管道增强
适应性检索
基于规则
如：FLARE通过生成过程中的概率主动决定是否以及何时进行搜索
基于模型
Self-RAG使用训练好的生成器根据不同指令确定是否执行检索，并通过自反思标记评估检索文本的相关性和支持程度
迭代RAG

4.RAG技术应用


Applications
RAG for Text
Qusetion Answering
REALM, TKEGEN, Atlas, etc.
Fact verification
CONCRETE, etc.
Commonsense Reasoning
KG-BART, etc.
Human-Machine Conversation
BlenderBot3, ConceptFlow, Internet-Augmented-DG, etc.
Neural Machine Translation
TRIME, KNN-MT, etc.
Event Extraction
R-GQA, etc.
Summarization
RAMKG, Unlimiformer, etc.
RAG for Code
Code Generation
REDCODER, APICoder, COCOGEN, etc.
Code Summary
Rencos, Re2Com, EditSum, etc.
Code Completion
ReACC, RepoCoder, CoCoMic, etc.
Automatic Program Repair
RING, Codex, CEDAR, etc.
Text-to-SQL
XRICL, SYNCHROMESH, CodeICL, etc.
Others
StackSpotAI, E\&V, etc.
RAG for Audio
Audio Generation
Re-AudioLDM, Make-An-Audio, etc.
Audio Captioning
RECAP, etc.
RAG for Image
Image Generation
RetrieveGAN, IC-GAN,KNN-Diffusion, etc.
Image Captioning
RA-Transformer, Memory-augmented (MA), REVEAL, etc.
RAG for Video
Video Captioning
R-ConvED, CARE, EgoInstructor, etc.
Video QA\&Dialogue
MA-DRNN, VGNMN, etc.
Others
VidIL, RAG-Driver, etc.
RAG for 3D
Text-to-3D
ReMoDiffuse, AMD, RetDream, etc.
RAG for Knowledge
Knowledge Base Question Answering
ReTraCk, CBR-KBQA, GMT-KBQA, etc.
Knowledge-augmented Open-domain Question Answering
UniK-QA, KG-FID, OREOLM, etc.
Table for Question Answering
EfficientQA, Convinse, CORE, etc.
Others
Prototype-KRG, SURGE, RHO, etc.
RAG for Science
Drug Discovery
RetMol, PromptDiff, etc.
Biomedical Informatics Enhancement
PoET, Chat-Orthopedist, BIOREADER,etc.
Math Applications
LeanDojo, RAG-for-math-QA, etc.

5.RAG运行尝试

```
from haystack.document_stores import InMemoryDocumentStore
from haystack.utils import fetch_archive_from_http
import os
from haystack.pipelines.standard_pipelines import TextIndexingPipeline
from haystack.nodes import BM25Retriever
from haystack.nodes import FARMReader
from haystack.pipelines import ExtractiveQAPipeline
from haystack.utils import print_answers

document_store = InMemoryDocumentStore(use_bm25=True)
doc_dir = "data/build_your_first_question_answering_system"

fetch_archive_from_http(
        url="https://s3.eu-central-1.amazonaws.com/deepset.ai-farm-qa/datasets/documents/wiki_gameofthrones_txt1.zip",
        output_dir=doc_dir,
)

files_to_index = [doc_dir + "/" + f for f in os.listdir(doc_dir)]
indexing_pipeline = TextIndexingPipeline(document_store)
indexing_pipeline.run_batch(file_paths=files_to_index)

retriever = BM25Retriever(document_store=document_store)

reader = FARMReader(model_name_or_path="deepset/roberta-base-squad2", use_gpu=True)

pipe = ExtractiveQAPipeline(reader, retriever)
```

```
[11] prediction = pipe.run(
        query="Who is the father of Arya Stark?", params={"Retriever": {"top_k": 10}, "Reader": {"top_k": 5}}
    )
```

```
Inferencing Samples: 100%|██████████| 1/1 [00:02<00:00, 2.21s/ Batches]
```

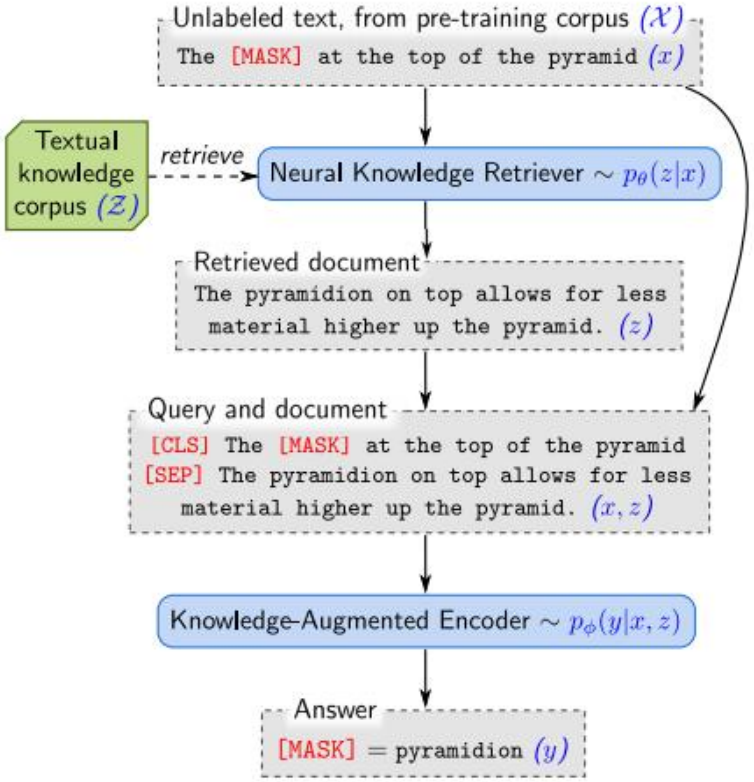
5.RAG运行尝试

```
from haystack.utils import print_answers

print_answers(prediction, details="minimum")    ## Choose from `minimum`, `medium`, and `all`
```

```
'Query: Who is the father of Arya Stark?'
'Answers:'
[   {   'answer': 'Eddard',
        'context': 's Nymeria after a legendary warrior queen. She travels '
                   "with her father, Eddard, to King's Landing when he is made "
                   'Hand of the King. Before she leaves,'},
    {   'answer': 'Ned',
        'context': 'k in the television series.\n'
                   '\n'
                   '====Season 1====\n'
                   'Arya accompanies her father Ned and her sister Sansa to '
                   "King's Landing. Before their departure, Arya's h"},
    {   'answer': 'Lord Eddard Stark',
        'context': 'rk daughters.\n'
                   '\n'
                   'During the Tourney of the Hand to honour her father Lord '
                   'Eddard Stark, Sansa Stark is enchanted by the knights '
                   'performing in the event.'},
    {   'answer': 'Ned',
        'context': ' girl disguised as a boy all along and is surprised to '
                   "learn she is Arya, Ned Stark's daughter. After the "
                   'Goldcloaks get help from Ser Amory Lorch and'},
    {   'answer': 'King Robert',
        'context': 'en refuses to yield Gendry, who is actually a bastard son '
                   "of the late King Robert, to the Lannisters.  The Night's "
                   'Watch convoy is overrun and massacr'}]
```

6.REALM



$$p(y \mid x) = \sum_{z \in \mathcal{Z}} p(y \mid z, x)\, p(z \mid x).$$

无论是预训练还是微调阶段，REALM都接收一些输入x，并学习输出y的可能性分布p(y|x)。在预训练阶段，x是从预训练语料库中抽取的一个句子，其中某些词被遮盖，模型需要预测这些缺失词；而在微调阶段，x是一个问题，y代表答案。

其中p(y|x)生成分为两个阶段：

检索：对于输入x，首先从知识语料库Z中检索可能有用的文档z。将其建模为分布p(z|x)的样本。

生成：以检索到的z和原始输入x为基础，生成输出y，建模为p(y | z,x)。为了获得生成y的总体可能性，将z视为一个潜在变量，并对所有可能的文档z进行边缘化。

6.REALM

$$p(z \mid x) = \frac{\exp f(x, z)}{\sum_{z'} \exp f(x, z')},$$
$$f(x, z) = \text{Embed}_{\text{input}}(x)^\top \text{Embed}_{\text{doc}}(z),$$

$$\text{Embed}_{\text{input}}(x) = \mathbf{W}_{\text{input}} \text{BERT}_{\text{CLS}}(\text{join}_{\text{BERT}}(x))$$
$$\text{Embed}_{\text{doc}}(z) = \mathbf{W}_{\text{doc}} \text{BERT}_{\text{CLS}}(\text{join}_{\text{BERT}}(z_{\text{title}}, z_{\text{body}}))$$

$$\text{join}_{\text{BERT}}(x) = \texttt{[CLS]}\, x\, \texttt{[SEP]}$$
$$\text{join}_{\text{BERT}}(x_1, x_2) = \texttt{[CLS]}\, x_1\, \texttt{[SEP]}\, x_2\, \texttt{[SEP]}$$

无论是预训练还是微调阶段，REALM都接收一些输入x，并学习输出y的可能性分布p(y|x)。在预训练阶段，x是从预训练语料库中抽取的一个句子，其中某些词被遮盖，模型需要预测这些缺失词；而在微调阶段，x是一个问题，y代表答案。

其中p(y|x)生成分为两个阶段：

检索：对于输入x，首先从知识语料库Z中检索可能有用的文档z。将其建模为分布p(z|x)的样本。

生成：以检索到的z和原始输入x为基础，生成输出y，建模为p(y | z,x)。为了获得生成y的总体可能性，将z视为一个潜在变量，并对所有可能的文档z进行边缘化。

6.REALM

$$p(y \mid z, x) = \prod_{j=1}^{J_x} p(y_j \mid z, x)$$

$$p(y_j \mid z, x) \propto \exp\left(w_j^\top \mathtt{BERT}_{\mathtt{MASK}(j)}(\mathtt{join}_{\mathtt{BERT}}(x, z_{\mathrm{body}}))\right)$$

$$p(y \mid z, x) \propto \sum_{s \in S(z,y)} \exp\left(\mathtt{MLP}\left(\left[h_{\mathtt{START}(s)}; h_{\mathtt{END}(s)}\right]\right)\right)$$

$$h_{\mathtt{START}(s)} = \mathtt{BERT}_{\mathtt{START}(s)}(\mathtt{join}_{\mathtt{BERT}}(x, z_{\mathrm{body}})),$$

$$h_{\mathtt{END}(s)} = \mathtt{BERT}_{\mathtt{END}(s)}(\mathtt{join}_{\mathtt{BERT}}(x, z_{\mathrm{body}})),$$

在预训练阶段，模型要预测每个掩码背后的词的真实值，使用的公式如下，其中BERTMASK为transformer为被掩盖的词生成的嵌入向量，wj表示被掩盖的词yi对应的被学习到的嵌入向量，Jx表示被掩盖的词的总数。

在微调阶段的开放问答中，模型需要生成答案字符串，假设答案y能够从文档z中找到，设S(z,y)为z中与y相匹配的片段的集合。则求p(y | z, x)的公式如下，其中BERTSTART(s)和BERTEND(s)对应S(z,y)的开始和结束标记的Transformer输出向量，而MLP表示前馈神经网络。

6.REALM

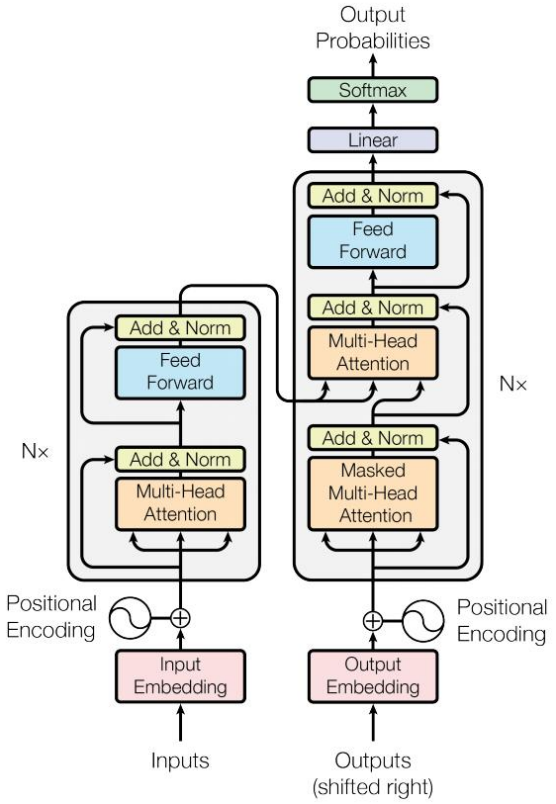
$$\nabla \log p(y \mid x) = \sum_{z \in \mathcal{Z}} r(z) \nabla f(x, z)$$

$$r(z) = \left[\frac{p(y \mid z, x)}{p(y \mid x)} - 1 \right] p(z \mid x).$$

在训练和微调阶段，使用最大化正确输出y的对数似然logp(y|x)进行训练，计算p(y|x)涉及到Z中所有文档z进行求和，为了简化计算，通过最大内积搜索计算近似的前k个文档。

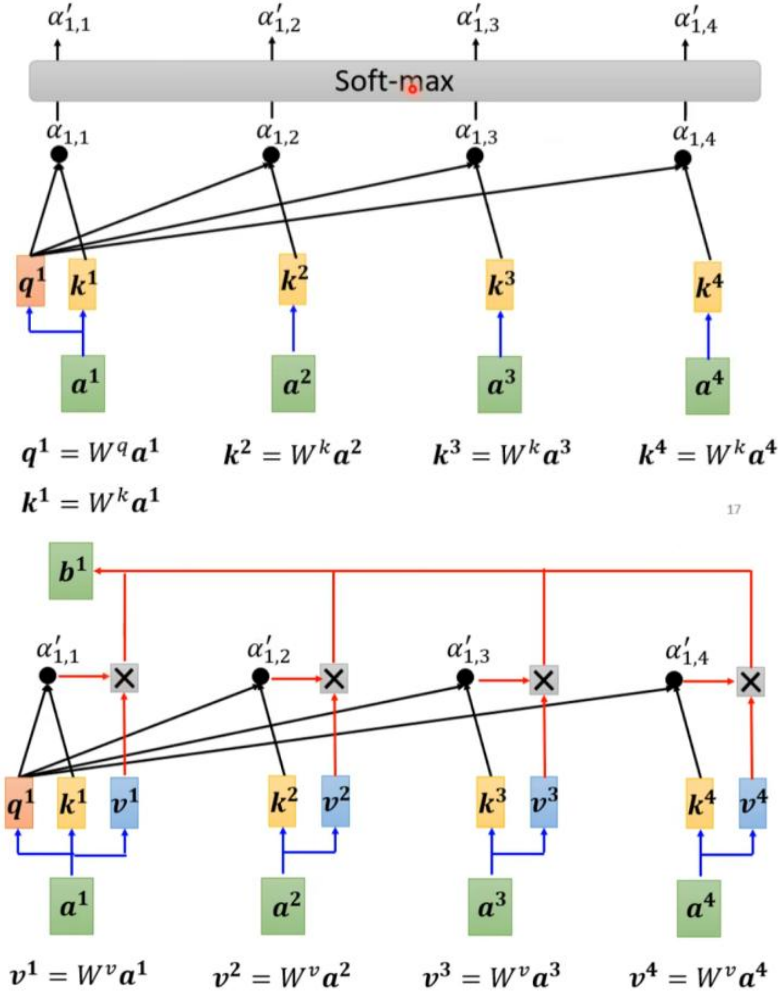
最后，文档根据r(z)改变得分f(x,z)，为正，如果r(z)为正则增加得分，如果为负，则减少得分。

7.Transformer



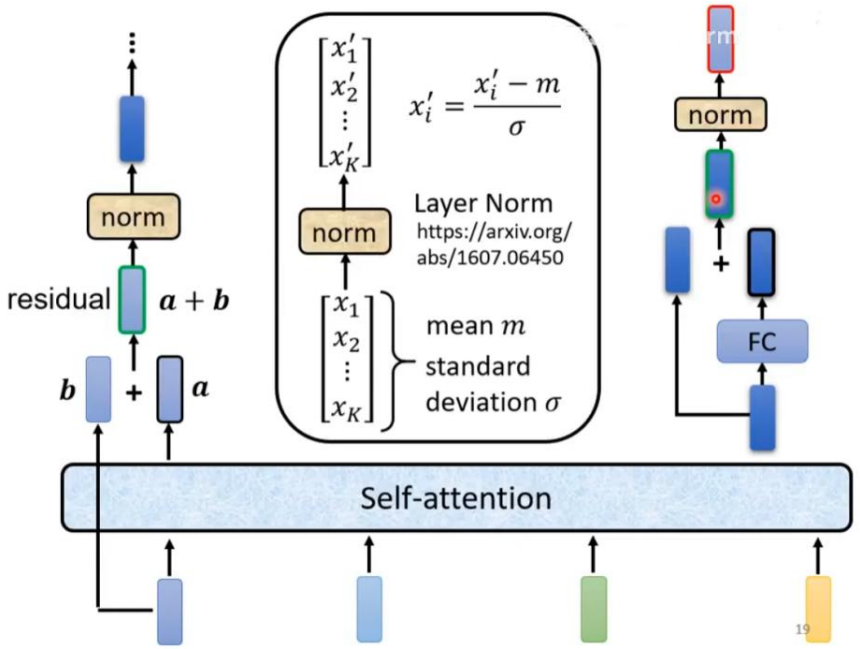
Transformer结构如下图所示，主要由编码器和解码器两部分组成。编码器由6个相同的层堆叠而成，每层有两个子层。第一层是一个多头的自注意力机制，第二层是一个简单的、位置上的全连接前馈网络。解码器也是由6个相同层的堆栈组成。除了每个编码器层的两个子层之外，解码器还插入了第三个子层，它对编码器堆栈的输出进行多头注意力。同时解码器中的注意力机制还进行了屏蔽，使得解码器只关注当前位置及之前位置的预测。

7.Transformer

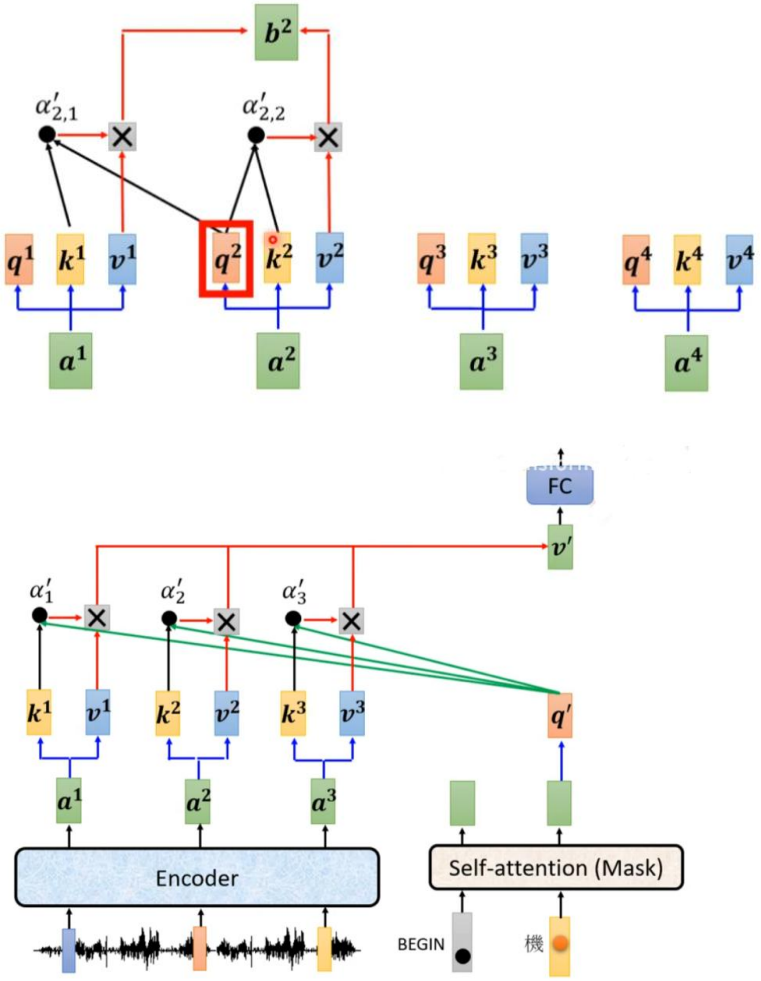


设a1-a4是模型的输入向量或者其他层的输出向量，Wq，Wk是可以学习的矩阵，以a1为例，输入向量a1与矩阵Wq相乘得到Query值，各向量ai与Wk相乘获得值Key，最后由Query与其他Key值做点积，获得a1,i即注意力得分，最后将得到的得分通过softmax层，获得最终的得分a`1,i。

求出a`1,i后，可以进一步求出与a1相关程度最高的部分，将输入向量ai与可学习矩阵Wv相乘，得到新向量vi，将vi与之前的a`1,i相乘后相加，即可得到最终结果，其中与a1关联程度最高的部分在最终结果中占比就越大。



输入向量经过注意力机制得到输出，与残差链接传来的输入向量相加后经过LayerNorm函数，再经过全连接层与输入相加后经过LayerNorm函数得到最终输出。



Cross Attention模块用于编码器和解码器之间传递资讯，当解码器接收到开始向量时，经过Masked Attention生成向量，再与矩阵相乘生成Query，Query与编码器各向量生成的Key做点积，再通过softmax函数等生成注意力分数，再将注意力分数与编码器输出的各向量生成的vi相乘在相加，得到v，通过全连接层等进行进一步处理，过程如图所示。

7.Transformer

```
text = "Jim Henson was a puppeteer"
```

```
['[CLS]', 'jim', 'henson', 'was', 'a', 'puppet', '##eer', '[SEP]', '[PAD]', '[PAD]']
```

```
{'input_ids': tensor([[101, 3958, 27227, 2001, 1037, 13997, 11510, 102]]),
'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1]])}
```