embedding model

RAG本身的原理并不难理解，但要将其推广到生产环境中则会面临多方面的挑战。这主要是因为 RAG 系统涉及多个不同的组件，每个组件都需要精心设计和优化。本文讨论的就是其中embedding的模块，也就是相关doc召回时的粗排阶段。

在本文中，将主要讨论以下几个问题：

- embedding模型的架构
- embedding模型的评测基准 MTEB
- 如何选择合适的embedding模型

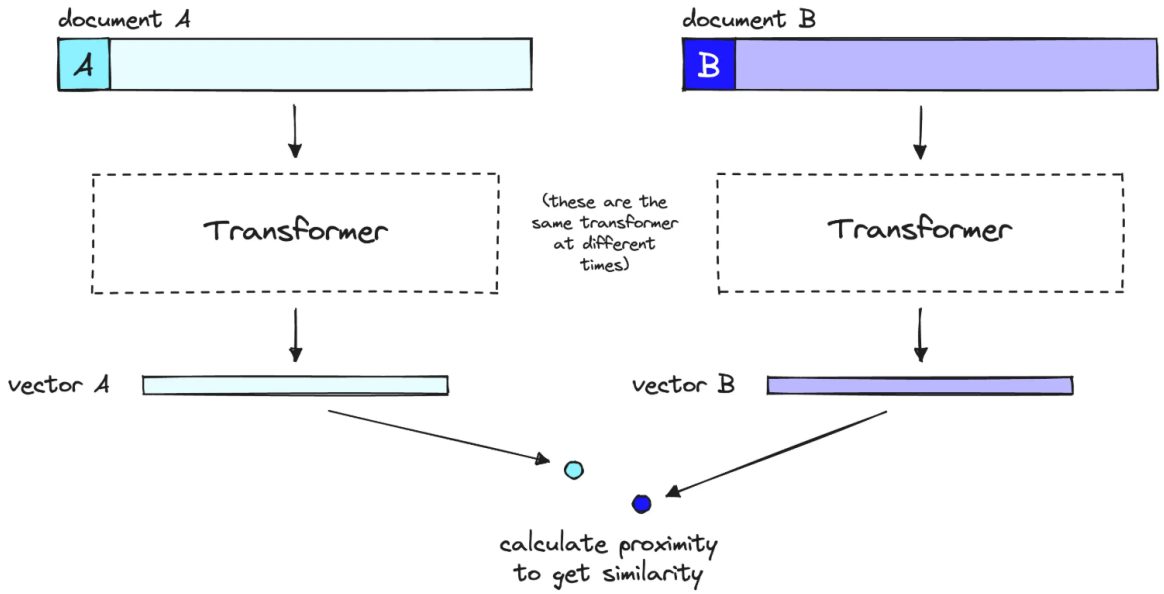
embedding模型的架构

在RAG框架中，常见的两种用于doc召回的embedding模型是双编码器（Bi-Encoder）和稀疏嵌入模型（Sparse Embedding Models）。

双编码器（Bi-Encoder）

双编码器的基本思想是使用两个独立的encoder来分别处理query和doc（或候选doc），然后将它们嵌入到相同的向量空间中。在检索阶段，query和doc会被转化为固定长度的向量表示，然后通过计算query向量和doc向量之间的相似度来进行匹配。

- **工作方式**：query和doc分别通过两个相同的encoder处理，每个编码器将输入转化为一个embedding。这两个embedding向量在同一个向量空间中表示它们的语义信息，之后根据相似度（例如余弦相似度）来判断查询与文档之间的相关性。
- **优点**：这种方法的优势在于它具有较高的计算效率，因为查询和文档的编码是独立进行的，适合用于大规模数据集。通常，使用双编码器进行检索时，检索过程会非常快速。



稀疏嵌入模型（Sparse Embedding Model）

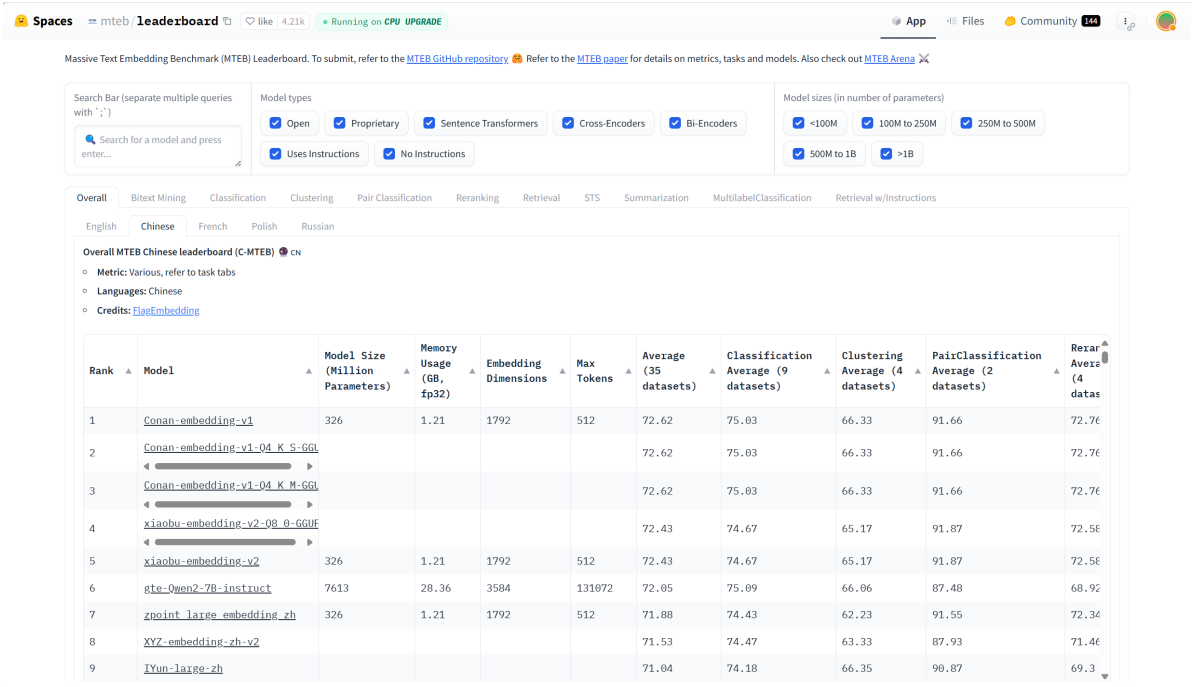
稀疏嵌入模型则是一种不同于密集嵌入（dense embedding）的模型，通常基于传统的词袋模型（如TF-IDF）或稀疏编码技术。这些模型生成的嵌入是稀疏的，意味着嵌入向量中大多数元素的值是零，仅有少量非零元素。

- **工作方式**：在稀疏嵌入模型中，文本的表示通常不是通过密集的向量（如BERT生成的嵌入向量）来表示，而是通过一种稀疏表示，其中很多维度的值为零，只在少数维度上有较高的值。这种稀疏表示通常是通过词频或其他特征的权重计算得到的，常见的实现包括基于词频的向量化方法（如TF-IDF、bm25）和一些稀疏编码方法（如LDA等）。
- **优点**：稀疏嵌入模型往往计算效率较高，并且可以避免高维密集向量所带来的计算开销，特别是在大型文档库的检索中。此外，稀疏表示有时能捕捉到更加显著的词汇特征，适用于特定的检索任务，如关键词匹配等。

embedding模型的评测基准 MTEB

MTEB（Massive Text Embedding Benchmark）是衡量文本嵌入模型（Embedding模型）的评估指标的合集，是目前业内评测文本向量模型性能的重要参考。

可以在huggingface上找到对应的leaderboard：



C-MTEB(Chinese Massive Text Embedding Benchmark)则是专门针对中文文本向量的评测基准，被认为是目前业界最全面、最权威的中文语义向量评测基准之一，涵盖了分类、聚类、检索、排序、文本相似度、STS等7个经典任务，共计35个数据集，为深度测试中文语义向量的全面性和可靠性提供了可靠的实验平台。

对于国内开发者而言，我们更加会专注C-MTEB。不过但也只能作为一个参考，这些模型在公开数据集上的 benchmark 在垂直领域、企业自身的业务领域不一定成立，具体选择哪个向量模型还需结合业务特点进行综合比较、权衡。

如何选择合适的embedding模型

可以从以下几个角度考虑：

1. **语言支持和性能**：大部分开源向量模型只支持单一或者有限的文本语言，所以需要确保 Embedding 模型支持的语言种类。多语言模型如 OpenAI Embedding 和 bge-m3 等模型能够处理多种语言。bge-m3 支持 100 多种语言，适合多语言需求的场景。另外，某些模型在主要语言（如中文）中的表现较好，但在处理较少使用的语言时可能会表现不佳。因此，需要评估模型在所有必需语言中的准确性，以确保一致的性能。
2. **处理长文本的能力**：切分的文本片段后续需要通过 Embedding 模型进行向量化，所以必须考虑向量模型对输入文本块的 tokens 长度限制，超出这个限制则会导致模型对文本进行截断，从而丢失

信息，影响下游任务的性能。不同的 Embedding 模型对文本块长度的支持能力不同。比如，BERT 及其变体通常支持最多 512 个tokens，处理长文本时则需要将文本分成更小的块，意味着需要更加精细化的分块策略。而 Jina AI 的 Embedding 模型和 bge-m3 模型则支持 8K 的 tokens 输入，适合处理长文本块。

3. **模型在特定领域的表现**：通用 Embedding 模型在特定垂直领域（如医学、法律和金融等）可能不如专用模型有效。这些领域通常需要专门训练 Embedding 模型来捕捉特定的专业术语和语境。为特定业务需求优化的 Embedding 模型能够显著提升检索和生成的质量。例如，通过结合向量检索和重排序（reranking）技术，可以进一步优化结果。
4. **存储和内存等资源需求**：高维向量需要更多的存储空间，这可能会带来长期成本。例如，较高维度的模型如 text-embedding-ada-002 需要更多的存储资源。另外，较大的模型可能会占用更多内存，因此不适合内存有限的设备。
5. **模型响应时间**：Embedding 模型的处理速度在实时应用中尤为关键。例如，intfloat/e5-base-v2 模型在处理速度上表现优异，但需要在 GPU上 运行以达到最佳性能。在选择模型时，需要评估其在嵌入和检索过程中的延迟。例如，OpenAI 的 Embedding 模型在许多基准测试中显示出较高的性能和较低的延迟。

通用的 Embedding 模型通常是在大规模、多样化的数据集上训练的，可能不完全适合特定领域的任务，比如医学、法律等专业领域，它们无法很好的理解一些专有词汇。如果模型在业务数据集上表现不能满足预期，可以通过微调，让模型学习到特定领域的词汇和概念，使其在特定应用场景中表现更佳。

你了解哪些embedding模型

bge v1

BGE，全称BAAI General Embedding，是智源研究院提出的开源通用向量模型，在过去短短一年时间内，在huggingface上总下载量已超数亿次，是目前下载量最多的国产AI系列模型。

论文：C-Pack: Packed Resources For General Chinese Embeddings

bge训练的3个阶段：

（1）预训练：用Wudao纯文本语料训练，利用了RetroMAE，重建污染的编码向量；

（2）弱监督学习：用C-MTP无标签数据集训练，对比学习从负样本学习中如何区分出成对的文本；

（3）有监督微调：用C-MTP有监督数据集训练，由于标签数据是多任务的，所以加入了指令微调实现多任务下的微调。

BGE v1由6个模型组成，每种语言有'large', 'base'和'small'三款不同规模的模型。用户可根据需求平衡挑选更大能力更强的模型，或更小速度更快的模型。

模型	语言	参数量	模型大小	描述	基座模型
BAAI/bge-large-en	英语	500M	1.34 GB	将文本转化为向量的向量模型	BERT
BAAI/bge-base-en	英语	109M	438 MB	基础规模模型，能达到与 bge-large-en 近似的能力	BERT
BAAI/bge-small-en	英语	33.4M	133 MB	大幅减小模型规模，依旧有很好的表现	BERT
BAAI/bge-large-zh	中文	326M	1.3 GB	将文本转化为向量的向量模型	BERT
BAAI/bge-base-zh	中文	102M	409 MB	基础规模模型，能达到与 bge-large-zh 近似的能力	BERT
BAAI/bge-small-zh	中文	24M	95.8 MB	大幅减小模型规模，依旧有很好的表现	BERT

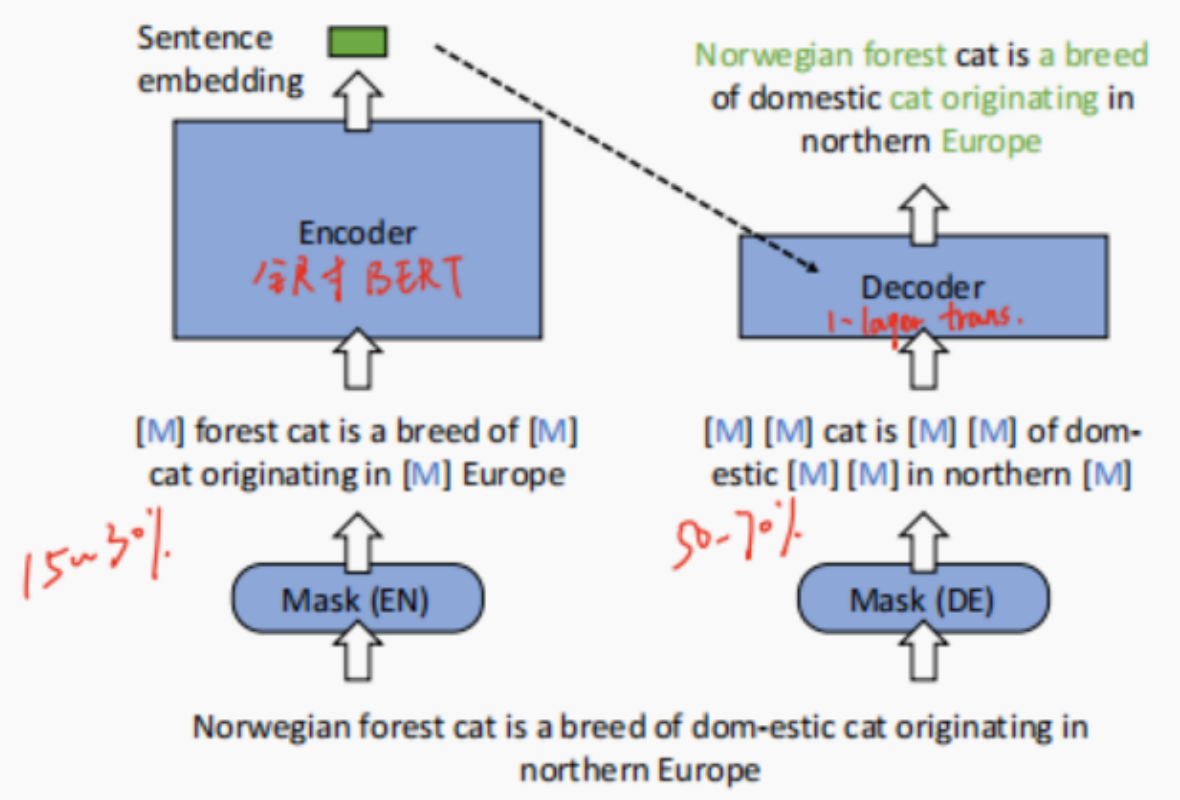
后面又出了v1.5版本，主要缓解了相似度分布问题，并提升无指令情况下的检索能力。

下面分别对训练的每个阶段做详细的分析：

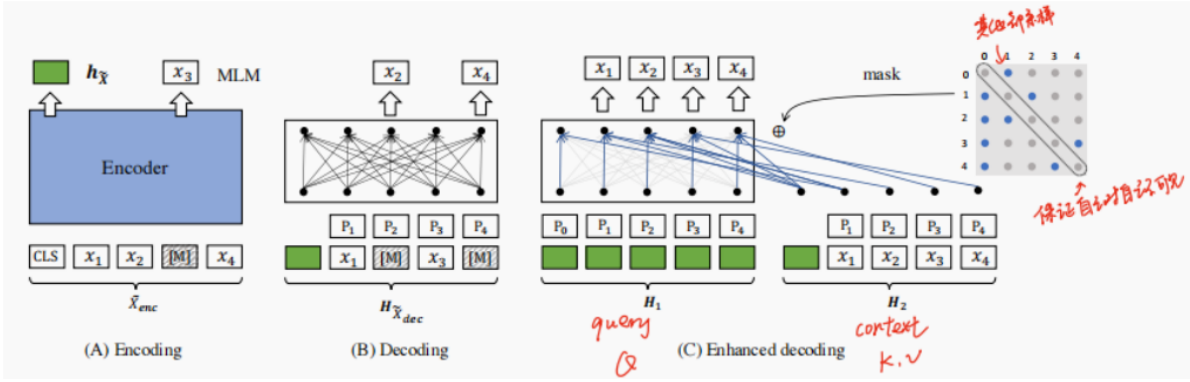
一、预训练

预训练阶段用的是2022年EMNLP上提出的RetroMAE。RetroMAE由两个主要部分组成：Encoder和Decoder。首先，输入文本经过掩码处理后被送入Encoder部分，Encoder使用BERT结构来处理输入并获取最终的`[CLS]`标记对应的隐层表示，这个表示作为整个句子的向量（绿色小方块）。然后，Decoder接收两个输入：一是经过掩码的句子，二是从Encoder中提取的`[CLS]`隐层表示，作为句子向量。Decoder部分是一个单层的Transformer，用于根据掩码输入和句子向量进行重建。

在训练过程中，模型的损失函数由两部分组成：一是Encoder的MLM（Masked Language Modeling）损失，负责通过掩码预测来学习语言的语法和语义；二是Decoder的重建损失，负责根据掩码的输入和Encoder的句子向量来恢复原始句子。整个模型的优化目标是同时最小化这两种损失。



工作流程如下：

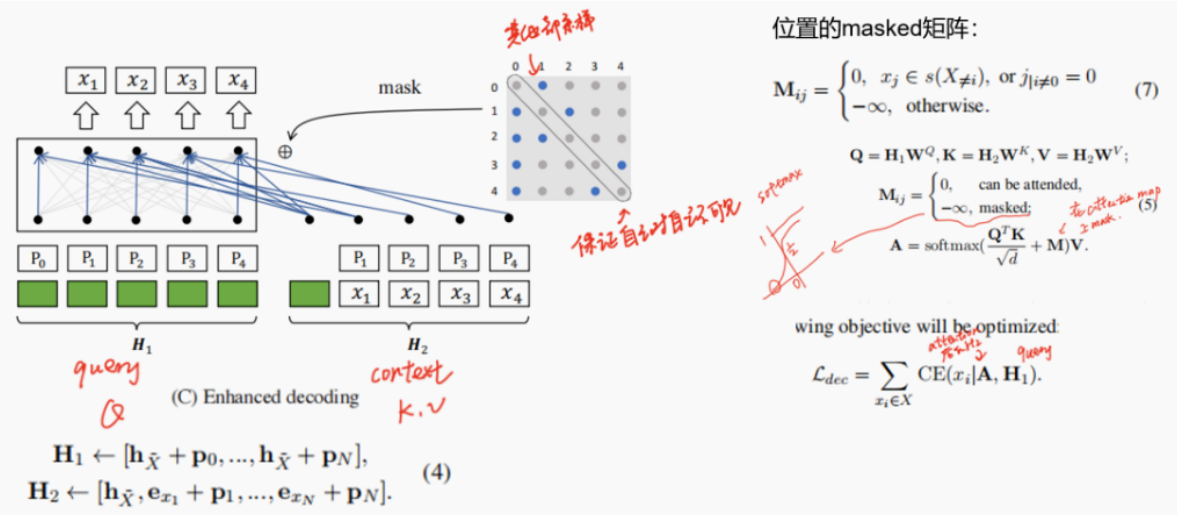


通过三步构建了一个逐步细化的自监督学习框架：首先通过Encoder获取句子的嵌入表示，然后通过Decoder利用该句子表示来重建掩码部分，最后通过增强编码阶段进一步利用上下文信息和句子嵌入进行全局恢复。

RetroMAE的各组件包括：

- **Encoder**：输入的掩码率为15%到30%。Encoder部分使用全尺寸的BERT，并执行**MLM任务**，通过该任务获取`[CLS]`标记的隐层表示作为句子的表示。
- **Decoder**：输入的掩码率为50%到70%。Decoder仅由一层**Transformer**组成，负责执行**重建任务**，即还原掩码部分的内容。

为了进一步提升性能，作者对Decoder进行了优化。具体来说，作者认为每个掩码token基于相同的上下文进行重建过于单一，因此引入了**双流自注意力**（two-stream self-attention）和**位置关注掩码**（position-specific attention mask）。简单来说，Q（查询）和KV（键值）分别处理不同的上下文信息，然后在计算attention时，通过mask机制使得每个掩码token能够使用不同部分的上下文。这样做的目的是增加模型的复杂度，并提升其泛化能力。如下图所示：



$$\mathbf{H}_1 \leftarrow [\mathbf{h}_{\tilde{X}} + \mathbf{p}_0, ..., \mathbf{h}_{\tilde{X}} + \mathbf{p}_N],$$
$$\mathbf{H}_2 \leftarrow [\mathbf{h}_{\tilde{X}}, \mathbf{e}_{x_1} + \mathbf{p}_1, ..., \mathbf{e}_{x_N} + \mathbf{p}_N]. \quad (4)$$

$$\mathbf{M}_{ij} = \begin{cases} 0, & x_j \in s(X_{\neq i}), \text{ or } j_{|i\neq 0} = 0 \\ -\infty, & \text{otherwise.} \end{cases} \quad (7)$$

$$\mathbf{Q} = \mathbf{H}_1\mathbf{W}^Q, \mathbf{K} = \mathbf{H}_2\mathbf{W}^K, \mathbf{V} = \mathbf{H}_2\mathbf{W}^V;$$
$$\mathbf{M}_{ij} = \begin{cases} 0, & \text{can be attended,} \\ -\infty, & \text{masked;} \end{cases} \quad (5)$$
$$\mathbf{A} = \text{softmax}(\frac{\mathbf{Q}^T\mathbf{K}}{\sqrt{d}} + \mathbf{M})\mathbf{V}.$$

$$\mathcal{L}_{dec} = \sum_{x_i \in X} \text{CE}(x_i | \mathbf{A}, \mathbf{H}_1).$$

Q：为什么编码器和解码器是非对称的?

A： 从论文创新的角度来看，这种设计是经过深思熟虑的。传统上，BERT主要用于MLM（Masked Language Modeling）和NSP（Next Sentence Prediction）任务，通过获取`[CLS]`向量或最后一层隐层的输出，能捕捉一定的语义信息，作为语义嵌入用于文本检索等任务。自监督学习除了MLM，还包括对比学习和自编码器等方法，但对比学习的应用较多，并且对负采样有较高要求。现在，RetroMAE的创新点在于将自编码器的思想引入BERT，利用自编码器作为特征表征学习的方式，进而提升模型的性能。

BERT本身作为一个强大的编码器，而通过加一个解码器，采用自编码的方式去牵引原本的BERT模型，使得特征表征能更好地学习和优化，相当于为原有的BERT任务添加了一个新的学习目标——即"重建掩码输入"。这本质上是一个**多任务学习**的问题，其中一个任务是学习有效的语义嵌入，另一个任务则是通过解码器恢复原始输入，从而强化特征表征的学习。

从掩码率和模型的尺寸上，我们可以看到对解码器的要求非常严格，解码器的设计较为紧凑。通过这种方式，RetroMAE强迫编码器学习更高质量的特征表征，因为解码器的学习任务相对复杂，它要求编码器提供更精准的表示来进行有效的重建。因此，整个模型设计是为了迫使编码器在学习过程中更加注重语义表征的精度。

总的来说，RetroMAE将编码器和解码器设计为非对称结构，正是为了通过多任务学习的方式，提升模型的表征学习能力。

需要注意的是，最终的embedding（如BGE）主要来自于Encoder的`[CLS]`标记的最后隐层表示。

二、弱监督学习

在对比学习中，我们的目标是通过无标签数据集让模型学习区分正负样本的能力。具体来说，给定一个正文本对 p 和 q，以及一个负样本 q'，我们希望通过一个损失函数使得正文本对的相似度高于负样本对的相似度，从而提升模型的区分能力。这个目标的损失函数可以表示为：

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(e_p, e_q)/\tau)}{\exp(\text{sim}(e_p, e_q)/\tau) + \exp(\text{sim}(e_p, e_{q'})/\tau)}$$

其中：

- $\text{sim}(e_p, e_q)$表示正样本对 p 和 q 的相似度，通常使用点积或余弦相似度来计算；
- τ 是温度参数，控制平滑程度；
- e_p 和 e_q 分别是正样本 p 和 q 的嵌入向量。

从抽象的角度来看，这个损失函数与 **softmax** 类似，正文本对的相似度占所有文本对相似度之和的比例越大，损失越小。温度参数 τ 的作用是调节这种比例的平滑程度：当 τ 越大时，概率分布越平滑，这会降低正样本对之间的区分度，帮助提升模型的泛化能力，减少过拟合。

难负样本采样

在对比学习中，负样本采样的质量对模型性能至关重要。特别是**难负样本**的采样非常重要，因为如果所有负样本都很容易区分，那么模型的损失会很小，梯度也会很小，导致模型收敛慢，且在复杂语义场景下，学习到的表示可能无法有效地区分正样本和难负样本。

然而，在本研究中，作者并未专门设计难负样本采样方法，而是采用了**in-batch负采样**（in-batch negative sampling）。具体来说，假设一个batch中有 m 条文本，作者为每条文本 p配对了一个正样本 q，即相关文本。这样，在一个batch中，除了正样本对 (p, q) 外，其他文本都与 p 无关，因此可以作为负样本 q' 来参与训练。这样一来，负样本集合就是这个batch中所有不与 p 相关的文本。

为了增加难负样本被包含的概率，作者使用了一个较大的batch size，设定为 19200，这样可以增加batch中包含难负样本的可能性。通过这种**in-batch负采样**方式，模型能够在每个batch中同时学习正样本和负样本，且由于batch size较大，能够确保难负样本得到有效利用，从而提升模型的区分能力并加速收敛。

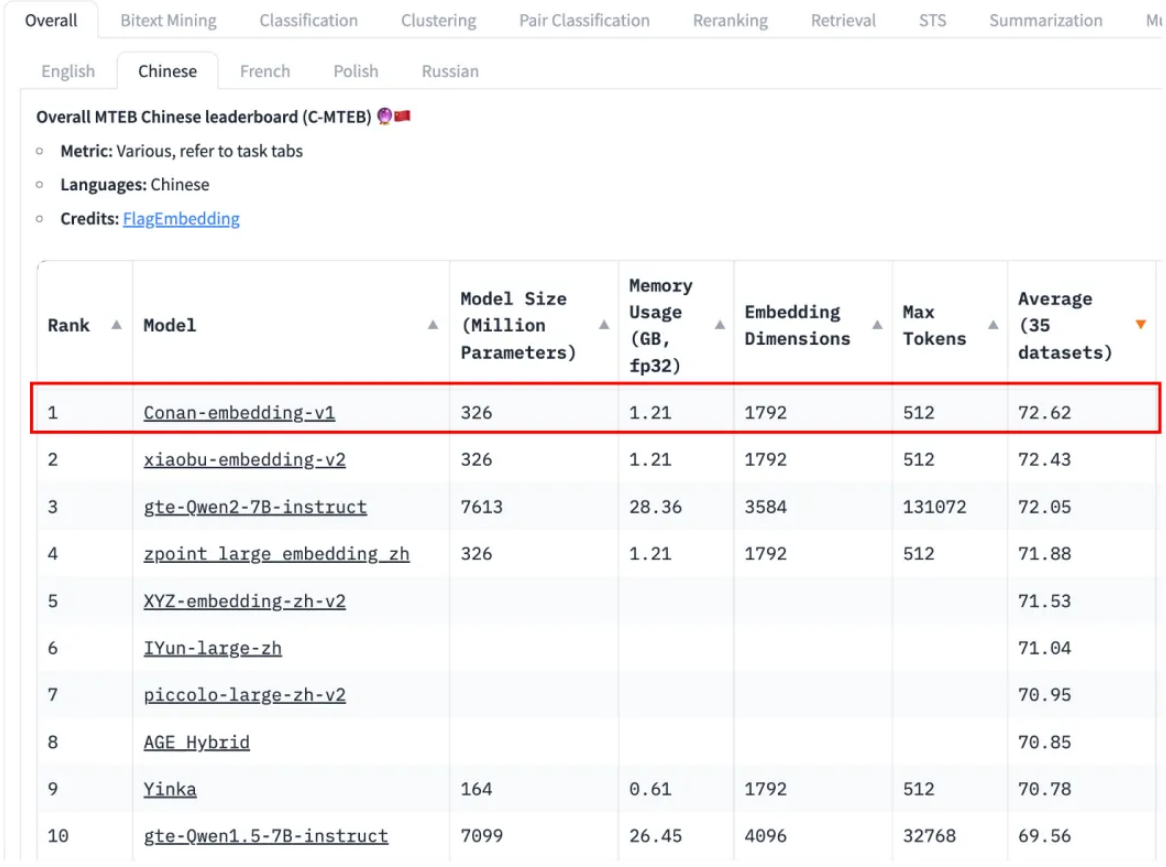
三、有监督微调

由于标签数据含有不同有监督任务的数据，所以加入了指令微调实现多任务下的微调。具体做法是在query前加入指令做微调，比如：指令“search relevant passages for the query” + query。此外，除了in-batch负采样，还采用ANN-style采样策略从给定任务的原始语料中挖掘难负样本。具体来说，ANN索引（近似最近邻索引）被用来找到与正样本相似度较高但仍属于负样本的文档，从而进一步增加负样本的难度，帮助模型更好地区分正负样本，提升训练效果。

Conan-Embedding

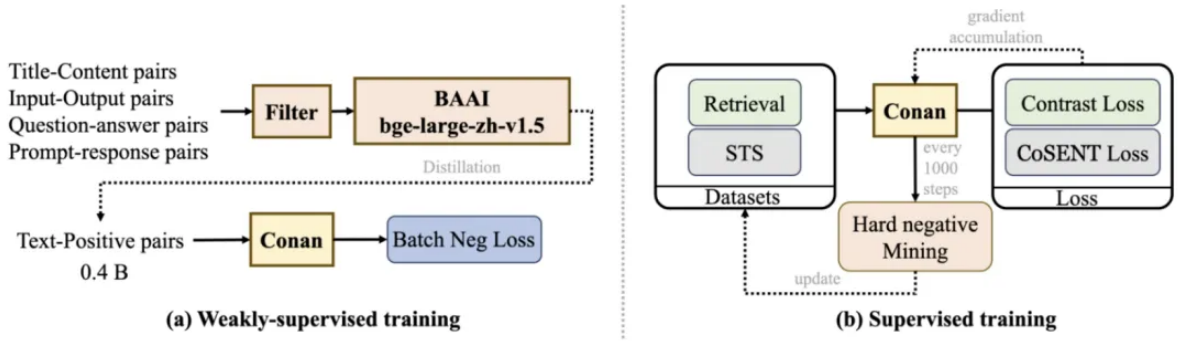
最近在C_MTEB霸榜的embedding模型，该工作来自腾讯。

论文：Conan-embedding: General Text Embedding with More and Better Negative Samples



Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average (35 datasets)
1	Conan-embedding-v1	326	1.21	1792	512	72.62
2	xiaobu-embedding-v2	326	1.21	1792	512	72.43
3	gte-Qwen2-7B-instruct	7613	28.36	3584	131072	72.05
4	zpoint_large_embedding_zh	326	1.21	1792	512	71.88
5	XYZ-embedding-zh-v2					71.53
6	IYun-large-zh					71.04
7	piccolo-large-zh-v2					70.95
8	AGE_Hybrid					70.85
9	Yinka	164	0.61	1792	512	70.78
10	gte-Qwen1.5-7B-instruct	7099	26.45	4096	32768	69.56

训练过程主要分为两个阶段：弱监督预训练和有监督微调



一、弱监督预训练

在预训练阶段，收集了7.5亿的文本对，参考了Internlm2.5中描述的标准数据过滤方法，通过以下四步进行过滤：

1、通过文档提取和语言识别进行格式化处理；

2、在基于规则的阶段，文本会经过规范化和启发式过滤；

3、通过MinHash方法进行去重；在安全过滤阶段，执行域名阻止、毒性分类和色情内容分类；

4、在质量过滤阶段，文本会经过广告分类和流畅度分类，以确保输出文本的高质量。

通过过滤，筛选了约 4.5 亿对数据，留存率约60%。

然后使用bge-large-zh-v1.5对数据进行评分，过滤掉得分低于0.4的低质量数据，最终筛选出4亿对数据。

预训练的方法和bge类似，采用InfoNCE loss，将in-batch内的其他文本对作为负样本进行训练。

二、有监督微调

在这个阶段针对不同的下游任务进行微调，将训练数据分为retrieval（非对称型）和STS（对称型）两种任务类型，其中retrieval任务使用InfoNCE loss，STS任务使用CoSENT loss。

这个阶段还使用了两种优化技巧：动态难负例挖掘训练、跨GPU的Batch均衡训练

动态难负例挖掘训练（Dynamic Hard Negative Mining，Dynamic-HNM）

Embedding模型的训练通常依赖对比学习，其关键在于正负例的选择质量。难负例（Hard Negatives）是与Query有一定相关性但与正例的区分较难的负例，能有效提高模型的对比损失效率。

传统的负例挖掘多在数据预处理阶段完成，这意味着负例是固定的。随着训练的进行，模型的**权重更新**可能导致这些固定负例对模型来说变得不再困难，从而降低训练效率。

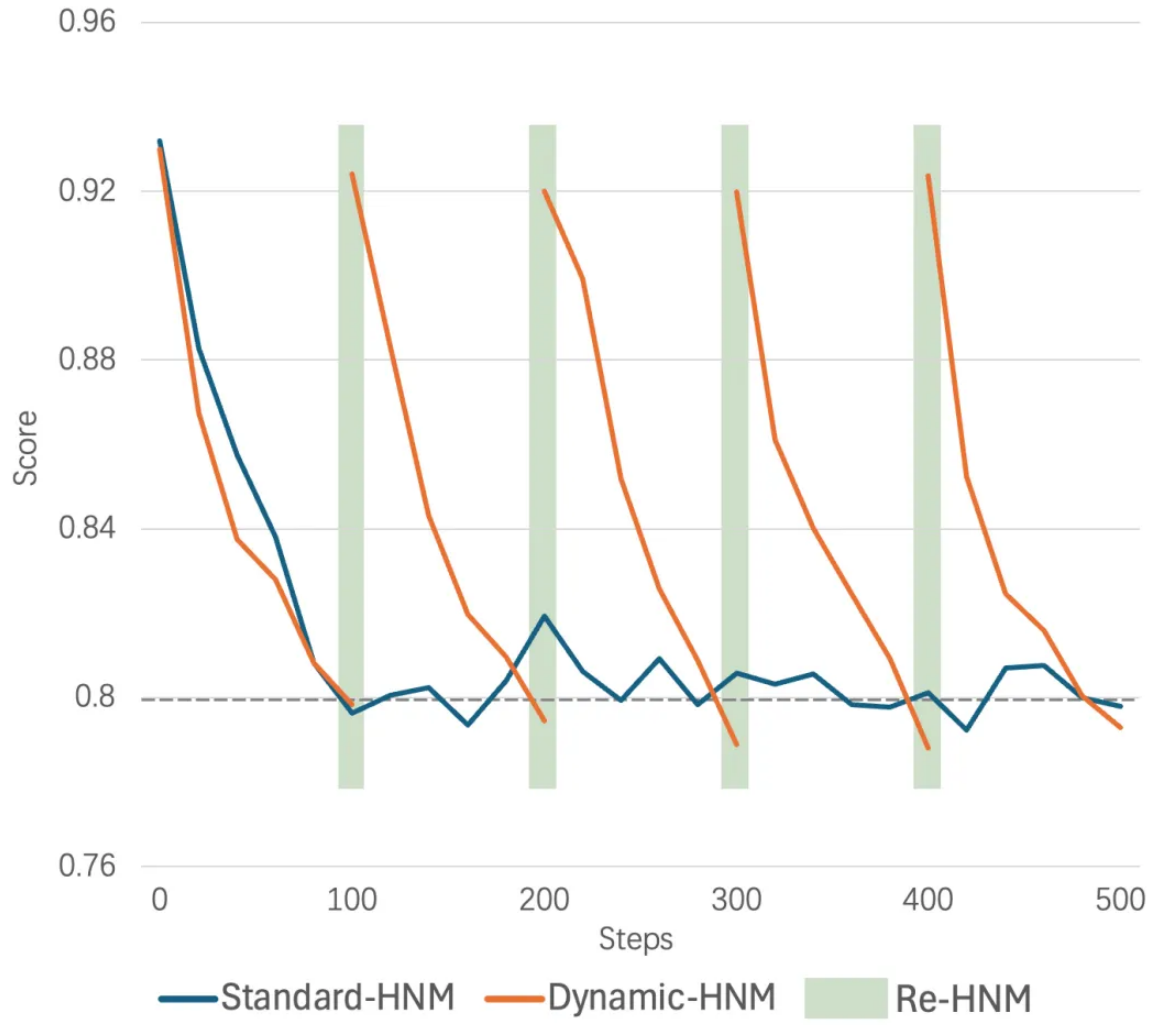
为解决上述问题，Dynamic-HNM在训练过程中动态调整负例，主要步骤如下：

1. **基于Teacher模型初始化难负例**：
    - 使用预训练Teacher模型为Query选择初始负例，这些负例需满足“有一定相关性但区分度低”的条件。
2. **动态更新机制**：
    - 在每次权重更新后，记录当前负例与Query的相似性得分（例如Cosine相似度）。
    - 每隔一定迭代步数（如100步），根据以下规则检测负例是否需要替换：
        - 若负例与Query的相似性得分的1.15倍小于初始得分，且绝对值低于0.8，则判定该负例“不再困难”。
    - 替换规则：使用最新的模型权重重新挖掘负例。
3. **负例替换方案**：

    - 每次替换使用区间中的案例（如第$(i-1)\times n+10$到$i\times n+10$个案例），确保负例质量与多样性。其中i表示第次替换，n表示每次使用的难负例数
4. **低成本实现**：
    - 通过简单的Score更新与筛选逻辑，动态挖掘的成本仅相当于一个训练Step的计算代价。

下图展示了Dynamic-HNM和Standard-HNM的正负例相似性得分随训练步数的变化曲线：

- Standard-HNM：负例得分在早期下降后趋于震荡，表明模型已"学会"这些负例。
- Dynamic-HNM：在负例学习完成后自动替换新负例，使负例得分继续下降。



跨GPU的Batch均衡训练（Cross-GPU Batch Balancing, CBB）

这个优化点主要是通过优化任务训练流程和样本利用率，从而提高训练稳定性和模型性能。

将训练数据分为两种任务类型：

- Retrieve任务：检索任务，通过对比学习优化Query和正/负样本的表示关系。使用InfoNCE loss。
- STS任务：语义文本相似性任务，通过监督学习训练模型，使其能够量化句子对之间的语义相似性。使用CoSENT loss。

传统的做法通常是在**顺序随机任务训练**中，每个训练Step只处理一个任务（如iter0处理STS任务，iter1处理Retrieve任务）。这种任务分配方式存在以下问题：

- **单任务优化方向不一致**：单次优化方向可能与Embedding模型的全局优化目标偏离，导致梯度震荡和收敛困难。
- **负样本利用不足**：Retrieve任务中，负样本数量受单GPU计算能力限制，无法充分挖掘更多难负例。

正对这两个问题CBB策略主要从以下两方面优化：

跨任务均衡

每个训练Step同时引入所有任务的Loss，确保优化方向与全局目标更一致。在单次Forward-Loss-Backward-Update中，计算所有任务的Loss并合并。

例如：Retrieve任务，从多个GPU中收集负样本，计算对比损失；STS任务，在另一个GPU上计算STS任务的Loss。最后将所有Loss汇总，计算全局梯度并更新模型权重。

跨GPU负例共享

多个GPU共享相同的Query和正样本（确保一致性），但每个GPU有不同的负样本。例如：4个GPU分别处理不同的负样本集，并计算对应的对比Loss，汇总后，整合所有负样本信息，提高训练效率和样本利用率。

loss函数如下：

$$\mathcal{L}_{\text{CBB}} = -\frac{1}{n}\sum_{i}\log\frac{\exp(s(x_i, y_i^+)/\tau)}{\exp(s(x_i, y_i^+)/\tau) + \sum_{k=1}^{N}\sum_{j=1}^{n}\exp(s(x_i, y_j^-)/\tau)} + \beta \times \mathcal{L}_{\cos}$$

- $s(q, y)$：Query和样本之间的相似性评分函数，通常为余弦相似度。
- τ：温度缩放参数，控制对高相似性样本的敏感度。
- N：Query q和正样本共享的GPU数量。
- β：权重因子，控制两个任务在总Loss中的占比（经验值为0.8）。

下图展示了CBB策略的具体流程，包括：

- 多个GPU如何分配负样本。
- 各任务如何在单次Iter中计算Loss并合并。