大语言模型中的知识检索与嵌入技术综述

徐善若 5 月 20 日学习笔记

Contents

1 引言 **2**

2 信息检索技术 **2**

2.1 稀疏检索 2

2.2 稠密检索 2

2.2.1 近似最近邻（ANN）检索概述 2

2.2.2 图索引：HNSW 算法 2

2.2.3 聚类索引：IVF 算法 3

2.3 检索效果评估与指标 3

2.4 检索结果融合（RRF） 3

3 嵌入表示技术 **3**

3.1 Word2Vec 静态词嵌入 3

3.2 BERT 上下文嵌入 3

4 模型部署与私有化实践 **3**

4.1 向量检索框架：Faiss 与 Milvus 3

4.2 神经搜索引擎：Jina 框架 3

4.3 私有化部署考虑 4

5 总结 **4**

1 引言

大型语言模型（LLM）在纯粹依赖参数学习知识时存在 “封闭书本” 的局限，模型回答的知识范围通常受训练语料限制，且更新成本高。这催生了知识检索与检索增强生成（Retrieval–Augmented Generation, RAG）等技术，通过在生成答案时实时检索外部知识来提高答案的准确性和时效性[1]。在这种架构中，检索阶段的质量至关重要：如果检索器无法找到相关文档，LLM 的回答精确度将很低且更易产生幻觉。因此，为了充分发挥检索的作用，我们需要系统了解信息检索的方法（稀疏 vs. 稠密）、嵌入向量表示技术，以及相关的索引算法和部署方案。

本文将对这些知识点进行综述分类，搭建一个清晰的技术架构体系，并结合关键公式、算法示例和实践框架进行说明。

2 信息检索技术

信息检索（Information Retrieval）旨在从海量数据中找到满足用户查询的信息。根据匹配信号的不同，可将检索方法分为**稀疏检索**（sparse retrieval）和**稠密检索**（dense retrieval）两大类。前者基于显式关键词匹配，典型方法如 TF–IDF 与 Okapi BM25；后者利用嵌入向量度量语义相似度，通过向量索引快速定位相关文档。

2.1 稀疏检索

稀疏检索使用离散的词项特征来匹配查询和文档，其索引通常是倒排表结构。例如 BM25（Best Match 25）是一种广泛应用的稀疏排序函数，被众多搜索引擎采用为默认排名策略。其得分可写为

$$\mathrm{BM25}(q,d) = \sum_{t \in q} IDF(t) \cdot \frac{tf_{t,d}\,(k_1+1)}{tf_{t,d} + k_1\big(1 - b + b \cdot \frac{|d|}{\mathrm{avgdl}}\big)},$$

其中 $tf_{t,d}$ 为词项 t 在文档 d 中的频次，$|d|$ 为文档长度，k_1, b 为可调超参。

稀疏检索可解释性强、实现相对简单，但无法衡量词语的语义相似度，对表述变化较大的自然语言问题往往力不从心。

2.2 稠密检索

稠密检索利用分布式向量表示（embedding）进行语义级匹配，将查询和文档分别映射至向量空间并计算余弦相似度、欧氏距离或内积等度量。但在百万量级以上的向量库中做全局比对开销巨大，故实际系统采用**近似最近邻**（Approximate Nearest Neighbor, ANN）算法加速。

2.2.1 近似最近邻（ANN）检索概述

ANN 通过构建高效数据结构，减少查询时需要比对的向量数量，以牺牲极少精度换取显著速度提升。衡量其效果常用**召回率**：

$$\mathrm{Recall} = \frac{\text{找到的真值近邻数}}{\text{总真实近邻数}}. \tag{1}$$

2.2.2 图索引：HNSW 算法

Hierarchical Navigable Small World (HNSW) [5] 通过分层近邻图结构实现对数级查询复杂度。高层图节点稀疏、连边长，可做全局跳跃；底层图节点密集、连边短，负责局部精细搜索。查询流程为 “自顶向下的贪婪爬山 + 层内 KNN”，可在高维空间取得优异的召回–速度平衡。

[1] 相关讨论见 Lewis (2020) 等。

2.2.3 聚类索引：IVF 算法

Inverted File Index (IVF) 首先对数据向量做 k-means 聚类得到 n_{list} 个簇中心；向量被分配至最近中心构成倒排链表。查询向量仅在 *nprobe* 个最近中心对应的链表内比对，大幅减少计算量。IVF 常与 Product Quantization (PQ) 结合，实现存储压缩与进一步加速。

2.3 检索效果评估与指标

除召回率（公式 1）外，常用**精准率** (Precision)、**平均准确率** (AP)、**平均倒数排名** (MRR) 等指标。MRR 公式：

$$\text{MRR} = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\text{rank}_i},$$

其中 rank_i 为第 i 个查询第一个相关文档的排名。

2.4 检索结果融合（RRF）

互惠排序融合（Reciprocal Rank Fusion, RRF）[1] 将多个检索器排名列表合并，其得分为：

$$\text{RRF}(d) = \sum_{r\in R}\frac{1}{k + r(d)}, \quad k = 60. \tag{2}$$

R 为检索器集合，$r(d)$ 为文档 d 在检索器 r 中的名次。RRF 简单易实现，却在多种场景接近或优于复杂学习融合方法。

3 嵌入表示技术

3.1 Word2Vec 静态词嵌入

Tomas Mikolov *et al.* 于 2013 年在 Google 提出 Word2Vec [3, 4]，通过 CBOW 与 Skip–Gram 两种无监督预测任务学习词向量，可捕捉类比关系：

$$\text{vec}(\text{king}) - \text{vec}(\text{man}) + \text{vec}(\text{woman}) \approx \text{vec}(\text{queen}).$$

3.2 BERT 上下文嵌入

BERT [2] 采用双向 Transformer 通过 Masked LM 预训练，生成上下文敏感的向量。经微调或 Sentence–BERT、SimCSE 等派生模型优化，可直接用于稠密检索场景。

4 模型部署与私有化实践

4.1 向量检索框架：Faiss 与 Milvus

Faiss [6] 由 Meta AI 开源，提供 GPU/CPU 高效索引；Milvus [7] 是云原生向量数据库，支持持久化、分布式部署与 SQL 风格查询，两者均可私有化运行。

4.2 神经搜索引擎：Jina 框架

Jina [8] 提供微服务级流水线编排，天然支持多模态向量检索与容器化部署，可快速构建端到端 RAG 系统。

4.3 私有化部署考虑

- **模型选择：**优先开源模型（LLaMA2、GPT-NeoX 等）或蒸馏压缩模型，避免敏感数据外传。
- **硬件资源：**需评估内存、GPU 显存与并发需求；可通过量化、剪枝降低推理成本。
- **数据流程安全：**文档分段、向量化、索引更新均在内网执行；结合日志与反馈机制防止违规输出。

5 总结

本文综述了大语言模型时代的知识检索与嵌入核心技术：从稀疏 BM25 到稠密 ANN 检索（HNSW、IVF），从 Word2Vec 静态词向量到 BERT 上下文嵌入；同时探讨了检索评估指标、RRF 融合策略以及 Faiss/Milvus/Jina 等私有化部署方案。通过掌握并实践这些技术，开发者可构建安全、高效且可扩展的检索增强大模型系统。

进一步阅读与实践建议：

- 深入阅读 HNSW [5] 与 BERT [2] 原论文，理解算法与模型细节；
- 在小规模数据集上动手实现 “ANN + RRF + LLM” 的简易 RAG 系统；
- 关注向量数据库新特性（如集成 LlamaIndex / LangChain）以应对持续增长的知识库需求。

References

[1] Nick Craswell, et al. *Reciprocal Rank Fusion Outperforms Condorcet and Bayesian Combination Methods.* SIGIR, 2009.

[2] Jacob Devlin, et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* NAACL, 2019.

[3] Tomas Mikolov, et al. *Efficient Estimation of Word Representations in Vector Space.* arXiv:1301.3781, 2013.

[4] Tomas Mikolov, et al. *Distributed Representations of Words and Phrases and their Compositionality.* NIPS, 2013.

[5] Y. A. Malkov, et al. *Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs.* TPAMI, 2018.

[6] J. Johnson, et al. *Billion-scale similarity search with GPUs.* arXiv:1702.08734, 2017.

[7] Milvus: https://milvus.io

[8] Jina AI: https://github.com/jina-ai/jina