

PolarDB与大模型结合：提升RAG效果的关键技术与实践

欢迎各位技术爱好者参加本次关于检索增强生成（RAG）技术与PolarDB结合的技术分享。在人工智能与数据库技术快速发展的今天，如何有效地将大型语言模型与结构化数据存储结合，成为了许多开发者关注的焦点。

本次分享将系统介绍提升RAG效果的关键技术，从数据分段、向量化到混合检索策略，并通过真实案例展示PolarDB在RAG应用中的优势与实践方法。我们将深入探讨如何利用先进的数据库技术与大模型能力，构建更高效、更精准的知识检索系统。

希望通过这次分享，能够帮助大家掌握RAG技术的核心要点，并在实际项目中灵活应用这些技术，提升应用的智能化水平。

 作者：**digoal zhou**



提升RAG效果的关键手段一览



提升RAG效果需要从多个维度同时发力。首先，文本分段策略是基础，合理的分段能确保检索单元的语义完整性和相关性。其次，通过大模型提取QA pairs和标签，可以丰富检索入口和语义理解维度。

在检索技术方面，需要综合应用全文检索、模糊查询、标签检索和向量检索等多种手段，形成互补。特别是向量检索能捕捉语义相似性，而rerank则能进一步优化检索结果的排序，提升最终匹配精度。

这些技术的有机结合，构成了一个完整的RAG优化体系，能够显著提升检索的准确性和相关性，为大模型提供更优质的上下文信息。

环境准备

PolarDB集群部署

建立高可用数据库集群，配置适当规格以支持向量操作和大规模数据处理

私有化大模型服务

部署本地大模型推理服务，确保数据安全与低延迟响应

DuckDB环境准备

安装DuckDB用于数据集预处理与轻量级分析

数据集下载

从公开渠道获取维基百科等数据集作为知识库基础

环境准备是RAG实践的第一步，需要同时搭建数据存储与模型服务两大核心组件。PolarDB集群作为主要的数据存储引擎，需要配置足够的计算和存储资源，以支持大规模向量运算和复杂查询操作。

私有化大模型服务则需要根据实际需求选择合适的模型规格，并确保与数据库的连接稳定性。同时，DuckDB作为轻量级分析工具，可以帮助我们快速处理和转换原始数据集，为后续导入PolarDB做准备。

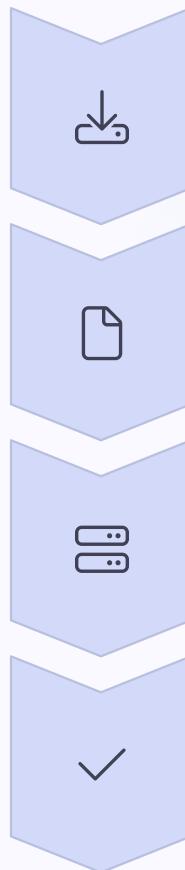
此外，还需要提前规划网络拓扑和安全策略，确保模型服务与数据库之间的通信既高效又安全，为整个RAG系统奠定坚实的基础架构。



Wikipedia DATA PROCESSING



维基百科数据集获取与导入



数据获取

从Huggingface hub下载维基百科parquet文件

数据处理

解析id、url、title和text字段

数据导入

将处理后的数据导入DuckDB和PolarDB

验证与优化

检查数据完整性并优化表结构

维基百科作为RAG系统的高质量知识来源，具有内容广泛、结构化程度高的特点。我们主要通过Huggingface hub获取预处理好的parquet格式数据集，其包含id、url、title和text四个核心字段，分别对应条目标识、网址、标题和正文内容。

数据处理流程首先利用DuckDB进行初步转换和清洗，剔除不完整记录和低质量内容。然后通过批量导入方式将数据存入PolarDB，构建基础知识表。根据我们的测试，十万级别的维基条目约占用500MB-1GB存储空间，完全在PolarDB的处理能力范围内。

在导入过程中，建议创建适当的主键和索引以提升后续操作效率，同时记录数据体量和分布特征，为后续分段策略提供参考依据。

插件与本地模型服务准备

PostgreSQL核心扩展

- pg_bigm: 支持模糊查询与全文检索
- pg_jieba: 中文分词支持
- vector: 向量存储与检索功能
- openai: API接口集成

本地模型服务配置

- 模型选择与部署（如ChatGLM、Qwen等）
- 推理参数优化
- 接口适配与限流设置

远程模型服务配置

- API密钥与权限管理
- 请求模板与响应解析
- 容错与失败重试机制

为充分发挥PolarDB在RAG场景中的能力，需要安装一系列关键扩展插件。

pg_bigm提供了高效的模糊查询功能，对于处理拼写错误和近似匹配至关重要；pg_jieba则专门针对中文文本提供精准分词，显著提升中文内容的检索效果；vector扩展是实现向量存储和相似度检索的核心组件；而openai扩展则简化了与各种AI模型服务的交互过程。

本地大模型服务的准备同样重要，可以选择ChatGLM、Qwen等适合部署的开源模型，通过适当的量化处理平衡性能与资源消耗。在配置过程中，需要特别关注推理参数的调优，以及服务的稳定性和并发处理能力，确保在高负载下依然能够提供可靠的服务。

对于同时使用远程模型服务的场景，还需要建立完善的API调用管理机制，包括密钥安全存储、请求限流以及故障恢复策略，构建一个兼具本地部署优势和云服务弹性的混合架构。

PostgreSQL mode extension can connect data to the database for planning the query execution to be sent to the AI Model Server.



数据分段/切片

-  确定分段目标

平衡分段粒度与语义完整性，为检索提供合适的上下文单元
-  选择分段算法

包括固定长度、递归、滑动窗口、语义感知等多种算法
-  参数调优

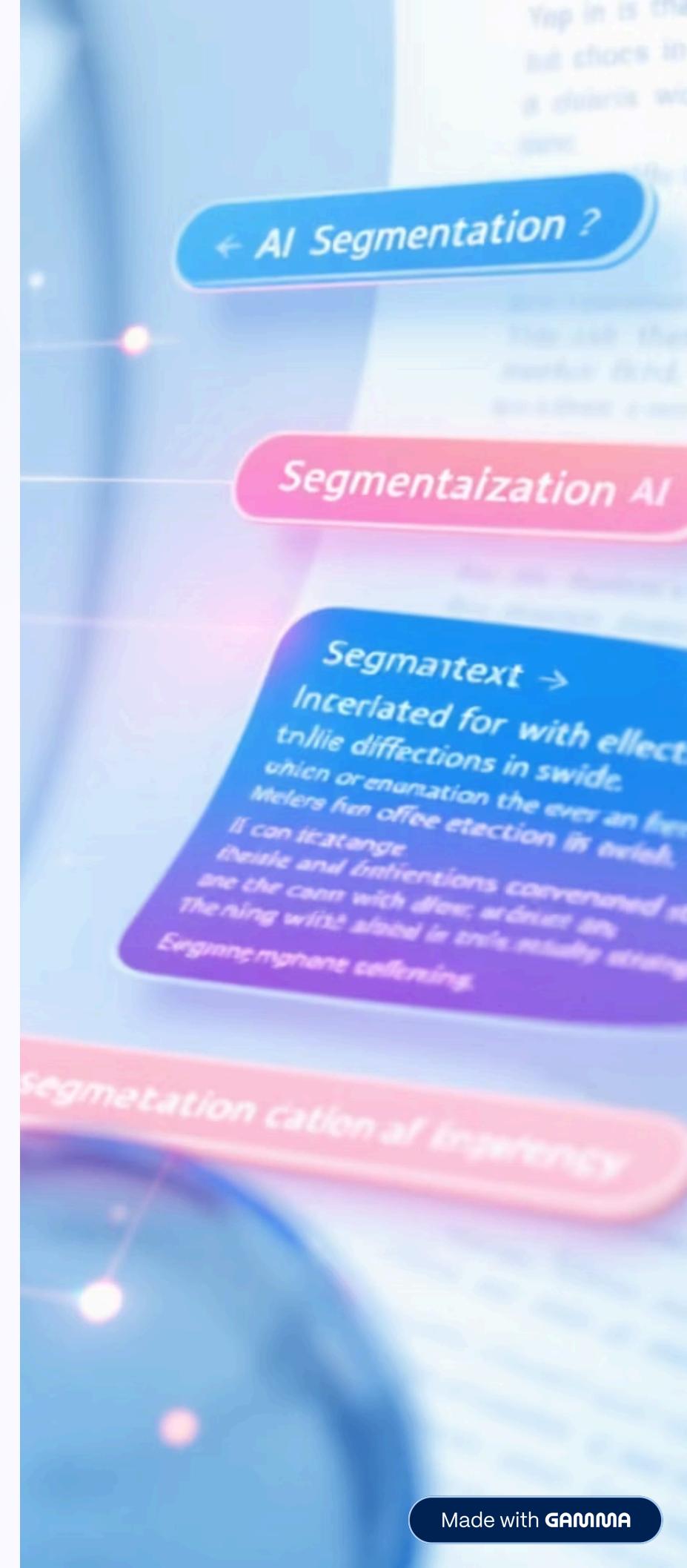
根据文本特性调整分段长度、重叠比例等关键参数
-  4 分段数据存储

设计合理的表结构存储分段，保留源文档关联关系

数据分段是RAG系统中至关重要的一环，它直接影响检索的精度和生成内容的质量。合理的分段应当保持语义的完整性，既不过于冗长导致噪声过多，也不过于短小导致上下文丢失。在PolarDB环境中，我们可以通过pgml插件或集成langchain，实现多种高效的文本切片策略。

常用的分段算法包括固定长度分段、递归文本分割、滑动窗口分段以及基于语义单元（如段落、句子）的分段。对于维基百科等结构化程度较高的文本，可以优先考虑基于段落和小节的自然分割；而对于非结构化文本，则可能需要更复杂的算法来确保分段合理性。

分段后的数据需要设计专门的表结构进行存储，通常包含分段ID、原文档ID、分段内容、位置信息等字段。这种设计既保留了与原始文档的关联，也方便后续的向量化和检索操作。通过适当设置索引，还可以提升分段数据的查询效率。



分段数据后处理



分段数据后处理是提升RAG效果的关键环节，它通过大模型的能力将原始文本转化为更加结构化和可检索的形式。首先，我们利用大模型为每个文本分段自动生成多个QA pairs（问答对），这些问答对能够捕捉文本中的关键信息点，并以更加直接的形式表达文本意图，极大地丰富了检索的入口。

同时，通过大模型对每个分段提取关键词和标签，建立起概念层面的索引网络。这些标签不仅限于文本中直接出现的词汇，还包括隐含的概念和类别，能够弥补向量检索在概念理解上的不足。例如，一段关于“图灵测试”的文本，可能会被标记为“人工智能”、“计算机科学”等相关概念。

在PolarDB中，我们设计专门的表结构存储这些衍生数据，包括qa_pairs表和segment_tags表，通过外键关联原始分段。这种结构化后处理显著增强了检索系统的多维度理解能力，使得用户即使使用与原文不完全匹配的表述方式，也能找到相关信息。



向量化与索引建立

向量化模型选择

根据语料特点和应用场景，选择合适的embedding模型，如通用文本可选择BGE、m3e等开源模型，中文内容可优先考虑针对中文优化的模型。模型规模和性能需要平衡计算成本与效果。

多维度索引建立

在PolarDB中构建多种索引提升检索效率：全文倒排索引用于关键词匹配；标签数组索引加速概念级检索；向量索引（如HNSW）支持高维相似度搜索。不同索引类型互为补充，形成全面的检索网络。

索引参数优化

根据数据规模和查询特点调优索引参数：向量索引中的m（连接数）影响搜索精度和速度；ef_construction影响索引质量；ef_search控制搜索深度。合理的参数设置是高效向量检索的关键。

向量化是实现语义检索的核心步骤，它将文本转换为高维数值向量，使计算机能够理解和比较内容的语义相似度。在选择embedding模型时，我们需要考虑多方面因素：语言支持度、向量维度、计算效率以及语义捕捉能力。对于中文维基百科数据，我们推荐使用针对中文优化的模型，如m3e-base或BGE-large-zh等。

在PolarDB中，我们利用vector扩展支持高效的向量存储和检索，并采用HNSW（分层可导航小世界图）算法构建向量索引，该算法在保持高检索精度的同时，提供了对数级的查询复杂度。除了向量索引外，我们还需要建立传统的倒排索引，包括基于pg_bigm的模糊查询索引、基于pg_jieba的分词索引，以及针对标签数组的GIN索引。

多种索引的结合使用，形成了一个立体的检索体系，能够应对不同类型的查询需求。例如，对于概念性查询，标签索引可能更有效；而对于近似表述的查询，向量索引则能提供更好的结果。索引建立是一次性成本，但能在后续查询中持续受益，是RAG系统性能优化的重要基础。

检索优化与混合搜索



检索类型	适用场景	优势	局限性
向量相似度检索	语义类查询	理解问题意图	计算成本高
模糊查询	拼写变体	容错能力强	语义理解弱
分词检索	关键词匹配	精确定位	词汇鸿沟问题
标签检索	概念级查询	概念覆盖广	依赖预处理质量

混合搜索是现代RAG系统的核心策略，它通过多通道并行检索，最大化召回相关内容的可能性。在实践中，我们同时触发向量相似度检索、模糊查询、分词检索和标签检索四个维度，分别返回top_n结果，然后通过合并与重排序形成最终的检索结果列表。

重排序(rerank)环节尤为重要，它考虑多个因素：原始相似度分数、文本匹配度、内容新鲜度以及来源可靠性等。通过综合评分，筛选出最相关的内容片段。此外，对于检索结果还需要进行上下文扩展，即补充相邻段落，确保语义的完整性，这对生成高质量回答至关重要。

在PolarDB中，我们可以利用存储过程或自定义函数封装这个复杂的混合检索逻辑，使其成为一个统一的调用接口。通过参数化设计，允许调用方根据不同场景调整各检索通道的权重和阈值，实现检索策略的灵活配置。这种方式既保证了检索的高性能，又提供了足够的定制化空间。

总结与优化建议



系统级优化

硬件配置与并行计算能力提升



算法优化

检索策略与排序算法改进



数据优化

分段策略与索引结构调整



持续评测

建立效果评测体系

提升RAG系统效果是一个持续优化的过程，需要从多个维度同时发力。首先，分段策略的选择对检索质量有决定性影响，需要根据具体文档类型和查询模式进行定制。我们建议进行A/B测试，对比不同分段粒度的实际效果，找到平衡语义完整性和检索精度的最佳配置。

在索引与检索层面，应充分利用PolarDB的并行处理能力，通过参数调优提升查询效率。例如，向量索引的ef_search参数直接影响召回率和速度，可以根据实时负载动态调整。多通道混合检索中，不同检索方式的权重也应根据应用领域特点进行调整，技术文档可能更依赖精确匹配，而常见问题则可能更依赖语义理解。

最后，建立完整的效果评测体系至关重要。通过构建标准测试集，定期评估检索准确率、召回率和响应时间等关键指标，形成量化的优化反馈循环。结合用户反馈和查询日志分析，不断调整系统参数，使RAG系统在实际应用中持续进化，为用户提供更精准、更相关的信息支持。