边缘侧检索增强生成系统高效缓存算法研究

詹慧悠,倪宏秋,谈海生*,王天竹,李向阳
(中国科学技术大学计算机科学与技术学院,安徽合肥 230027)

摘 要: 随着大语言模型在边缘智能场景中的部署需求日益增长,检索增强生成(Retrieval-Augmented Generation, RAG)因其可降低模型依赖、提升领域知识覆盖与隐私保障能力而成为边缘侧部署的重要范式. 然而,RAG系统在资源受限的边缘设备上仍面临显著挑战:大规模高维嵌入向量索引难以全量载入内存,频繁的缓存置换与低速存储访问导致检索延迟显著上升. 当前嵌入向量获取主要依赖磁盘加载、在线生成和内存缓存三种路径,三者在延迟、计算开销与资源占用方面差异显著,缺乏统一高效的调度机制. 针对上述挑战,本文提出一种专为资源受限边缘RAG系统定制的高效在线缓存算法——BP-Cache. 该算法的核心创新在于引入了多路径访问代价建模与动态分级缓存管理机制,旨在通过细粒度的资源调度解决边缘侧的性能冲突. 首先,本文通过分析揭示了边缘环境下嵌入向量访问的两个关键特性:一是向量生成代价呈现显著的长尾分布,大尺寸向量簇的在线计算延迟远超磁盘加载;二是访问模式存在极强的局部性与稀疏性,超过60%的向量在生命周期内仅被单次访问. 基于此观测,BP-Cache引入轻量级准入过滤器机制,利用小缓存区作为"观察窗"暂存新到达向量,实现对低价值访问请求的快速旁路,有效遏制了缓存污染问题. 同时,算法构建了代价-大小联合评分模型,将向量簇的在线生成计算代价、磁盘I/O读取延迟及其内存占用大小纳入统一评价体系,在缓存空间不足时,动态优先保留单位内存效益最高的向量簇,从而在无需预知未来访问序列的前提下逼近最优离线策略的性能. 本文在基于NVIDIA Jetson AGX Orin的真实边缘平台上部署了该系统,并基于BEIR基准中的多个数据集开展了广泛的对比实验. 实验结果表明,与EdgeRAG等当前最先进的边缘RAG方案相比,BP-Cache在多组数据集上平均降低检索延迟约29%,提升缓存命中率约21%,并显著优化了系统的长尾延迟表现. 进一步的参数敏感性实验证实,该算法在不同缓存容量配置、小缓存配比及向量簇规模下均表现出优异的鲁棒性与适应性.

关键词: 边缘计算;检索增强生成;缓存优化;在线算法;低延迟检索
基金项目: 国家自然科学基金(No.62132009)
中图分类号: TP303;TP393 **文献标识码:** A **文章编号:** 0372-2112(2025)12-4444-16
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250461

Efficient Caching Algorithm for Retrieval Augmented Generation Systems on Edge Devices

ZHAN Hui-you, NI Hong-qiu, TAN Hai-sheng*, WANG Tian-zhu, LI Xiang-yang
(*School of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei*, *Anhui* 230027, *China*)

Abstract: With the growing demand for deploying large language models (LLMs) in edge intelligence scenarios, retrieval-augmented generation (RAG) has emerged as a pivotal paradigm for edge-side deployment due to its ability to reduce dependency on large models while enhancing domain-specific knowledge coverage and privacy protection. However, RAG systems still face significant challenges on resource-constrained edge devices: large-scale, high-dimensional embedding vector indexes cannot be fully loaded into memory, and frequent cache evictions coupled with slow storage accesses lead to substantially increased retrieval latency. Currently, embedding vectors are primarily acquired through three approaches—disk loading, online generation, and in-memory caching—which differ significantly in terms of latency, computational overhead, and resource consumption, and lack a unified, efficient scheduling mechanism. To address these challenges, this paper proposes BP-Cache, an efficient online caching algorithm specifically tailored for resource-constrained edge RAG systems. The core innovation of BP-Cache lies in its multi-path access cost modeling and dynamic hierarchical cache

收稿日期:2025-05-31;录用日期:2025-10-28;责任编辑:覃怀银
*通讯作者:谈海生

management mechanism, designed to resolve performance conflicts at the edge through fine-grained resource scheduling. First, we uncover two key characteristics of embedding vector access patterns in edge environments through empirical analysis: (1) the cost of online vector generation exhibits a pronounced long-tail distribution, where the computational latency for large vector clusters far exceeds that of disk loading; and (2) access patterns show strong locality and sparsity, with over 60% of vectors accessed only once throughout their lifetime. Based on these observations, BP-Cache introduces a lightweight admission filtering mechanism that uses a small buffer cache as an "observation window" to temporarily hold newly arriving vectors, enabling rapid bypass of low-value access requests and effectively mitigating cache pollution. Simultaneously, the algorithm constructs a joint cost–size scoring model that integrates the online generation cost, disk I/O latency, and memory footprint of each vector cluster into a unified evaluation framework. When cache capacity is insufficient, BP-Cache dynamically prioritizes retaining vector clusters that deliver the highest utility per unit of memory, thereby approaching the performance of an optimal offline strategy—without requiring any knowledge of future access sequences. We implement and evaluate our system on a real-world edge platform based on the NVIDIA Jetson AGX Orin, conducting extensive experiments across multiple datasets from the BEIR benchmark suite. Results show that, compared to state-of-the-art edge RAG solutions such as EdgeRAG, BP-Cache reduces average retrieval latency by approximately 29% and improves cache hit rate by about 21% across multiple datasets, while significantly optimizing tail latency performance. Further sensitivity analyses confirm that the algorithm demonstrates excellent robustness and adaptability under varying cache capacities, small-buffer ratios, and vector cluster granularities.

Key words: edge computing; retrieval augmented generation; cache optimization; online algorithm; low-latency retrieval

Foundation Item(s): National Natural Science Foundation of China (No.62132009)

1 引言

大语言模型(Large Language Model,LLM)在自然语言处理领域有着广泛应用,将其部署于本地终端以实现智能问答、个性化助手等场景的需求日益增强. 然而,LLM通常模型规模庞大、计算资源需求高,难以直接运行于资源受限的边缘设备上[1~3]. 为此,近年来一种新型的系统架构检索增强生成(Retrieval Augmented Generation,RAG)[4~6]受到关注. RAG架构通过引入外部知识库,将轻量级的生成模型与高效的向量检索模块相结合,在生成响应前先从知识库中检索出与输入问题语义相关的内容,并将其作为上下文注入模型生成过程. 这种方法不仅有效降低了对模型参数规模的依赖,还提升了模型在特定领域、特定语境下的知识覆盖能力,实现基于用户私有数据的定制化问答,从而在提升响应质量、降低服务延迟与增强数据隐私保障等方面展现出显著优势.

尽管边缘RAG系统具有重要的实用价值,其在实际部署过程中仍面临严峻的资源约束挑战. 首先,内存限制问题突出. RAG的核心组件——向量数据库需存储大量高维嵌入向量,以支持语义检索的准确性与鲁棒性. 在典型任务场景中,知识库可能包含数十万甚至上百万条文档,每条文档对应一个维度为768或更高的浮点向量,这将使得索引规模轻易达到数十GB. 而当前主流边缘平台(如智能手机、树莓派、嵌入式ARM板卡等)所具备的可用内存资源普遍有限,仅为4~12 GB不等,远无法容纳完整的向量索引结构,导致系统频繁发生内存页置换、缓存失效等问题,从而带来显著的性能退化[7,8]. 其次,访问延迟亦不可忽视. 从本地磁盘存储(如eMMC[9]或SD卡)加载大规模嵌入索引时,由于磁盘I/O延迟远高于内存访问延迟,尤其在缺乏高性能NVMe[10]接口的低端设备上,频繁的向量加载与检索会显著延长整体查询响应时间,严重影响用户体验与系统实时性要求. 因此,如何在上述硬件瓶颈下实现高效、低延迟、可扩展的检索增强生成系统设计,已成为边缘智能研究中的重要问题.

在边缘设备上部署检索增强生成(RAG)系统时,嵌入向量的检索效率是影响整个系统的响应性能的重要组成. 目前检索阶段嵌入向量的获取方式主要包括以下三种路径:

(1)从本地磁盘加载. 系统将预先生成的嵌入向量存储在本地存储设备中,查询时按需调入内存,该方式无需实时计算,但磁盘I/O速度较慢,访问延迟较高,尤其在频繁检索场景下更为明显.

(2)在线生成嵌入向量. 系统基于存储的原始数据内容,在检索过程中生成对应嵌入向量,可有效节省存储空间,但生成过程计算开销大,且在计算能力受限的边缘设备上易引发长尾延迟问题.

(3)从缓存中获取. 对于高频访问或延迟敏感的数据,提前将其嵌入向量保存在内存中,可大幅缩短访问路径,是当前优化系统响应性能的重要手段.

然而,这三种方式在访问成本、延迟表现与资源占用上存在显著差异,如何在具体应用场景中平衡三者

使用比例成为系统设计的关键问题. 以EdgeRAG[11]为代表的现有方案,已尝试结合在线生成与磁盘加载的混合策略,并通过仅基于生成成本的启发式缓存机制进行部分嵌入向量的内存驻留管理. 但这一策略仍存在局限:其一,未对三种获取路径的访问代价进行统一建模,缺乏系统性的准入与替换机制;其二,缓存策略无法根据访问模式变化进行动态调整,难以在多种检索负载下保持稳定的高效性能. 为此,本文提出将边缘RAG系统中嵌入向量的三种获取方式作为整体调度对象,通过设计具备动态适应性的缓存管理策略,综合考虑生成、加载与缓存命中三类代价,对缓存准入与淘汰机制进行统一优化,以显著降低检索延迟,提升边缘侧RAG系统的响应效率和资源利用率.

在边缘设备上部署检索增强生成系统的过程中,本文通过深入分析数据访问模式和嵌入向量的生成特性,得出了以下两个关键性观察结果.

(1)嵌入向量生成代价具有显著差异性和尾部分布特征. 不同大小的数据块所对应的嵌入向量生成时间存在较大差异. 对于较小的数据块,在线生成其嵌入向量所需的时间甚至低于从磁盘加载预计算向量所需的时间;而对于较大的数据块,生成嵌入向量则可能引入显著延迟. 此外,嵌入向量的生成成本呈现出明显的"长尾"分布,即少数大规模数据块占据了大部分的计算资源消耗.

(2)大量嵌入向量仅被访问一次或极少重复使用. 约有60%以上的嵌入向量在整个生命周期中仅被访问一次,且仅有少数高频访问的嵌入向量会被反复调用. 这一现象表明,将这些低频访问的嵌入向量缓存至内存中不仅浪费有限的缓存空间,还可能导致缓存污染,降低整体命中率. 因此,应设计一种机制避免或快速淘汰这类"冷向量"进入缓存系统.

基于以上观察,本文提出了一种融合多种获取成本建模的在线缓存算法框架,通过引入小型过滤器实现高效缓存准入控制. 该算法在不依赖未来信息的前提下,动态决策每个嵌入向量是否应当直接旁路(即不进入缓存)、缓存或替换. 具体而言,本文的主要技术贡献包括:

(1)为边缘RAG场景构建了一个统一的代价感知缓存模型,对嵌入向量缓存问题进行了系统性建模与形式化定义,明确了多种向量获取路径下的访问代价指标与优化目标,并证明任意确定性在线缓存算法在这个问题下的竞争比至少为K(K为缓存容量).

(2)基于对边缘RAG访问模式的深入分析,构建出面向多路径访问代价的分级缓存框架. 与场景特有的代价-大小联合建模相融合,提出一种考虑旁路与文件大小的边缘RAG系统上的在线缓存算法,用于动态判断是否将某嵌入向量缓存或绕过缓存,并进一步引入小型过滤器控制准入门槛,设计了高效的二级缓存结构BP-Cache,提升系统鲁棒性与实用性.

(3)在Jetson AGX Orin等典型边缘设备平台上进行了系统部署与验证实验,结果表明,与现有最先进的边缘RAG方案相比,BP-Cache在多个数据集上平均减少了约29%的检索延迟和提高了平均约21%的缓存命中率,同时维持了较高的生成质量与较低的内存占用.

2 相关工作

2.1 检索增强生成系统

检索增强生成系统是一种结合信息检索和自然语言生成的混合方法,旨在提升大型语言模型的性能,尤其在知识密集型任务中表现突出,例如问答和事实验证. 通过将响应建立在检索的证据之上,RAG系统有效缓解了生成模型的幻觉问题,如图1所示,其工作流程可分为三个主要阶段:索引、检索和生成.

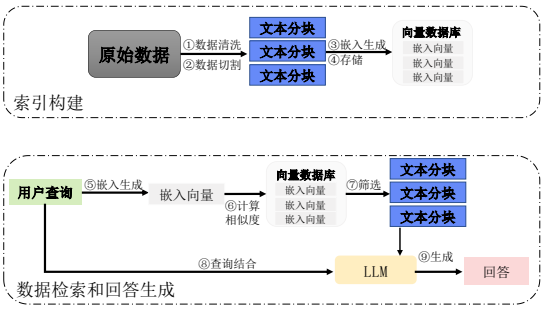


图1 检索生成系统(RAG)的工作流程

索引阶段. 在索引构建阶段,首要任务是对原始数据进行清洗(步骤1). 原始数据往往包含格式混乱、噪声数据、重复内容等问题,需要通过正则表达式、数据过滤规则去除无效字符、HTML标签等,利用去重算法剔除重复文本,以提升数据质量.

清洗后的数据会被分割成合适长度的文本分块(步骤2),文本分割的粒度直接影响检索效率. 基于句子边界的细粒度分割适用于问答场景的精准匹配;固定长度滑动窗口分割能够平衡上下文完整性与计算复杂度;段落级分割则更适合长文本摘要任务. 这些分割后的文本单元通常借助预训练的嵌入模型(如BERT[12]、Sentence-BERT[13])转化为高维稠密向量(步骤3),这些向量存储在单独数据库中(步骤4),通常使用近似最近邻(ANN)搜索算法(例如FAISS[14]和HNSW[15])以实现高效、可扩展的检索. 这一阶段的目的是为后续检索提供快速访问的结构化数据.

检索阶段. 当系统接收到用户查询时,首先通过在索引阶段使用的嵌入模型将自然语言输入编码为语义向量(步骤5),随后在索引空间中执行相似度匹配(步

骤6). 主流的相似度度量方法包括余弦相似度、欧氏距离、内积空间匹配等,部分系统还会引入基于Transformer[16]的重排序模块,对初始检索结果进行二次筛选,以提升答案相关性. 在实际应用中,检索策略需综合考虑响应时效性与内容准确性. 对于实时性要求较高的场景,通常采用粗筛(如倒排索引快速过滤)+细排(向量检索精确匹配)的级联检索架构;而对于专业领域问答,则更注重多跳检索、跨模态检索等复杂策略,以挖掘深层关联知识. 最终筛选出的高相关文档片段将作为外部上下文(步骤7),与用户查询共同构成生成模型的输入.

生成阶段. 在检索到相关文档后,系统将相关文档与用户的查询结合(步骤8),输入到生成模型中. 生成模型通常采用序列到序列模型(如T5[17])或自回归模型(如GPT[18]系列),模型通过注意力机制聚焦关键信息生成连贯且事实准确的响应(步骤9).

2.2　检索增强生成系统的索引方法

平面索引(Flat Index)[19]是最基础的向量索引结构,它将所有向量数据直接存储在索引空间中,不进行任何其他处理. 在查询时,Flat Index会计算查询向量与索引库中所有向量的相似度.

早期索引方法更侧重基于统计特征的索引方法,例如基于词袋模型和基于向量空间模型的索引方式. 词袋模型(Bag of Words,BoW)[20]通过统计文档中词频构建索引,经典实现包括倒排索引(Inverted Index)[21]. 该方法将每个单词映射到包含该词的文档列表,通过布尔检索或TF-IDF加权匹配实现快速查询,具有结构简单、查询效率高的优势. 二级倒排文件索引(Two-level Inverted File Index,IVF)[12]是传统倒排索引在向量检索领域的扩展,专为大规模向量数据处理设计. 其核心架构由倒排列表和聚类中心构成. 构建时,通过聚类算法将相似向量分组,每个聚类由中心向量表征,倒排列表则存储对应聚类的向量及元数据. 如图2所示,检索时,系统先计算查询向量与聚类中心的相似度,筛选出候选聚类(步骤1),再对这些聚类的倒排列表进行深度匹配(步骤2),大幅降低计算量. 最后,系统检索与嵌入关联的数据(步骤3),相较于传统倒排索引,二级倒排文件索引通过聚类优化检索效率,但检索精度高度依赖聚类质量. 在RAG系统中,IVF常与语义索引结合,先用IVF快速过滤,再用精确索引精排,平衡检索速度与准确性. 这也是当前主流边缘侧RAG采用的索引方式. 向量空间模型(Vector Space Model,VSM)[13]将文档和查询映射为高维空间中的向量,通过余弦相似度等度量计算相关性. BM25算法[22]作为改进方案,引入词频衰减、文档长度归一化等策略,在传统信息检索任务中表现出色. 此类方法通过量化语义距离提升检索精度,但依赖人工设计特征权重,难以适应动态语义变化.

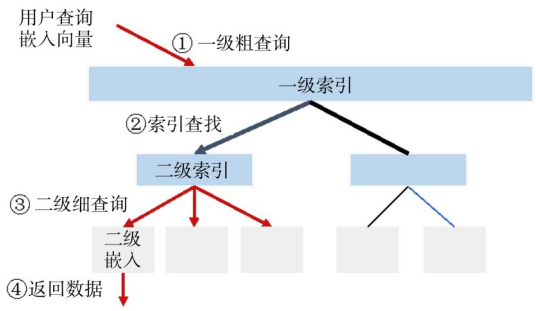


图2　二级倒排文件索引(IVF)示意图

近年来基于Transformer的嵌入模型通过预训练捕捉深层语义信息将文档和查询编码为固定维度的稠密向量. 这种语义索引支持跨语言、上下文感知的检索,结合FAISS、HNSW等高效向量检索库,可实现百万级文档的毫秒级检索. 但其训练成本较高,且存在向量维度灾难问题,需配合聚类、分层索引等技术优化性能. 实际应用中常采用混合索引策略,结合传统倒排索引的精确匹配能力与向量索引的语义泛化能力. 例如,Karpukhin等人[23]先用BM25快速筛选候选文档,再用密集向量重新排序,兼顾效率和准确性. 而针对超大规模数据集,多采用层次结构(如HNSW图[15]). 将文档聚类为多级,较高层次表示大范围聚类,较低层次细化嵌入. 这种方法提升了可扩展性,但设计和维护成本较高.

2.3　边缘侧检索增强生成系统

随着边缘计算的兴起,将检索增强生成(RAG)系统部署在边缘设备(如智能手机、平板电脑和物联网设备)上成为研究热点. 但由于这些设备内存和计算能力有限,传统RAG系统的大型语言模型和向量数据库难以直接运行. EdgeRAG[11]是一种专为资源受限的边缘设备设计的RAG系统. 它采用两级倒排文件IVF索引结构,第一级存储聚类中心及其引用,第二级存储文档嵌入. 通过在聚类内修剪嵌入,仅保留第二级搜索所需的必要数据,EdgeRAG显著降低了向量数据库的内存需求. 在检索时,EdgeRAG按需生成嵌入,而不是预先存储所有嵌入,进一步减少内存占用. 同时,为避免生成嵌入导致的延迟(特别是在大型尾部聚类中),EdgeRAG对这些聚类的嵌入进行预计算和存储,从而避免长尾延迟. EdgeRAG还设计了一套自适应缓存机制,通过分析查询的重用模式,识别频繁访问的聚类并缓存其嵌入. 这种机制减少了冗余计算,优化了检索速度. MiniRAG[24]针对传统RAG向小型语言模型(SLMs)迁移时性能下滑的问题从架构设计上做出革新. 其核心优化有这两部分:异构图索引与轻量级基于图的知识

检索. 异构图索引整合文本块与命名实体,设立实体节点来提取关键语义元素,文本块节点保留原始文本上下文,让文本块直接参与检索,克服SLMs文本总结能力的不足. 轻量级基于图的知识检索机制一方面利用SLMs实体提取优势,简化查询解析,实现查询语义映射;另一方面采用两阶段检索在边缘系统资源受限的情况下,达成高效知识获取. Adaptive Contextual Caching[25]通过深度强化学习驱动的缓存机制动态优化检索增强生成流程. 通过语义相似度、用户上下文和缓存缺失开销的联合建模,实现缓存内容的主动预取与替换策略优化. RAG允许LLM利用外部知识生成更好的响应,但使用更多外部知识会导致更高的响应延迟,为了平衡质量和响应延迟,METIS[26]构建了快速且质量感知的RAG系统并支持配置自适应. LEANN[27]针对存储资源受限的端设备提出了一种基于近似最近邻搜索索引的低存储向量索引方法. MobileRAG[28]则针对功耗受限的移动设备提出了移动友好的向量搜索算法与轻量级选择性内容缩减方法以大幅减少内存占用和CPU使用率.

2.4 缓存算法及其在大模型推理中的应用

缓存替换策略一直是系统优化的重要研究方向. 最常见LRU[29]和LFU[30]算法在多种应用场景中被广泛采用,但在复杂访问模式下常常存在缓存命中率低的问题. 为此,研究者提出了改进方法,例如2Q算法[31]与ARC算法[32],通过多级队列与自适应调节机制提升缓存命中率与稳健性. 在资源受限设备中,还出现了大小感知的缓存策略,以减少大对象“挤占”缓存的负面影响[33],以及旁路机制[34],用于在特定条件下直接跳过缓存低价值对象. 近年来,为了适配多样的缓存工作负载,基于学习的缓存替换方法逐渐[35~39]兴起,例如受最优离线缓存算法启发的预测方法[35]和专家集成驱动的轻量学习方法[36~39],它们在在线环境中能够在较低开销下接近最优替换策略. 然而,这类方法通常因引入学习过程而产生额外开销,在资源受限的边缘计算场景中应用仍较为有限. 缓存算法的应用领域极为广泛,例如近年就被创造性地应用于大型语言模型的推理优化中. 对于云服务提供商而言,提供大型语言模型服务至关重要,而处理每个请求后缓存中间结果(KV)可以显著提高服务吞吐量和延迟. 多项研究借鉴经典缓存思想,设计了高效的KV缓存驱逐策略以最大化KV重用、加速LLM推理[40~45]. 例如,H_2O[40]通过评估数据的重要性来选择性地保留和驱逐KV缓存,同时也有来自大型云服务商的实证研究[41]对KV缓存的特性分析进行了深入分析,并提出了一种基于工作负载感知的KV缓存驱逐策略. CacheBlend[43]聚焦于在RAG场景下高效融合多个预计算的KV缓存以加速LLM推理,ClusterKV[44]通过聚类、选择、索引和缓存实现高效准确的KV缓存压缩. 尽管云端LLM的KV缓存优化已取得显著进展,当前研究对边缘侧大模型所特有的缓存行为与约束条件(如缓存未命中时嵌入生成与磁盘加载的异构代价、严格的内存与能效约束)仍缺乏系统探索,相关缓存机制的设计尚处于初步阶段.

3 问题模型

本节将系统性地建立边缘侧检索增强生成系统的缓存优化问题模型. 首先,我们将定义缓存系统的组成与约束条件,对嵌入向量的请求模式进行形式化描述,包括请求序列的生成方式;接着,结合边缘RAG系统的实际运行场景,量化分析不同嵌入向量获取路径(磁盘加载、在线生成、缓存命中)的延迟代价;最后,基于上述模型组件,形式化定义优化目标,并分析问题的计算复杂度与理论求解难度. 通过这一完整的建模过程,为后续算法设计奠定理论基础.

3.1 缓存模型

本文考虑基于二级倒排文件索引(IVF)的边缘侧RAG系统上的缓存问题,其缓存空间由有限的内存资源构成,用于在内存中存储高频访问的嵌入向量簇. 设缓存总容量为K,单位为向量尺寸的标准块大小(如768维FP16向量所占内存),系统中所有可能的嵌入向量簇集合为 $\mathcal{F}=\{f_1,f_2,\cdots,f_N\}$,其中每个向量簇$f_i$具有一个固定大小$s_i$,表示嵌入向量簇占用的存储空间,由向量维度与精度决定. 缓存需满足容量约束:任意时刻缓存中所有向量的总大小不超过K,即$\sum_{f\in\mathcal{C}}s_i\leqslant K$,其中$\mathcal{C}\subseteq\mathcal{F}$为当前缓存内容. 用户的请求以在线方式到达,即我们无法获得未来的信息,也没有对到达模式做出任何假设,其序列表示为$\mathcal{R}=\left(r_1,r_2,\cdots,r_T\right)$,$T$为序列长度. 在边缘RAG系统中,每个用户请求通常需要检索多个相关的嵌入向量簇以获取足够的上下文信息,即每个请求r_j需访问特定的嵌入向量簇子集$F_j\subseteq\mathcal{F}$. 对于每个对嵌入向量簇$f_{j_k}\in F_j$的访问,系统可选择以下三种处理方式之一.

(1)命中(Hit):若所请求的嵌入向量簇已在缓存中$f_{j_k}\in\mathcal{C}$,则直接从缓存读取,延迟记为0.

(2)生成(Generate):若该嵌入向量簇不在缓存中时,系统可选择在线生成其嵌入向量簇,延迟为其生成代价$c_{j_k}^{\text{gen}}$(单位:s),该代价依赖于簇的大小与模型计算开销.

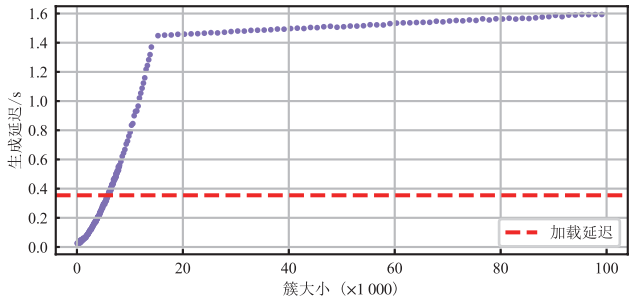
(3)加载(Load):系统亦可从本地磁盘加载预先计算的嵌入向量,延迟为从磁盘加载预计算嵌入向量簇的I/O延迟$c_{j_k}^{\text{Load}}$(单位:s).

对于不在缓存中的嵌入向量簇,通过生成或者加

载的方式获取后,可选择将其放入缓存中,也可以选择旁路(Bypass)操作,即直接选择不缓存该向量簇,仅临时处理请求.

3.2 检索延迟建模及问题形式化

在边缘RAG系统中,检索延迟是衡量系统性能的关键指标. 为了准确建模不同嵌入向量获取路径的延迟代价,我们需要结合实际运行场景和实验数据,对加载延迟、生成延迟以及缓存命中延迟进行量化分析. 本小节将基于图3所示的簇大小与延迟的关系,进一步细化检索延迟的建模.



注:簇大小的单位为向量尺寸的标准块,红色虚线表示从磁盘加载向量簇的延迟.

图3　不同大小的嵌入向量簇的生成延迟

图3展示了在Jetson AGX Orin平台上,在线生成嵌入向量时,其生成延迟随嵌入向量簇大小变化的实测数据. 我们可以观察到以下两个关键现象:

(1)生成延迟呈非线性增长趋势,且存在显著的长尾分布. 在簇大小小于约18 000个标准块时,生成延迟随着向量簇大小的增加而快速增长;而当簇大小超过18 000个标准块后,生成延迟增速变缓,并显著超过加载代价.

(2)对于小规模嵌入向量簇,即簇大小小于约8 000个标准块时,生成延迟始终低于磁盘加载延迟(约为0.35 s),且磁盘加载延迟近似为一常数M.

这是由于Jetson AGX Orin平台上的eMMC带宽约为200~400 MB/s,大于绝大多数向量簇的大小,导致读取延迟主要由固定启动开销主导. 因此,我们进一步定义每个嵌入向量簇f_i的检索延迟为

$$d_i=\begin{cases}0, & \text{if } f_i\in C\\ \min\left(c_i^{\text{gen}},M\right), & \text{if } f_i\notin C\end{cases}.$$

在上述缓存模型与延迟建模的基础上,边缘RAG系统中的优化目标为:在在线请求序列下,最小化总检索延迟,同时满足缓存容量限制. 该优化目标可形式化为

$$\min_{\pi}\ \sum_{j=1}^{T}\sum_{f_i\in F_j} d_i\ ,$$

其中,π表示缓存管理策略,包括淘汰策略及旁路决策,即每次访问选择缓存命中、生成或加载;新项是否进入缓存;若缓存已满,淘汰何项.

3.3 问题难度分析

设ALG表示在线算法的代价,OPT表示具有完全信息的最优离线算法的代价,则该在线算法的竞争比定义为$\sup\limits_{I}\dfrac{\text{ALG}(I)}{\text{OPT}(I)}$,其中$\sup\limits_{I}$表示对所有可能的输入序列$I$的上确界. 竞争比反映了在线算法在最坏情况下的性能损失上限.

在不考虑旁路时,边缘RAG系统中的在线缓存问题退化为具有非均匀文件大小与获取代价的一般缓存问题,已有研究[46,47]表明该问题的最优在线算法的竞争比至今仅能达到$O(\log k)$. 本文进一步考虑支持旁路决策的边缘RAG系统上的在线缓存问题,并证明对于确定性在线算法具有以下下界.

定理1　对于任意确定性在线缓存算法ALG,存在一个请求序列,使得其竞争比至少为K.

证明　假设所有嵌入向量簇的大小均为一个使得加载代价大于生成代价的簇大小$s_i=S$,缓存总容量为KS,则此时每个簇的未命中代价为固定值M,即$\min\left(c_i^{\text{gen}},M\right)=M$. 假设初始缓存包含$K$个初始簇$f_1,f_2,\cdots,f_K$. 现在构造一个针对任意在线策略$\eta$的对抗性请求序列$\mathcal{R}=z_1^{\delta_1},z_2^{\delta_2},\cdots,z_K^{\delta_K}$,其中,每个$z_u$是未被初始缓存包含的新簇,$\delta_u$为$Z_u$的连续请求次数,其值将被选择为足够大,以迫使在线算法将其加入缓存并淘汰已有簇.

接下来逐步构造如下请求过程.

(1)对于z_1,若策略η永远不将其加入缓存,则令$\delta_1\to\infty$,从而使在线算法在每次请求中都产生代价M,总开销为$\delta_1\cdot M$,而最优策略OPT只需第一次请求时将其加入缓存,代价仅为M,故此时竞争比趋近于δ_1,可以任意大. 因此我们可以假设η在请求到第δ_1次时将z_1加入缓存,并替换掉某个初始簇$f_{r_1}\in\{f_1,f_2,\cdots,f_K\}$.

(2)之后请求新的簇z_2,设其为上一步中被替换掉的簇f_{r_1},由于其不在缓存中,若η不选择将其加入缓存,则同样令δ_2足够大可迫使其加入缓存,产生一次替换. 记被替换掉的簇为f_{r_2}.

(3)重复上述过程,直到我们构造了K个新簇,每个簇z_u产生δ_u次请求,均足以让η将其加入缓存并替换掉已有缓存内容.

在整个过程中,在线策略η共执行了K次缓存替换操作,且每个新簇在加入前至少访问一次未命中,因此其总开销至少为$\text{ALG}\geqslant K\cdot M$. 而对于最优策略OPT,由于请求的所有簇仅包括初始的K个缓存簇与新引入的簇z_1,共$K+1$个簇,其可在一开始选择一个不被请求的

初始簇f^*替换为第一个被请求的z_1，之后所有请求均为缓存命中，总成本为一次加载OPT=M.

因此，竞争比满足：

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} \geqslant \frac{K \cdot M}{M} = K .$$

若嵌入向量簇大小s_i及生成/加载代价$c_i^{\mathrm{gen}}, c_i^{\mathrm{Load}}$均不同，则缓存算法的竞争比，最坏情况下至少为$K^* := \left\lfloor \frac{K}{s_{\min}} \right\rfloor$，其中$s_{\min}$为所有簇中最小的簇大小. 证明过程如下：选取$K^*+1$个嵌入向量簇，分别为$(f_1, f_2, \cdots, f_{K^*+1})$，每个簇的大小均为$s_{\min}$，都可以被缓存一次塞入$K^*$个. 假设每个簇的检索代价均为最大值$M := \max_i d_i$，即所有选的簇都设置为高代价（这是最坏情况）. 缓存初始状态包含前K^*个簇. 请求的构造序列与证明过程与同构情况相同，可证明竞争比下界为K^*. 证毕.

4 在线算法设计

在本节中，我们首先结合系统运行数据与访问分布，进一步阐述算法设计的动机与直觉基础，明确设计目标与关键策略；随后，详细介绍我们提出的在线缓存算法框架，包括准入判断、旁路机制、替换策略等核心组件，并给出完整伪代码与决策流程；最后，我们对算法的时间与空间复杂度进行分析，说明该算法在边缘设备上运行的可行性.

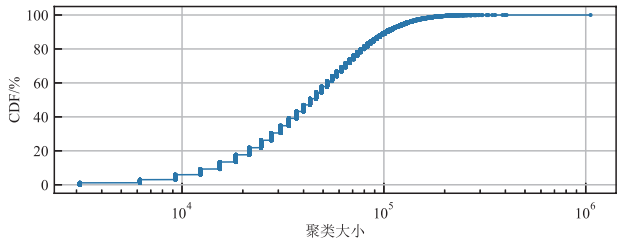
4.1 设计动机与关键思想

为实现低延迟、高效率的边缘RAG系统部署，缓存策略必须应对资源约束、访问动态性强与延迟成本差异大的挑战. 我们通过对嵌入向量簇的生成行为和访问模式进行深入数据分析，得到两个关键观察结果，这为在线算法的设计提供了重要启发.

观察一：嵌入向量簇的大小具有显著差异性和长尾分布，导致生成延迟高度不均衡.

图4展示了嵌入向量簇大小在真实系统中的累积分布函数（Cumulative Distribution Function，CDF）. 可以看到，尽管大部分嵌入向量簇较小，但仍有相当比例的簇规模极大，整体分布呈现出明显的长尾特性. 这一结构性不均衡会直接影响在线生成嵌入向量的延迟代价. 如前文3.2节图3所示，嵌入向量簇的生成时间随着簇大小的增加快速增长，且当超过某一临界规模后，生成代价甚至高于从磁盘加载的成本. 因此，一个理想的缓存策略应综合考虑簇的大小与访问代价，在缓存空间有限的条件下优先缓存生成成本高、但空间占用较小的“高收益”簇，从而提升单位内存资源的使用效益.

观察二：嵌入向量簇访问高度不均，存在大量“一次性访问”的冷数据.



注：聚类大小的单位为向量尺寸的标准块.

图4 不同嵌入向量簇大小的累积分布函数

表1所示为RAG问答数据集的统计结果. 我们可以看到，在多达百万量级的嵌入向量中，平均约有56%以上的嵌入向量在整个生命周期中仅被访问一次，仅有少部分高频向量簇承担了主要的访问负载. 例如，在nq数据集中，在多达百万量级的嵌入向量中，80%的嵌入向量仅被短期调用而后不再使用. 这表明，将这类“冷向量”加入缓存不仅难以被再次命中，反而会挤占稀缺的缓存空间，降低整体命中率与系统性能. 因此，缓存机制需要具备一定的判断机制，避免将明显为一次性访问的簇纳入缓存，或在其短时间未被复用时快速驱逐.

基于上述两点关键观察，我们的在线缓存算法设计遵循以下核心思想. 代价-大小联合建模：通过结合嵌入簇的大小、生成代价与加载代价进行评分，选择高代价但缓存性价比更高的项优先保留；轻量级准入过滤器：设计高效、低开销的过滤机制，在在线场景下判断向量是否具备“缓存价值”，避免频繁错误纳入.

基于以上设计理念，下一节将详细介绍我们提出的在线缓存算法框架BP-Cache的具体实现流程、策略结构与伪代码说明.

表1 评测数据集的嵌入向量访问统计

数据集名称	记录数	嵌入向量总大小	独立访问数	总访问次数	一次性访问占比/%
nfcorpus	3 633	226 MB	1 157	2 000	57.9
scidocs	25 657	434 MB	2 974	13 286	22.4
fiqa	57 638	974 MB	12 789	18 685	68.4
quora	522 931	3.1 GB	15 672	30 000	52.2
nq	2 681 468	16.6 GB	8 186	10 235	80.0

4.2 BP-Cache设计

4.2.1 总体设计

为在边缘设备上的检索增强实现兼顾命中率与响应延迟的在线缓存机制，BP-Cache采用双缓存结构的设计架构，如图5所示. 该结构由三个核心组件构成：

（1）主缓存区（Main）：用于长期保存访问频次高、复用性强的嵌入向量簇.

(2)小缓存区(Small):作为前置过滤缓冲区,用于临时缓存新到达、尚不明确价值的嵌入向量簇.

(3)Ghost表($\mathcal{G}$):记录曾短暂进入小缓存但未被复用的冷向量簇,用于辅助后续准入判断.

BP-Cache的核心思想是通过小缓存区拦截大部分短生命周期访问对象,避免其直接污染主缓存;同时借助Ghost表记录"失败的历史",提升后续准入判断的精度.

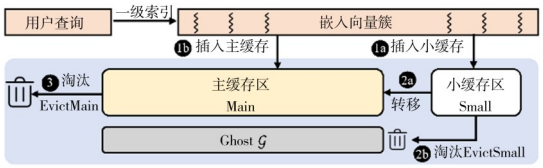


图5　BP-Cache的双缓存结构与访问路径示意图

具体而言,BP-Cache的缓存行为可分为以下三类操作.

插入(Insert):若某个向量簇从未出现于Ghost表中,则判定为首次访问,插入到小缓存Small(路径1a);若其曾出现在Ghost表中,则说明其历史上曾进入缓存但未复用,则直接插入主缓存Main(路径1b).

小缓存淘汰(EvictSmall):若该向量簇在缓存期间未被访问,则认为该向量簇均为冷向量,将其名称插入Ghost表用于记录并直接淘汰向量簇(路径2a);否则,若该向量簇被访问过,说明具有一定价值,则请求转移至主缓存区(路径2b).

主缓存淘汰(EvictMain):若主缓存区的缓存已满,且有新向量簇要求进入该缓存区,则按照算法淘汰嵌入向量簇(路径3a).

通过上述设计,BP-Cache实现了"首次访问→缓存观察→转入主缓存"的快速淘汰过程,同时利用Ghost表对历史行为进行惩罚性记录,有效提升了缓存命中率.

4.2.2　小缓存的缓存算法

随小缓存Small作为BP-Cache的前置缓冲区,用于临时存放首次访问的嵌入向量簇,其设计目标是快速识别冷向量簇并避免其污染主缓存. 小缓存采用LRU(最近最少使用)策略进行淘汰管理,确保优先保留近期更可能被重复访问的项.

具体而言,如算法1描述的,当一个新向量簇f被插入小缓存时,若缓存容量已满,则从队首淘汰多个最近最久未被访问的向量簇f_{evict}直至可容纳新对象(第1~2行). 随后根据淘汰的向量簇在驻留期间是否被访问过,执行以下分支逻辑:(1)若f_{evict}曾被访问,则认为具备一定复用价值,直接将其转入主缓存(第3~4行);(2)若f_{evict}在缓存期间未被访问,则视为"冷向量簇",其标识被加入Ghost表(第5~6). 最后将新向量簇f加入队列队尾,并将f标记为未访问. 这种"先观察后升级"的机制有效避免了将大量一次性访问的嵌入向量簇直接写入主缓存,从而提高整体命中率与缓存空间利用率.

算法1　小缓存的插入与淘汰算法

输入:小缓存队列Small,请求插入的向量簇f,Ghost表 $\mathcal{G}$
输出:小缓存更新后的状态
1. WHILE Small容量已满 DO
2. f_{evict} ← 从队列队首移除向量簇
3. IF f_{evict}.VISITED( ) THEN
4. 将f_{evict}转移至主缓存中
5. ELSE
6. 将f_{evict}的ID插入 $\mathcal{G}$
7. 将f插入队列尾部
8. 标记f.VISITED( )为 False

4.2.3　主缓存的缓存算法

主缓存Main用于存放经小缓存筛选后,具备一定复用价值的嵌入向量簇,其目标是在有限空间内保留单位内存贡献最大的对象,从而最小化整体检索延迟. 为实现这一目标,我们设计了基于访问代价和簇大小的权重分配式替换策略.

在边缘RAG场景下,我们将每个向量簇f的访问代价定义为$\text{cost}(f)=\min\left(c_f^{\text{gen}},M\right)$,该定义反映了在当前边缘设备上检索该嵌入向量簇所需的最小延迟代价. 如算法2所示,主缓存算法维护每个缓存项的信用值$\text{credit}(f)$,用于衡量其保留价值. 算法整体逻辑如下:每当一个新对象f需要插入主缓存时,首先初始化其信用值为其访问代价$\text{cost}(f)$,然后将其与当前缓存内的所有向量簇组成临时集合G(第1~2行). 若G中的嵌入向量簇的总大小超过主缓存容量,则先计算单位信用消耗最小值λ,然后所有在集合G中的嵌入向量簇需要将自身的信用值根据大小扣减相应份额(第3~6行),并从集合G删除所有信用值为0的对象,直到G满足容量限制(第7行). 在这个时候,若最终$f\in G$,则将其插入缓存(第8~10行);否则旁路处理(第11~12行). 在该算法中,高成本、体积小的对象更易留存,反之则更可能被淘汰或绕过. 这种算法相比现有缓存算法的优势在于不依赖未来访问信息,也不基于启发式频率估计,而是直接对访问代价建模,适合处理例如边缘设备等访问代价异质性强、缓存空间有限的场景.

该信用值扣减算法在每次插入请求时,通过对临时集合中所有项进行等比例信用扣减,直至满足容量约束. 由于每次扣减量λ为正且信用值非负,该策略使得每次至少扣减一个对象的信用值为0,该过程必然在有限步内终止,因此算法具有确定的收敛性. 其竞争比

算法2 主缓存的插入与淘汰算法

输入：主缓存集合Main,请求插入的向量簇 f

输出：主缓存更新后的状态

1. $\text{credit}(f) \leftarrow \text{cost}(f)$
2. $G \leftarrow \text{Main} \cup f$
3. WHILE G 的容量大于Main的容量DO
4. $\lambda \leftarrow \min_{g \in G} \text{credit}(g)/\text{size}(g)$
5. FOR EACH $g \in G$ DO
6. $\text{credit}(g) \leftarrow \text{credit}(g) - \lambda \times \text{size}(g)$
7. $G \leftarrow G - \{g \in G \mid \text{credit}(g) = 0\}$
8. $\text{Main} \leftarrow G \cap \text{Main}$
9. IF $f \in G$ THEN
10. 将f插入到主缓存Main中
11. ELSE
12. RETURN //旁路f, 处理请求但不缓存

与收敛性在现有理论工作中已有深入分析[47]. 出于篇幅限制,我们未在正文中展开完整的收敛性证明.

4.3 开销分析

BP-Cache作为一种面向边缘RAG系统的在线缓存算法,在设计中兼顾了算法性能可控性与边缘设备运行可行性. 本小节从时间复杂度和空间开销两个方面,对该算法进行系统分析.

在时间复杂度方面,BP-Cache的运行流程可分为三个主要模块:(1)小缓存模块采用LRU策略,通过双向链表实现,插入、访问与淘汰操作的时间复杂度均为$O(1)$;(2)Ghost仅存储已被淘汰的向量簇ID,可通过哈希表实现,查询与插入均为$O(1)$,属常数级开销;(3)主缓存模块采用基于信用值的替换策略,其插入操作涉及对缓存集中所有向量簇的信用进行遍历更新. 设主缓存容量为K,则每次插入最多执行$O(K)$次信用更新;在最坏情况下,需执行$O(K)$次对象移除. 因此,BP-Cache每次请求的最坏时间复杂度为$O(K^2)$,适用于多数中小型边缘设备部署场景.

在空间复杂度方面,设主缓存容量为K,小缓存容量为K_s,Ghost表作为辅助数据结构表,最大保留N_g个历史记录项. 考虑到实际部署中通常设置$K_s \approx 0.1K$,且Ghost表仅存储冷向量簇的标识符,一般远小于K,因此整体内存占用为线性级别$O(K)$. 在Jetson Orin Nano、Jetson AGX Orin、Raspberry Pi 4等主流边缘平台上,Ghost表存储向量簇ID为32位整数(4 Byte),而一个典型嵌入向量簇(如768维FP16)占用约1.5 KB内存. 因此,Ghost表的空间开销仅为所记录向量实际内存占用的0.2%左右. 实验表明Ghost表的内存占用低于10 MB,相较于主缓存空间,此项管理开销对系统整体性能的影响可忽略不计.

综上所述,BP-Cache兼顾了边缘部署可行性与响应延迟优化,是一种适用于边缘RAG系统的高效、稳定的在线缓存管理方案.

5 实验评测

本节通过在真实边缘平台上部署检索增强生成系统来评估BP-Cache在边缘RAG系统中的性能表现,涵盖整体性能对比、关键组件分析与参数敏感性测试三部分. 实验重点衡量缓存策略对检索延迟、端到端响应时间与命中率的影响,并验证其在不同负载与配置下的稳定性与适应性.

5.1 实验设置

实验平台:本文在NVIDIA Jetson AGX Orin边缘平台上对所提出的算法进行实地部署与性能评估. 实验系统构建基于LlamaIndex框架实现检索增强生成(RAG)流程,底层向量索引采用FAISS实现. 大语言模型通过NanoLLM部署,使用gte-base-en-v1.5作为嵌入编码模型,Sheared-LLaMA-2.7B-ShareGPT作为生成模型完成端到端问答任务. 具体软硬件环境配置详见表2.

表2 实验平台软硬件规格

	软硬件规格
CPU	CortexA78AE 2.2 GHz 12cores
GPU	Ampere, 2048 CUDA cores, 64 Tensor cores
内存	64 GB
RAG框架	LLamaIndex v0.12.35
向量数据库	FAISS 1.7.4
LLM 引擎	NanoLLM r36.4.0
嵌入模型	gte-base-en-v1.5 (Dim=768)
生成模型	Sheared-LLaMA-2.7B-ShareGPT
系统软件	Jetpack 6.2

数据集:本文在BEIR基准测试集上评估BP-Cache性能,BEIR数据集是一个综合性的基准测试套件,用于评估信息检索系统在不同领域的性能,如科学论文、金融问答、网络问题、自然问题、多跳问答和索赔验证等. 本文选取了其中五个具有代表性的数据集nfcorpus、scidocs、fiqa、quora、nq以从多个维度来评估BP-Cache的算法性能,数据集的信息如4.1节中表1所示.

基线算法:本文将BP-Cache与以下三种具有代表性的基线算法进行对比,覆盖了不同的调度策略.

Online[22]:具有在线集群嵌入生成的两级IVF索引. 该策略将所有查询请求所需的文档嵌入均在查询时由在线模型即时生成,不依赖任何预存索引或缓存机制. 此方式可确保嵌入表示的最新性,节约了存储空间,但同时也带来了较高的计算延迟.

Hybrid:该策略采用在线生成与从本地存储的预计

算嵌入向量加载相结合的方式. 系统根据预估的生成延迟与加载延迟,动态选择嵌入向量的获取方式. 该方法在性能和实时性之间取得一定平衡,但在高并发场景下仍可能出现延迟波动.

EdgeRAG[11]:当前最先进的边缘检索增强生成方法,主要特点在于其通过聚类内嵌入剪枝技术缓解内存压力,并在检索阶段按需生成嵌入向量. 针对长尾聚类可能引发的延迟问题,系统预计算并存储其嵌入向量,同时使用缓存处理剩余嵌入,从而实现更优的存储利用与检索效率.

本文涉及的相关算法的实现代码已公开,访问地址为 https://github.com/091827sys/BPCache/. 该仓库包含了算法核心模块、实验配置及运行说明.

5.2　总体性能

5.2.1　检索延迟分析

我们评估了不同边缘 RAG 方案的尾部延迟. 图 6 说明了各检索策略在 fiqa 数据集上的检索延迟分布. 可以观察到,BP-Cache 在整体延迟控制方面表现最优,绝大多数查询请求的完成时间显著缩短,例如,在 80% 分位点下,BP-Cache 将检索延迟控制在约 3 s 以内,而 Online 策略延迟超过 10 s,Hybrid 与 EdgeRAG 则分别在 6~7 s 附近. 这表明,BP-Cache 相比 EdgeRAG 所进行的缓存优化,显著优化了检索效率. 在尾部延迟控制方面,BP-Cache 的优势更为明显. 传统 Online 与 Hybrid 策略在 95% 分位点之后延迟增长迅速,最坏情况下达到 20~30 s;而 EdgeRAG 虽通过部分嵌入缓存减少了延迟,但仍存在长尾波动,尾部请求在 15~20 s 范围内. 相比之下,BP-Cache 通过动态旁路和基于成本权衡的缓存机制,有效规避了长尾生成,能将最坏情况的最大检索延迟控制在 13 s 以内,这一能力对于边缘设备在实时问答、工业监控等延迟敏感任务中具有关键意义.

5.2.2　端到端延迟分析

端到端延迟可分为三个主要阶段:(1)检索延迟,

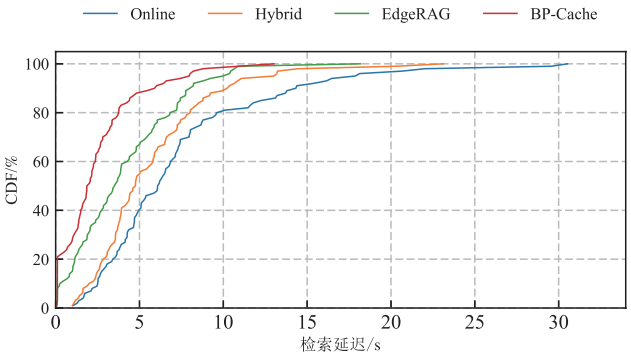


图 6　不同优化策略下的检索延迟分布对比

即通过向量相似度搜索识别相关文本块所需的时间;(2)预填充延迟,从模型接收到输入(包括查询语句和检索到的文本块)到生成第一个输出 token 的时间;(3)生成延迟,即从生成第一个 token 到生成最后一个 token 所需的时间. 检索延迟与预填充延迟之和(即首 token 响应时间)直接影响用户的感知延迟,因为它反映了从用户提交查询到系统返回初始响应之间的等待时间,是用户体验的关键指标.

图 7 展示了在五个不同数据集上,各方案在三个关键阶段(检索、预填充、生成)上的平均延迟,构成了完整的端到端响应时间. 整体来看,BP-Cache 在所有数据集上均实现了最低的总延迟,且其优势主要体现在检索阶段的显著优化,相比 EdgeRAG 平均减少了约 29% 的检索延迟. 例如,在 nfcorpus 和 nq 数据集上,BP-Cache 将检索延迟控制在约 1~2 s 范围,而 Online 与 Hybrid 策略的检索阶段耗时普遍在 5 s 以上. 相比之下,EdgeRAG 虽在部分数据集上有所缓解,但仍存在缓存污染与加载瓶颈,检索延迟整体高于 BP-Cache. 此外 BP-Cache 并不显著改变生成阶段的延迟,这说明它的主要贡献在于优化模型前处理路径,避免了直接干扰语言模型本身.

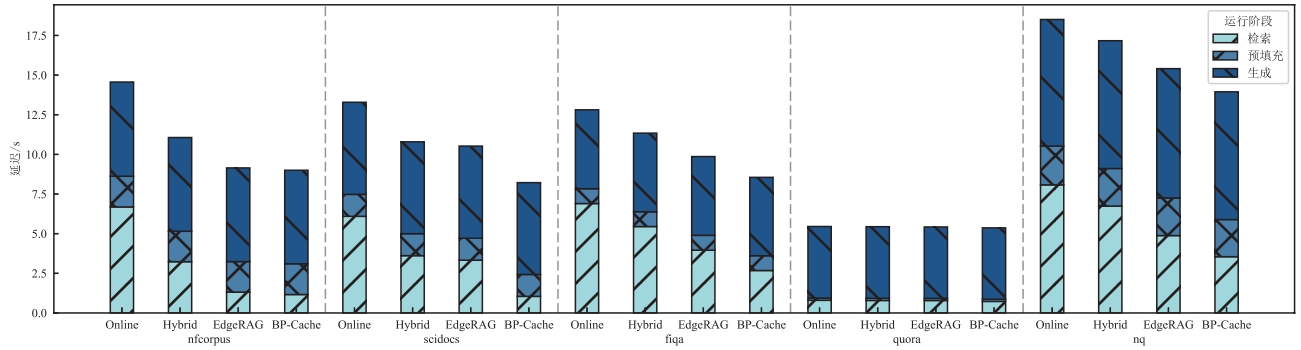


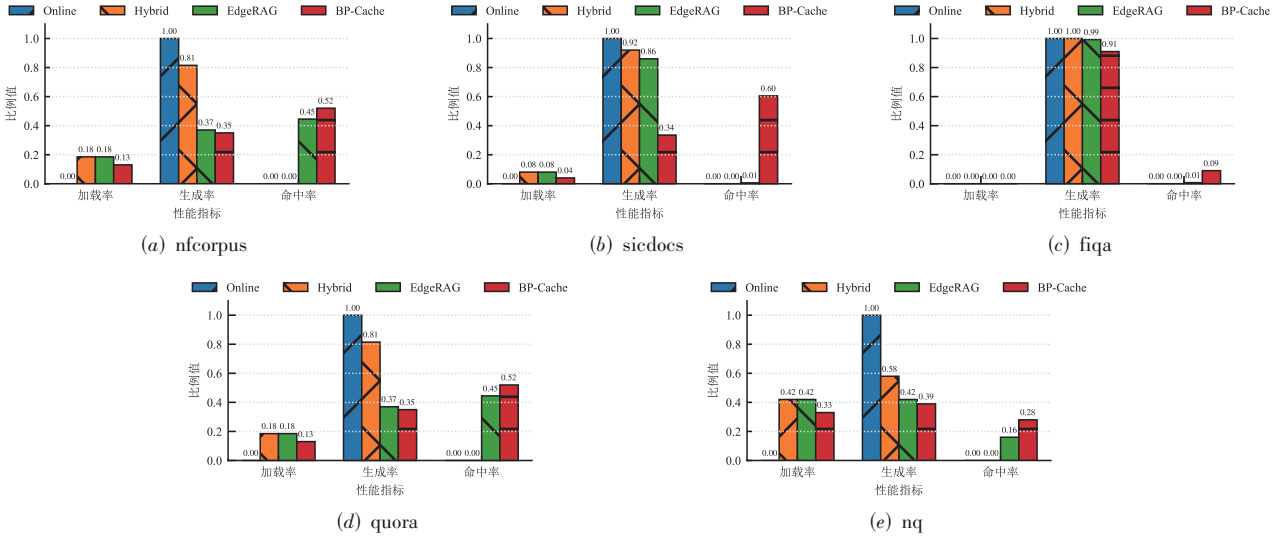
图 7　不同数据集上的检索、预填充、生成延迟对比

在 nfcorpus 数据集上,BP-Cache 相比 EdgeRAG 算法的优势并不显著. 这主要是因为 nfcorpus 数据集规模较小,可以完全加载到缓存中. 在这种情况下,EdgeRAG 的缓存机制已经能够充分发挥作用,使得

BP-Cache 的进一步优化空间有限. 因此,BP-Cache 和 EdgeRAG 在该数据集上的性能差距较小. 在 quora 数据集上,四种算法的表现几乎接近. 这是由于 quora 数据集的访问模式更偏向于扫描模式,这是一种对缓存不友好的模式,使得缓存策略无法充分发挥其优势. 在这种情况下,检索操作对缓存机制的依赖性较低,导致 BP-Cache 的性能提升空间有限.

5.2.3 缓存命中率分析

为了进一步分析 BP-Cache 缓存策略在检索延迟方面的影响,本文对四种算法的实际运行性能进行了详细对比,实验指标包括加载率(从本地磁盘加载的向量簇占比)、生成率(在线生成的向量簇占比)和缓存命中率,旨在全面评估 BP-Cache 算法的效果. 如图 8 所示,对于 Online 算法而言,由于所有检索向量均通过在线生成完成,因此其加载率和缓存命中率均为 0. 相比之下,Hybrid 算法虽然结合了两种策略,但由于未引入缓存机制,其缓存命中率同样为 0.



(a) nfcorpus (b) sicdocs (c) fiqa

(d) quora (e) nq

图 8 不同数据集下的各算法加载率、生成率和命中率对比

进一步分析可以发现,在五个数据集上,BP-Cache 算法相比 EdgeRAG 算法提升了约 21% 的缓存命中率,特别是在 scidocs 数据集上,命中率提升了近 60%,这表明 BP-Cache 算法能够更高效地利用缓存机制,减少重复数据加载所带来的性能开销. 而在加载率和生成率方面,BP-Cache 算法始终低于 EdgeRAG 算法,这说明 BP-Cache 通过缓存降低了其他高代价获取向量方式的占比,能够在降低生成负担的同时,减少对预计算和磁盘加载的依赖. 相比之下,EdgeRAG 虽然也利用了缓存机制,但其缓存策略仅基于生成成本判断,缺乏对各种调度成本的综合考虑.

5.3 关键组件分析

在这一节中将探究小缓存对于算法的影响. 我们在表 3 中展示了四种算法及其变体 BP-Cache-NoS(即未启用小缓存机制)的性能表现,涵盖检索延迟、加载率与缓存命中三个关键指标. 通过对比可以观察到,不使用小缓存的 BP-Cache-NoS 算法在部分数据集(如 nfcorpus)上的表现甚至劣于 EdgeRAG,尤其在检索延迟方面未体现出显著优势. 这一现象表明,在缓存机制的设计中,仅依赖主缓存进行嵌入向量管理,容易受到一次性访问请求的干扰,导致缓存空间被频繁替换、污染,最终削弱了命中效率. 引入小缓存机制后,BP-Cache 在多个数据集上均表现出显著的性能提升,尤其是在检索延迟方面取得了稳定的优化. 该机制采用了一种“先观察,后升级”的缓存策略,有效抑制了主缓存中“一次性访问”内容的涌入,避免了重要簇因短期噪声而被替换,从而提升了缓存的命中率与空间利用率. 此外,加载率的变化也从侧面反映出缓存使用效率的差异. 相比 BP-Cache-NoS,启用小缓存后的算法在保持较高命中率的同时,显著降低了不必要的加载开销,进一步缩短了用户请求的响应时间.

5.4 敏感性实验

本节通过对缓存大小和簇数量这两个 BP-Cache 算法中的核心变量展开分析与敏感性实验.

5.4.1 缓存大小对算法性能的影响

如图 9 所示,不同缓存大小下的四种算法在检索延迟、生成率和命中率上的表现存在显著差异. 随着缓存大小的增加,BP-Cache 算法的检索延迟呈现出明显的下降趋势,且在缓存增加的初始区间,其下降速度显著快于 EdgeRAG 算法. 这表明 BP-Cache 能够更高效地利用有限的缓存资源,通过优先缓存高收益簇,有效降低

表3　基线算法、未启用小缓存的BP-Cache与BP-Cache的检索延迟、加载率及缓存命中率对比

	Online			Hybrid			EdgeRAG			BP-Cache-NoS			BP-Cache		
	检索延迟/s	加载率	命中率	检索延迟/s	加载率	命中率	检索延迟/s	加载率	命中率	检索延迟/s	加载率	命中率	检索延迟/s	加载率	命中率
nfcorpus	6.685	0	0	3.233	0.185	0	1.316	0.185	0.445 0	1.550	0.16	0.48	1.162	0.13	0.520
scidocs	6.094	0	0	3.614	0.080	0	3.332	0.080	0.060 0	1.453	0.06	0.42	1.046	0.04	0.605
fiqa	6.900	0	0	5.450	0.010	0	3.963	0.010	0.240 0	3.358	0	0.37	2.677	0	0.405
quora	0.798	0	0	0.788	0	0	0.787	0	0.007 5	0.740	0	0.04	0.730	0	0.090
nq	8.079	0	0	6.741	0.420	0	4.879	0.420	0.160 0	3.954	0.37	0.25	3.549	0.33	0.280

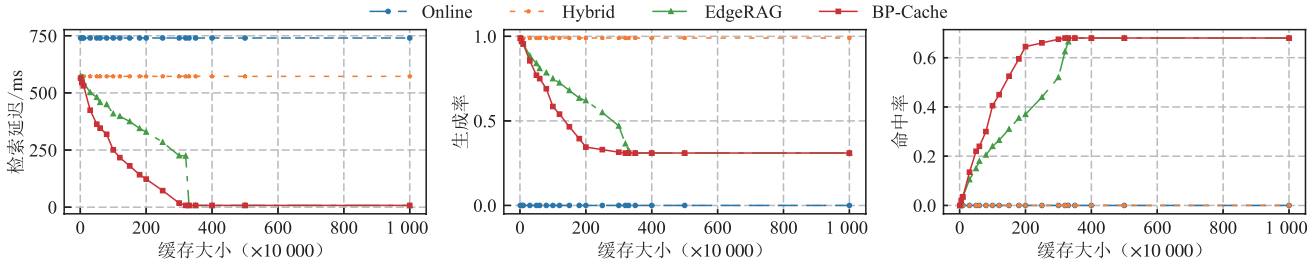


图9　不同缓存大小下各算法的检索延迟、生成率、缓存命中率对比,缓存大小的单位为向量尺寸的标准块

检索延迟. 从图9还可以观察到,BP-Cache的检索延迟下降趋势更加平稳,而EdgeRAG算法在缓存接近上限时出现了突降现象. 这一现象反映了EdgeRAG算法在缓存利用上的不足,即其在小缓存场景下无法充分挖掘缓存的潜力,而仅在缓存接近上限时才显现出性能提升. 这种突降行为表明EdgeRAG在缓存资源管理上存在较大改进空间. 相比之下,BP-Cache通过双级缓存策略的优化,不仅提升了小缓存场景下的性能,还在缓存逐步增加时保持了性能提升的稳定性,使得其在不同的缓存大小场景下均能更好地发挥性能优势.

进一步分析生成率和命中率的变化趋势也可以验证BP-Cache算法的设计合理性. 与其他算法相比,BP-Cache在缓存大小较小时便能保持较高的缓存命中率,这得益于其智能化的缓存分配策略,能够优先缓存生成成本较高但空间占用较小的簇,提高了单位缓存资源的使用效率. 这种策略不仅减少了生成操作的开销,还有效平衡了计算资源与存储资源之间的关系,从而在不同缓存大小下都能保持较优的性能表现.

5.4.2　缓存占比对算法性能的影响

为深入探究小缓存区容量的设置对系统性能与Ghost表过滤精度的影响,本文进一步设计了敏感性实验,评估了不同小缓存区容量占比(相对于总缓存容量)下的检索延迟、命中率及误判率. 误判率定义为:一个被Ghost表记录(即首次进入小缓存后因未被复用而被淘汰)的向量簇,在后续请求序列中又被访问的情况占总请求的比例. 结果如表4所示.

首先,完全移除小缓存区(占比0%)时,系统退化为单一主缓存结构,其检索延迟高达4.22 s,命中率仅为21.6%. 这验证了小缓存区作为过滤“一次性访问”冷向量的前置缓冲区至关重要,能有效避免主缓存被污染. 其次,随着小缓存区占比从1%逐步增加,系统性能显著提升. 当小缓存区占比达到10%时,系统取得了最优的综合性能,检索延迟降至最低的2.68 s,同时命中率提升至40.5%. 这表明一个适度规模的小缓存区能最有效地识别出有复用价值的对象,在不过度挤占主缓存空间的前提下,高效地过滤掉大部分冷数据.

表4　BP-Cache在不同小缓存大小占比下的检索延迟、命中率及误判率

小缓存区占比/%	主缓存区占比/%	检索延迟/s	命中率/%	误判率/%
0	100	4.217 5	21.6	0
1	99	3.975 5	24.2	17.3
5	95	3.367 4	32.7	12.2
10	90	2.677 2	40.5	3.6
25	75	2.702 4	40.1	2.8
50	50	2.974 1	38.7	2.0
75	25	3.426 7	30.4	0.9
100	0	4.015 6	24.3	0.5

值得注意的是,误判率随着小缓存区占比的增加而单调下降. 当小缓存区占比仅为1%时,由于容量极度有限,大量向量簇尚未等到第二次访问即被淘汰并标记入Ghost表,导致后续访问被误判为“冷数据”而无法直接进入主缓存,因此误判率高达17.3%. 随着小缓存区容量扩大,向量簇在其中获得更长的观察期,误判率迅速下降,但是盲目增大小缓存占比并非良策. 当小缓存区占比超过25%后,由于过度挤占了主缓存的空间,用于存储高频热向量簇的资源减少,导致整体命中率开始下降,检索延迟也随之上升. 当小缓存区占比达

到100%时，系统退化为仅使用LRU缓存，性能与EdgeRAG相近．这一现象表明，小缓存区的容量配置是在过滤精度（低误判率）与热数据驻留空间（高命中率）之间进行权衡的关键．实验表明，将总缓存容量的10%至25%分配给小缓存区是一个合理的设置区间，能在控制误判率的同时，最大化系统的整体性能．

5.4.3 向量簇数目对算法性能的影响

图10中展示了在不同向量簇个数设置（即聚类中心数目）下，四种调度算法在检索任务中的延迟表现．整体来看，随着簇个数的增加，检索延迟呈现出一个“先下降后上升”的变化趋势，说明簇的划分粒度对系统性能有显著影响．

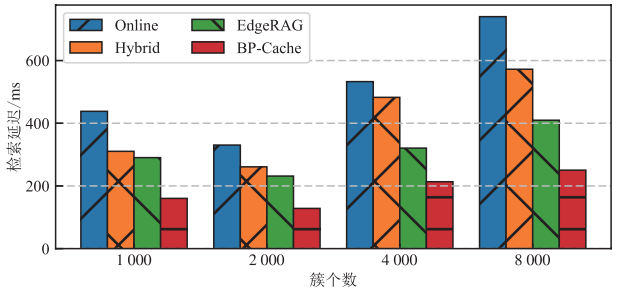


图10 不同向量簇个数下各算法的检索延迟对比

具体来说，在簇个数较少（如个数为1 000）时，检索延迟相对较高．这是由于每个簇内包含的数据量较大，导致局部检索范围过大，即使先通过簇划分缩小了候选集，但仍需在一个较大的簇中完成精确匹配，因此整体延迟较高．随着簇个数的增加，单个簇中的样本数量减少，局部检索空间进一步缩小，系统可以更快速地定位到目标数据，从而显著降低检索延迟．当簇个数增加至2 000时，系统的检索延迟达到最优．这表明在该粒度下，簇划分实现了较为合理的局部性与检索成本之间的平衡：既有效减少了每次检索所需遍历的数据量，又未引入过多额外的簇选择开销．然而，继续增加簇个数（超过2 000）后，检索延迟反而上升．这是因为，簇数量过多会导致簇选择过程变得复杂，簇匹配阶段的计算量增加，从而拖慢整体响应时间；其次，在簇划分过细的情况下，相邻簇之间的语义相似性提高，容易导致目标数据分布碎片化，降低了划分策略的效果，进而影响最终检索效率．

图11展示出的生成率和图10展示出的检索延迟趋势基本吻合．而折线展示出的命中率部分，可以看到虽然命中率随着簇的个数增加而增加，但由于簇数量过多会导致簇选择过程变得复杂，尽管命中率有所提升，但计算量的增加仍会拖慢整体响应时间．此外，我们所设计的BP-Cache算法在不同簇个数下，相对其他算法仍然保持比较明显的性能领先，这表明BP-Cache算法具有良好的稳定性．

接下来，本文探究了不同簇个数下边缘RAG系统

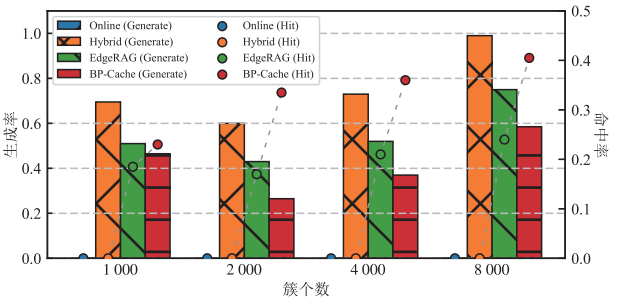


图11 不同向量簇个数下各算法的生成率、命中率对比

的生成质量对比．Online、Hybrid、EdgeRAG、BP-Cache四种检索方式均基于二级倒排索引（Two-level IVF Index）[22]架构，由于生成质量主要依赖于倒排索引的聚类精度与检索覆盖范围，而缓存策略仅影响检索延迟（如是否命中或生成/加载），因此上述方法的生成质量差异主要源于簇划分粒度．本文将其与基于Flat检索方式[19]的生成质量进行对比．图12中可以看出，随着簇个数的增加，生成质量呈现出下降的变化趋势．当簇个数较少（如1 000）时，单个簇内数据量大、检索范围广，生成质量较高（如nq数据集评分达3.3），但是缓存检索延迟较高、命中率较低；随着簇个数增至2 000~4 000，簇内数据量减少，检索范围缩小，生成质量稍有降低；簇个数进一步增至8 000后，簇选择阶段的计算开销增加，且语义相似性导致的数据碎片化降低了划分效果，造成生成质量下降、检索延迟增加．与精确匹配的Flat方法相比，IVF-Based方法在合理簇划分下的生成质量接近甚至超越Flat（如quora数据集IVF-Based-1 000的评分为3.2，Flat的评分为3.0），同时避免了内存占用过高的问题．不同数据集表现差异反映了内容分布特性的影响，如语义清晰的数据集（如nq）生成质量更高．综上，簇个数需平衡检索效率与生成质量以实现性能与质量的最优权衡．

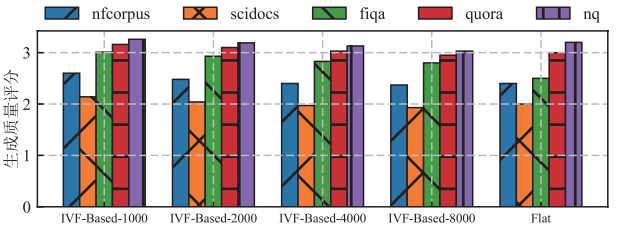


图12 不同向量簇个数下，基于IVF(Online、Hybrid、EdgeRAG、BP-Cache)与基于Flat检索方式的生成质量对比

5.4.4 边缘平台对算法性能的影响

为验证BP-Cache在不同边缘硬件配置下的适应性与鲁棒性，我们在Jetson AGX Orin与JetsonOrin Nano两类典型边缘平台上部署了相同的RAG系统并运行对比实验．两平台主要差异在于计算资源，Nano的GPU CUDA核数仅为AGX Orin的1/4，CPU核数减半，内存

从64 GB降至4 GB.

如表5所示,由于Nano的计算与I/O资源更为紧张,所有算法的绝对检索延迟均显著高于在Orin平台上的结果. 例如,BP-Cache的延迟从2.677 s增加至5.396 s,这主要源于嵌入向量生成速度的下降和内存容量变小导致的缓存命中率下降. 然而,BP-Cache相对于其他算法的性能优势依然稳固. 在Nano上,BP-Cache相比性能最接近的基线算法EdgeRAG,仍能降低约27%的检索延迟. 这表明BP-Cache的优化效果对硬件性能变化不敏感,其优势源于算法逻辑本身对工作负载特性的有效利用,具有良好的平台鲁棒性. 在Nano上,我们还观察到Hybrid和EdgeRAG算法的加载率从Orin平台上的极低水平(0.01)显著上升至0.43以上. 这是因为在计算能力较弱的Nano平台上,在线生成的代价变得更高,使得系统更倾向于从磁盘加载预计算的向量. 然而,BP-Cache的加载率仍然低于基线算法. 这得益于BP-Cache高效的缓存机制,它通过更高的命中率满足了更多请求,从而减少了对高代价的生成和加载操作的依赖. 这一对比表明,在资源越受限的平台下,精准的缓存决策对于平衡计算和I/O开销越关键.

表5　不同边缘平台上各算法的检索延迟、加载率、缓存命中率对比

	Online			Hybrid			EdgeRAG			BP-Cache		
	检索延迟/s	加载率	命中率	检索延迟/s	加载率	命中率	检索延迟/s	加载率	命中率	检索延迟/s	加载率	命中率
Orin	6.900	0	0	5.450	0.01	0	3.963	0.01	0.24	2.677	0	0.405
Nano	15.657	0	0	10.731	0.43	0	7.425	0.43	0.11	5.396	0.36	0.210

6　结论

本文围绕边缘侧检索增强生成(RAG)系统的高效部署问题,系统性地分析了当前在嵌入向量检索路径选择与缓存机制方面存在的性能瓶颈. 针对边缘设备受限的内存与计算资源条件,结合真实运行特性和数据分布规律,本文提出了一种高效的在线缓存算法BP-Cache. 该方法基于对检索代价的尾部分布特性和访问频次模式的关键观察,设计了包含旁路机制、小缓存过滤器与主缓存信用管理的分级缓存结构,可统一调度三类嵌入向量获取路径,提升缓存效率、减少冗余计算. 在Jetson Orin等实际边缘平台上的部署验证显示,BP-Cache在多个基准数据集上平均减少了约29%的检索延迟,提升了约21%的缓存命中率,同时保持了低内存占用和稳定的生成质量,验证了其在边缘智能问答等延迟敏感任务中的实用价值.

参考文献

[1] GUO S T, LIU J D, YANG Y Y, et al. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing[J]. IEEE Transactions on Mobile Computing, 2019, 18(2): 319-333.

[2] WANG T, LIANG Y Z, SHEN X W, et al. Edge computing and sensor-cloud: Overview, solutions, and directions[J]. ACM Computing Surveys, 2023, 55(13s): 1-37.

[3] JIN Y Y, ZHONG R X, LONG S Q, et al. Efficient inference for pruned CNN models on mobile devices with holistic sparsity alignment[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(11): 2208-2223.

[4] FAN W Q, DING Y J, NING L B, et al. A survey on RAG meeting LLMs: Towards retrieval-augmented large language models[C]//Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. New York: ACM, 2024: 6491-6501.

[5] LEWIS P, PEREZ E, PIKTUS A, et al. Retrieval-augmented generation for knowledge-intensive NLPtasks[EB/OL]. (2021-04-12)[2025-10-01]. https: //arXiv.org/abs/2005.11401.

[6] YU H, GAN A R, ZHANG K, et al. Evaluation of retrieval-augmented generation: A survey[C]//CCF Conference on Big Data. Singapore: Springer Nature Singapore, 2024: 102-120.

[7] NING H S, LI Y F, SHI F F, et al. Heterogeneous edge computing open platforms and tools for Internet of Things[J]. Future Generation Computer Systems, 2020, 106: 67-76.

[8] XU M W, FU Z, MA X, et al. From cloud to edge: A first look at public edge platforms[C]//Proceedings of the 21st ACM Internet Measurement Conference. New York: ACM, 2021: 37-53.

[9] KIM C, LEE C. Design of eMMC controller with multiple channels[C]//2016 International SoC Design Conference. Piscataway: IEEE, 2016: 317-318.

[10] XU Q M, SIYAMWALA H, GHOSH M, et al. Performance analysis of NVMe SSDs and their implication on real world databases[C]//Proceedings of the 8th ACM International Systems and Storage Conference. New York: ACM, 2015: 2757684.

[11] SEEMAKHUPT K, LIU S H, KHAN S. EdgeRAG: Online-indexed RAG for edge devices[EB/OL]. (2024-12-31)[2025-10-01]. https://arXiv.org/abs/2412.21023.

[12] SIVIC, ZISSERMAN. Video Google: A text retrieval approach to object matching in videos[C]//Proceedings of Ninth IEEE International Conference on Computer Vision. Piscataway: IEEE, 2008: 1470-1477.

[13] SALTON G, WONG A, YANG C S. A vector space model

for automatic indexing[J]. Communications of the ACM, 1975, 18(11): 613-620.

[14] GITHUB. GitHub-facebookresearch/faiss: A library for efficient similarity search and clustering of dense vectors[EB/OL]. (2025-10-15) [2025-10-20]. https://github. com/huggingface/diffusers?tab=contributing-ov-file.

[15] MALKOV Y A, YASHUNIN D A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, 42(4): 824-836.

[16] HAN K, XIAO A, WU E, et al. Transformer in transformer[EB/OL]. (2021-10-26)[2025-10-01]. https://arxiv.org/abs/2103. 00112.

[17] RAFFEL C, SHAZEER N, ROBERTS A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer[J]. Journal of Machine Learning Research, 2020, 21(1): 5485-5551.

[18] BROWN T B, MANN B, RYDER N, et al. Language models are few-shot learners[C]//Proceedings of the 34th International Conference on Neural Information Processing Systems. New York: ACM, 2020: 1877-1901.

[19] WEAVIATE. Weaviate Database[EB/OL]. [2025-10-01]. https://weaviate.io/developers/academy/py/vector_index/flat.

[20] ZHANG Y, JIN R, ZHOU Z H. Understanding bag-of-words model: A statistical framework[J]. International Journal of Machine Learning and Cybernetics, 2010, 1(1): 43-52.

[21] YAN H, DING S, SUEL T. Inverted index compression and query processing with optimized document ordering[C]// Proceedings of the 18th International Conference on World Wide Web. New York: ACM, 2009: 401-410.

[22] ROBERTSON S, ZARAGOZA H. The probabilistic relevance framework: BM25 and beyond[J]. Foundations and Trends in Information Retrieval, 2009, 3(4): 333-389.

[23] KARPUKHIN V, OGUZ B, MIN S, et al. Dense passage retrieval for open-domain question answering[EB/OL]. (2020-09-30)[2025-10-10]. https://arxiv.org/abs/2004.04906.

[24] FAN T Y, WANG J Y, REN X B, et al. MiniRAG: Towards extremely simple retrieval-augmented generation[EB/OL]. (2025-01-26)[2025-10-10]. https://arXiv.org/abs/2501.06713.

[25] LIU G Y, LIU Y Q, WANG J C, et al. Adaptive contextual caching for mobile edge large language model service[EB/OL]. (2025-01-16)[2025-10-02].https://arXiv.org/abs/2501.09383.

[26] RAY S, PAN R, GU Z, et al. METIS: Fast quality-aware RAG systems with configuration adaptation[C]//Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles. NewYork: ACM, 2025: 606-622.

[27] WANG Y C, LIU S, LI Z F, et al. LEANN: A low-storage vector index[EB/OL]. (2025-06-09) [2025-10-10]. https://arXiv.org/abs/2506.08276.

[28] PARK T, LEE G, KIM M S. MobileRAG: A fast, memory-efficient, and energy-efficient method for on-device RAG[EB/OL]. (2025-07-01) [2025-10-10]. https://arXiv.org/abs/2507.01079.

[29] SLEATOR D D, TARJAN R E. Amortized efficiency of list update and paging rules[J]. Communications of the ACM, 1985, 28(2): 202-208.

[30] LEE D, CHOI J, KIM J H, et al. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies[J]. IEEE Transactions on Computers, 2001, 50(12): 1352-1361.

[31] JOHNSON T, SHASHA D. 2Q: A low overhead high performance buffer management replacement algorithm[C]// Proceedings of the 20th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc, 1994: 439-450.

[32] MEGIDDO N, MODHA D S. ARC: A self-tuning, low overhead replacement cache[C]//2nd USENIX Conference on File and Storage Technologies (FAST 03). California: USENIX Association, 2003: 115-130.

[33] YANG J C, ZHANG Y Z, QIU Z Y, et al. FIFO queues are all you need for cache eviction[C]//Proceedings of the 29th Symposium on Operating Systems Principles. New York: ACM, 2023: 130-149.

[34] TAN H S, JIANG S H, HAN Z H, et al. Camul: Online caching on multiple caches with relaying and bypassing[C]//IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. Piscataway: IEEE, 2019: 244-252.

[35] HU X Y, RAMADAN E, YE W, et al. Raven: Belady-guided, predictive (deep) learning for in-memory and content caching[C]//Proceedings of the 18th International Conference on Emerging Networking Experiments and Technologies. New York: ACM, 2022: 72-90.

[36] ZHOU W B, NIU Z X, XIONG Y Q, et al. 3L-Cache: Low overhead and precise learning-based eviction policy for caches[C]//23rd USENIX Conference on File and Storage Technologies (FAST 25). California: USENIX Association, 2025: 237-254.

[37] RODRIGUEZ L V, YUSUF F B, LYONS S, et al. Learning cache replacement with CACHEUS[C]//19th USENIX Conference on File and Storage Technologies. California: USENIX Association, 2021: 341-354.

[38] CHEN J Y, SHARMA N, KHAN T, et al. Darwin: Flexible learning-based CDN caching[C]//Proceedings of the ACM SIGCOMM 2023 Conference. New York: ACM, 2023: 981-999.

[39] 王玉庆, 杨秋松, 李明树 . 基于指令流混合模式学习的缓存预取算法[J]. 电子学报, 2023,51(2): 342-354.
WANG Y Q, YANG Q S, LI M S. A cache prefetching mechanism based on hybrid pattern learning of instruction

flow[J]. Acta Electronica Sinica, 2023, 51(2): 342-354. (in Chinese)
[40] ZHANG Z Y, SHENG Y, ZHOU T Y, et al. H_2O: Heavy-hitter oracle for efficient generative inference of large language models[EB/OL]. (2023-12-18)[2025-10-01]. https://arXiv.org/abs/2306.14048.
[41] WANG J H, HAN J B, WEI X D, et al. KVCachecache in the wild: Characterizing and optimizing KVCache cache at a large cloud provider[EB/OL]. (2025-07-23)[2025-10-01]. https://arXiv.org/abs/2506.02634.
[42] GAO B, HE Z M, SHARMA P, et al. Cost-efficient large language model serving for multi-turn conversations with Cached Attention[C]//2024 USENIX Annual Technical Conference.California: USENIX Association, 2024: 111-126.
[43] YAO J Y, LI H C, LIU Y H, et al. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion[C]//Proceedings of the Twentieth European Conference on Computer Systems. New York: ACM, 2025: 94-109.
[44] GIM I, CHEN G J, LEE SS, et al. Prompt cache: Modular attention reuse for low-latency inference[EB/OL]. (2024-04-25)[2025-10-10].https://arXiv.org/abs/2311.04934.
[45] LIU G D, LI C W, ZHAO J R, et al. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression[C]//2025 62nd ACM/IEEE Design Automation Conference. Piscataway: IEEE, 2025: 1-7.
[46] ADAMASZEK A, CZUMAJ A, ENGLERT M, et al. An $O(\log k)$-competitive algorithm for generalized caching[J]. ACM Transactions on Algorithms, 2018, 15(1): 1-18.
[47] EPSTEIN L, IMREH C, LEVIN A, et al. Online file caching with rejection penalties[J]. Algorithmica, 2015, 71(2): 279-306.

作者简介

詹慧悠　女,1998年1月生,广东韶关人.现为中国科学技术大学计算机科学与技术学院博士研究生.主要研究方向为边缘智能、在线算法等.
E-mail: zhanhuiyou@foxmail.com

倪宏秋　女,2000年生,安徽合肥人.中国科学技术大学计算机科学与技术专业博士研究生.主要研究方向为大模型推理调度.
E-mail: nhq0806@mail.ustc.edu.cn

谈海生　男,1981年生.中国科学技术大学人工智能与数据科学学院,教授,博士生导师,副院长.主要研究方向为边缘计算、工业互联网、AI+Edge、网络算法设计与系统实现.
E-mail: hstan@ustc.edu.cn

王天竹　男,2003年生,河北唐山人.中国科学技术大学数据科学专业硕士研究生.主要研究方向为边缘智能与大语言模型等.
E-mail: wtz2333@mail.ustc.edu.cn

李向阳　男,1971年10月生,江苏泰兴人.中国科学技术大学计算机科学与技术学院教授、博士生导师.现任中国科大信息与智能学部执行部长,计算机科学与技术学院执行院长,中国科学院无线光电通信重点实验室主任.主要研究方向为智能物联网、边缘计算、数据共享计算、物联网安全和数据安全隐私等.中国电子学会会员编号:E190072308M.
E-mail: xiangyangli@ustc.edu.cn