FaTRQ：面向远内存感知近似最近邻搜索系统的大语言模型向量搜索分层残差量化

Tianqi Zhang加州
大学圣地亚哥分校美国
加利福尼亚州拉霍亚
tiz014@ucsd.edu

Flavio Ponzina圣迭戈
州立大学美国加利福尼亚州
圣迭戈
fponzina@sdsu.edu

Tajana Rosing加
州大学圣地亚哥分校美
国加利福尼亚州拉霍亚
tajana@ucsd.edu

arXiv:2601.09985v1 [cs.LG] 15 Jan 2026

摘要—近似最近邻搜索（ANNS）是检索增强生成（RAG）中的一项关键技术，可从海量向量数据库中快速识别最相关的高维嵌入。现代近似最近邻搜索引擎利用预构建索引加速这一过程，并将压缩后的向量量化表示存储在快速内存中。然而，它们仍依赖成本高昂的第二阶段精化流程，需要从固态硬盘等较慢的存储介质读取全精度向量。对于现代文本和多模态嵌入，这些读取如今已成为整个查询延迟的主要来源。我们提出 FaTRQ，这是一种采用分层内存的远内存感知精化系统，无需从存储中获取完整向量。该系统引入渐进式距离估计器，利用从远端内存流式读取的紧凑残差来精化粗略分数。一旦能够确定某个候选项不可能进入前 k 名，精化过程便会提前停止。为此，我们提出分层残差量化，将残差编码为三元值，并高效地存储在远端内存中。我们在 CXL Type-2 设备中部署了定制加速器，以便在本地执行低延迟精化。综合而言，与最先进的 GPU 近似最近邻搜索系统相比，FaTRQ 将存储效率提升了 2.4× ，吞吐量最高提升了 9×。

索引词—近似最近邻搜索、残差量化、分层内存、CXL

I. 引言

最近邻搜索（NNS）是现代数据系统的核心组件，支持语义检索、推荐、欺诈检测和大规模排序等任务。尽管精确 NNS 在大规模场景下难以实现，近似最近邻搜索（ANNS）能够以低得多的成本实现高召回率，如今已成为生产级系统的标准配置。

ANNS 的一个主要应用场景是检索增强生成（RAG）。如图 1 所示，检索增强生成流水线会为每个文档块生成一次嵌入，将生成的向量缓存起来，并在查询时为用户提示生成嵌入，随后启动 ANNS 搜索，再将搜索结果传递给语言模型。现代稠密嵌入模型（例如 OpenAI 的模型 [1]）会生成 1536 维向量，每个向量约占 6 kB，
全精度。对于包含数百万甚至数十亿条目 [2],的知识库，这很快就会超出主存容量，使精确搜索变得不可行。

为减轻内存压力并加快搜索，现代引擎将索引数据结构（例如 IVF [3]）与紧凑的向量编码相结合，例如乘积量化系列 [4]–[6]。这些方法将高维向量量化为短编码，例如将一个 6 kB 的浮点向量压缩到约 200B，从而使整个索引都能存入主存。查询时，图会引导搜索过程，并使用紧凑编码估算距离。

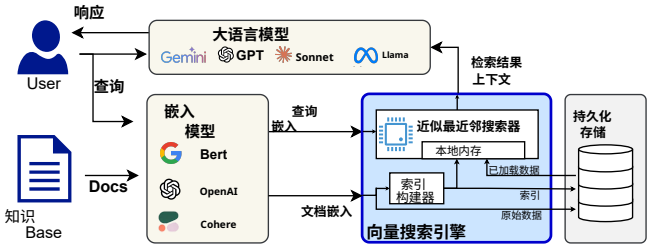

图1. 近似最近邻搜索是检索增强生成流水线的关键组成部分。

然而，量化也有代价：为了恢复完整精度，系统会从存储中读取原始全精度向量，并重新计算精确距离，以便对较长的候选列表重新排序。对于现代稠密嵌入 [1], [7], ，这一步精化如今已成为查询延迟的主要来源。我们在 II-A节 中的详细分析显示，超过 90% 的查询时间可能耗费在从存储中读取向量上。更糟的是，内存与存储带宽之间的差距不断扩大，使这一瓶颈日益严重。

与此同时，CXL 内存扩展器和存储级内存（SCM）等分层内存技术，为重新审视这一精炼阶段带来了机遇。尽管紧凑编码比完整精度向量小得多，但这些编码的总工作集仍可能超出本地DRAM（快速内存）的容量，尤其是在大规模数据库或多租户工作负载中。在这类情况下，这些分层内存（远端内存）提供了一种颇具吸引力的选择：它们在容量和延迟方面都介于本地DRAM与固态硬盘之间，因此非常适合存储紧凑的中间数据结构，而非完整向量。然而，现有近似最近邻搜索系统 [8]–[11]很少利用这些层级，仍将精炼阶段的输入/输出视为不可避免的瓶颈。

为利用这一机会，我们提出了**FaTRQ**，一种面向远端内存的精化系统，通过**分层残差量化**，消除了高精度近似最近邻搜索中大部分开销高昂的输入/输出。其核心思想是渐进式距离估计：FaTRQ 不获取全精度向量，而是利用紧凑的残差编码逐步精化粗略距离。为此，FaTRQ 引入了分层残差量化 (TRQ)，其中“T”既体现了其分层感知 内存设计，也体现了其三元 码本。TRQ 将记录与其粗量化结果之间的残差编码为三元值，存储在远端内存中，并在查询时以流式方式读取。该估计器基于向量分解和轻量级距离估计

校准模型，无需重构完整向量即可累积残差贡献。直接处理三元编码时，它仅通过加法和减法来精化距离，避免乘法运算，同时保持高精度。尽管 FaTRQ 可在分层内存系统上以软件方式实现，我们还构建了一个采用轻量级定制逻辑的 CXL Type-2 加速器概念验证，展示了 FaTRQ 如何集成到新兴的远端内存硬件中。

我们的贡献总结如下：

- 面向分层内存的多级向量量化：我们设计了一种分层压缩格式，将粗略量化向量存储在快速内存中，并将紧凑的残差码存储在较慢的内存层级中，从而以最少的输入/输出实现细粒度的精度控制。
- 渐进式距离估计：我们提出了一种增量评分机制，无需重构向量即可更新粗略距离，将内存流量从数百字节降低至 4 字节。
- 加速精炼流水线：我们为残差精化开发了一种远端内存加速器原型，从而减少数据移动和主机 CPU 开销。
- 主要结果：与图形处理器近似最近邻搜索系统 [9], 相比，近似最近邻搜索吞吐量提升了 2.6× 至9.4× ，存储效率较最先进流水线中的精炼方案 [12]提高了 2.4× 。

II. 背景与相关工作

A. 近似最近邻搜索系统

现代近似最近邻搜索引擎将索引（例如 IVF [3], CAGRA [13]）与向量量化结合起来，使大规模集合能够保留在内存中，同时支持快速距离估计 [3], [11]–[15]。索引负责剪枝搜索空间，而量化编码则支持低成本评分。在索引遍历期间，系统会获取量化编码并确定一部分候选项。然而，在检索增强生成工作负载中，现代高维嵌入（例如 768 维 [7], [16]）需要进行激进量化才能装入内存，这会降低召回率，因此必须进行二次精炼，从固态硬盘中读取全精度向量 [14], [17]。图 2 显示，在 GPU 加速流水线 [9] 中，这种精炼占据了大部分延迟；在该流水线中，索引和量化向量位于 GPU 内存中，而全精度向量则通过 mmap 映射在主机上。在配备 40 线程 CPU 的 A10 GPU 上，由于 GPU 加速，索引遍历仅占查询时间的 2–15%；相比之下，随机固态硬盘输入/输出和距离计算使精炼成为主要耗时。如果所有向量都能驻留在主内存中

内存，这在大规模场景下是不切实际的假设；性能最多可提升 14×。这一无法实现的上限促使我们采用渐进式距离估计方法，通过紧凑的残差编码逐步精化候选项：减少完整向量读取、降低细化开销，并避免以硬件为中心的解决方案所面临的容量上限。

B. 向量量化与距离估计

向量量化将高维向量压缩为紧凑的代码，从而降低存储成本并实现高效

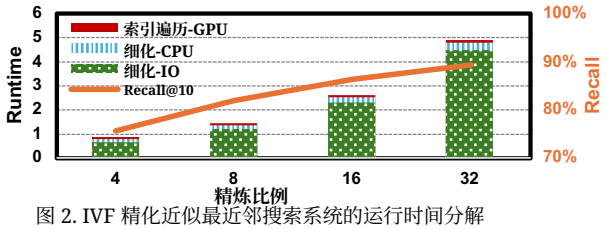

图 2. IVF 精化近似最近邻搜索系统的运行时间分解

距离计算。最简单的形式是标量量化(SQ)，它对每个维度独立离散化，但随着向量长度增加，扩展性较差。乘积量化(PQ) [3],[4], [18]通过将向量划分为多个子空间，并使用单独的码本对每个子空间进行量化，从而提升扩展性，实现基于查找表的高效距离查询和图形处理器加速 [3], [12]–[14]。残差量化(RQ) [19], [20]则通过将向量编码为粗略近似值加上连续的残差修正，进一步提高精度，通常在每个阶段使用乘积量化。为估计距离，大多数系统采用非对称距离计算 (ADC) [21], , 将全精度查询向量与量化后的数据库代码进行比较：

$$\hat{d}(q,x) = d\left(q, \sum_{i=1}^{L} Q^{-1}(c_i)\right),$$

其中 c_i 是第 i 层的残差量化代码。虽然现有系统效果良好，但在遍历过程中，它们会为每个向量解码所有量化层级，尽管大多数候选项很快就会被过滤掉。因此，解码的候选项比必要数量多出 10-100× ，浪费了带宽，因为这些向量会立即被丢弃。FaTRQ 通过支持对残差量化进行渐进式非对称距离计算，解决了这一低效问题：较早的残差量化层级存储在快速内存中，而更深层的残差编码则分层存放在较慢的远端内存中。距离估计逐步进行，从而可以提前剪除不理想的候选项，避免对大多数向量进行完整重构。

C. 相关工作

近期研究利用 CXL [8], 持久内存 [10], 、近数据处理（NDP） [22]–[25], 和图形处理器 [12]–[14]等高带宽内存系统，降低近似最近邻搜索的内存开销。这些方法要么加速量化代码的查找表查询，要么假设可以直接获取全精度向量，并将精化留给慢速存储读取。CXL-ANNS [8]将全精度向量存储在 CXL 内存中，并将距离计算卸载到 FPGA，但当原始数据集无法装入内存容量时，其可扩展性便会崩溃。HM-ANN [10] 将分层可导航小世界（HNSW） [26] 索引的入口层映射到快速 DRAM–NVM层级，但对于高维向量，增加层数几乎不能带来额外的剪枝收益，反而会增加管理开销 [27]。近数据处理设计 [22]–[25] 将量化向量的距离计算直接集成到内存设备中，但没有解决精化问题。近数据处理加速与我们的工作互为补充。我们聚焦于渐进式、层级感知的精化，以尽量减少固态硬盘访问，并能自然地受益于或结合快速内存和远端内存中的近数据处理加速。

III. FATRQ：渐进式距离估计

为减少二次精炼带来的高昂输入/输出开销和计算开销，我们提出 FaTRQ，这是一种能够

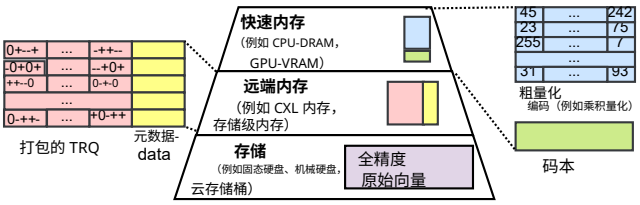

图 3. FaTRQ框架的分层内存布局

逐步提升粗略距离估计精度的分层感知框架。如图3所示，粗粒度PQ编码和码本保留在快速内存中，而紧凑的残差编码和元数据则存储在远端内存中，从而减少高延迟的固态硬盘访问。该精化方法基于将 L2 距离分解为粗略项和残差项，无需重建向量即可逐步估计这些项。残差被编码为紧凑的三元编码，既无需乘法运算，又能高效地存入远端内存。这种紧凑格式支持渐进式细化：首先进行残差距离估计，其次采用无乘法编码，最后通过整合所有组成部分的增强型估计器，准确计算 L2 距离。通过这些步骤，FaTRQ 能够及早剪枝不具潜力的候选项，并减少固态硬盘流量。

*A. L2*距离分解

与以往使用专为精化而构建的独立量化编码进行重排序的系统不同，FaTRQ 会逐步复用早期阶段已计算出的粗略近似。这种复用避免丢弃粗编码中的有用信息，并使精化过程能够通过 L2 距离分解清晰地表示：

$$\|x-q\|_2^2 = \|x_c-q\|_2^2 + \|x_c-x\|_2^2 - 2\langle q-x_c, x-x_c\rangle,$$

其中，q 是查询，x_c 是原始向量 x 经粗编码重构得到的向量，而 $\langle\cdot,\ \cdot\rangle$ 是内积。该距离分解为三项：粗略近似、压缩失真和残差内积。

图 4 展示了 Wiki 数据集 [28]上的这一分解。对于每个记录向量，我们将其粗量化重构结果 x_c 平移至原点（圆心），并将残差 $x-x_c$缩放至单位圆上。查询偏移量 $q-x_c$沿 x 轴放置，其长度与 $\frac{\|q-x_c\|}{\|x-x_c\|}$成正比。在这种表示中，原始记录 x（蓝色点）位于单位圆上，而查询 q （红色点）位于 x 轴上。这表明残差几乎与查询偏移量正交，因此两者的内积很小。

这引出了如下一阶近似：

$$\hat{d}_1(q,x) = \|x_c-q\|_2^2 + \|x_c-x\|_2^2 = \hat{d}_0(q,x) + \|x_c-x\|_2^2,$$

其中 $\hat{}\ d_0(q,\ x) = \|x_c-q\|_2^2$。请注意，$\|x_c-x\|_2^2$ 是与每个记录向量关联的标量值，可预先离线计算。为了完全还原真实距离，我们必须估计残差项 $\langle q-x_c,\ x-x_c\rangle$。

令 $\delta = x-x_c$ 表示记录与其粗略近似之间的残差，则有

$$\|x-q\|_2^2 = \|q-x_c\|_2^2 + \|\delta\|_2^2 + 2\langle x_c,\delta\rangle - 2\langle q,\delta\rangle.$$

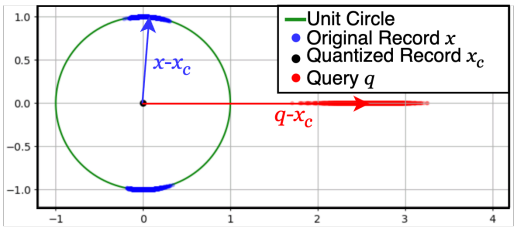

图 4. 残差向量与查询向量的可视化

前三项仅使用粗量化编码和预计算标量即可计算。最后一项，$\langle q,\ \delta\rangle$，通过量化残差进行估计，无需解码基向量。与存储完整的原始向量相比，残差占用空间更小，因为基编码已经捕获了向量结构的大部分信息。将残差存储在远端内存中，可以更好地平衡延迟与容量。我们将这种使用量化残差向量进行的精化称为二阶近似。由于残差量化天然可层叠应用，因此可以逐步精化距离估计。例如，我们可以先在粗编码的基础上对残差进行编码，然后再对剩余误差编码更细粒度的残差，以逐步获得更精确的距离估计。在不失一般性的前提下，以下讨论将聚焦于二阶估计。下一节将详细说明如何量化 δ，并利用它得到内积项的无偏估计。

B. FaTRQ 残差距离估计

为估计内积 $\langle q,\ \delta\rangle$，我们首先将残差向量 δ 量化为码字 δ_c，并得到

$$\langle q,\delta\rangle = \|q\|_2\ \|\delta\|_2\ \langle e_q, e_\delta\rangle,$$

其中 $e_q = q/\|q\|_2$和 $e_\delta = \delta/\|\delta\|_2$分别是查询 q 和残差 δ的归一化方向。我们的目标是获得方向项 $\langle e_q,\ e_\delta\rangle$的无偏估计器。

注意，通过加上再减去 e_q 在 e_{δ_c}（即 δ_c 的方向）上的投影，我们可以写成

$$\begin{aligned}\langle e_q,e_\delta\rangle &= \langle e_q,e_\delta\rangle + \langle e_q,e_{\delta_c}\rangle\langle e_\delta,e_{\delta_c}\rangle - \langle e_q,e_{\delta_c}\rangle\langle e_\delta,e_{\delta_c}\rangle \\ &= \langle e_q,e_{\delta_c}\rangle\langle e_{\delta_c},e_\delta\rangle + \langle e_q - \langle e_q,e_{\delta_c}\rangle e_{\delta_c}, e_\delta\rangle \\ &= \langle e_q,e_{\delta_c}\rangle\langle e_{\delta_c},e_\delta\rangle + \|e_q - \langle e_q,e_{\delta_c}\rangle\|_2\ \langle e_\perp, e_\delta\rangle. \quad (1)\end{aligned}$$

该方程将 e_q 分解为沿 e_{δ_c} 方向和平行于 e_{δ_c} 的正交分量。$e_\perp$ 是位于由 e_{δ_c} 和 e_q 张成的平面内、与 e_{δ_c} 正交的单位向量。

这可以视为先前工作中查询解耦的对偶形式：该工作将查询分解为原始方向 e_δ，而不是我们工作中的量化方向 e_{δ_c}，以及相应的正交基。该工作表明，在温和条件下，正交项（公式 1 的第二项）高度集中且其期望为零；具体来说，残差方向均匀分布且与查询不相关。在我们的情形中，由于粗量化已经捕获了大部分向量相似度，残差向量 δ 近似各向同性，且与查询不相关。因此，无需额外的随机投影步骤，我们可以充分利用编码在

每个残差码中的信息。因此，我们估算第一项，并将第二项视为期望为零的误差项，从而得到一个无偏的残差距离估算器。

C. FaTRQ 无乘法编码

为实现对 $\langle q,\ \delta\rangle$ 的高效估计，FaTRQ 使用稀疏三元向量对残差方向进行编码。这种紧凑表示可避免精化过程中的乘法运算，并降低内存占用。具体而言，给定残差向量 $e_\delta \in \mathbb{R}^D$，我们在码本$\mathcal{C} = \{-1, 0, 1\}^D$中寻找代码 $\bar{e}_{\delta_c}$，使其归一化版本

$$e_{\delta_c} = \frac{\bar{e}_{\delta_c}}{\|\bar{e}_{\delta_c}\|_2}$$

最小化 $\|e_\delta - e_{\delta_c}\|_2$。观察到

$$\left\|e_\delta - e_{\delta_c}\right\|_2^2 = 2 - 2\,\langle e_{\delta_c}, e_\delta\rangle = 2 - 2\,\langle \tfrac{\bar{e}_{\delta_c}}{\|\bar{e}_{\delta_c}\|_2}, e_\delta\rangle,$$

我们可以等价地选择

$$\bar{e}_{\delta_c} = \arg\max_{c\in\mathcal{C}} \langle \frac{c}{\|c\|_2}, e_\delta\rangle = \arg\max_{c\in\mathcal{C}} \frac{\sum_{i=1}^D c_i\, e_{\delta,i}}{\|c\|_2}.$$

直接枚举所有 3^D 个候选项不可行。我们通过观察到任何最优的 c 都必须恰好选取 k 个非零项（每项取 $e_{\delta,\ i}$ 的符号），且这些项来自 e_δ 中绝对值最大的 k 个元素，从而求解这一优化问题。若用较小绝对值的$e_{\delta,\ i}$ 替代较大的元素，会严格降低内积的分子；而改变非零项总数 k 只会影响分母$\sqrt{k}$。因此，对于给定的 k，最佳选择是取排序列表中的前 k 个元素。具体而言，将绝对值排序为 $x_1 \geq x_2 \geq \cdots \geq x_D,$ ，其中 $x_i = \left|e_{\delta,\ (i)}\right|$，即可将内积项化简为

$$\langle \frac{c}{\|c\|_2}, x\rangle = \frac{\sum_{i=1}^k x_i}{\sqrt{k}} \quad \text{whenever } \sum_i c_i^2 = k.$$

因此，对 $\mathcal{C}$ 的全局最大化可归结为一维搜索

$$\max_{c\in\mathcal{C}} \langle \frac{c}{\|c\|_2}, x\rangle \ = \ \max_{1\leq k\leq D} \frac{\sum_{i=1}^k x_i}{\sqrt{k}}.$$

所提出的方法只需 $O(D\log D)$ 时间对 x_i 个值进行排序，需 $O(D)$ 时间构建 x_i 的前缀和，并再用一次 $O(D)$ 遍历来计算每个比值 $S_k/\sqrt{k}$ 并确定最大化点 k^*。已知 k^* 后，我们通过将 $\mathrm{sign}(e_{\delta,(i)})$ 赋给排名靠前的 k^* 个条目、其余位置赋零来构造 $\bar{e}_{\delta_c}$，然后用 $\sqrt{k^*}$ 进行归一化。该过程无需枚举整个码本 $\mathcal{C}$，即可在 $O(D\log D)$ 时间内得到精确的最优三元码字。

D. 紧凑 FaTRQ 代码

图3展示了 FaTRQ 在细化阶段用于将每条记录的信息存储在远端内存中的数据布局。每条记录包含两个标量值：$\langle x_c,\ \delta\rangle$ 和 $\|\delta\|_2$。它们充当缩放元数据，用于计算最终的距离估计，同时还包含一个量化残差向量 $\bar{e}_{\delta_c}$，用于记录第一级近似后的剩余误差。

残差向量 $\bar{e}_{\delta_c}$ 包含来自三级量化集合 $\{-1, 0, 1\}$ 的元素。因此，每个元素需要超过 1 位进行编码，但没有必要也不应浪费地为每个值使用完整的 2 位，因为这样会编码 4 种状态，而实际只需要 3 种。均匀分布的 3 级变量的熵为 $\log_2 3 \approx 1.58$ 位，这构成了无损压缩的理论极限。

为在实践中接近这一界限，FaTRQ 使用固定长度的紧凑编码，将残差向量每 5 个维度打包为一个字节。具体而言，首先通过变换 x_i+1 将每个值 $x_i \in \{-1, 0, 1\}$ 映射到集合 $\{0, 1, 2\}$，然后使用下方公式将所得的 5 位三进制数编码为一个字节。这种编码使每个维度的存储开销为 1.6 比特。

$$y = \sum_{i=0}^{4} 3^i (x_i + 1).$$

E. FaTRQ 增强细化距离估计器

引入距离函数分解和残差编码后，最后一步是将这些组件整合到细化估计器中。具体而言，L2 距离可以表示为粗略近似 $\hat{}\ d_0$、预计算标量 $\|\delta\|_2^2$ 和 $2\langle x_c,\ \delta\rangle$以及估计的残差项 $-2\langle q,\ \delta\rangle$的组合。

评估这些估计的有效性时，会发现一个重要结论：距离近似的均方误差 (MSE) 较低，并不总能带来更高的近似最近邻搜索召回率。这是因为召回率主要取决于候选项在顶部-k 决策边界附近的排序准确度。对于非常近或非常远的点，即使估计误差很大，影响也很小，因为这些误差不会影响关键区域内的排序。因此，提高整体距离准确度并不能保证召回率更高；真正重要的是决策边界附近的局部精度。

基于这一观察，FaTRQ 通过对少量样本—邻居对组成的校准集进行子采样，离线学习一个轻量级线性校准模型，直接优化边界区域的召回率。具体而言，它会随机抽取少量数据库向量（实际约占 0.3%，我们发现这足以使模型达到饱和）。对于每个抽取的向量 x，它利用现有索引识别近似邻居，无需进行代价高昂的精确 k近邻搜索：如果索引基于图（例如 CAGRA [13] 或分层可导航小世界 [26]），则使用 x的图邻接顶点；如果索引基于 IVF，则使用同一倒排列表中的向量。这些点自然聚集在原始空间中 x 附近，并对其局部决策边界提供密集覆盖。

对于每个样本—邻居对，我们构造特征向量：

$$A = \left[\ \hat{d}_0,\ \hat{d}_{ip},\ \|\delta\|_2^2,\ \langle x_c, \delta\rangle\ \right],$$

其中，$\hat{}\ d_0 = \|q - x_c\|_2^2$ 是粗略近似，$\hat{}\ d_{ip}$ 是通过三元编码得到的 $-2\langle q,\ \delta\rangle$ 精化估计，$\|\delta\|_2^2$ 是残差范数，$\langle x_c,\ \delta\rangle$ 是预先计算的交叉项。令 D表示真实距离。校准模型在离线阶段通过求解以下问题进行学习

$$\hat{W} \ = \ \arg\min_W \left\|D - A\,W\right\|_2^2,$$

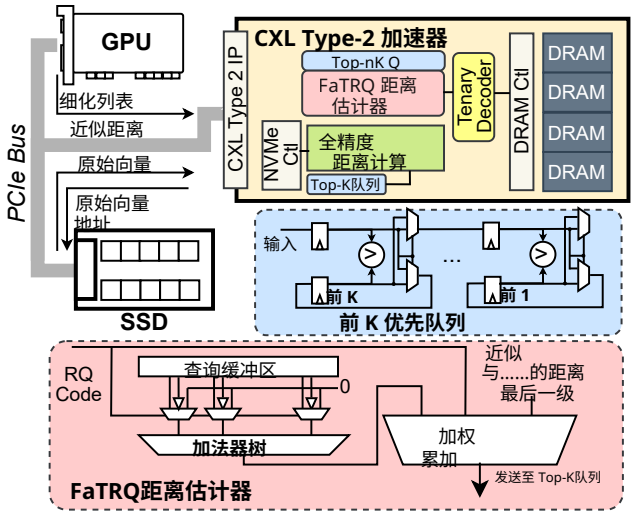

图 5. FaTRQ 增强型 ANNS 流水线的系统架构。

通过普通最小二乘法。在查询时，精化距离ˆ的估计可简化为对 AW进行轻量级计算。

IV. FaTRQ 增强型 ANNS 系统

虽然 FaTRQ 面向广泛的分层内存系统，但我们也探索了其在现代远端内存设备上的优势。CXL Type-2 模块兼具远端内存（例如 SCM）的大容量和轻量级计算能力。我们的原型将定制的精炼逻辑集成到此类设备中，将精炼操作直接推送到远端内存，以减少主机与设备间的数据传输。鉴于计算型大容量内存设备已经实现商业化 [30]，，这表明 FaTRQ 能够利用新兴的硬件模型，而我们的软件实现则表明它并不依赖专用的远端内存硬件。

如图 5 所示，前级 GPU 使用粗量化码处理索引，以估算近似距离并生成较大的候选列表。无需传输数百字节的粗编码，只需将每个候选项的 4 字节粗距离值发送到配备 FaTRQ 精炼逻辑的远端内存设备。不同于以往的方法 [19]，[20]从粗量化码和残差编码中重构记录向量，FaTRQ 直接对紧凑的残差编码进行精化，从而避免重构。远端内存加速器利用其设备上的距离估算器对候选项重新排序，并根据更新后的分数截减列表。随后，仅从固态硬盘中获取排名靠前的结果进行最终计算，使精炼成本降低 85%（V-B 节）。

图 5 还展示了详细的实现方式。系统使用两个硬件优先队列，通过寄存器和比较器实现，用于在细化过程中追踪前 K 个最近邻。一个队列维护由 FaTRQ 残差量化路径估计的距离排序的前 K 个候选项，另一个队列则根据全精度向量距离进行最终排序。队列中的每个条目都存储一个距离值和指向相应向量的指针。新候选项通过将其距离与队列中的距离进行比较后插入，并在比较器流水线中将较小的值逐步向前移动。每个队列最多支持 1024 个条目。一个包含 256 个条目的

表I FaTRQ评估参数

参数	值
DRAM配置	8Gb x16 DDR5-4800
时序（tRCD-tCAS-tRP）	34-34-34
通道 / 每通道存储体组数	8 / 8
SSD延迟 / 吞吐量 [31]	45 μ 秒 / 1200K IOPS
CXL延迟 / 吞吐量 [30]	271 纳秒：22 GB/秒

查找表实现了三进制解码器，用于将残差编码解包为三进制表示。由于 FaTRQ 中的残差编码采用三进制表示，因此查询与残差向量之间的内积 $\langle e_q,\ e_{\delta_c}\rangle$可以通过简单的加法器和多路复用器计算。加权累加单元以 MAC 阵列实现，用于合并这些结果并生成最终的估计距离，如第 III-E 节所述。

V. 评估

A. 实验设置

软件与基线： 我们将 FaTRQ 实现为 cuVS [9] 和 FAISS [3]库的扩展。我们的设计将所提出的残差编码置于作为粗量化器的乘积量化（PQ）之上，并将其与 IVF [3]和 CAGRA [13] 索引结构相结合，构成完整的近似最近邻搜索系统。在评估中，我们使用用于 IVF 的 FAISS GPU版本，以及 cuVS 中面向图形处理器的图索引 CAGRA。所有参数均通过网格搜索 [13]进行调优。我们将这些经过 FaTRQ 增强的系统与 cuVS [13], [17]提供的基线流程进行比较。

数据集： 我们使用两个预嵌入的近似最近邻搜索数据集。a）*Wiki* 数据集 [28]（251 GB）包含 8,800 万个 SBERT [7] 多语言句子嵌入（768 维）。b）*LAION* 数据集 [32],是 VDBBench [33], 中最大规模的预嵌入数据集合，包含 1 亿个源自 LAION- 5B [32], 的 CLIP [16] 嵌入，总计 286 GB。两个数据集均使用欧几里得距离作为相似度度量，并包含 1 万个查询。

平台： 我们扩展了 Ramulator [34]，以模拟 CXL Type-2 设备的远端内存访问模式，DRAM 参数列于表 I 中。我们使用 ASAP7 [35]，以 1 GHz 的频率通过 Verilog 综合我们的加速器，从而评估硬件开销。片上静态随机存取存储器使用 FinCACTI [36]进行建模。所有 GPU端索引遍历和粗距离计算均在 NVIDIA A10 GPU（24 GB 显存）上运行，而基线系统中的精化操作则在配备 128 GB DRAM 和 1 TB 固态硬盘的 40 线程 I ntel Xeon Gold 6230 CPU 上执行。

B. 总体性能

图 6 比较了 FaTRQ 与 SoTAGPU 流水线、IVF-FAISS [3] 和 CAGRA-cuVS [9]，在前 10 查询的 85% 、 90% 和 95% 召回率下的端到端吞吐量。在 LAION 数据集上，当乘积量化编码适合存放在图形处理器显存中时，召回率会在 94% 处饱和，因此省略了 LAION-95 的结果。基线方法将量化向量存储在图形处理器内存中，并通过从固态硬盘获取全精度向量，在 CPU 上执行精化。我们评估了将残差编码存储在远端内存中的 FaTRQ：在软件模式（-SW）下，编码存储于 CXL 内存中，但

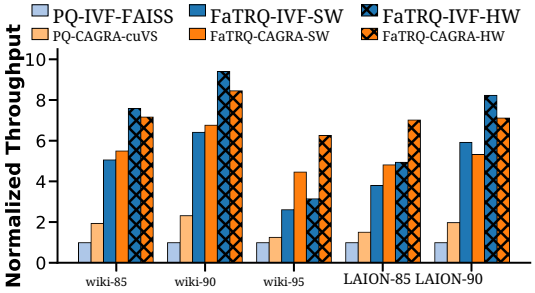

图 6. 在 IVF 和 CAGRA 前置索引上，针对不同前 10 名查询召回率的 FaTRQ 归一化吞吐量评估。

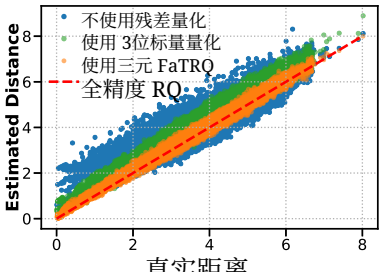

图 7. 相对于前 100 名真实结果的平方欧几里得距离估计误差。

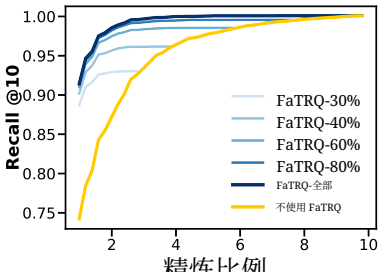

图 8. FaTRQ 在过滤不同比例候选项时，Recall@10 与精炼比例的关系。

筛选在 CPU 上执行；而在硬件模式（- HW）下，筛选被卸载到 CXL Type-2 加速器。

FaTRQ-HW 相较于 IVF-FAISS 可实现 3.1×–9.4× 倍加速，相较于 CAGRA-cuVS 可实现 2.6×–4.9× 倍加速。由于 IVF 需要进行更多精化，因此从中获益更大。例如，在 Wiki 上达到 90% 召回率时，IVF 每个查询需要精化 320 个候选项，而 CAGRA 为 120 个。在基线方案中，这意味着分别需要从固态硬盘读取 320 次和 120 次。采用 FaTRQ 后，IVF 需要 28 次固态硬盘读取和 320 次 CXL 访问，CAGRA 则需要 17 次固态硬盘读取和 120 次 CXL 访问，从而将大部分精化流量从固态硬盘转移出去，使 IVF 获得更大的性能提升。召回率达到 95% 时，加速幅度有所缩小，因为实现高精度所需的更深层遍历和粗粒度过滤开始成为运行时间的主要开销。我们还发现，硬件版本相较于软件路径可将吞吐量提升 1.2×–1.5×，候选项过滤速度最高可提升 3.7× 倍，这得益于直接访问远端内存以及消除了主机数据移动。

C. 距离估计失真评估

如第 III-E 节所述，近似最近邻搜索的准确性取决于查询与集合中距离查询较近的一小部分数据之间的距离估计。为评估 FaTRQ 引入的距离失真，我们通过穷举搜索计算每个查询与前 100 名真实结果之间的距离。图 7 比较了 Wiki 数据集上的三种情况的距离失真：INT8 量化（不使用 RQ）、采用标量量化（SQ）残差码的 PQ [12], , 以及采用 FaTRQ 残差码的 PQ。SQ 和 FaTRQ 均重复使用先前实验中的同一 PQ 码本，以确保不同方法之间的一致性。

与基于 SQ 的残差不同，后者会重建记录向量；FaTRQ 无需传输上层代码，而是通过增强距离估计直接细化距离。如图所示，与 3 位标量量化相比，FaTRQ 的结果更接近预言机线（全精度残差向量），均方误差为 0.0159，而非 0.258。

此外，FaTRQ 具有更高的存储效率。对于一个 768 维向量，FaTRQ 仅需 $768/5+8=162$ 字节（将五个三元值打包为一个字节，另加四个字节用于存储预计算值）；而要达到相近的均方误差（0.0134），4 位 SQ 则需要 $768\times 4/8=384$ 字节。

D. 精炼减少分析

我们研究了过滤比例对候选项重排序的影响。对于每个查询，我们根据乘积量化距离收集了真实的前 100 个结果，并考察了重排序行为。图 8 展示了召回率与精炼比例之间的关系，精炼比例定义为按最终前-k = 10归一化的固态硬盘输入/输出。每条蓝色曲线对应一种过滤比例：只有 FaTRQ 排序队列中排名前-X%的项会访问全精度向量；黄色曲线则表示基线方案，即整个列表都需要进行固态硬盘输入/输出。没有 FaTRQ 时，要以 99% 的概率找回真实的前 10 个结果，需要从 100 个候选项中最多扫描 70 个全精度向量。使用 FaTRQ 后，只需扫描 25 个即可达到同样的保证，使精炼量减少了 2.8×。

E. 开销分析

集成在 CXL Type-2 设备中的定制处理单元仅增加 0.729 mm^2 的面积和 897 mW 的功耗。FaTRQ 距离估算器占面积的 29% 和功耗的 27%，而优先队列分别增加 6% 和 8%。与配备 16 个 ARM Neoverse V2 核心的 aCXL 内存控制器 [37]（每个核心为 2.5 mm^2, 1.4 W [38]）相比，其开销不到面积的 1.8% 和功耗的 4%，表明 FaTRQ 为整个内存扩展器带来的额外负担很轻。

FaTRQ 的离线开销也很小。训练轻量级校准模型并构建残差编码，每个向量只需进行一次并行扫描，总共约需 10 分钟；相比之下，在我们的实验中，为任一数据集构建 CAGRA 索引约需 3 小时。

VI. 结论

我们提出了 FaTRQ，这是一种分层感知精化层，可将残差压缩为三元编码，并在无需重建完整向量的情况下逐步精化粗略距离。FaTRQ 针对现代近似最近邻搜索系统中的一个关键瓶颈：二次精炼需要从慢速存储中读取全精度向量。随着嵌入维度和数据集规模持续增长，这种输入/输出开销日益成为查询延迟的主要因素。通过从远端内存流式读取紧凑残差并即时更新距离，FaTRQ 能够提前剪枝候选项、减少不必要的数据读取，与 GPU [13] 相比，吞吐量最高可提升 9×。

致谢

本工作部分由 PRISM 和 Co-CoSys 资助；它们是 JUMP 2.0 中心，JUMP 2.0 是由美国国防高级研究计划局资助的 SRC 项目；此外还获得了太平洋西北国家实验室的美国能源部 DeCoDe 项目（编号 84245）以及美国国家科学基金会的资助，项目编号为 2112665、2112167、2052809 和 2211386。

参考文献

[1] [在线]。网址：https://platform.openai.com/docs/guides/embeddings [2] J. Wang、X. Yi、R. Guo、H. Jin、P. Xu、S. Li、X. Wang、X. Guo、C. Li、X. Xu 等人，“Milvus：专为向量数据管理打造的系统”，载于《2021年国际数据管理会议论文集》，2021年，第 2614–2627 页。[3] M. Douze、A. Guzhva、C. Deng、J. Johnson、G. Szilvasy、P.-E. Mazaré、M. Lomeli、L. Hosseini 和 H. Jégou，“Faiss 库”，2024年。[4] T. Ge、K. He、Q. Ke 和 J. Sun，“优化乘积量化”，《IEEE模式分析与机器智能汇刊》，第 36 卷，第 4 期，第 744–755 页，2013年。[5] R. Wang 和 D. Deng，“Deltapq：面向高维相似性搜索的无损乘积量化编码压缩”，《VLDB基金会会议录》，第 13 卷，第 13 期，第 3603–3616 页，2020年。[6] Y. Kalantidis 和 Y. Avrithis，“面向近似最近邻搜索的局部优化乘积量化”，载于《IEEE计算机视觉与模式识别会议论文集》，2014年，第 2321–2328 页。[7] B. Wang 和 C.-C. J. Kuo，“Sbert-wk：一种通过剖析基于BERT的词模型实现的句子嵌入方法”，《IEEE/ACM音频、语音与语言处理汇刊》，第 28 卷，第 2146–2157 页，2020年。[8] J. Jang、H. Choi、H. Bae、S. Lee、M. Kwon 和 M. Jung，“CXL-ANNS：面向十亿级近似最近邻搜索的软件—硬件协同内存解聚与计算”，载于《2023年USENIX年度技术会议（USENIX ATC 23）》，2023年，第 585–600 页。[9] “cuvs：在图形处理器上进行向量搜索与聚类。”[在线]。网址：https://github.com/rapidsai/cuvs [10] J. Ren、M. Zhang 和 D. Li，“HM-ANN：在异构内存上高效进行十亿点最近邻搜索”，《神经信息处理系统进展》，第 33 卷，第 10 672–10 684 页，2020年。[11] S. Jayaram Subramanya、F. Devvrit、H. V. Simhadri、R. Krishnawamy 和 R. Kadekodi，“Diskann：在单节点上快速、精确地进行十亿点最近邻搜索”，《神经信息处理系统进展》，第 32 卷，2019年。[12] V. Karthik、S. Khan、S. Singh、H. V. Simhadri 和 J. Vedurada，“Bang：使用单个GPU进行十亿级近似最近邻搜索”，arXiv预印本 arxiv:2401.11324，2024年。[13] H. Ootomo、A. Naruse、C. Nolet、R. Wang、T. Feher 和 Y. Wang，“CAGRA：面向图形处理器的高度并行图构建与近似最近邻搜索”，载于《2024年第40届IEEE国际数据工程会议》（ICDE）。IEEE，2024年，第 4236–4247 页。[14] B. Tian、H. Liu、Y. Tang、S. Xiao、Z. Duan、X. Liao、X. Zhang、J. Zhu 和 Y. Zhang，“Fusionanns：面向十亿级近似最近邻搜索的高效CPU/GPU协同处理架构”，arXiv预印本 arXiv:2409.16576，2024年。[15] B. Tian、H. Liu、Y. Tang、S. Xiao、Z. Duan、X. Liao、H. Jin、X. Zhang、J. Zhu 和 Y. Zhang，“通过 {CPU/GPU} 协同过滤与重排序，实现高吞吐量、低延迟的十亿级向量搜索”，载于第 23 届《USENIX文件与存储技术会议》（FAST 25），2025年，第 171–185 页。[16] G. Ilharco、M. Wortsman、R. Wightman、C. Gordon、N. Carlini、R. Taori、A. Dave、V. Shankar、H. Namkoong、J. Miller、H. Hajishirzi、A. Farhadi 和 L. Schmidt，“Openclip”，2021年7月；如果您使用此软件，请按以下格式引用。 [ ] 10 5281 5143773 在线。[17] [ ] 网址：https://doi.org/。/zenodo。C. Nolet，“主题演讲：使用 RAPIDS RAFT 加速图形处理器上的向量搜索”，NeurIPS 2023 大型近似最近邻挑战赛主题演讲，2023年，在线发表于 https://neurips.cc/virtual/2023/83767（访问日期：2025年9月2日）。[18] H. Jegou、M. Douze 和 C. Schmid，“用于最近邻搜索的乘积量化”，《IEEE模式分析与机器智能汇刊》，第 33 卷，第 1 期，第 117–128 页，2010年。[19] S. Liu、H. Lu 和 J. Shao，“面向高维近似最近邻搜索的改进残差向量量化”，arXiv预印本 arXiv:1509.05195，2015年。[20] J. Yuan 和 X. Liu，“面向近似最近邻搜索的变换残差量化”，arXiv预印本 arXiv:1512.06925，2015年。[21] H. Jegou、M. Douze 和 C. Schmid，“用于最近邻搜索的乘积量化”，《IEEE模式分析与机器智能汇刊》，第 33 卷，第 1 期，第 117–128 页，2010年。[22] M. Chen、T. Han、C. Liu、S. Liang、K. Yu、L. Dai、Z. Yuan、Y. Wang、L. Zhang、H. Li 等人，“Drim-ann：基于商用DRAM-PIM的近似最近邻搜索引擎”，arXiv预印本 arXiv:2410.15621，2024年。[23] T. Yu、B. Wu、K. Chen、C. Yan、G. Zhang 和 W. Liu，“Hdanns：面向十亿级近似最近邻搜索加速的内存内超维计算”，*IEEE*人工智能电路与系统汇刊，2025年。[24] Z. Zhu、J. Liu、G. Dai、S. Zeng、B. Li、H. Yang 和 Y. Wang，“面向十亿级近似最近邻搜索的层次内存处理架构”，载于*2023*年第*60*届*ACM/IEEE*设计自动化会议（*DAC*）。IEEE，2023年，第1–6页。[25] T. Zhang、F. Ponzina 和 T. Rosing，“SpANNS：利用近内存处理优化稀疏向量的近似最近邻搜索”，载于第*31*届亚洲及南太平洋设计自动化会议，2026年。[26] Y. A. Malkov 和 D. A. Yashunin，“利用分层可导航小世界图实现高效稳健的近似最近邻搜索”，*IEEE*模式分析与机器智能汇刊，第42卷，第4期，第824–836页，2018年。[27] B. Munyampirwa、V. Lakshman 和 B. Coleman，“摒弃层级：HNSW 中的‘H’代表‘枢纽’”，*arXiv*预印本 *arXiv:2412 01940*，2024年。。[28] “Cohere/Wikipedia-22-12。”[在线]。可访问：https://huggingface.co/datasets/Cohere/wikipedia-22 12 - J. Gao 和 C. Long，[29] “RaBitQ：针对近似最近邻搜索，利用理论误差界量化高维向量”，*ACM*数据管理会议录，第2卷，第3期，第1–27页，2024年。[30] “CXL近内存加速器和内存扩展控制器。”[在线]。可访问：”https://www.marvell.com/products/cxl.html” [31] “三星V-NAND SSD 990 PRO 2022年数据表。” [在线]。可访问：”https://download.semiconductor.samsung.com/resources/data-sheet/Samsung NVMe SSD 990 PRO Datasheet Rev.1.0.pdf” [32] C. Schuhmann、R. Beaumont、R. Vencu、C. Gordon、R. Wightman、M. Cherti、T. Coombes、A. Katta、C. Mullis、M. Wortsman等，“LAION-5B：用于训练新一代图文模型的开放大规模数据集”，神经信息处理系统进展，第35卷，第25 278–25 294页，2022年。[33] “VectorDBBench（VDBBench）：VectorDB基准测试工具。” [在线]。可访问：https://github.com/zilliztech/VectorDBBench [34] Y. Kim、W. Yang 和 O. Mutlu，“Ramulator：快速且可扩展的DRAM模拟器”，*IEEE*计算机体系结构快报，第15卷，49，2015年1，45第。页–。[35] L. T. Clark、V. Vashishtha、L. Shifren、A. Gujja、S. Sinha、B. Cline、C. Ramamurthy 和 G. Yeric，“ASAP7：7纳米鳍式场效应晶体管预测性工艺设计套件”，微电子学杂志，第53卷，第105–115页，2016年。[36] A. Shafaei、Y. Wang、X. Lin 和 M. Pedram，“FinCACTI：深度缩放鳍式场效应晶体管器件缓存的体系结构分析与建模”，载于*2014*年*IEEE*计算机学会超大规模集成电路年度研讨会。IEEE，2014年，第290–295页。[37] “Structera A 2504内存扩展控制器。”[在线]。可访问：”https://www.marvell.com/content/dam/marvell/en/public-collateral/assets/marvell-structera-a-2504-near-memory-accelerator-product-brief.pdf” [38] M. Bruce，“ARM Neoverse V2平台：为下一代云计算、机器学习和高性能计算工作负载提供领先的性能与能效”，载于*2023*年*IEEE Hot Chips*第*35*届研讨会（*HCS*），2023年，第1–25页。