arXiv:2602.10258v1 [cs.IR] 10 Feb 2026

JAG：面向过滤最近邻搜索的联合属性图

Haike Xu [1] **Guy Blelloch** [2 3] **Laxman Dhulipala** [4 3] **Lars Gottesbüren** [3] **Rajesh Jayaram** [3] **Jakub Ł .acki** [3]

摘要

尽管过滤最近邻搜索是现代向量搜索系统中的一项基本任务，现有算法的性能对查询选择率和过滤器类型高度敏感。具体而言，现有方案要么仅在特定过滤器类别（例如标签相等）上表现优异，要么仅在狭窄的选择率区间内有效（例如针对低选择率的预过滤），因此难以满足实际部署对泛化到新过滤器类型和未知查询选择性的要求。本文提出 JAG（联合属性图），一种基于图的算法，旨在整个选择性范围内提供稳健性能，并支持多样化的过滤器类型。我们的关键创新是引入*属性* 和 *过滤器距离*，将二元过滤约束转化为连续的导航引导。通过构建同时优化向量相似性和属性邻近性的邻近图，JAG 避免了导航死胡同，并使其能够持续优于先前的基于图的过滤近邻搜索方法。我们在五个数据集和四种过滤器类型（标签、范围、子集、布尔）上的实验结果表明，JAG 在吞吐量和召回率稳健性方面均显著优于现有最先进的基线方法[1]。

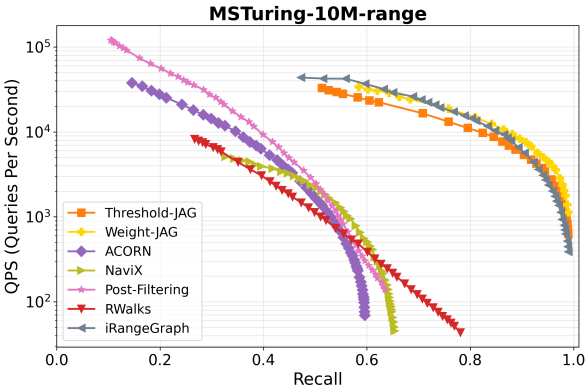


图1。MSTuring-10M 数据集上范围过滤器的每秒查询数与召回率关系图。实验详情请参阅第4 节。

1. 引言

向量搜索已成为现代数据管理系统的基本组成部分，其驱动力来自管理非结构化数据的需求日益增长，以及深度学习嵌入的广泛普及。尽管针对原始向量相似度的搜索已得到充分研究，但实际的向量搜索查询很少仅依赖相似性。相反，用户通常会发出混合查询，将高维向量相似度与结构化元数据约束相结合。例如，用户可能会查询产品目录，以查找与参考图像在视觉上相似的商品（这是使用图像嵌入的向量搜索查询），但将搜索限制在特定的价格范围或类别内。这一范式通常被称为*过滤向量搜索*，它引入了复杂的优化权衡，与标准的关系查询处理有显著差异。

定义过滤向量搜索性能格局的一个关键参数是*选择率*。我们明确将选择率定义为索引中满足过滤条件的项所占的比例。因此，*高选择率*查询意味着数据集中有很大一部分满足过滤条件，而*低选择率* 查询则意味着只有一小部分项是有效候选项。在选择性范围的两端，现有技术都能提供稳健的解决方案。在高选择率范围（大多数项都有效）中，*后过滤*通常效果很好。这种方法使用标准的非过滤向量搜索解决方案来检索一组近邻，随后丢弃不满足过滤条件的项。由于过滤条件较宽松，在前-k 个结果中找到足够多有效近邻的概率仍然很高。相反，在低选择率范围（只有极少数项有效）中，*预过滤* 是首选策略。在这种情况下，系统会先识别数据中的有效子集——现代数据库对此类任务进行了大量优化——然后在缩减后的候选集上执行精确搜索。

[1]麻省理工学院 [2]卡内基梅隆大学 [3]Google Research [4]马里兰大学。通信请联系：Haike Xu <haikexu@mit.edu>、Rajesh Jayaram <rkja-yaram@google.com>。

预印本。*2026*年*2*月*12*日。[1]我们的代码可在此处获取：https://github.com/xuhaike/JAG

然而，在这两个极端之间的宽广选择率范围内，存在一个显著的挑战。在这一过渡区域，有效子集太小，导致后过滤效率低下（需要扫描向量索引中数量过多的项，才能找到 k 个有效结果），但又太大，导致预过滤性能不佳（因为“缩减后”的集合中可能仍有过多项，无法进行暴力计算）。解决这一缺口是数据库社区当前的一个研究热点。人们提出了多种解决方案，从专用图遍历（Patel 等，2024；Gollapudi 等人，2023；Ait Aomar 等人，2025；Wang 等人，2022；Li 等，2025）到混合索引结构（Gupta etal.,2023；Mohoney 等，2023；Zuo 等，2024）。这些现有方法往往呈现出不同的性能特征：有些专门针对特定过滤器类型进行了优化（例如标签相等（Gollapudi 等人，2023）或范围搜索（Zuo 等，2024；Engels 等，2024）），另一些则主要在特定的选择率子范围内表现出色。

为阐明现有方法的贡献并定位我们的工作，我们根据混合搜索算法使用元数据的方式，将其分为三个不同类别：

(1) **完全无感知（过滤器无关）：** 这些方法（例如，采用后过滤的标准 HNSW、ACORN（Patel 等人，2024年）、NaviX（Sehgal 和 Salihoglu，2025年）仅基于向量数据构建索引，在构建过程中忽略属性。它们具有普遍适用性，但当过滤条件较严格时，性能会下降，因为索引无法引导搜索找到符合条件的项。

(2) **过滤器感知：** 这些算法针对事先已知的特定查询谓词类别专门优化索引结构，例如范围查询（Zuo 等人，2024年；Xu 等人，2024年）或标签相等、基于析取的过滤条件（Gollapudi 等人，2023年；Landrum 等人，2025年；Cai 等人，2024年）。虽然它们在目标查询类型上表现良好，但无法稳健地处理其他类型的查询：针对范围优化的索引无法高效处理布尔查询或子集查询，因此每种过滤器类型都需要使用不同的索引。

(3) **属性感知（也称过滤条件无关）：** 这些方法（Wang 等人，2022年；Ait Aomar 等人，2025年）在构建索引时利用属性数据分布来组织索引，但不会针对特定的过滤逻辑硬编码索引结构。不过，现有方法仍主要针对二元匹配/不匹配过滤器（例如标签相等）设计，通用性不足，无法支持任意过滤器类型。相比之下，我们的目标是在单一统一索引结构中支持多种过滤器类型，包括但不限于标签、范围、子集和布尔过滤器。

专用的过滤器感知 算法在其针对的特定任务上自然会优于通用方法。然而，现代工作负载具有动态性和多样性，为每种可能的过滤器类型维护单独的索引通常会带来难以承受的计算开销。因此，设计与过滤器无关的算法具有很强的吸引力，这类算法能够在广泛的过滤器类型和选择性范围内保持竞争力。

本文提出了一种新的过滤向量搜索算法，旨在适用于各种选择率，并支持多种过滤器类型。因此，我们的算法不依赖特定过滤条件，其设计动机是：给定的索引配置很少能适用于所有选择率。为此，我们提出了一种方法，将针对不同选择率调优的多个索引结构有效地组合到一个紧凑索引中。具体而言，该算法构建一种联合属性图，简称***JAG***。它是一种单一基于图的索引，整合了对应不同查询选择率的多个导航层。为了使这种结构能够灵活支持广泛的过滤器类型——包括类别、范围、子集和布尔过滤器——我们对属性距离和过滤距离的概念进行了形式化并加以应用。

“属性距离 ”($dist_A$；构建时使用)用于衡量两个数据点的属性之间的语义邻近度（即用于确定数据点是否满足过滤条件的元数据），且不依赖任何特定查询。通过使用向量距离（两个向量之间的距离）和属性距离的联合 度量来构建 JAG，我们可以确保即使在向量空间稀疏但属性相似的区域中，图也保持连通。关键在于，构建 JAG 时我们采用“封顶”的属性距离机制。通过对 $dist_A$ 应用不同的阈值，我们生成层次结构 的导航边：一些连接严格的属性邻居（对于低选择率至关重要），另一些则连接更广泛的属性邻域（足以应对高选择率）。JAG 中这种统一索引结构使搜索算法能够根据查询的选择率，无缝调整其遍历策略。

该过滤距离（$dist_F$；仅在查询时、已知过滤条件后使用）可作为二元过滤约束的代理，用于量化候选项的元数据距离满足查询要求还有多远。即使邻居并不严格满足过滤条件，这也能提供导航代理，避免查询在 JAG 中陷入“死胡同”。请参阅图2 中的示例，了解这一代理的工作方式。

我们使用 JAG 得到的实验结果表明，我们的想法相比以往最先进的过滤向量搜索算法有显著提升。例如，图1展示了在带范围过滤器的 MSTuring-10M 数据集上的结果。所有过滤器无关算法在每秒查询数低于 50 时，召回率最高只能达到 0.8；而我们的 JAG 在召回率为 0.8 时，每秒查询数可达到> 10,000，并且仍能达到完美召回率，同时每秒查询数为

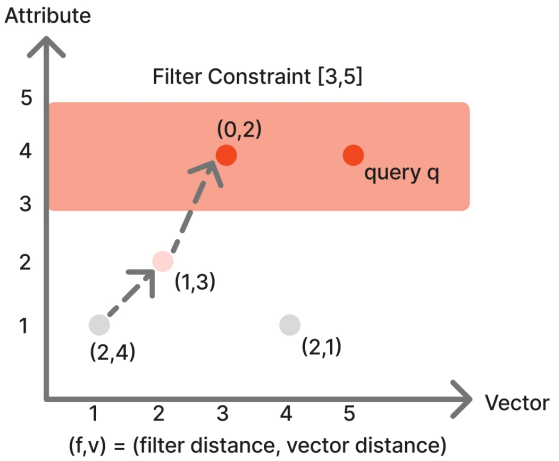


图2。此示例展示了 JAG 如何利用过滤距离和向量距离来执行范围查询。在图中，向量值（x 轴）和属性值（y 轴）均为一维实数。查询指定了一个过滤范围 [3,5]。虚线箭头展示了过滤距离和向量距离如何引导贪心搜索朝范围查询的目标前进。红色越深表示过滤距离有所改善（即减小）。

每秒查询数 >500。此外，JAG 的性能与 iRangeGraph 相当。iRangeGraph 是专为范围过滤器设计的算法。

我们的具体贡献如下：

(1) 我们提出了一种统一的基于图的索引，通过整合针对不同过滤严格程度优化的边，使其在整个选择性范围内都能保持稳健的性能。

(2) 我们定义了通用的属性距离和过滤距离，使索引能够在单一框架内支持广泛的约束，包括标签、范围、子集和布尔逻辑。

(3) 我们在五个数据集上进行了广泛的实验评估，证明了该方法的优越性能。与最先进的基线方法相比，在所有过滤器无关算法中，我们的方法在查询吞吐量和召回率方面始终表现最佳。尤为重要的是，在具有混合查询工作负载的标准且具有挑战性的数据集（例如 LAION-25M 和 YFCC）上，与表现第二好的过滤器无关算法相比，JAG 在召回率为 0.9 时的每秒查询数最高可提升 10×。

2. 预备知识

近邻搜索。 我们将近邻搜索 问题定义如下。设 $X = \{x_1, \ldots, x_n\}$ 为由 n 个 $x \in \mathbb{R}^d$ 点组成的集合，其中每个 $x \in \mathbb{R}^d$ 都是一个 d维向量。给定查询 $x_q \in \mathbb{R}^d$，近邻搜索的目标是设计一种数据结构，以高效地找到 $\arg\min_{x_p \in X} \text{dist}(x_p, x_q)$，其中 $\text{dist}(\cdot, \cdot)$ 表示距离（或相似性）函数。返回候选项的质量通常通过近似比（从理论角度）或召回率 （从经验角度）进行评估。

带过滤条件的近邻搜索。 在带过滤条件的场景中，每个点 $p = (x_p, a_p)$ 都带有一个属性 $a_p \in \mathcal{A}$，每个查询 $q = (x_q, f_q)$ 都带有一个过滤条件 $f_q \in \mathcal{F}$。搜索必须满足以下约束：点的属性符合过滤条件，即 $g(a_p, f_q) = 1$，其中g： $\mathcal{A} \times \mathcal{F} \to \{0, 1\}$ 是一个二元匹配函数。形式上，目标是求 $\arg\min_{p \in P:\, g(a_p, f_q)=1} \text{dist}(x_p, x_q)$ 过滤约束。

理想情况下，过滤约束可以是将属性和过滤条件组成的配对映射为通过/不通过的任意函数。接下来，我们将介绍本文考虑的几种常见过滤约束形式：

(1) **Equality filter.** $\mathcal{A} = [L], \mathcal{F} = [L]$, and
$$g(a, f) = \mathbf{1}[a = f].$$
这对应于类别过滤，其中每个点恰好属于一个类别。

(2) **范围过滤器**。 $\mathcal{A} = \mathbb{R}, \mathcal{F} = \mathbb{R} \times \mathbb{R}$，以及
$$g(a, f) = \mathbf{1}[f_{\min} \le a \le f_{\max}],$$
其中 $f = (f_{\min}, f_{\max})$。这描述了数值范围过滤器。

(3) **Subset filter.** $\mathcal{A} = \mathcal{F} = \{0, 1\}^L$, and
$$g(a, f) = \mathbf{1}[f \wedge a = f],$$
其中 $f \wedge a$ 表示逐元素与。这描述了子集包含。

（4）**布尔筛选器**。 $\mathcal{A} = \{0, 1\}^L$， $\mathcal{F}$ 是一个关于 L 变量的布尔函数，并且
$$g(a, f) = \mathbf{1}[f(a) = 1].$$
这表示一般的逻辑过滤，其中过滤条件是在属性上求值的任意布尔谓词。

3. 联合属性图（JAG）

本节将详细介绍联合属性图（JAG）的设计与实现。这是一种统一索引策略，能够在不同选择率下保持稳健的性能。首先，我们将形式化定义过滤距离 和属性距离的概念，它们对于将二元过滤约束转换为连续的导航梯度至关重要。然后，我们将说明如何通过截断属性距离机制和统一比较规则，将这些度量与标准向量距离相结合，以引导图遍历。最后，我们将介绍索引

构建与搜索的具体算法，并介绍我们的主要方法 Threshold-JAG，以及加权变体 Weight-JAG。

3.1. 过滤器与属性距离。

与二元约束函数 $g: \mathcal{A} \times \mathcal{F} \to \{0,1\}$ 不同，它仅指示属性–过滤器对是否满足约束，我们定义了两个连续距离函数：

(i) $dist_F : \mathcal{A} \times \mathcal{F} \to \mathbb{R}_{\geq 0}$, and (ii) $dist_A : \mathcal{A} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$, 其中，$dist_F(f, a)$ 衡量属性 a 距离满足过滤条件 f 有多远，而 $dist_A(a_1, a_2)$ 衡量两个属性在满足某个（索引时未知的）过滤条件方面有多大差异。

定义。函数 $dist_F$ 必须满足以下两个性质：

(1) **有效性**对于任意属性 $a \in \mathcal{A}$和过滤条件 $f \in \mathcal{F}$

$$dist_F(a, f) = 0 \quad \Longleftrightarrow \quad g(a, f) = 1$$

换言之，当且仅当属性完全满足过滤条件时，过滤距离为零。

(2) **一致性**。对于任意过滤条件 $f \in \mathcal{F}$ 和属性$a_1, a_2 \in \mathcal{A}$,

$$dist_F(a_1, f) < dist_F(a_2, f)$$

这与启发式解释一致：a_1 比 a_2更接近满足 f。$dist_F(\cdot, f)$ 会形成一种属性排序，该排序与接近满足过滤条件的程度相关，尽管满足与否是离散的。

类似地，属性距离 $dist_A$ 必须满足两个类似的准则：

(1) **有效性**。$dist_A(a_1, a_2) = 0$ 当且仅当 $a_1 = a_2$。

(2) **一致性**。对于任意属性 $a_1, a_2, a_3 \in \mathcal{A}$,

$$dist_A(a_1, a_2) < dist_A(a_1, a_3)$$

这与以下启发式解释一致：a_1 和a_2 更有可能同时通过或未通过某个未知的过滤条件[2]，而非 a_1 和 a_3。$dist_A$ 会在属性对之间建立一种排序，该排序与过滤结果的相似性相关。

这些度量提供了属性与筛选条件之间细粒度的关系信息，在构建属性上的邻近图以及通过图路由查询时，都可以加以利用。

示例。对于几种常见的过滤场景，可以自然地实例化所提出的距离：

(1) **等值过滤器**。属性 / 过滤条件表示一个离散类别。令 $\mathcal{A} = \mathcal{F} = [L]$，我们定义

$$dist_F(a, f) = \begin{cases} 0, & a = f, \\ 1, & a \neq f, \end{cases} \quad dist_A(a_1, a_2) = \mathbf{1}[a_1 \neq a_2].$$

(2) **范围过滤器**。属性为标量值（例如时间戳、价格）。设 $\mathcal{A} = \mathbb{R}$ 和 $\mathcal{F} = \mathbb{R} \times \mathbb{R}$，过滤条件为 $f = (f_{\min}, f_{\max})$。我们定义

$$dist_F(a, f) = \begin{cases} f_{\min} - a, & a < f_{\min}, \\ 0, & a \in [f_{\min}, f_{\max}], \\ a - f_{\max}, & a > f_{\max}, \end{cases}$$

$$dist_A(a_1, a_2) = |a_1 - a_2|,$$

其中 $dist_A$ 用于衡量属性之间的数值邻近性。

(3) **子集过滤器**。属性用于编码集合成员资格（例如，标签、权限）。令 $\mathcal{A} = \mathcal{F} = \{0,1\}^L$。我们定义

$$dist_F(a, f) = |f \setminus a|, \qquad dist_A(a_1, a_2) = |a_1 \oplus a_2|,$$

其中，$\oplus$ 表示按位异或。这里，$dist_F$ 衡量 a 需要覆盖 f 中的元素数量，而 $dist_A$ 衡量二者不同的元素数量。

(4) **布尔过滤器**。属性用于编码变量的布尔赋值。令 $\mathcal{A} = \{0,1\}^L$，并令 $\mathcal{F}$ 表示所有布尔谓词构成的集合。我们定义

$$dist_F(a, f) = \min_{a' : f(a')=1} |a - a'|,$$

$$dist_A(a_1, a_2) = |a_1 - a_2|,$$

其中，$|\cdot|$ 表示汉明距离。这里，$dist_F(a, f)$衡量满足布尔谓词 f所需的最少比特翻转次数。

讨论。$dist_F$和 $dist_A$ 的具体选择并非唯一，且可能因应用场景而异。在最坏情况下，始终可以定义平凡距离：

$$dist_F(a, f) = \mathbf{1}[g(f, a) = 0], \quad dist_A(a_1, a_2) = \mathbf{1}[a_1 \neq a_2]$$

这保证了在任何过滤方案下，都可以纳入过滤距离和属性/过滤距离。我们的算法兼容所有此类 $dist_F$ 和 $dist_A$选择，但性能可能因属性/过滤距离的信息量而异。

3.2. 合并属性距离与标准距离

给定属性距离 $dist_A$ 和过滤距离 $dist_F$的定义，一个自然的想法是使用 $dist_A$在属性上构建邻近图，然后依据 $dist_F$进行贪心搜索。直观上，这应该能够帮助我们快速找到其属性最有可能满足查询过滤器的点。然而，仍然存在两个主要挑战：

(1) **属性相似性与过滤约束之间的不一致性**。我们将属性相似性定义为一种代理指标，用于近似两个属性同时通过未知过滤条件的可能性。

[2]这个概念略显非正式，因为如果事先不知道过滤条件，就无法严格定义确切的可能性。

然而，如果索引过度依赖属性相似性，这种近似可能会产生误导。在极端情况下，未经过滤的查询完全不依赖属性信息。使用固定的属性相似性来近似所有未知过滤条件，本质上是十分困难的。

(2) 结合属性邻近关系与标准邻近关系。将属性/过滤邻近图与数据向量上的标准邻近图整合起来并非易事。嵌入空间中彼此接近的点，其属性可能截然不同，反之亦然，因此难以统一这两种距离概念。

先前的工作，例如（Wang 等人，2022年）和（Ait Aomar 等人，2025年），提出了基于标准向量距离与过滤距离的加权组合来构建邻近图。虽然这一方法直观，但存在两项局限：（i）这两种距离的尺度往往无法直接比较，因为一种通常源自欧几里得空间（例如ℓ_2 距离），而另一种则反映离散或类别不匹配（例如关键词重叠）；（ii）最佳权重可能取决于查询选择率，而构建时无法得知该选择率。

我们的解决方案。为应对这些挑战，我们提出一种新方法，引入封顶属性距离的概念，以及用于候选项评估的统一比较规则。

截断属性距离。我们定义一种依赖阈值的属性距离变体：

$$dist_A(a_1, a_2; t) = \max(dist_A(a_1, a_2) - t, 0),$$

其中，$t \geq 0$ 是一个可调阈值。这种形式意味着，一旦两个属性足够接近（距离在 t 以内），在索引过程中就将它们视为等价。从概念上讲，这引入了一种类似“指示器”的性质：假设所有位于 t 邻域内的属性都满足“假设的”过滤条件，而距离更远的属性则根据其超出部分的属性距离按比例受到惩罚。

统一比较规则。给定一个基点 $p = (x_p, a_p)$ 和阈值 t，我们先计算候选点 $u = (x_u, a_u)$ 与 $v = (x_v, a_v)$ 相对于 p 的截断属性距离 $dist_A(a_p, a_u; t)$ 和 $dist_A(a_p, a_v; t)$，再对两者进行比较。如果两个候选项的属性距离相同，则使用它们的标准向量距离 $dist(x_p, x_u)$ 和 $dist(x_p, x_v)$ 打破平局。

在查询时，给定查询 q 和两个候选点 u和 v，我们类似地使用 $dist_F(f_q, a_u)$ 和 $dist_F(f_q, a_v)$ 对它们进行比较；如果相等，则使用它们到 x_q的标准向量距离打破平局。

该设计将离散过滤约束与连续向量距离统一到单一的基于图的框架中，使搜索能够根据不同的查询选择率平滑地进行调整。

形式化地，对于任意阈值参数 $t \geq 0$和两个点u和 v，我们将统一距离定义为一个有序对

$$D_A^t(u, v) = \big(dist_A(a_u, a_v; t),\, dist(x_u, x_v)\big),$$

类似地，对于查询 q 和点 u，我们定义

$$D_F(q, u) = \big(dist_F(f_q, a_u),\, dist(x_q, x_u)\big),$$

我们按照字典序比较 $D_A^t(u, v)$ 或 $D_F(q, u)$。

3.3. Threshold-JAG

为了支持选择率各异的筛选条件，我们使用一组阈值 T 构建邻近图，并在由此得到的统一索引上执行贪心搜索。我们将此版本的算法称为 Threshold-JAG。

我们的索引构建和查询过程依赖于三个组件：贪心搜索、插入和JointRobust-Prune。所有过程都在有向邻近图 $G = (P, E)$ 上运行，并使用融合属性距离和过滤距离的定制比较器，具体如第3.2小节所述。

贪心搜索（算法1）：为在比较器 x 下查找点 D的近邻（该点可以是查询或现有数据点），算法从固定入口点 s开始，在图上执行贪心束搜索。每次迭代中，它扩展距离最近的未探索顶点 p，计算 x 到所有出邻居 $N_{out}(p)$ 的比较器距离，并将这些邻居添加到候选列表中；候选列表由按$D(x, \cdot)$ 排序的优先队列维护。当队列中的所有前l_s 个顶点都已被探索后，搜索终止。最后，它返回 k距离最近的已探索顶点作为搜索结果。

查询（算法 2）：为回答过滤最近邻查询 (q, f)，我们调用贪心搜索，使用比较器$D_F(f)$，该比较器根据候选项的过滤器距离 $dist_F$进行优先排序（以向量距离打破平局）。

插入（算法3）：图以增量方式构建。要插入新点 p（属性为 a_p），我们执行贪心搜索，使用比较器 $D_A(t)$，针对每个阈值 t（属于预定义列表 T），并收集所有已访问顶点的并集 V。随后将候选集 V剪枝为子集 V'，$|V'| \leq R$，通过调用 JointRobustPrune。在 p 与每个 $v \in V'$ 之间建立双向边。若任一顶点 v 的度超过 R，我们再次应用 JointRobustPrune 以强制满足度约束。

JointRobustPrune（算法4）：当顶点的出度超过最大值时 R，此过程会选取一个多样化的邻居子集。令 T 为索引中维护的阈值列表，deg 为总度数预算。我们将度预算划分为 $|T|$ 个桶，并分配

算法 1 贪心搜索(G, q, k, l_s, D) 1：输入：图 $G = (P, E)$、查询点 q、束宽 l_s、比较器 D2：输出：距 q最近的前 k 个顶点、已访问集合 V3：初始化候选列表 $L \leftarrow \{s\}$ {s 为入口顶点} 4：初始化已访问集合 $V \leftarrow \emptyset$5：当 $L \setminus V \neq \emptyset$ 时执行 6：$p \leftarrow \arg\min_{v \in L \setminus V} D(q, v)$ 7：$V \leftarrow V \cup \{p\}$8：对于每个 $u \in N_{\text{out}}(p)$，执行 9：如果 $u \notin L$，则 10：$L \leftarrow L \cup \{u\}$11：结束如果 12：结束循环 13：如果 $|L| > l_s$，则 14：保留 L 中按 $D(q, \cdot)$ 排序的前 l_s 个顶点 15：结束如果 16：结束循环 17：返回 V 中按 $D(q, \cdot)$ 排序的前 k 个顶点，以及 V

算法 2 查询(G，q，l_s，k)

1：**输入：**图 $G =$（P，E），查询点 q，搜索束宽 l_s，返回邻居数量 k2：**输出：**最接近k 的前若干个顶点 q3：$[A, V] \leftarrow$（贪心搜索（G，q，k，l_s，D_F））4：**返回** A

每个阈值对应的度为 $/|T|$ 个邻居。对于每个 $t \in T$，我们按照比较器 $D_A(t)$对候选项进行排序，然后按顺序遍历排序后的列表：如果候选项 v 未被之前选中的顶点支配，则将其加入 V'，并剪枝后续顶点 v'，对于这些顶点，$dist_V(v, v') < dist_V(p, v')/\alpha$，具体依据 (Sub- ramanya et al., 2019) 中的标准鲁棒剪枝准则。该过程持续进行，直到每个桶达到其局部度数限制 deg $/|T|$。最后，我们合并所有桶，形成新的邻居列表 V'，确保$|V'| \leq$ deg。

3.4. Weight-JAG

我们还实现了 JAG 的加权变体，其中属性距离和向量距离通过不同的权重组合。对于权重 w，我们定义

$$D_A^w(u, v) = w \cdot \text{dist}_A(a_u, a_v) + \text{dist}(x_u, x_v).$$

我们使用一组这样的权重构建索引图。与 Threshold-JAG 类似，在调用插入 和 JointRobust-Prune时，我们遍历权重列表，并使用 $D_A^w(u, v)$作为比较器。在查询处理期间，我们仍然先比较 $\text{dist}_F(\cdot)$，然后比较标准向量距离。我们将这一变体称为 Weight-JAG。实验对比请参阅附录C 。

Alg法 3 插入 $(G\ p, T, l_b, R, \alpha$ **) 1：输入：图** $G = (P, E)$**，新点** p**，阈值列表** T**，构建束宽** l_b**，度界** R**，剪枝参数** α**2：输出：更新后的图** $G' = (P', E')$ **3：初始化** $V \leftarrow \emptyset$**4：对每个** $t \in T$ **执行 5：** $[A_t, V_t] \leftarrow$ **贪心搜索 (**$G, p,$ **1,** $l_b, D_A(t)$**) 6：** $V \leftarrow V \cup V_t$**7：结束循环 8：** $N_{\text{out}}(p) \leftarrow$ **JointRobustPrune(**$G, p, V, R,$ α, T**) 9：对每个** $v \in N_{\text{out}}(p)$**执行 10：** $N_{\text{out}}(v) \leftarrow N_{\text{out}}(v) \cup \{p\}$**11：如果** $|N_{\text{out}}(v)| > R$ **则 12：** $N_{\text{out}}(v) \leftarrow$ **JointRobustPrune(**$G, v, N_{\text{out}}(v)$ R, α, T**) 13：结束如果 14：结束循环 15：返回** G'

4. 实验

我们在五个数据集上将算法 JAG 与十种基线算法进行了比较。请参阅补充材料中的代码。下面总结我们的发现。

- 在所有数据集（SIFT、ARXIV、LAION、YFCC 和 MSTuring）上，JAG 的表现始终优于所有与过滤条件无关的算法（见图 1、3、5、4）。
- 对于具有低选择率查询的数据集（例如，MSTuring 上的选择率 $< 1/100$），JAG 是唯一能够实现完美召回率的过滤器无关方法。所有其他过滤器无关算法的召回率都停滞在 0.8 以下，即使吞吐量极低（每秒查询数 < 50），而 JAG 在召回率达到 0.8 时的每秒查询数为 > 1000（参见图1、5、4）。
- 随着数据规模增大，JAG 的每秒查询数表现最佳（参见图7）。
- 当查询过滤器与向量空间之间存在相关性时，JAG 的每秒查询数最高（参见图6）。

4.1. 数据集

我们在五个数据集上、采用各种过滤条件设置评估算法，包括**SIFT**、**ARXIV**、**LAION**、**YFCC**和**MSTuring**，每个数据集配有一种或多种标签、范围、子集和布尔过滤器类型。我们在表1中汇总了数据集大小和平均选择率。所有数据集的详细信息请参阅附录D.2。

4.2. 算法

我们在实验中实现了 Threshold-JAG 和 Weight-JAG。实现细节请参阅第 D.3节。我们将算法与以下方法进行比较

算法 4 JointRobustPrune(G, p, V, R, α, T)

输入： 图 $G = (P, E)$、顶点 p、候选集 V、总度数预算 R、剪枝参数 α、阈值列表 T
输出： 剪枝后的邻居列表 V'
初始化 $V' \leftarrow \emptyset$
对每个 $t \in T$ **执行**
  初始化 $V'_t \leftarrow \emptyset$
  按 $D_A(t)$ 的递增顺序对 V 排序
  对 每个 $v \in V$（按顺序）**执行**
    如果 $\forall u \in V'_t,\ \alpha \cdot \text{dist}(u, v) > \text{dist}(p, v)$ **则**
      $V'_t \leftarrow V'_t \cup \{v\}$
    结束如果
    如果 $|V'_t| \geq R/|T|$ **则**
      跳出
    结束如果
  结束循环
  $V' \leftarrow V' \cup V'_t$
结束循环
返回 V'

基线算法。

基线方法。 我们将 JAG 与一系列全面的基线算法进行比较，包括 ACORN（Patel 等人，2024年）、NaviX（Sehgal 和 Salihoglu，2025年）、RWalks（Ait Aomar 等人，2025年）、后过滤。我们还测试了其他一些仅支持特定筛选条件的基线算法。例如，我们在标签和子集筛选条件下测试了 FilteredVamana（Gollapudi 等人，2023年）、StitchedVa-mana（Gollapudi 等人，2023年）和 UNG（Cai 等人，2024年），在标签筛选条件下测试了 NHQ（Wang 等人，2022年），并在范围过滤器下测试了 iRangeGraph（Xu 等人，2024年）。我们在表 2中总结了它们的兼容性。我们在附录 D.4 中介绍了各个基线算法，并在附录 D.5中说明了参数选择。

4.3. 结果

请参阅图 3、5、4 中每秒查询数与召回率的关系图，以及附录 10、11、13 中的距离计算与召回率的关系图 D.7，以及表 3中的索引时间。对于预筛选算法，它始终能够达到完美召回率，但其每秒查询数通常过低，无法在图中显示。我们在表 1中报告了它的每秒查询数及其执行的距离计算次数。下面我们将针对不同筛选条件详细分析算法。

标签筛选器。 图 3 展示了所有算法在带标签筛选器的 SIFT1M 和 ARXIV 数据集上的性能。对于 SIFT 数据集，所有算法都实现了完美召回率，但每秒查询数各不相同。我们的算法比 StitchedVamana、FilteredVamana 和 UNG 等专门针对过滤条件设计的算法慢，但优于其他所有方法。这是意料之中的，因为对于互不重叠的标签筛选器，最优解是为每个标签分别构建一个索引

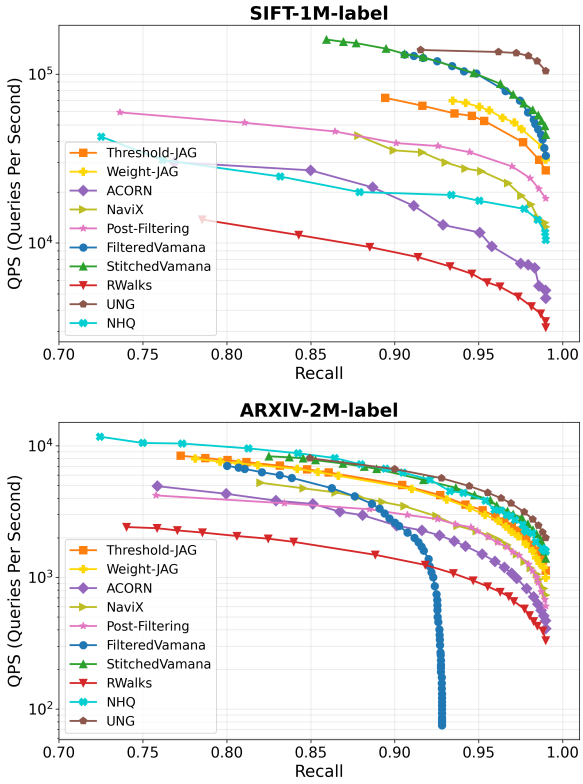


图 3. SIFT 和 ARXIV 数据集上标签过滤器的每秒查询数与召回率关系图。请注意，NHQ 是专为标签过滤器设计的。FilteredVamana、StitchedVamana 和 UNG 是专为标签和子集过滤器设计的。

然后在共享该标签的点子集内执行标准的最近邻搜索——这正是 StitchedVamana、FilteredVamana 和 UNG 在此类情况下所做的。对于 ARXIV 数据集，我们的算法与 StitchedVamana、FilteredVamana 和 UNG 的性能几乎相同，并且优于其他所有算法。

范围过滤器。 图 1和上方的图 5 展示了所有算法在采用范围过滤器的 ARXIV 和 MSTuring 数据集上的性能。在 ARXIV 数据集上，所有算法都达到完美召回率；我们的算法取得了所有其他过滤器无关算法中最高的每秒查询数，但效率不及专为范围查询设计的算法 iRangeGraph。在 MSTuring 数据集上，值得注意的是，只有我们的算法和 iRangeGraph 能够达到完美召回率，而所有其他通用算法的召回率最高仅为 0.8。这是因为我们的合成范围过滤器包含选择率各异的过滤范围；我们推测，其他算法无法有效处理选择率极高的情况。更详细的分析请参阅图 8 中的消融研究。

子集过滤器。图 4展示了所有算法在 MSTuring、YFCC 和 LAION 数据集上的性能

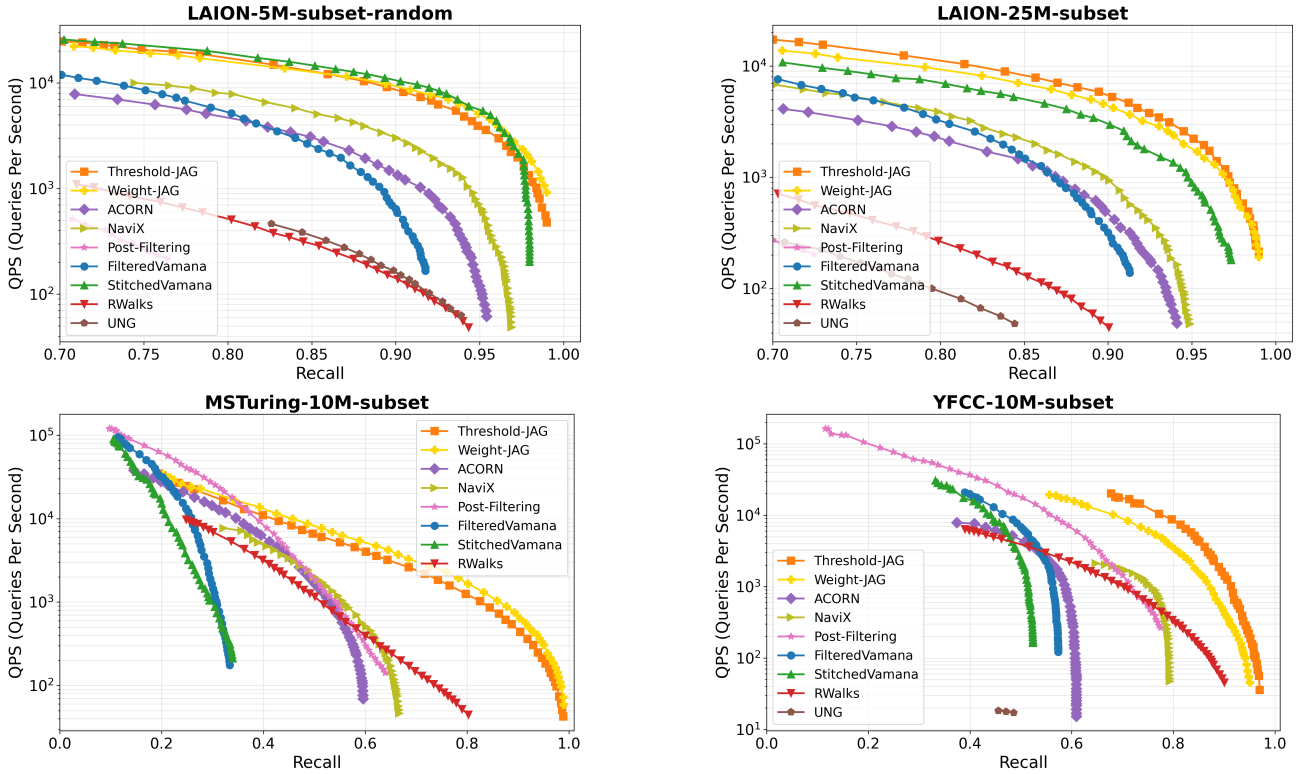


图4. MSTuring-10M、LAION-5M 和 2500万数据集上子集过滤器的每秒查询数与召回率关系图，以及 MSTuring-10M 数据集上布尔过滤器的对应关系图。请注意，FilteredVamana 和 StitchedVamana 仅适用于标签和子集过滤器。

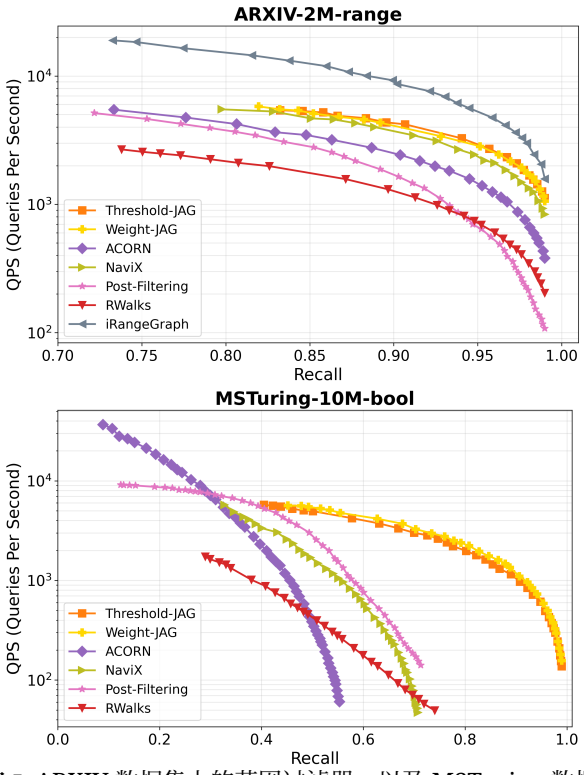


图 5. ARXIV 数据集上的范围过滤器，以及 MSTuring 数据集上的布尔过滤器的每秒查询数与召回率关系图。请注意，iRange-Graph 专为范围过滤器设计

采用子集过滤器时。对于 MSTuring 和 YFCC 数据集，只有我们的算法能够达到完美召回率；在每秒查询数降至 50 以下之前，其他所有算法在 MSTuring 上的召回率最高为 0.8，在 YFCC 上最高为 0.9。正如（Ait Aomar 等人，2025年）所指出的，YFCC 数据集中的大多数查询都具有极强的选择性（<1%）；此前没有任何基于图的算法能在不使用辅助 IVF 结构或预过滤的情况下达到高于 0.9 的召回率。在此，我们首次提出了能够达到近乎完美召回率（>0.95）、每秒查询数为 >50 的基于图的算法。我们的算法在 LAION-5M 上的表现与 StitchedVamana 相当；随着数据集大小增至 2500万，我们的优势进一步扩大。

布尔过滤器。 下图5 展示了所有算法在 MSTuring 数据集上使用布尔过滤器时的性能。只有我们的算法能够达到完美召回率，而在每秒查询数降至 50 以下之前，其他所有算法在 MSTuring 上的召回率最高仅为 0.8。

5. 结论

在本文中，我们提出了JAG（联合属性图），这是一种专为过滤最近邻搜索设计的统一基于图的索引策略。JAG能够在不同过滤器类型和选择率下保持稳健，关键在于属性距离、过滤距离和截断属性距离这些新概念；它们使阈值距离函数能够提供导航引导，并防止导航-

死胡同。与以往仅限于特定过滤器类型、或难以在广泛的选择性范围内取得良好性能的方法不同，JAG在构建过程中统一了针对不同距离阈值定制的边，因此能够处理广泛选择率范围内的筛选条件。

影响声明

本文旨在推动机器学习领域的发展。我们的工作可能产生许多社会影响，但我们认为无需在此特别强调其中任何一项。

参考文献

Ait Aomar, A.、Echihabi, K.、Arnaboldi, M.、Alagiannis, I.、Hilloulin, D. 和 Cherkaoui, M. Rwalks：将随机游走用作属性扩散器，以实现过滤向量搜索。《*ACM*数据管理会议录》，3(3):1–26，2025年。Cai, Y.、Shi, J.、Chen, Y. 和 Zheng, W. Navigating labels and vectors：一种用于过滤近似近邻搜索的统一方法。《*ACM*数据管理会议录》，2(6):1–27，2024年。Engels, J.、Landrum, B.、Yu, S.、Dhulipala, L. 和 Shun, J. 使用窗口过滤器进行近似最近邻搜索。载于国际机器学习会议，第12469–12490页。PMLR，2024年。Gollapudi, S.、Karia, N.、Sivashankar, V.、Krishnaswamy, R.、Begwani, N.、Raz, S.、Lin, Y.、Zhang, Y.、Mahapatro, N.、Srinivasan, P. 等。Filtered-DiskANN：带筛选条件的近似最近邻搜索图算法。载于*2023*年*ACM Web*会议，第3406–3416页，2023年。Gupta, G.、Yi, J.、Coleman, B.、Luo, C.、Lakshman, V. 和 Shrivastava, A. Caps：用于过滤相似性搜索的实用分区索引。*ARXIV*预印本 *arXiv:2308.15014*，2023年。Iff, P.、Brügger, P.、Chrapek, M.、Besta, M. 和 Hoefler, T. 对基于Transformer 的嵌入向量上的过滤近似近邻搜索算法进行基准测试。*ARXIV*预印本 *arXiv:2507.21989*，2025年。Jegou, H.、Douze, M. 和 Schmid, C. 用于大规模图像搜索的汉明嵌入与弱几何一致性。载于欧洲计算机视觉会议，第304–317页。Springer，2008年。Landrum, B.、Manohar, M. D.、Karjikar, M. 和 Dhulipala, L. ivf^2 索引：融合经典倒排索引与空间倒排索引以实现快速过滤 ANN。载于第一届向量数据库研讨会，2025年。

Li, Z., Huang, S., Ding, W., Park, Y., and Chen, J. Sieve：利用索引集合实现高效的过滤向量搜索。*arXiv* 预印本 *arXiv:2507.11907*，2025年。

Lowe, D. G. 基于尺度不变关键点的独特图像特征。《国际计算机视觉期刊》，60（2）：91–110，2004年。

Malkov, Y. A. 和 Yashunin, D. A. 使用分层可导航小世界图进行高效且稳健的近似最近邻搜索。*CoRR*，abs/1603.09320，2016年。网址：http://arxiv.org/abs/1603.09320。

Mohoney, J.、Pacaci, A.、Chowdhury, S. R.、Mousavi, A.、Ilyas, I. F.、Minhas, U. F.、Pound, J. 和 Rekatsinas, T. 知识图谱中的高吞吐量向量相似性搜索。《*ACM*数据管理会议录》，1(2):1–25，2023年。Patel, L.、Kraft, P.、Guestrin, C. 和 Zaharia, M. Acorn：针对向量嵌入和结构化数据的高性能谓词无关搜索。《*ACM*数据管理会议录》，2(3)，2024年5月。doi: 10.1145/3654923。URLhttps://doi.org/10.1145/3654923。Sehgal, G. 和 Salihoglu, S. Navix：面向图数据库管理系统的原生向量索引设计，具备稳健的谓词无关搜索性能。*ARXIV*预印本 *arXiv:2506.23397*，2025年。Subramanya, S. J.、Devvrit, F.、Kadekodi, R.、Krish- nawamy, R. 和 Simhadri, H. V. Diskann：单节点上快速、准确地进行十亿点最近邻搜索。载于 Wallach, H. M.、Larochelle, H.、Beygelz- imer, A.、d' Alché-Buc, F.、Fox, E. B. 和 Garnett, R.（编），《神经信息处理系统进展 *32*：*2019*年神经信息处理系统年会，*NeurIPS 2019*，*2019*年*12*月*8*日至*14*日，加拿大不列颠哥伦比亚省温哥华》，第13748–13758页，2019年。URL https://dl.acm.org/doi/abs/10.5555/3454287.3455520。Thomee, B.、Shamma, D. A.、Friedland, G.、Elizalde, B.、Ni, K.、Poland, D.、Borth, D. 和 Li, L.-J. YFCC100M：多媒体研究中的新数据。《*ACM*通讯》，59(2):64–73，2016年。Wang, M.、Lv, L.、Xu, X.、Wang, Y.、Yue, Q. 和 Ni, J. 基于可导航邻近图驱动的原生混合查询：带有结构化和非结构化约束。*ARXIV*预印本 *arXiv:2203.13601*，2022年。Xu, Y.、Gao, J.、Gou, Y.、Long, C. 和 Jensen, C. S. iRange-图：改进用于范围过滤最近邻搜索的范围专用图。《*ACM*数据管理会议录》，2(6):1–26，2024年。

Zhang, D.、Li, J.、Zeng, Z. 和 Wang, F. Jasper 和 stella：最先进嵌入模型的蒸馏。*ARXIV*预印本 *arXiv:2412.19048*，2024年。

Zuo, C., Qiao, M., Zhou, W., Li, F. 和 Deng, D. Serf：用于范围过滤近似最近邻搜索的分段图。美国计算机协会数据管理会议论文集，2(1):1–26，2024年。

A. 相关工作

一种解决过滤向量搜索的方法是直接利用数据库处理过滤条件。预过滤首先筛选出满足元数据约束的点子集（例如，使用 SQL 查询），随后在这个缩减后的候选集中计算距离。尽管此方法能够保证正确性，但在过滤子集仍然很大的高选择率场景中，计算成本会变得很高。

完全无感知方法。我们将完全无感知方法定义为在索引构建时忽略属性的方法。后过滤使用标准的未过滤索引（例如 HNSW（Malkov 和 Yashunin，2016年）或 Vamana（Subramanya 等人，2019年））生成初始近邻列表，然后回溯删除不符合过滤条件的项。当筛选条件较宽松（选择率较高）时，这种方法效果良好；但当有效项稀疏，或在向量空间中远离查询点时，往往无法检索到足够多的有效邻居。

为缓解标准后过滤方法的召回率问题，ACORN（Patel 等人，2024年）和 NaviX（Sehgal 和 Salihoglu，2025年）等更先进的完全无感知方法，仅使用向量数据构建索引，但会修改搜索过程。这些方法通常采用扩展邻域策略（如两跳遍历），以探索更大的向量空间区域，从而期望找到足够数量的有效候选项。然而，这些方法隐含地假设满足过滤条件的点在向量邻域中均匀分布。因此，当过滤条件与向量距离呈负相关或选择率较低时，性能会显著下降。

过滤器感知方法。 过滤器感知算法会针对预先已知的特定类别查询谓词定制索引结构。例如，FilteredVamana 和 StitchedVamana（Gollapudi 等人，2023年）修改 DiskANN 架构以适应标签约束：前者将导航限制在有效点上，而后者则合并针对各个过滤条件分别构建的图。类似地，其他专用方法会使用辅助数据结构（如字典树或线段树）增强邻近图，以处理特定的过滤器类型。尽管这些解决方案在其针对的任务（例如标签相等或范围搜索）上表现出色，但它们缺乏泛化能力；针对一种过滤器类型进行优化，往往会妨碍对其他类型（例如布尔或子集查询）的高效处理。

属性感知（过滤器无关）方法。 属性感知（或过滤器无关）方法会在索引构建时利用属性，构建针对多种（未知）查询进行优化的索引。据我们所知，能够归类为属性感知的方法相对较少。NHQ（Wang 等人，2022年）是一种较早的过滤搜索算法，它仅根据两个点的属性是否相等来定义属性距离，并使用加权平均将其与标准向量距离结合起来。因此，NHQ 只能处理基于相等性的过滤器（因而也可以合理地将其归类为过滤器感知方法）。据我们所知，唯一的另一种属性感知方法是近期提出的 RWalks（Ait Aomar 等人，2025年），它使用标准的（未过滤）索引，并对其进行增强以支持过滤搜索。在索引构建时，该算法会在未过滤索引上执行随机游走，以计算聚合属性向量。随后，它将过滤距离定义为该向量中与查询过滤器匹配的标签所占的比例。在查询时，该过滤距离会通过加权评分函数与通常的向量距离相结合，以引导贪心搜索。

B. 补充实验结果

规模扩展实验。 对于 LAION 数据集，我们选取表现最好的三种算法——我们的方法、ACORN 和 StitchedVamana——并在不同大小的数据集（100万、500万和2500万）上运行，以评估其规模扩展表现。结果请见图 7。在100万和500万规模下，我们的算法与 StitchedVamana 的表现相当；到了2500万规模，我们的优势则更加明显。随着数据集大小从100万增加到500万再到2500万，所有每秒查询数曲线都呈下降趋势，而我们的算法仍能达到完美召回率，并保持每秒查询数 > 50。

相关性。我们测试查询过滤器与向量在空间上相关时各算法的表现。对于 LAION-5M 数据集，每个关键词都代表向量空间中的一个点簇。为建立查询与过滤条件之间的正相关，我们将查询过滤器设为距离查询图像嵌入最近的关键词簇。为建立负相关，我们则选择距离查询图像嵌入最远的关键词簇作为查询过滤器。结果见图6 。在正相关情况下，我们的算法 FilteredVamana、后过滤和 NaviX 的表现几乎相同。有趣的是，后过滤在其他子集过滤器场景中表现不佳，但在此场景下速度极快。这是因为我们选择查询过滤器的方式，使查询邻域与过滤子集之间有显著重叠——也就是说，距离查询向量最近的点很可能符合过滤条件。在负相关情况下，我们的算法、NaviX 和 UNG 均能达到 > 0的 0.95 召回率，而我们的算法每秒查询数最高。

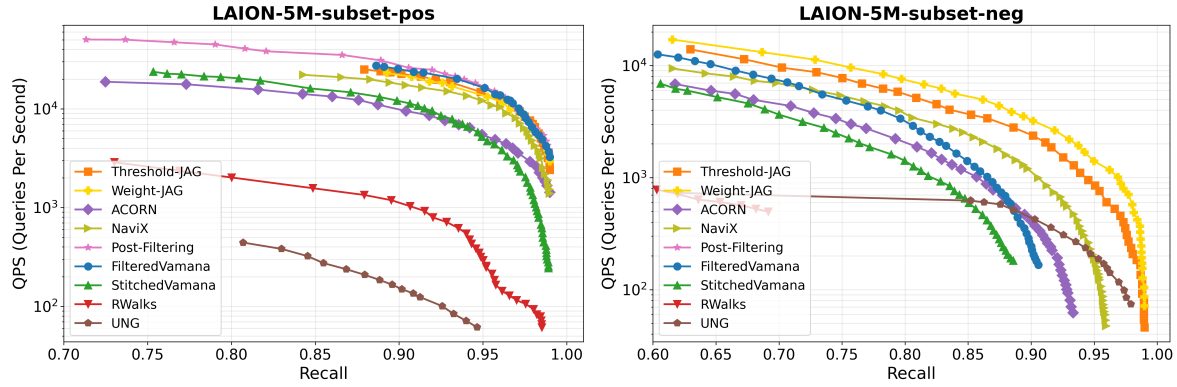


图 6. 每秒查询数 v s. 召回率曲线，展示了 LAION-5M 数据集上具有正相关和负相关的子集过滤器。

过滤器选择率变化。我们评估了各算法在不同查询选择性下的表现。我们使用范围过滤器在 MSTuring-10M 上进行实验，并分别测量查询过滤器选择率为 1， 10^{-1}， 10^{-2}， 10^{-3}， 10^{-4}， 10^{-5} 时各算法的召回率。对于每个选择率，我们报告在保持每秒查询数 >5000的情况下所能达到的最高召回率。</output>

如图 8所示，所有通用过滤向量搜索算法仅在选择率较高时表现良好；随着选择率降低，它们的召回率逐渐降至零。相比之下，我们的 JAG 算法与 iRangeGraph 表现相当，后者专为范围过滤条件设计。

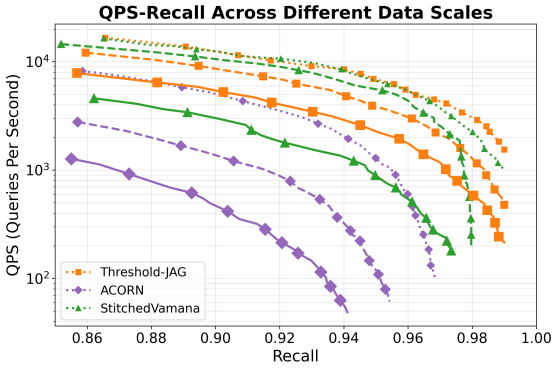


图 7. LAION 数据集上子集过滤器的每秒查询数与召回率。每种算法以不同颜色表示；不同的线型表示不同的数据集规模：点线表示 1 M，虚线表示 5M，实线表示 2500万。

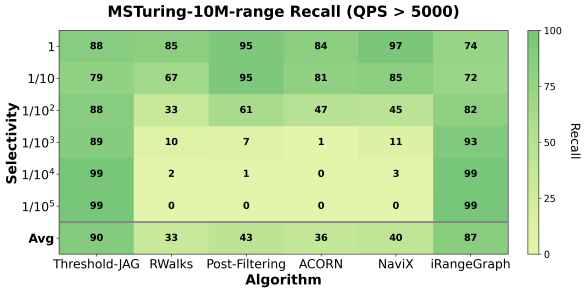


图 8. 在 MSTuring-10M 数据集上维持每秒查询数 >5000 时，不同过滤器选择率下范围过滤器的召回率。

C. 消融研究

多个阈值。我们分别在 MSTuring-10M 的范围过滤器和 LAION-5M 的子集过滤器上进行消融实验，以考察不同阈值如何帮助我们的算法处理选择率各异的过滤条件。请注意，第4 节中的原始查询过滤器包含混合选择性。此处，我们评估采用不同单一阈值构建算法时的性能，并测量算法在选择率不同的过滤条件下的表现。

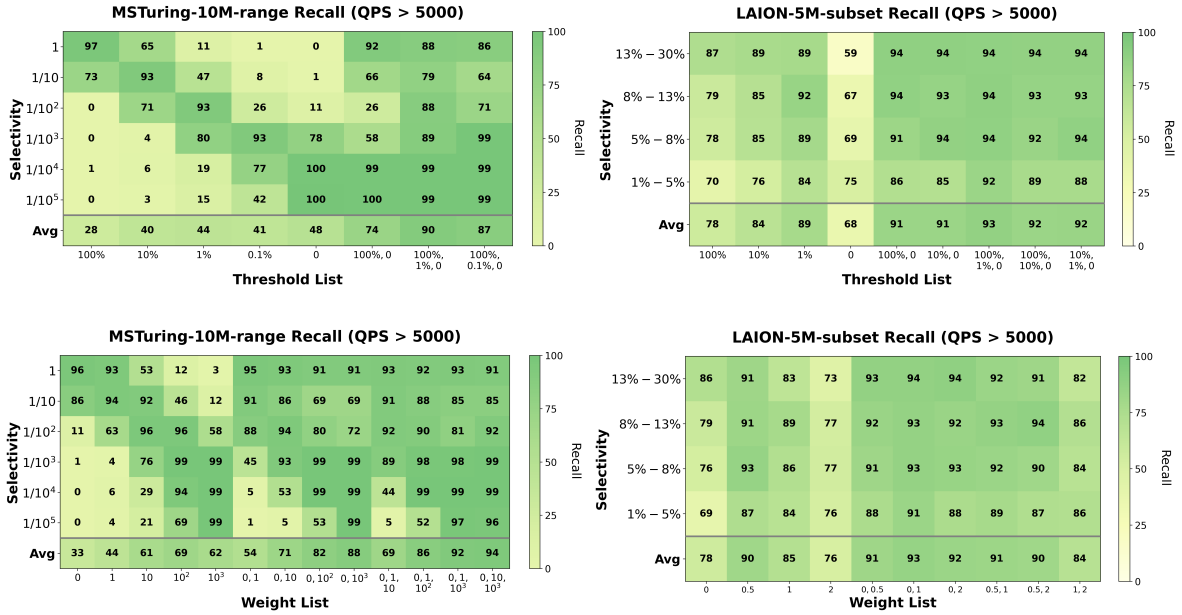


图9. 在 MSTuring-10M 数据集上使用范围过滤器（左上）和在 LAION-5M 数据集上使用子集过滤器（右上）时，不同阈值下的查询召回率；以及在 MSTuring-10M 数据集上使用范围过滤器（左下）和在 LAION-5M 数据集上使用子集过滤器（右下）时，不同权重下的查询召回率

对于 MSTuring-10M-range，我们考虑六种查询选择性：1，10^{-1}，10^{-2}，10^{-3}，10^{-4}，10^{-5}。对于 LAION-5M-subset，我们将所有查询选择性排序，并将其均匀划分为四个范围（$13\% - 30\%$，$8\% - 13\%$，$5\% - 8\%$，$1\% - 5\%$）。所有阈值选项均采用相同的度数限制。结果请参见图9 中的上方两幅图。

当单一过滤阈值设为 0% 时——这意味着我们严格按照过滤距离进行比较，并且只有在过滤距离相同时才使用标准距离——所得索引在低选择率情况下（对两种过滤器类型均如此）可实现完美召回率。然而，随着选择率提高，其召回率逐渐下降。这是因为选择率变低时，过滤条件的重要性随之降低，但查询比较仍然过度强调过滤距离。

相反，阈值为 100% 时，效果等同于仅根据标准距离进行搜索。该索引在高选择率情况下表现良好，但随着选择率变得更具挑战性，性能会下降。有趣的是，每个中间阈值都只在选择性范围内的某个特定区间表现最佳；当选择率偏离其“舒适区”时，性能便会下降。例如，在子集过滤器情况下，1% 阈值在选择率介于 8% – 13% 的查询上表现最佳（即按查询选择率排序后位于前 25–50% 的查询）。当选择率更高或更低时，其性能都会下降。同样，在范围过滤器情况下，1% 阈值在 $1/100$ 时表现最佳，并且向两侧变化时性能都会下降。

通过合并三个不同的阈值 $\{100\%,\ 1\%\ 0\}$（共享相同的度界），最终索引在所有查询选择率水平下都实现了强劲且稳定的性能。

Weight-JAG 我们还实现了 JAG 的加权变体，其中属性距离和向量距离通过不同权重进行组合。对于权重 w，我们定义

$$D_A^w(u,v) = w \cdot \mathrm{dist}_A(a_u, a_v) + \mathrm{dist}(x_u, x_v).$$

我们使用一组这样的权重构建索引图。与 Threshold-JAG 类似，在调用 Insert 和 JointRobustPrune 时，我们遍历权重列表，并使用 $D_A^w(u,\ v)$ 作为比较器。

在查询处理过程中，我们仍然先比较 $\mathrm{dist}_F(\cdot)$，然后比较标准向量距离。我们将此变体称为 Weight-JAG。

多种权重。 我们还对 Weight-JAG 进行了消融实验，以考察不同权重如何帮助处理选择率各异的过滤条件。从图 9下方的两幅图中可以看出，权重为 0 时，邻近图完全依据标准向量距离构建，对标准最近邻查询（选择率为 1 ）取得了最高召回率。随着权重增加，生成的邻近图会更加适用于低选择率查询。通过

表 1。数据集规模和平均选择率。预过滤算法的每秒查询数（QPS）和距离计算次数（DC）。

		SIFT1M 标签	ARXIV 标签	ARXIV 范围	LAION-1M 子集	LAION-5M 子集	LAION-25M 子集	YFCC 子集	MSTuring10M 子集	MSTuring10M 范围	MSTuring10M Bool
数据集	Size	10^6	$2.7 \cdot 10^6$	$2.7 \cdot 10^6$	10^6	$5 \cdot 10^6$	$2.5 \cdot 10^7$	10^7	10^7	10^7	10^7
	Avg 选择率	0.083	0.37	0.43	0.099	0.098	0.099	0.018	0.15	0.17	0.15
Pre	QPS	412	5.4	4.6	146	28	5.6	5.2	27	31	2.3
过滤	DC	$8.3 \cdot 10^4$	$1.0 \cdot 10^6$	$1.1 \cdot 10^6$	$9.9 \cdot 10^4$	$4.8 \cdot 10^5$	$2.4 \cdot 10^6$	$1.8 \cdot 10^5$	$1.5 \cdot 10^6$	$1.7 \cdot 10^6$	$1.5 \cdot 10^6$

合并构建的图　　采用不同权重时，最终索引在所有查询选择率水平上都能实现较高的召回率　　s.

比较 ***Threshold-JAG*** 和 ***Weight-JAG***。 尽管我们的 JAG 的两种实现（Threshold 和 Weight）都能处理选择率各异的一般过滤搜索，但它们各有优势。Weight-JAG 在 MSTuring-10M-range 上的召回率可能略高，但其权重选择并不十分稳健。例如，MSTuring-10M-range 上的最优权重组合是 $\{0, 10, 1000\}$，而 LAION-5M-subset 上的最优组合则是 $\{0, 2\}$。相比之下，对于 Threshold-JAG，我们在整个实验中均从 $\{100$ %，10%、1% 0。1% $0\}$ 中选择阈值。

总结这两种算法变体的实验结果，我们发现，Threshold-JAG 在不同数据集上始终表现良好，并且所需调参较少。另一方面，Weight-JAG 偶尔能以微小优势取得最佳结果；然而，它在不同数据集上的稳健性较差，并且需要更多调参。我们建议将 Threshold-JAG 作为稳健且质量良好的默认选择。

统一多个图。 一个自然的问题是： 如果提前知道查询选择率，我们是否可以简单地构建多个相互独立的索引，每个索引针对特定的选择率范围进行定制，然后将每个查询路由到相应的索引？我们的结果表明，这种策略并非最优。

从图 9中的 LAION-5M 实验可以观察到，对于选择率在 $1\% - 5\%$范围内的查询，阈值组合 $\{100\%,\ 1\%\ 0\}$ 优于每一个单独的阈值。类似地，对于 Weight-JAG，权重组合$\{0, 2\}$ 优于任何单独的权重配置。

这些发现表明，贪心搜索能够从基于多个 阈值或权重构建的边中获益。搜索可以自适应地遍历这些异构边，从而高效地到达目标——单个特定选择率的索引无法做到这一点。

D. 实验设置与细节

D.1. 机器

所有实验均在谷歌云虚拟机上进行，配置如下：n2-highmem-64 实例，配备 32 个可见的英特尔 Ice Lake 虚拟 CPU（每个物理核心对应 1 个虚拟 CPU）和 512 GB 随机存取存储器。实验使用 32 个线程运行。

D.2. 其他数据集细节

SIFT。 SIFT 数据集（Lowe，2004年；Jegou 等人，2008年）是近似最近邻（ANN）和过滤近似最近邻研究的标准基准之一。它包含一百万个点，位于 $\mathbb{R}^{128}$。按照（Patel 等人，2024年；Cai 等人，2024年），我们为每个点分配一个从 $\{1, \ldots, 12\}$中均匀采样的整数属性，并为每个查询分配一个标签筛选器。如果某个点的属性与查询的标签匹配，则该点满足过滤条件。

ARXIV。 我们采用近期基准测试（Iff 等人，2025年）中的数据集和过滤条件设置。该基准包含使用`stella_en_400M_v5` 模型生成的 270 万篇 arXiv 论文的文本嵌入（Zhang 等人，2024年）。我们同时使用范围过滤器和标签过滤器：对于范围过滤器，属性是发布日期，查询指定目标时间范围；对于标签过滤器，属性是分配给每篇论文的子类别数量，查询检索具有指定子类别数量的邻居。

LAION。 我们使用（Patel 等人，2024年；Cai 等人，2024年）中采用的 LAION数据集，其中包含网络抓取图像的 CLIP 嵌入。按照（Patel 等人，2024年）中的设置，我们构建了一个包含 30 个常用关键词的词汇表

并为每张图像分配与该图像向量嵌入最接近的三个关键词。每个查询指定一个关键词作为过滤条件，搜索会检索其分配的关键词集合中包含该指定词语的图像向量。我们在三个规模下实例化这一设置：100万、500万和2500万个点。由于关键词能够捕捉图像的语义内容，我们通过选择不同类型的查询过滤器来改变过滤条件与向量相似度之间的相关性：正向 （与查询图像最接近的关键词）、随机 （随机选取的关键词）和负向 （距离查询图像嵌入最远的关键词）。

YFCC。 YFCC 数据集是 NeurIPS 2023 竞赛过滤搜索赛道采用的标准过滤近似近邻基准。它由从 YFCC100M（Flickr）中随机选取的 1000 万张图片的 CLIP 嵌入组成（Thomee 等，2016年）。每张图片都关联着一个从描述文本、相机型号、年份和国家等元数据中提取的“标签包”，其中包含超过 20 万个独特属性。每个查询指定一组标签作为过滤条件，目标是检索标签集包含查询中所有过滤条件词项的图片。

MSTuring。 MSTuring 数据集由 Bing 搜索查询及其对应答案组成。我们使用一个包含 1000 万个点的子集，其嵌入维度为 100，并通过合成方式构造具有混合选择率的子集、范围和布尔值 筛选条件。

子集过滤器： 我们定义 30 个二元属性；每个点都以 $1/2$ 的概率独立包含每个属性。对于一个查询，我们随机选取 $k \in \{0, 2, 4, 6, 8, 10, 12, 14, 16\}$ 个属性，并要求候选点包含所有选定属性。

范围过滤器：每个点都被分配一个位于 $[0, 10^6]$, 中的整数属性，查询指定长度为 $10^6/k$ 的随机区间，用于 $k \in \{1, 10, 100, 1000, 10^4, 10^5\}$ 以控制选择率。

布尔过滤器：每个查询过滤器都是关于15个布尔变量的随机布尔函数 f，每个点的属性是这些变量的一个随机实例化 x。如果 $f(x) = $ 为真，则该点满足过滤条件。我们通过生成通过率落入四个范围的函数来控制选择率：$(1/2^4, 1)$、$(1/2^8, 1/2^4)$、$(1/2^{12}, 1/2^8)$和$(0, 1/2^{12})$。

D.3. JAG 的实现细节

对于 Threshold-JAG，在实践中，我们通过在特定点处选择阈值来确定 Threshold-JAG 的阈值。对于属性为 a_p 的每个点 p，我们采样一组点 V（例如 $|V| = 500$），计算$dist_A(a_p, a_V)$的分布，并从分位数集合 $\{100\%, 10\%, 1\%, 0\%\}$中考虑候选阈值。然后，我们选择性能最佳的阈值列表。

对于 Weight-JAG，在实际应用中，对于每个具有属性 a_p 的点 p，我们抽取一组 V 个点，计算属性距离 $\text{dist}_A(a_p, a_V)$ 的标准差σ_A，以及向量距离 $\text{dist}(x_p, x_V)$ 的标准差 σ。令 $h = \sigma/\sigma_A$。然后，我们从集合中选择权重

$$\{0, h, 2h, 5h, 10h, 100h, 1000h\}$$

并选择性能最佳的子集。请注意，与 Threshold-JAG 相比，Weight-JAG 使用的权重范围大得多，而且一些极端值（例如 $1000h$）在我们的实验中证明很有用。这表明 Weight-JAG 的稳健性不如 Threshold-JAG。由于在先前的研究中已经探索过属性距离和向量距离的加权组合（Wang 等人，2022年；Ait Aomar 等人，2025年），我们纳入这一变体主要是为了进行消融研究。综合性能和稳健性考虑，我们推荐使用 Threshold-JAG 版本。

在算法 4 的第 8 行，我们令 V'_t包含 v，如果 $v \in V'$ 已由先前的阈值 t 添加。这样做是为了实现更好的剪枝并提高边利用率。

为了加快索引构建过程，同时避免触发过多次 JointRobustPrune 调用，我们在算法4 的第 10 行引入了一个提前退出条件：当满足 $|V'_t| \geq 0.9 \cdot \deg/|T|$时终止。这意味着，执行 JointRobustPrune 后，我们只存储90% 的度数限制，预期这样可以避免每次添加新边时都触发 JointRobustPrune。这是以往实现中常用的一种技术。

对于 YFCC 和 LAION 数据集，由于其标签分布的方差较大，我们对属性距离设计进行了调整。例如，在 YFCC 中，这种方差体现在两个方面：

1. 该数据集包含 $L = 200363$ 个不同的标签，单个数据点可能关联 0 到 1517 个标签。

2. 属性频率存在较大差异：某些标签在超过 30% 的数据集中共享，而

表 2. 算法对不同过滤器类型的兼容性。✓ 表示支持，✗表示不支持。

过滤条件 Type	JAG	Post 过滤	ACORN	NaviX	RWalks[3]	过滤条件 Vamana	拼接 Vamana	NHQ	UNG	iRange 图
标签	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
范围	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓
子集	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗
布尔	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗

其他项仅出现一次。

为缓解这种不平衡并优先考虑频率较低的属性，我们收集每个属性的频率 p_i $i \in [L]$，并按 $\log(1/p_i)$ 对其加权。随后，我们将属性距离定义为：

$$dist_A(a_u, a_v) = C - \sum_{i \in a_u \cap a_v} \log(1/p_i)$$

对于一个足够大的常数 C。该设计优先考虑在数据集中出现频率较低的筛选条件，并关注共享标签，而非它们之间的差异。

D.4. 算法补充细节

下面简要介绍我们评估的各个基线算法，以及如何选择其参数。

具体而言，如果某个算法和数据集曾在原论文中接受过评估，我们便直接采用原论文中的参数设置。否则，我们参考原论文推荐的参数选项，并选择在召回率为 0.95 时每秒查询数最高的配置。如果该算法在每秒查询数为 =50 时所能达到的最佳召回率低于 0.95，则选择能够达到最高召回率的配置。

ACORN。(Patel 等人，2024年) 提出了两个改进变体：ACORN-1 和 ACORN-γ。在我们的实验中，我们重点关注 ACORN-γ，因为它的搜索质量更高。ACORN-γ 构建了一个更稠密的图，并在搜索过程中采用两跳扩展。假设最小过滤器选择率为 γ 且 M 是标准 HNSW 中使用的度参数。ACORN-γ 使用两跳扩展，确保每个节点都能访问约 $M \cdot \gamma$ 个附近的点。过滤掉不满足查询过滤器的点后，剩余邻居应近似于仅在满足过滤条件的点上构建标准 HNSW 图时得到的邻居。

对于 ACORN 论文中使用的 SIFT 和 LAION 数据集，我们采用作者建议的参数设置。对于其他数据集，我们从集合 $\{M = 32, M_\beta = 32\}$、$\{M = 32, M_\beta = 64\}$ 和 $\{M = 64, M_\beta = 64\}$ 中选择参数。对于 γ，我们将其设为选择率最高的 5% 查询数量与 30 中的较小值。

NaviX。(Sehgal 和 Salihoglu，2025年) 在 ACORN 的基础上提出了一种改进的搜索启发式。它构建标准的 HNSW 图，并在查询时根据局部选择率估计，自适应地决定扩展一跳还是两跳邻居。我们使用作者的 FAISS-Navix 实现，并启用 *adaptive-local* 启发式。我们测试的参数配置为 $\{M = 32, \texttt{efc} = 200\}$ 和 $\{M = 64, \texttt{efc} = 200\}$，并为每个数据集选择性能较好的配置。

FilteredVamana。(Gollapudi 等人，2023年) 提出了一种专为子集过滤搜索设计的 Vamana 算法扩展版本。原始实验侧重于单过滤器情形，这自然可以推广到多个 OR 过滤器。在我们的评估中，我们还测试了该算法在多个 AND 过滤器（即我们的子集设定）下的表现。其核心思想是，插入一个点时，算法只遍历并连接与该点至少共享一个共同属性的已有点。在执行“FilteredRobustPruning”时，如果边 (p, p') 被剪枝并由边 (p, p^*) 替代，则替代点 p^* 必须涵盖 p 与 p' 之间共享的所有属性。查询时，该算法会将遍历范围限制在满足查询过滤器的点中。我们从 $\texttt{deg} \in \{64, 96, 128\}$ 和 $\alpha = 1.2$ 中选取最优参数。

[3]RWalks 所使用的原始过滤距离仅支持二元匹配/不匹配过滤器（例如标签过滤器和子集过滤器）。在我们的实验中，我们采用所提出的过滤距离来实现 RWalks，该过滤距离可推广到范围更广的过滤器类型，并带来一致的性能提升。

StitchedVamana。这是（Gollapudi 等人，2023年）中提出的第二种算法。它主要针对单过滤器设置而设计，但我们也在多个 AND 过滤器（即我们的子集情形）下对其进行评估。对于数据集中的每个不同标签，该算法在包含该标签的点上构建图时使用较小的度参数 R_{small}。对于具有多个标签的点，它将来自相应逐标签图的出边进行合并，合并时使用 *FilteredRobustPruning*，并采用更大的度 R_{stitched}。在查询处理期间，遍历被限制在满足过滤条件的点上。我们测试的参数配置为 $R_{\text{stitched}} \in \{64, 96, 128\}$， $R_{\text{small}} = R_{\text{stitched}}/2$，以及 $L = 1.5 \cdot R_{\text{stitched}}$。

RWalks。（Ait Aomar 等人，2025年）基于标准未过滤索引构建。对于每个点，该算法执行随机游走以计算聚合属性向量。当查询到来时，它将点与查询之间的过滤距离定义为聚合属性向量中与查询过滤条件相匹配的标签所占的比例。在我们的实验中，我们使用所提出的过滤距离来实现 RWalks；该距离严格推广了原论文所采用的二元匹配/不匹配判据，并一致地提升了性能。在搜索过程中，贪心遍历由过滤距离与标准向量距离的加权组合引导，并使用权重参数 h。超参数 h 据报告在设置为0.1（归一化之后）时效果最佳。

我们在 Vamana 之上实现了 RWalks 的算法思想，并引入了少量修改以提升性能（例如，我们使用广义的属性与过滤距离定义，而非其二元相等定义）。我们测试的度参数为 deg $\in \{64, 96, 128\}$。对于所有其他参数，我们采用原论文推荐的取值： $m = 5$、 $d = 3$、 $\tau = 0$以及 $h = 0.1$。

后过滤。 我们首先在不应用任何过滤约束的情况下检索近邻，然后对结果执行过滤检查，以获得满足查询过滤器的前 k 个点。我们测试度参数 deg $\in \{32, 64, 96, 128\}$。

预过滤。 我们首先对点应用过滤约束，然后计算它们的距离，以获得排名前 k 的结果。这种预过滤方法不需要索引。

NHQ。(Wang 等人，2022年)提出了 NHQ 算法，其针对标签相等过滤器。它仅根据两个点的属性是否相等来定义它们之间的属性距离，并使用加权平均将其与标准向量距离相结合。由于该属性距离只能区分相等性，NHQ 在我们的实验中只能处理基于相等性的标签过滤器。我们采用（Iff 等人，2025年）中为 ARXIV 和 SIFT 数据集推荐的参数设置。

UNG。（Cai 等人，2024年）提出了统一导航图（UNG），专为子集过滤查询设计。该算法首先为每个可能的标签集构建一个邻近图。随后，通过在一个标签集包含另一个标签集时连接一条有向边，在这些标签集上构建标签导航图。对于子集查询，UNG 会维护一个字典树数据结构以确定可行的起始点，然后从这些入口点开始执行贪心搜索。我们主要采用原论文推荐的参数设置： $\alpha = 1.2$， $\delta = 6$，以及 $\sigma = 16$。对于度 R 和队列长度 L，我们选择了以下配置中性能最佳的一组： （R， L） $\in$ {（32，100）、（48，150）、（64，200）}。

iRangeGraph。（Xu 等人，2024年）提出了用于范围过滤查询的 iRangeGraph。它构建一棵线段树，将属性范围分解为长度各异的区间，为每个区间构建图索引，并且仅在与查询范围重叠的区间图上进行搜索。我们采用作者推荐的参数设置： $m = 16$ 和$\texttt{EF} = 100$。

D.5. 参数设置

下面汇总各算法和数据集所使用的参数。

- Threshold-JAG。我们对所有数据集使用 $\alpha = 1.2$ s.

  – SIFT-1M（标签）：度数 $= 96$**，阈值** $= \{10\%\ 0\}$**– ARXIV（标签）：度数** $= 64$**，阈值** $= \{100\%\ 0\}$**– ARXIV（范围）：度数** $= 96$**，阈值** $= \{100\%,$ **10%**$\}$**– LAION-1M / LAION-5M / LAION-25M：度数** $= 96$**，阈值** $= \{10\%,$ **1%** $0\}$**– YFCC：度数** $= 128$**，阈值** $= \{10\%,$ **1%** $0\}$**– MSTuring（子集、范围、布尔值）：度数** $= 128$**，阈值** $= \{100\%,$ **1%** $0\}$

• Weight-JAG。我们对所有数据集均使用 $\alpha = 1.2$ 。

– **SIFT-1M（标签）：** 度 $= 96$，权重 $= \{1\}$– **ARXIV（标签）：** 度 $= 64$，权重 $= \{0, 10\}$– **ARXIV（范围）：** 度 $= 96$，权重 $= \{0, 2\}$– **LAION-1M / LAION-5M / LAION-25M（子集）：** 度 $= 96$，权重 $= \{0, 1\}$– **YFCC（子集）：** 度 $= 128$，权重 $= \{0, 1\}$– **MSTuring（子集）：** 度 $= 128$，权重 $= \{0, 1, 5\}$– **MSTuring（范围）：** 度 $= 128$，权重 $= \{0, 10, 1000\}$– **MSTuring（布尔值）：** 度 $= 128$，权重 $= \{0, 1, 10\}$

• ACORN。

– **SIFT-1M（标签）：** $M = 32$，$\gamma = 12$，$M_\beta = 64$– **ARXIV（标签）：** $M = 32$，$\gamma = 25$，$M_\beta = 32$– **ARXIV（范围）：** $M = 32$，$\gamma = 10$，$M_\beta = 32$– **LAION-1M / LAION-5M / LAION-25M（子集）：** $M = 32$，$\gamma = 30$，$M_\beta = 32$– **YFCC（子集）：** $M = 64$，$\gamma = 30$，$M_\beta = 64$– **MSTuring（子集）：** $M = 64$，$\gamma = 30$，$M_\beta = 64$– **MSTuring（范围）：** $M = 64$，$\gamma = 30$，$M_\beta = 64$– **MSTuring（布尔值）：** $M = 32$，$\gamma = 30$，$M_\beta = 64$

• NaviX。

– **SIFT-1M（标签）：** $M = 32$，$efc = 200$– **ARXIV（标签）：** $M = 64$，$efc = 200$– **ARXIV（范围）：** $M = 64$，$efc = 200$– **LAION-1M / LAION-5M / LAION-25M（子集）：** $M = 32$，$efc = 200$– **YFCC（子集）：** $M = 64$，$efc = 200$– **MSTuring（子集）：** $M = 64$，$efc = 200$– **MSTuring（范围）：** $M = 64$，$efc = 200$– **MSTuring（布尔值）：** $M = 64$，$efc = 200$

• RWalks。 $\alpha = 1.2$、 $m = 5$、 $d = 3$、 $\tau = 0$、 $h = 0.1$

– **SIFT-1M（标签）：** 度 $= 64$– **ARXIV（标签）：** 度 $= 64$– **ARXIV（范围）：** 度 $= 64$– **LAION-1M（子集）：** 度 $= 64$– **LAION-5M / LAION-25M（子集）：** 度 $= 96$– **YFCC （子集）：** 度 $= 96$– **MSTuring （子集）：** 度 $= 96$– **MSTuring（范围）：** 度 $= 128$– **MSTuring（布尔值）：** 度 $= 96$

• 后过滤。 $\alpha = 1.2$

– **SIFT-1M（标签）：** 度 $= 32$– **ARXIV（标签）：** 度 $= 96$– **ARXIV（范围）：** 度 $= 32$– **LAION-1M（子集）：** 度 $= 32$– **LAION-5M（子集）：** 度 $= 64$– **LAION-25M（子集）：** 度 $= 128$

– **YFCC（子集）：** 度 $= 64$–
MSTuring（子集）： 度 $= 96$–
MSTuring（范围）： 度 $= 64$–
MSTuring（布尔值）： 度 $= 64$

• FilteredVamana。 $\alpha = 1.2$

– **SIFT-1M（标签）：** 度 $= 64$–
ARXIV（标签）： 度 $= 96$–
LAION-1M（子集）： 度 $= 96$–
LAION-5M（子集）： 度 $= 96$–
LAION-25M（子集）： 度 $= 128$–
YFCC（子集）： 度 $= 128$–
MSTuring（子集）： 度 $= 128$

• StitchedVamana。 $\alpha = 1.2$

– **SIFT-1M（标签）：** 度 $= 64$–
ARXIV（标签）： 度 $= 96$–
LAION-1M（子集）： 度 $= 96$–
LAION-5M（子集）： 度 $= 96$–
LAION-25M（子集）： 度 $= 96$–
YFCC（子集）： 度 $= 64$–
MSTuring（子集）： 度 $= 128$

• NHQ– **SIFT-1M（标签）：** $K = 80$， $L = 60$， $S = 10$， $R = 200$，范围 $= 60$， $PL = 300$， $B = 0.6$，权重_搜索 =1000000– **ARXIV（标签）：** $K = 80$， $L = 60$， $S = 10$， $R = 200$，范围 $= 60$， $PL = 300$， $B = 0.6$，权重_搜索 =1000000

• UNG。 $\alpha = 1.2$， $\delta = 6$， $\sigma = 16$

– **SIFT-1M（标签）：** 度 $= 32$，L $= 100$–
ARXIV（标签）： 度 $= 48$，L $= 150$–
LAION-1M（子集）： 度 $= 32$，L $= 100$–
LAION-5M（子集）： 度 $= 32$，L $= 100$–
LAION-25M（子集）： 度 $= 32$，L $= 100$–
YFCC （子集）： 度 $= 32$，L $= 100$

• iRangeGraph。

– **ARXIV（范围）：** $M = 16$， $ef = 100$–
MSTuring（范围）： $M = 16$， $ef = 100$

D.6. 索引时间

我们在表 3 中报告了使用第 D.5 节所指定参数时，各算法在所有数据集上的索引时间。如第 4.2 节所述，我们的参数选择指南是在召回率为 0.95 时最大化每秒查询数，而非优化索引时间或索引大小。

D.7. 距离计算与召回率

我们在图 10、11、12、13 中报告了召回率@10 与距离计算量的关系。

表3. 不同数据集上所有算法的索引时间。我们展示了 Threshold-JAG 的索引时间；Weight-JAG 的索引时间与之相近。构建索引所用的参数请参阅附录 D.5 。

	JAG	ACORN	NaviX	过滤版 Vamana	拼接式 Vamana	Post 过滤	RWalks	UNG	NHQ	iRange 图
SIFT-1M-标签	100s	222s	54s	24s	26s	11s	122s	15s	28s	N/A
ARXIV-2M-标签	3174秒	16016秒	2378秒	3011秒	5599秒	3409秒	19685秒	3779秒	515s	N/A
ARXIV-2M-range	5978秒	5211秒	2081 秒	N/A	N/A	750s	19642 秒	N/A	N/A	7424 秒
LAION-1M 子集	261s	1148 秒	145s	211s	1249 秒	31s	365s	13s	N/A	N/A
LAION-5M-subset	1537s	6960s	872s	1232s	3683s	328s	3293s	89s	N/A	N/A
LAION-25M 子集	8687s	45662s	4981s	9401s	36179s	3209s	17733s	569s	N/A	N/A
YFCC-10M 子集	7898s	22572s	1220s	4515秒	42812秒	300s	3355秒	8453秒	N/A	N/A
MSTuring-10M 子集	3760秒	21742s	1276s	1257s	33376s	500s	4140s	>15h	N/A	N/A
MSTuring-10M-range	3348秒	21322秒	1281秒	N/A	N/A	311s	7228秒	N/A	N/A	6596秒
MSTuring-10M-bool	3760秒	21185秒	1445秒	N/A	N/A	310s	4140秒	N/A	N/A	N/A

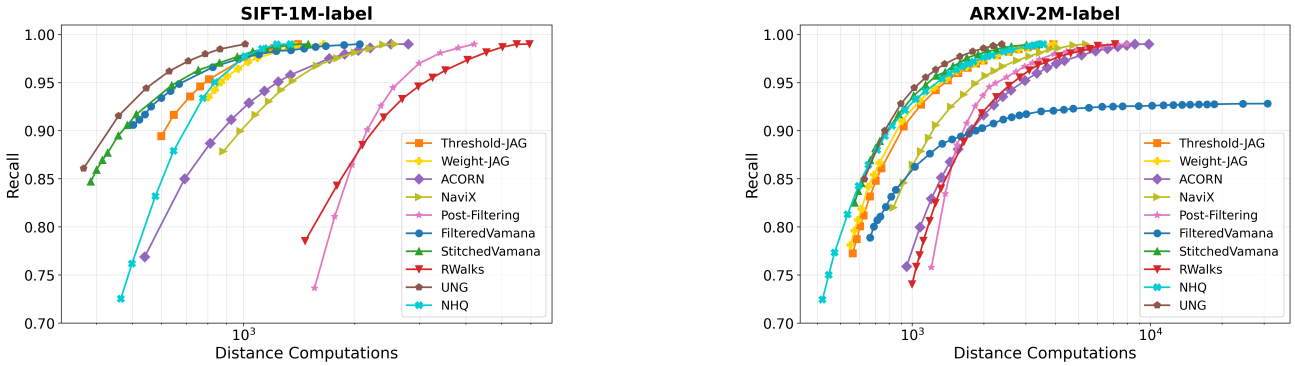


图10。 SIFT 和 ARXIV 数据集上标签过滤器的距离计算与召回率关系图。请注意，NHQ 专门针对标签筛选器设计。FilteredVamana、StitchedVamana 和 UNG 专门针对标签筛选器和子集过滤器设计。

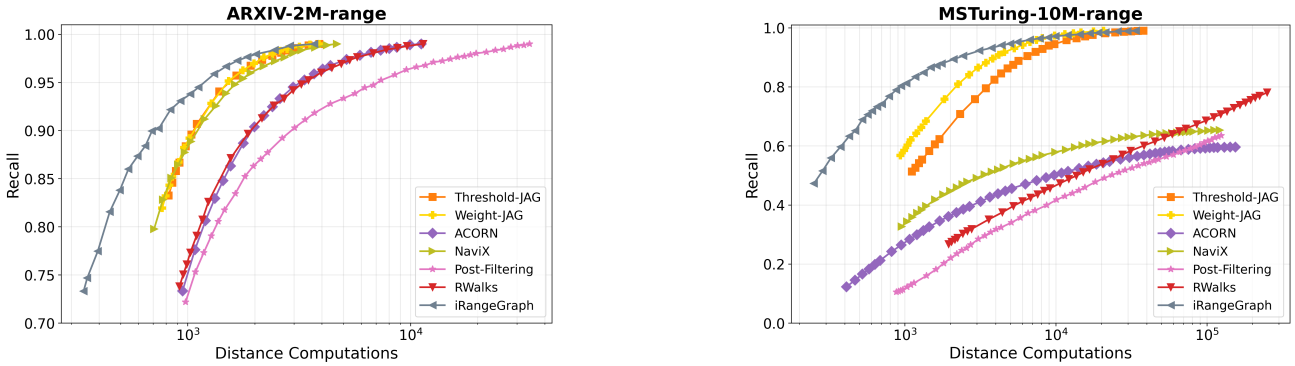


图11。 ARXIV 和 MSTuring 数据集上范围过滤器的距离计算与召回率关系图。请注意，iRangeGraph 专门针对范围过滤器设计。

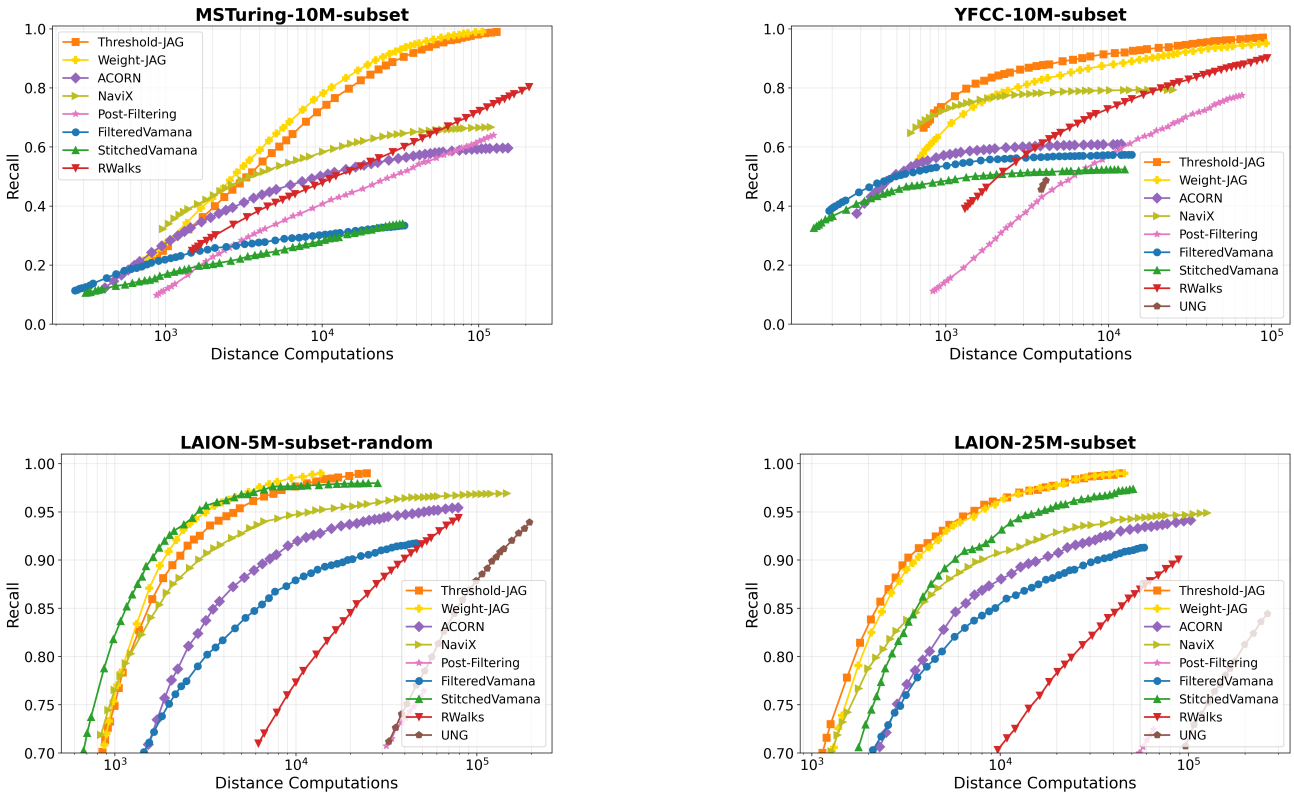


图12. 在 MSTuring-10M、LAION-5M、LAION-25M 和 YFCC 数据集上，子集过滤器的距离计算与召回率曲线。请注意，FilteredVamana、StitchedVamana 和 UNG 专为标签和子集过滤器而设计。

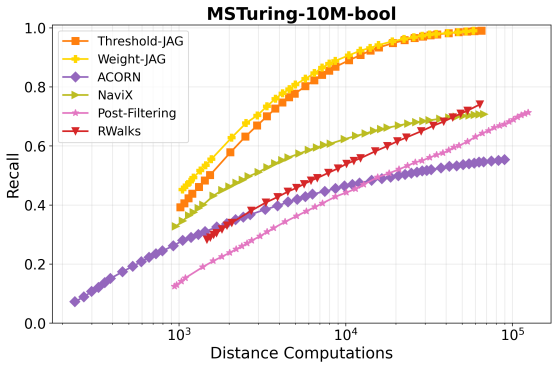


图 13. MSTuring 数据集上，布尔过滤器的距离计算与召回率。