arXiv:2305.02375v2 [cs.DB] 8 Jan 2024

MaskSearch：大规模图像掩码查询

Dong He、Jieyu Zhang、Maureen Daum、Alexander Ratner、Magdalena Balazinska 华盛顿大学，{donghe,jieyuz2,mdaum,ajratner,magda}@cs.washington.edu

摘要

图像数据库上的机器学习任务通常会生成用于标注图像内容的掩码（例如显著性图、分割图、深度图），并支持多种应用（例如，确定模型是否正在学习虚假相关性，或图像是否遭到恶意篡改以误导模型）。根据掩码属性检索示例的查询对从业者很有价值，但现有系统无法高效支持这类查询。本文将该问题形式化，并提出 MaskSearch：一个专注于加速图像掩码数据库查询、同时保证查询结果正确性的系统。MaskSearch 利用一种新颖的索引技术和高效的过滤验证查询执行框架。对原型系统的实验表明，MaskSearch 使用约为压缩数据大小 5% 的索引，可将单条查询加速最多两个数量级，并且在模拟数据集探索和分析流程的各种多查询工作负载中，始终优于现有方法。

1 引言

图像数据库上的许多机器学习（ML）任务通常会生成掩码，用于标注图像中的各个像素。例如，模型解释技术 [52–55, 64] 会生成显著性图，以突出显示各个像素对模型输出的重要性。在图像分割任务 [26, 35, 48],中，掩码表示像素属于特定类别或实例的概率。深度估计模型 [6, 44] 会生成反映每个像素深度的掩码，而人体姿态估计模型 [11, 22]则会提供掩码，指示像素对应身体关节的概率。图 1 展示了一些示例。

探索这些掩码的属性能够解锁大量应用。例如，在模型解释领域，检查显著性图是了解模型是否依赖输入数据中的虚假相关性（即偏离领域知识的信号）的最常见方法 [8, 18, 41, 42, 46, 59]。其他基于掩码属性的应用还包括使用显著性图识别恶意攻击样本 [。

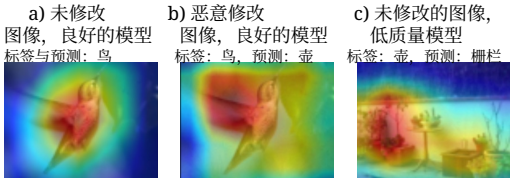
58, 62, 63],分布外检测也使用显著性图 [29], 监控模型错误 [1, 2, 34] 使用分割掩码，以及使用分割掩码进行交通监控和零售分析 [16, 17],等。

这些广泛的应用凸显了机器学习从业者日益迫切的需求：能够根据图像掩码的属性，高效地从图像数据库中查询和检索图像示例及其对应的掩码 [15, 35, 46]。目前，机器学习从业者尚缺乏能够高效、大规模地支持这一任务的系统。

请考虑以下两个受文献启发的场景：场景 ***1*** （受 *[62]* 启发）：*Bob* 是一名负责监控图像分类模型性能的数据工程师。他注意到，过去一周模型的准确率显著下降。为了了解原因，*Bob* 检查了误分类图像的显著性图



(a) 分割掩码 (b) 深度估计掩码 (c) 显著性图 图 1：机器学习任务为 ImageNet [49] 图像生成的图像掩码示例，用于标注图像内容。



*红色（或蓝色）像素表示对模型预测的重要性高（或低）。

图 2：图像掩码示例：叠加了显著性图的 ImageNet [49]图像。b) 和 c) 列中的显著性图表明，模型依赖无关像素进行预测。检索更多具有相似掩码属性的示例，有助于更好地探究模型的行为。

，发现高值像素并未集中在前景物体上，而是分散在无关的背景区域中。图 *2* 展示了三幅叠加显著性图的示例图像。他怀疑，这些误分类可能是由误导模型关注无关像素的恶意修改造成的。*Bob* 希望识别并检索其他高值像素分散在图像大部分区域中的图像。通过分析这些示例，他可以更好地了解恶意修改的影响范围，并努力提升模型抵御此类攻击的能力。

场景 ***2*** （灵感来源于 *[18]*）：*Alice* 是一位科学家，正在开发一种基于胸部 *X* 光图像检测 *COVID-19* 的模型。她训练了一个模型，该模型在公共数据集的训练集和验证集上均取得了很高的准确率。然而，当模型部署到当地医院后，其预测往往与 *PCR* 检测得出的诊断结果相矛盾。为了弄清楚为什么这个高准确率模型在真实环境中表现不佳，*Alice* 检查了模型为训练集中的胸部 *X* 光图像生成的显著性图。她发现，显著性图中的高值像素集中在图像边缘的标记上，而非肺部区域。这一观察表明，模型学到的是图像中的混杂因素（即侧位标记），而不是肺部的医学病理。 *[18]* 中的图 *3* 展示了体现这一现象的 *X* 射线图像及其显著性图。为了进一步探究，*Alice* 希望检索更多具有相似掩码属性的示例。

如上面的示例所示，在机器学习应用中查询掩码数据库非常重要。遗憾的是，目前缺乏能够高效执行这些查询的系统支持 [28]。根据

为了 [46], 识别模型依赖虚假相关性的示例，研究人员不得不手动检查每张图像的解释图。这种繁琐的方法显然难以为继，因此需要一个能够高效支持基于掩码的查询的系统。

针对现有挑战，我们提出了 MaskSearch，这是一种能够根据掩码属性高效检索示例的系统。为了构建 MaskSearch，我们首先形式化定义了一类新颖且适用范围广泛的查询，这类查询依据图像上计算得到的掩码属性，从图像数据库中检索图像（及其掩码）。这些查询的核心是针对图像掩码的谓词，它们对感兴趣区域（ROI）内的像素值应用过滤器和聚合（即像素计数）。我们还进一步扩展了查询，以支持跨掩码聚合和 top-k 计算，从而增强所支持查询的灵活性。跨掩码聚合是比较不同掩码趋势的有力工具，例如研究模型显著性图与人类注意力图之间的差异 [15]。top-k 计算也得到广泛应用。例如，Alice 可能希望找到显著性图中肺部区域高值像素数量最少的 top-k 张 X 射线图像。

高效执行构造出的查询颇具挑战。掩码数据库过大，无法装入内存；从磁盘加载所有掩码速度缓慢，且占据了查询执行时间的大部分；压缩图像也无济于事，因为解压缩会带来额外开销。现有方法和系统也无法高效支持这些查询。例如，使用 NumPy 或 PostgreSQL 加载并处理掩码时，在 ImageNet 上执行一个基于感兴趣区域（ROI）内的像素数量和像素值范围过滤掩码的查询，需要 30 多分钟才能完成（见图 7）。尽管 SciDB [10] 和 TileDB [43] 等数组数据库旨在处理多维稠密数组，但它们并未针对高效搜索大量小型数组集合进行优化，而这正是这些查询所需要的（见图 7）。此外，现有的多维索引技术也无法缩短执行时间，因为掩码是稠密数组。

MaskSearch 通过引入一种新型索引和高效的过滤验证查询执行框架，在不损失查询准确性的前提下加速上述查询。这两种技术协同工作，减少查询执行期间必须从磁盘加载的掩码数量，同时确保查询结果正确。我们将这种索引技术称为累积直方图索引（CHI）；它能够为掩码中感兴趣区域（ROI）内、给定像素值范围的像素计数提供上下界。该索引旨在适用于任意感兴趣区域（包括掩码特定区域和固定区域）以及用户在查询时指定的像素值范围。在查询执行期间，系统利用这些上下界判断是否应从磁盘加载并处理某个掩码，同时确保查询结果正确。

MaskSearch 的查询执行采用了预过滤的思路。利用预过滤技术避免高昂的计算开销或磁盘 I/O，已在许多其他问题中得到探索并证明有效，例如加速相似性连接 [31, 38] 和包含机器学习模型 [3, 30, 33, 37]的查询；在这些情况下，查询执行期间计算相似度函数或运行模型推理的成本很高。MaskSearch 的过滤验证执行框架利用 CHI，跳过加载那些已确定满足或不满足查询谓词的掩码。只有无法被过滤掉的掩码才会从磁盘加载并处理。通过这种方式，MaskSearch 克服了现有系统的局限，减少了处理查询时必须加载的掩码数量。此外，MaskSearch 采用了一种增量索引方法，避免了可能高昂的前期索引成本，并使其能够在线运行。

I总之，　　h本文的贡献 ibuti包括 f thi本文的贡献包括:

- 我们形式化定义了一类新颖且适用范围广泛的查询，可根据掩码的属性从图像数据库中检索图像及其掩码，并进一步扩展这些查询，以支持跨掩码聚合和 top-k 计算（§2）。
- 我们提出了一种新颖的索引技术，以及一个高效的过滤—验证查询执行框架（§3）。
- 我们在一个原型系统 MaskSearch 中实现了这些算法，并证明：对于单个查询，其速度最高可比现有方法快两个数量级；对于各种模拟数据集探索与分析过程的多查询工作负载，它也始终优于现有方法（§4）。

总的来说，MaskSearch 是迈向基于机器学习模型生成的掩码，无缝且快速地探索数据集的重要一步。它也是用于机器学习模型可解释性与调试的方法工具箱中的重要组成部分。

2 掩码查询

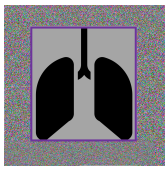
本节将形式化定义 MaskSearch 支持的查询，并讨论高效执行这些查询所面临的挑战。

2.1 数据与查询模型

数据模型。 图像是由像素值构成的二维数组。图像上的掩码同样是由像素值构成的二维数组。不过，掩码中的值仅限于范围 $[0, 1.0)$。图 3 展示了一个玩具示例，说明一张 X 射线图像及其对应掩码。示例展示了一张显著性图，其中数值越高，表示该像素对模型的决策越重要。我们可以用以下概念性关系视图来描述这一数据模型，

MasksDatabaseView（掩码 id INTEGER
PRIMARY KEY, image_id
INTEGER, 模型_id I
NTEGER, 掩码_type
INTEGER, 掩码 REAL[][],...);

其中 mask_id、image_id 和 model_id 分别存储生成该掩码的掩码、图像和模型的唯一标识符。mask_type 是掩码类型的标识符（ENUM 类型），例如显著性图、人体注意力图、分割掩码、深度掩码等。mask 列存储掩码本身。每个掩码都是一个二维浮点数组，数值范围为 $[0, 1)$。其他列可存储其他信息，例如真实标签、预测标签和图像拍摄时间。稍微宽泛地说，上述视图中的一个示例元组可以是 (6，4，ResNet-50，SaliencyMap [[0.9, 0.5, . . .], . . .])，表示使用 ResNet-50 [27] 为图像 #4 计算出的显著性图（掩码 #6）。请注意，mask_id 与 image_id 没有直接关系，因为一张图像可以有多个掩码，也可以没有掩码。



(a) 示例图像

0.1	0.2	0.4	0.1	0.1
0.4	0.8	0.5	0.1	0.1
0.5	0.9	0.5	0.1	0.1
0.1	0.9	0.6	0.1	0.1
0.3	0.3	0.5	0.1	0.1

(b) 掩码

图 3：一张受 [18] 及其掩码启发的示例图像。紫色框表示感兴趣区域（ROI）。针对掩码的谓词通常涉及统计感兴趣区域内取值处于某个范围内的像素数量，例如，感兴趣区域内取值位于 # 像素中的 (0.85,1.0) 的像素数量为 2。

感兴趣区域（ROI）是一个边界框，b，可以由用户指定，也可以通过查询计算得出。图 3 展示了一个由用户指定的感兴趣区域，它对应于图像中包含肺部的部分。感兴趣区域取决于查询，因此不会包含在 MasksDatabaseView 中，而是在查询执行期间计算得出。

基本查询。 MaskSearch 旨在支持指定以下内容的查询：(1) 图像中的感兴趣区域（例如，用户预计肺部所在的位置），(2) 掩码中像素值的过滤谓词（例如，所有高于阈值、表示重要性的像素值），以及对满足谓词的这些像素进行聚合（即像素计数）。对掩码的查询可用以下 SQL 查询表示，其中 opt. 表示某个子句是可选的，而 roi等概念将在下文详细说明，

```
SELECT *, CP(mask, roi, (lv, uv)) as val
来自 MasksDatabaseView
WHERE  <按 CP(...)> [AND  |OR] ... 进行过滤 -- 可选。ORDER BY
val [ASC  | DESC] [LIMITK] -- 可选。
```

感兴趣区域（ROI）。 ROI（roi）是一个边界框，由两对坐标表示，分别对应框的左上角和右下角。它可以对所有掩码都相同，也可以因掩码而异，例如每张图像中前景对象的边界框。ROI 由用户在查询时指定，或从与 MasksDatabaseView 连接的另一张表中获取。

覆盖概率 函数。 查询的核心是 CP 函数。它以掩码、ROI、下界（lv）和上界（uv）作为输入，并返回该掩码 ROI 中值处于 [lv,uv) 范围内的像素数量。CP 的形式化定义如下：

$$\texttt{CP}(mask, roi, (lv, uv)) = \sum_{(x,y)\in roi} \mathbb{1}_{lv \le mask[x][y] < uv}$$

其中，$\mathbb{1}$condition 是一个示性函数：条件为真时取 1，否则取 0。请注意，CP 的输出是一个标量值，可以对其进行算术运算。在我们的查询中，CP 通常出现在过滤谓词中，例如 CP($mask, roi, (lv, uv)$) $> T$；也会出现在 ORDER BY 子句中，例如 ORDER BY CP($mask, roi, (lv, uv)$) ASC。一个查询中可以使用多个 CP 函数，例如指定多个 ROI，或计算不同像素值范围内像素数量的多个比率。

示例 ***1***：考虑 *§1* 中的场景 *2*。科学家 *Alice* 正在构建一个模型，以 X 射线图像为输入，并将其分类为 *COVID-19* 或非 *COVID-19*。模型部署后运行效果不佳。为了调查问题，*Alice* 想确认模型是否将注意力集中在图像中对应肺部的区域。因此，她编写了一个查询，用于计算其中显著（即重要，或值为 > 0.85）像素的数量她将对应肺部的感兴趣区域手动指定为边界框，感兴趣区域[1]。她检索出显著像素数量少于 *10,000* 的所有图像，

```
SELECT image_id FROM MasksDatabaseView
其中 CP(掩码, 感兴趣区域, (0.85, 1.0)) < 10000;
```

她还可以计算肺部区域内显著像素数量与图像中显著像素总数的比值。她通过以下查询找出比值最低的前 *25* 张图像：

```
SELECT image_id,
    CP(掩码, 感兴趣区域, (0.85, 1.0)) / CP(掩码, -, (0.85,1.0)) AS r
FROM MasksDatabaseView ORDER BY r ASC LIMIT 25;
```

复杂查询。 MaskSearch 还支持对像素计数进行聚合，以及对聚合掩码进行像素计数。这些更完整的查询可以用以下 SQL 表示，

```
SELECT [mask_id  | image_id  | model_id  | ...],
[标量_聚合(覆盖概率(掩码, 感兴趣区域, (lv, uv)))
 | CP(MASK_AGG(掩码), 感兴趣区域, (lv, uv))] 作为聚合值
FROM MasksDatabaseView
WHERE  <对 CP(...) 进行过滤> [AND  |OR] ... -- 可选。GROUPBY [图像_id  |模
型_id  |掩码_类型]                --可选。HAVING  <对聚合值进行过滤> [AND  |
OR] ... -- 可选。ORDERBY聚合值 [ASC  |DESC] [LIMITK]                --可选。
```

标量聚合。 用户可以通过定义 SCALAR_AGG 函数，对同一图像、模型或掩码类型的掩码对应的 CP 函数输出进行聚合。该函数会聚合 CP 函数的输出。MaskSearch 支持 SUM、AVG、MIN 和 MAX 等常用函数，例如，按图像_id 分组，对不同模型生成的掩码应用多个 CP 函数后取平均值。

掩码聚合。 MASK_AGG 用于聚合掩码本身。它是一个用户定义函数，以掩码列表为输入并返回一个掩码：MASK_AGG $\rightarrow$ REAL[][]。MASK_AGG 的一个示例是 交集($m_1 > 0.8$,..., $m_n > 0.8$)，即对阈值处理为 0.8 后的 n 掩码取交集。

示例 ***2***：考虑这样一种情况：第 *1* 节场景 *2* 中的用户 *Alice* 想了解她的模型是否关注 X 射线图像中与人类专家相同的区域。将 roi 设置为完整掩码后，她可以编写下面的查询，其中显著性图具有 掩码_类型 $= 1$，而人类注意力图具有掩码_类型 $= 2$，

```
SELECT image_id, 覆盖概率(交集(掩码 > 0.7), 感兴趣区域, (0.7, 1.0)) AS s
FROM MasksDatabaseView WHERE 掩码_type IN (1, 2) GROUP BY image_i
d ORDER BY s DESC LIMIT 10;
```

2.2 挑战

高效处理上述查询颇具挑战。基线方法是在查询处理之前将掩码从磁盘加载到内存中，但由于磁盘读取带宽达到饱和，这种方法极其缓慢。对 ImageNet 执行单个查询需要超过 30 分钟才能完成（图 7）。另一种方法是存储压缩后的掩码，这虽然减少了从磁盘加载的数据量，却将瓶颈转移到了解压缩上，因此对 ImageNet 执行单个查询仍需约 30 分钟。

现有系统（例如 PostgreSQL）也存在从磁盘加载掩码这一瓶颈。由于掩码数据是稠密的，现有的多维索引技术无法高效支持我们的目标查询。它们需要将每个掩码的像素表示为空间（x，y，像素值）中的一个点，其中 x 和 y 表示像素坐标。在这个空间中，我们的查询是一个正交范围

[1]为便于阅读，我们将感兴趣区域指定为变量 roi。通常情况下，它是一组由四个数字组成的值，用于指定边界框的坐标。

查询，随后按掩码_id进行聚合。目前已知的最佳算法 [12, 13],范围树的查询时间为 $O(k+\log^2 n)$，预处理时间为 $O(n\log^2 n)$。这里， n表示数据集中掩码像素的数量， k表示由 ROI 和 (lv,uv) 定义的长方体中的像素数量。由于掩码数据是稠密的，n极其庞大（例如，ImageNet 中有 650 亿），因此无法实际使用这些索引。数组数据库 [10, 43] 专为处理稠密数组而设计，但它们针对的是对少量大型数组进行复杂计算，而非高效搜索大量数组。虽然它们可以加载所需 ROI 中的特定切片，而无需加载整个数组，但正如我们接下来将要说明的，MaskSearch 对很大一部分掩码完全无需加载任何像素。我们将在 §5 中进一步讨论相关工作。

3 MASKSEARCH

MaskSearch 能够高效地执行针对图像掩码数据库的查询，同时保证查询结果的正确性。如上所述，我们目标查询中的基本操作包括：根据感兴趣区域（ROI）内的像素值过滤掩码，随后执行可选的聚合、排序或 Top-k 计算。执行这些操作时面临的主要挑战在于，掩码数据库规模过大，无法全部装入内存，而扫描、加载和处理所有掩码的过程又十分缓慢。

为了加速此类查询，MaskSearch 引入了一种新型索引，称为累积直方图索引（CHI）（§3.1），以及一种高效的过滤验证查询执行框架（§3.2）。CHI 技术通过维护空间区域与像素值的关键组合所对应的像素计数，为每个掩码建立索引。CHI 构建了一种紧凑的数据结构，能够针对任意感兴趣区域（ROI）和像素值范围，快速计算 CP 函数的上下界。在查询执行过程中，利用这些界限可以高效地过滤掉那些必然无法满足查询谓词或必然能够满足查询谓词的掩码，而无需从磁盘加载这些掩码。该查询执行框架由两个阶段组成：过滤阶段和验证阶段。在过滤阶段，框架利用 CHI 计算 CP 函数的界限，无需从磁盘加载掩码即可将其过滤掉。随后，在验证阶段，框架通过从磁盘加载剩余掩码并应用完整谓词，对其进行验证。该框架能够保证查询结果的正确性，并通过大幅减少必须从磁盘加载的掩码数量，克服查询执行过程中的性能瓶颈。

3.1 累积直方图索引(CHI)

CHI 的主要目标是：**(G1)** 支持任意查询参数 lv 和 uv，用于指定 MaskSearch 事先未知的像素值范围；以及 **(G2)** 支持任意感兴趣区域 roi，并允许在单次查询中指定掩码专属的 roi。由于用户可以在查询时间任意指定 roi，因此 roi 也会事先未知。

核心思想。 MaskSearch 通过构建 CHI 来同时实现这两个目标，CHI 用于维护不同空间位置和像素值组合对应的像素计数。从概念上讲，MaskSearch 针对搜索键（$mask_id$， roi，像素值）构建索引。对于每个搜索键，CHI 都保存满足该条件的像素计数。给定 、 以及由 （， uv）指定的像素值范围，该索引支持以下查询：返回掩码 roi中像素值处于范围（lv， uv）内的像素数量，即 CP（$mask$， roi， （lv， uv））。

为每一种可能的（$mask_id$， roi 像素值）组合建立索引，在空间和时间复杂度方面都不可行，因为每个掩码的可能 roi 数量与掩码中的像素数量呈平方关系，更不用说掩码数量和像素值数量了。

相反，CHI 构建了一种能够高效提供谓词上下界的数据结构，而不是提供精确值。这种方法在有效剪枝那些必然无法满足谓词或必然满足谓词的掩码的同时，还能生成较小的索引。随后，只需从磁盘加载少量掩码并对其进行完整处理，即可验证谓词。

CHI 详情。CHI 利用了两个关键思想：离散化和累积计数。离散化可减少索引中的信息总量，而累积计数则能实现极高效的查找。下面分别介绍这两者。

为了构建小型索引，MaskSearch 将掩码划分为互不相交的区域，并将像素值离散化为互不相交的区间。随后，它针对（$mask_id$，区域，像素值区间）的组合建立索引。对于空间维度，MaskSearch 将每个掩码划分为由网格单元组成的网格，每个单元格的大小为 $w_c \times h_c$ 个像素。对于像素值维度，MaskSearch 将像素值离散化为 b 个桶（箱）。MaskSearch 可以使用等宽桶或等深桶。我们当前的原型使用等宽桶。

离散化之后，有多种实现索引的方案。一种直接的方案是针对搜索键（$mask_id$， cx_id， cy_id， bin_id）建立索引，其中 cx_id、 cy_id 和 bin_id 标识每个网格与像素值范围的唯一组合所对应的坐标（例如， cx_id 为 3 时，对应从像素 $w_c * 3$ 开始的网格单元； cy_id 和 bin_id 的情况也类似）。对于每个这样的键，索引可以存储掩码中坐标位于由（cx_id， cy_id）标识的单元格内且像素值位于区间 $[p_{min} + bin_id \cdot \Delta, p_{min} +$ （$bin_id + 1$）$\cdot \Delta$）内的像素数量，其中 p_{min} 是所有掩码中的最低像素值， Δ 是每个桶的宽度。采用此方案时，MaskSearch 需要找出与 roi 相交的所有单元格，以及与（lv， uv）相交的所有桶，并对这些单元格和桶中的像素计数执行我们的查询执行技术（见 §3.2）。我们采用了一种更高效的方案：针对搜索键（$mask_id$， cx_id， cy_id， bin_id）建立索引，但对于每个键，存储掩码中像素值位于区间 $[p_{min} + bin_id \cdot \Delta, p_{max}]$ 且坐标位于区域 $((1,1),(cx_id \cdot w_c, cy_id \cdot h_c))$ 内的像素计数的反向累积和。此索引记作 $H(mask_id, cx_id, cy_id, bin_id)$。我们还用 $H(mask_id, cx_id, cy_id)$ 表示所有桶的累积和数组，即

$$H(mask_id, cx_id, cy_id, [bin_id]) = (,,)$$

$H(mask_id, cx_id, cy_id, bin_id)$ $(,,,)$。回想一下，能够唯一标识 $mask$。该索引可形式化表示为：

$$H(mask_id, cx_id, cy_id, bin_id) = \mathrm{CP}(mask, ((1,1),(cx_id \cdot w_c, cy_id \cdot h_c)), (p_{min} + bin_id \cdot \Delta, p_{max})) \quad (1)$$

示例：在图4中，$MaskSearch$ 为示例掩码 M 构建 CHI，其中包含 $w_c = 2$、 $h_c = 2$ 和 $b = 2$。因此，每个以浅蓝色突出显示的单元格 (x_c, y_c) 都标示出一个离散化区域的角。

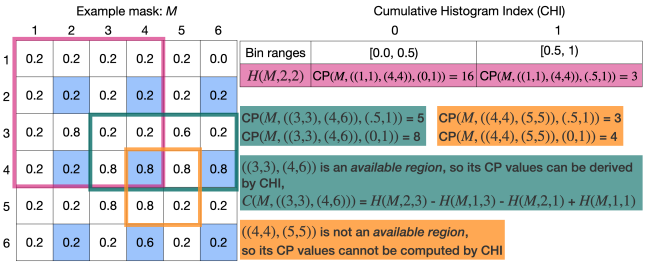


图4：CHI、覆盖概率、可用区域和 C的示例。

使用 $b=2$，像素值范围被离散化为 b 个区间，$[0,0.5)$ 和 $[0.5,1)$。*MaskSearch* 为每个角单元格构建 $H(M,x_c/w_c,y_c/h_c)$。例如，对于单元格 *(2, 2)*，有 $H(M,1,1)[0]=4$（四个像素都位于 (p_{min},p_{max}) 中），以及 $H(M,1,1)[1]=0$（没有像素位于 *0.5* 至 p_{max} 的范围内）。对于单元格 *(4, 4)*，$H(M,2,2)=[16,3]$。

本质上，$H(mask_id,cx_id,cy_id,bin_id)$ 存储的是像素计数的累积和，同时考虑空间维度和像素值维度。与存储原始值相比，存储累积和具有更高的效率，因为它只需执行简单的算术运算，即可快速计算特定范围内的像素计数，同时涵盖空间维度和像素值维度，而无需访问目标空间区域内所有单元格中目标像素值范围内的全部数据箱。为了说明这一点，我们首先介绍可用区域的概念。

定义 *3.1.* 令 X_c表示 $\{x_c|x_c \in [w_c,2w_c,3w_c\ldots,w]\}$，$Y_c$表示 $\{y_c|y_c y_1 \in x_2 [h_c,2h_c,3h_c,\ldots,h]\}$。若区域 ((x_2 y_2 $\in$ $X_c\times Y_c$ x_1-1 y_1-1 $\in$ $X_c\cup 0$ $\times$ $Y_c\cup 0$,) (,)) 满足 (,) 和 (,) ({ }) ({ })，则该区域在掩码的 CHI 中可用。

示例：图*4*中的可用区域是从蓝色单元格的右下角开始、到蓝色单元格的右下角结束的边界框[2]，例如，((3, 3), (4,6)) 是一个可用区域，以深绿色边界框突出显示；((4,4), (5, 5)) 不是可用区域，以橙色边界框突出显示。

在查询执行期间，利用可用区域 内的像素计数，计算任意感兴趣区域和像素值范围对应的 CP 函数界限（§3.2）。在介绍这些界限之前，我们先说明如何计算可用区域 内像素值落在两个分箱边界范围内的像素计数。MaskSearch 执行以下步骤：(1) 使用索引值计算指定区域的反向累积和；(2) 通过相减相关的累积和，计算两个分箱边界之间的像素计数。详情如下。

令 $C(mask_id,r)$ 表示掩码 $mask_id$ 中区域 r 的反向累积像素计数直方图，其中 $C(mask_id,r)[i]=\text{CP}(mask,r,(p_{min}+i\Delta,p_{max}))$。对于任意可用区域 $((x_1,y_1),(x_2,y_2))$，MaskSearch 都可以计算 $C(mask_id,((x_1,y_1),(x_2,y_2)))$。为简洁起见，令 M 表示 $mask_id$，则有：

$$\begin{aligned}&C(M,((x_1,y_1),(x_2,y_2)))\\&=H(M,x_2/w_c,y_2/h_c)-H(M,(x_1-1)/w_c,y_2/h_c)\\&-H(M,x_2/w_c,(y_1-1)/h_c)+H(M,(x_1-1)/w_c,(y_1-1)/h_c)\end{aligned}\tag{2}$$

其中，$-$ 和 $+$ 分别表示两个相同大小数组的逐元素减法和加法。公式（2）成立的原因是，$C(mask_id,region)$ 是一个关于互不相交空间划分的（有限）可加函数，因为 $C(mask_id,region)$ 的每个区间都是一个覆盖概率函数。

[2](0，0)，尽管图中未显示，也同样被视为蓝色单元格。

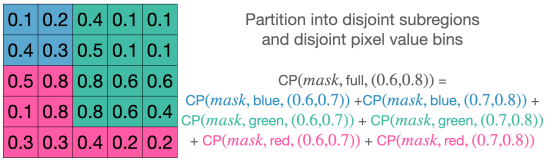


图 5：覆盖概率具有（有限）可加性的示意图。

其为（有限）可加的。图 5 展示了这一可加性的示例。请注意，对于任意 $mask_id$ 和 roi，C roi 0 $(mask_id,)$ $[\lceil p_{max}/\Delta\rceil]$ 始终如此，这只是为了简化符号表示。示例：图*4* 展示了 $C(M,((3,3),(4,6)))$ 的计算方式。

在 MaskSearch 计算出区域 r 中像素计数的反向累积和 C 后，即可通过相减两个分箱的累积和，得到任意两个分箱边界之间（针对像素值离散化）的像素计数。

给定一个谓词 $\text{CP}(mask,roi,(lv,uv))>T$，MaskSearch 使用 CHI 检查该谓词是否成立。从较高层面来看，MaskSearch 在掩码的 CHI 中识别出 可用区域，r_1 和 r_2，其中 r_1 是覆盖 roi 的最小区域，而 r_2 是被 roi覆盖的最大区域。随后，MaskSearch 使用方程（2）计算 $C(mask,r_1)$ 和 $C(mask,r_2)$，并利用它们计算 $\text{CP}(mask,roi,(lv,uv))$ 的上下界。最后，MaskSearch 通过将上下界与 T进行比较，检查 $mask$ 是否必然满足该谓词或必然不满足该谓词。详细内容将在 §3.2 中进一步解释。

由于 $mask_id$、cx_id、cy_id和 bin_id 都是整数，因此我们没有针对这些键构建 B 树索引或哈希索引，而是使用数组创建了一种优化的索引结构，其中 $mask_id$、cx_id、cy_id bin_id 和作为数组中查找操作的偏移量。我们将这种结构称为累积直方图索引（CHI），并将 H（$mask_id$）称为掩码 $mask_id$的 CHI。这种优化的索引结构具有多个优点。首先，它使 MaskSearch 只需存储 CHI 的值，从而避免存储 CHI 键的成本，以及构建 B 树索引或哈希索引的开销。其次，对于任意查找键，查找延迟的时间复杂度都是常数，并且避免了其他索引结构中常见的指针追逐。

计算 N个大小为$w\times h$的掩码的 CHI，时间复杂度为 $O(N\cdot w\cdot h)$；借助 §3.6 中所述的增量索引技术，这项开销可随时间摊销。对于 N 个掩码，MaskSearch 构建的 CHI 数量为 $N\cdot w\cdot h/(w_c\cdot h_c)$。每个 CHI 有 b个分箱，因此占用 $4\cdot b$字节。因此，N个掩码对应的 CHI 集合占用 $4\cdot b\cdot N\cdot w\cdot h/(w_c\cdot h_c)$ 字节的空间。在合理配置 b、w_c和 h_c 的情况下，中等规模数据集的 CHI 可以保存在内存中，MaskSearch 借此能够实现良好的查询性能（见 §4.2）。

3.2 过滤验证查询执行

不失一般性，本节将说明 MaskSearch 如何加速单侧过滤谓词 $\text{CP}(mask,\ roi,\ (lv,\ uv))>T$（记作 P）的执行，因为多个单侧过滤谓词可以组合成一个复杂谓词。在 §3.3 中，我们将说明，我们的技术也适用于加速形如 $\text{CP}(\ldots)<T$ 或涉及多个不同 CP 函数（例如 $\text{CP}(\ldots)<\text{CP}(\ldots)$）的谓词。聚合和 top-$k$查询将在 §3.4 和 §3.5 中分别讨论。

MaskSearch 接收一个过滤谓词 P 作为输入，其目标是查找并返回满足 P 的掩码的 $mask_id$。总体而言，MaskSearch 执行以下工作流程：

- **过滤阶段**：过滤掉 P谓词保证不满足的掩码，并将谓词 P 保证满足的掩码直接加入结果集，然后再从磁盘加载这些掩码到内存。
- **验证阶段**：从磁盘加载剩余未过滤的掩码到内存，并通过应用谓词 P对其进行验证。如果掩码满足 P，则将其加入结果集。

值得注意的是，MaskSearch 能够保证查询结果的正确性，因为它只剪枝那些保证不满足 P 的掩码，并将保证满足P 的掩码直接加入结果集；随后，它会验证所有不确定的掩码，以确保结果正确。

3.2.1 过滤阶段。在较高层次上，对于每个掩码，该算法的工作方式如下：MaskSearch 使用掩码的 CHI 计算 CP($mask, roi, (lv, uv)$) 的上下界，然后利用这些上下界确定该掩码是否满足 P。通过这种方式，MaskSearch 通过剪枝保证不满足 P 的掩码，并将保证满足 P 的掩码直接加入结果集 R，从而减少验证阶段（§3.2.2）期间从磁盘加载的掩码数量。推导 CP($mask, roi, (lv, uv)$) 的上下界颇具挑战，因为 roi和 (lv, uv) 可以是任意的，且无法预先知晓。MaskSearch 通过利用掩码的 CHI 以及 CHI 的（有限）可加性，为任意的 roi和 (lv, uv) 推导上下界，从而应对这一挑战。符号说明。 P表示 CP($mask, roi, (lv, uv)) > T$。 $mask$由 $mask_id$唯一确定。 θ表示 CP($mask, roi, (lv, uv)$) 的实际值。 $\bar{\theta}$和 $\underline{\theta}$分别表示 MaskSearch 计算得到的 θ的上界和下界。$C(mask_id, r)$ 表示掩码 $mask_id$中区域 r的像素值分箱的反向累积像素计数直方图，其中 $C(mask_id, r)[i] = \text{CP}(mask, r, (p_{min} + i\Delta, p_{max}))$。

MaskSearch 启动一个会话时，会将每个掩码的 CHI 从磁盘加载到内存，并在系统运行期间一直保留在内存中。如果 CHI 无法全部保留在内存中，MaskSearch 会在查询执行期间按需从磁盘加载掩码的 CHI。请注意，掩码的 CHI 远小于掩码本身，因此，即使掩码的 CHI 存储在磁盘上，计算界限的开销也远低于将掩码从磁盘加载到内存并在其上计算谓词 P 的开销。

给定谓词 P，MaskSearch 会并行处理过滤谓词所针对的每个掩码。对于每个由 $mask_id$ 唯一标识的 $mask$，MaskSearch 按以下步骤执行：

步骤 1：计算 $\bar{\theta}$和 $\underline{\theta}$。 在此步骤中，MaskSearch 使用 $mask_id$ 的 CHI 计算 $\bar{\theta}$ 和 $\underline{\theta}$。MaskSearch 采用两种方法计算 θ 的两个上界 $\bar{\theta}_1$ 和 $\bar{\theta}_2$，并将较小者用作 $\bar{\theta}$。这两种方法在不同场景下都能有效地产生界限（详情见下文）。

方法 1 首先在 CHI 中确定覆盖 $mask_id$ 的最小 可用区域（定义 3.1）。我们用 roi 表示该区域。然后， $C(mask_id, roi)$ 可通过 CHI 使用公式（2）计算得出。最后， $\bar{\theta}_1$ 的计算方式如下，

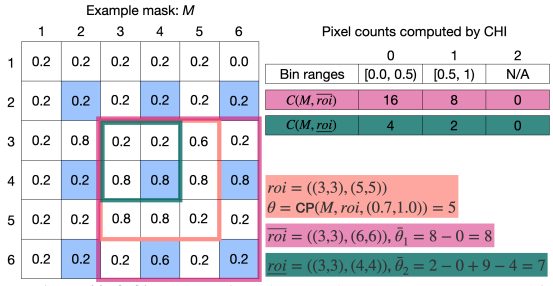


图6：示例：给定掩码 roi 和 (lv, uv)，MaskSearch 计算上界 $\bar{\theta}_1$ 和 $\bar{\theta}_2$。

$$\bar{\theta}_1 = C(mask_id, \overline{roi})[\lfloor lv/\Delta \rfloor] - C(mask_id, \overline{roi})[\lceil uv/\Delta \rceil] \quad (3),$$

其中 $\lfloor x \rfloor$ 和 $\lceil x \rceil$ 分别表示 x的向下取整和向上取整。

方法2首先为每个掩码识别出 CHI 中由 roi 覆盖的最大 可用区域 （定义3.1）。我们用 roi 表示该区域。然后，可以使用公式（2）计算 $C(mask_id, roi)$。最后， $\bar{\theta}_2$ 的计算如下：

$$\bar{\theta}_2 = C(mask_id, \underline{roi})[\lfloor lv/\Delta \rfloor] - C(mask_id, \underline{roi})[\lceil uv/\Delta \rceil] + |roi| - |\underline{roi}| \quad (4)$$

其中，$|\ |$ 表示一个区域的面积 .

最后， $\bar{\theta}$ 取 $\bar{\theta}_1$ 和 $\bar{\theta}_2$ 中的最小值。为证明 $\bar{\theta}$ 是 θ 的上界，我们首先证明以下不等式。由于（$\lfloor lv/\Delta \rfloor * \Delta$， $\lceil uv/\Delta \rceil * \Delta$）是（$lv$， uv）的超集，对于任意 $mask_id$ 和 roi，我们有：

$$C(mask_id, roi)[\lfloor lv/\Delta \rfloor] - C(mask_id, roi)[\lceil uv/\Delta \rceil] \geq \theta \quad (5)$$

现在我们证明以下定理。

定理 3.2。 $\bar{\theta}$ 是 θ的上界。

我们通过证明 $\bar{\theta}_1 \geq \theta$和 $\bar{\theta}_2 \geq \theta$来证明该定理。为简洁起见，在上下文明确时，我们省略 $C(mask_id, ...)$中的 $mask_id$，以及 CP($mask, ...$)中的$mask$；即， $C(Q)$表示$C(mask_id, Q)$， 而 CP($Q, (lv, uv)$)表示 CP($mask, Q, (lv, uv)$)。我们还用 CP($Q \setminus W, (lv, uv)$)表示空间区域 $Q \setminus W$中像素值位于(lv, uv)内的像素数量。

证明。我们首先证明 $\bar{\theta}_1 \geq \theta$。

$$\begin{aligned}
\bar{\theta}_1 &= C(\overline{roi})[\lfloor lv/\Delta \rfloor] - C(\overline{roi})[\lceil uv/\Delta \rceil] && (6)\\
&\geq \text{CP}(\overline{roi}, (lv, uv)) && (7)\\
&= \text{CP}(roi, (lv, uv)) + \text{CP}(\overline{roi} \setminus roi, (lv, uv)) && (8)\\
&\geq \text{CP}(roi, (lv, uv)) = \theta && (9)
\end{aligned}$$

其中，不等式 (7) 由方程 (5) 得出，而方程 (8) 则由 CP 在互不相交的空间区域上具有可加性得出。令 L 表示 $(\lfloor lv/\Delta \rfloor * \Delta, \lceil uv/\Delta \rceil * \Delta)$。现在我们证明 $\bar{\theta}_2 \geq \theta$。

$$\begin{aligned}
\theta &= \text{CP}(roi, (lv, uv)) && (10)\\
&\leq \text{CP}(roi, L) && (11)\\
&= \text{CP}(\underline{roi}, L) + \text{CP}(roi \setminus \underline{roi}, L) && (12)\\
&\leq \text{CP}(\underline{roi}, L) + |roi| - |\underline{roi}| && (13)\\
&= C(\underline{roi})[\lfloor lv/\Delta \rfloor] - C(\underline{roi})[\lceil uv/\Delta \rceil] + |roi| - |\underline{roi}| = \bar{\theta}_2 && (14)
\end{aligned}$$

其中，方程 (12) 是基于覆盖概率在互不相交的空间区域上具有可加性这一事实得出的。不等式 (13) 是因为，任何像素值处于任意范围内的区域，其像素数量都不超过该区域中的像素总数。□

示例：图6通过一个示例掩码展示了这两种方法。掩码数据与图4中的相同。第一种方法识别出 roi，即 ((3, 3), (6, 6))，并计算出 $C(M, roi)$

使用公式（2）。然后，使用公式（3）计算 $\bar{\theta}_1$，即 $C(M, roi)[1] - C(M, roi)[2] = 8 - 0 = 8$。第二种方法识别出 roi，即 $((3, 3), (4, 4))$，并使用公式（2）计算 $C(M, roi)$。然后，使用公式（4）计算 $\bar{\theta}_2$，即 $C(M, roi)[1] - C(M, roi)[2] + |roi| - |roi| = 2 - 0 + 9 - 4 = 7$。

这两种方法在不同场景下都能有效地产生界限。直观而言，当 roi 和 roi 彼此接近时，第一种方法更为有效，因为这会使 $\bar{\theta}_1$ 和 θ 之间的差异较小。当 roi 和 roi 彼此接近时，第二种方法更为有效。

下界 $\underline{\theta}$ 可以按照这两种方法以类似方式计算。由于篇幅限制，此处省略具体细节。

步骤 2：确定 $\bar{\theta}$ 和 $\underline{\theta}$ 以及 T 之间的关系。 在此步骤中，MaskSearch 根据 $\bar{\theta}$ 与 $\underline{\theta}$ 和 T 之间的关系，确定谓词 P 是否由该掩码满足。共有三种情况：

- 情况 1：$\bar{\theta} \leq T$。由于该掩码不可能满足谓词 P，因此将其剪枝。
- 情况 2：$\underline{\theta} > T$。由于该掩码必定满足谓词 P，因此将其直接添加到结果集 R 中。
- 情况 3：$\underline{\theta} \leq T < \bar{\theta}$。由于需要在验证阶段针对 P 对其进行验证，因此将该掩码添加到候选掩码集 S 中。

3.2.2 验证阶段。验证阶段旨在验证 S 中每个既未被剪枝、也未被直接加入结果集的候选掩码。MaskSearch 从磁盘加载该掩码，计算 θ 的实际值，然后评估谓词 P，以确定该掩码是否满足谓词 P。如果掩码满足谓词 P，则将其加入结果集 R。

3.3 通用谓词

到目前为止，我们介绍了 MaskSearch 如何高效处理形式为 $\text{CP}(mask, roi, (lv, uv)) > T$ 的谓词。处理形式为 $\text{CP}(mask, roi, (lv, uv)) < T$ 的谓词，与前一种情况类似。唯一的区别在于过滤阶段的步骤 2 中：如果满足 $\bar{\theta} < T$，MaskSearch 会直接将掩码加入结果集 R；如果满足 $\underline{\theta} \geq T$，则会剪枝该掩码。

MaskSearch 还支持涉及多个 CP 函数的通用谓词，即 $\text{CP}_1(...)\ \text{op}_1\ \text{CP}_2(...) \cdots \text{op}_{n-1}\ \text{CP}_n(...) > T$。令 $F = \text{CP}_1(...)\ \text{op}_1\ \text{CP}_2(...) \cdots \text{op}_{n-1}\ \text{CP}_n(...)$。只要 F 相对于每个 CP_i 函数都是单调的，MaskSearch 就可以利用每个 CP 函数的上下界推导出 F 的上下界，并在 §3.2.1 所述的过滤阶段利用这些界，高效剪枝那些必定不满足谓词或必定满足谓词的掩码。使 F 单调的常见运算符包括 +，−，×。

3.4 聚合

MaskSearch 支持对 CP 函数或掩码聚合上的 CP 函数进行标量聚合的查询，如 §2 所述。对于标量聚合上的过滤谓词，例如 $\text{SUM}(\text{CP}(mask, roi, (lv, uv))) > T$ 按 $image_id$ 分组，MaskSearch 使用与 §3.3 相同的方法，高效过滤掉与相同 $image_id$ 相关联、且已确定无法满足谓词或已确定能够满足谓词的掩码组，因为常见的标量聚合函数（SUM、AVG 等）相对于 CP 函数是单调的

。对于掩码聚合上的过滤谓词，例如 $\text{CP}(\text{MASK_AGG}(mask)\ roi, (lv, uv)) > T$，MaskSearch 将聚合掩码视为新掩码，并使用 §3.2 中描述的相同方法处理查询。聚合掩码的索引可以预先构建，也可以增量构建（§3.6）；这是当前原型的一项限制。不过，当掩码聚合具有单调性时，例如加权求和，MaskSearch 可以轻松扩展，以利用为单个掩码构建的索引，高效过滤聚合掩码。

3.5 Top-K

为了回答 Top-k 查询，MaskSearch 遵循 §3.2 中所述的类似思路，但交织执行过滤和验证阶段，以维护当前的 top-k 结果。不失一般性，考虑一种 top-K 查询，该查询旨在寻找 CP 函数值最高的掩码。Top-k 掩码集合可定义为一个集合 R，由 k 个掩码组成。R 初始为空，并在查询执行过程中，通过识别满足以下条件的下一个掩码 $mask$（及其 $\text{CP}(mask, roi, (lv, uv))$ 值）并将其添加到 R 中，以概念上逐步构建该集合，

$$\text{覆盖概率}(mask, roi, (lv, uv)) > \text{最小值}_{mask' \in R}\ \text{覆盖概率}(mask', roi, (lv, uv)) \quad (15)$$

MaskSearch 依次处理这些掩码，并将 $\bar{\theta}$ 与当前 R 的 CP 值进行比较。如果 $\bar{\theta} \leq \text{最小值}_{mask' \in R} \text{CP}(mask, roi, (lv, uv))$，则会剪枝该掩码，因为它不可能出现在 top-$k$ 结果中；否则，MaskSearch 会从磁盘加载该掩码，并计算 $\text{CP}(mask, roi, (lv, uv))$ 的实际值。然后，如果该掩码满足方程 (15)，它就会将该掩码添加到 R 中，从而更新 R。

3.6 增量索引

正如我们在 §4.2 和 §4.3 中所示，迄今为止介绍的原始 MaskSearch 系统能够在索引大小较小的情况下，相较于基线方法显著缩短查询时间。然而，原始 MaskSearch 所采用的方法在构建索引时，预处理开销可能很高。在处理任何查询之前，原始 MaskSearch 必须为数据库中的每个掩码构建 CHI，这可能导致用户需要等待很长时间才能获得第一个结果。

为应对这一挑战，我们提出在执行查询时逐步构建 CHI，从而只为当前查询所需的掩码建立索引。每当用户发出查询，MaskSearch 按照 §3.2 所述依次处理每个掩码时，都会检查该掩码的 CHI 是否已构建。如果已构建，MaskSearch 就直接按照 §3.2 所述继续处理。如果尚未构建，MaskSearch 会从磁盘加载掩码并判断它们是否满足查询谓词，以此执行查询。对于从磁盘加载的每个掩码，MaskSearch 随后会为其构建 CHI，并将其保存在内存中，以供同一会话中的后续查询使用。MaskSearch 会话结束时，该会话中所有掩码的 CHI 都会持久化到磁盘，以供后续会话使用。采用这种方法，只有在掩码首次从磁盘加载时，才会产生构建该掩码 CHI 的开销，而且仅当该掩码为查询所需时才会产生。在 §4.5 中，我们将展示，采用这种增量索引的 MaskSearch 能够快速摊销索引开销，并且在多查询工作负载上的表现显著优于其他基线方法。

表 1：根据研究动机和相关工作对评估查询进行的总结。

查询描述	
Q1	返回满足 CP（$mask$，roi，（lv，uv）$>5000, roi =$（（50，50），（200，200））， （lv，uv）=（0.6，1.0）的掩码，$model_id = 1$
Q2	返回掩码，使得覆盖概率$(mask, roi, (lv, uv)) > 15{,}000, roi =$对象$, (lv, uv) = (0.8, 1.0), model_id = 1$
Q3	返回 CP 最大的前 25 个掩码$(mask, roi, (lv, uv)), roi = ((50, 50), (200, 200)), (lv, uv) = (0.8, 1.0), model_id = 1$
Q4	返回均值最大的前 25 张图像（覆盖概率（$mask$，roi，（lv，uv））（按 $image$分组）$_id$）与两个模型关联的 $mask$s，$roi =$对象，（lv，uv）=（0.8，1.0）
Q5	返回覆盖概率（交集（$mask$）roi，（lv，uv））最大的前 25 张图像（按 $image_id$ 分组），用于两个模型关联的 $mask$s，$roi =$对象，（lv，uv）=（0.8，1.0）

4 评估

4.1 实验设置

实现。MaskSearch 使用 Python 编写，作为一个库提供。**数据集**。 我们在两组数据集和模型上评估 MaskSearch。第一组数据集和模型称为 *WILDS*，来自 [36]。*WILDS* 包含来自 iWildCam 数据集 [36] 的分布内（ID）和分布外（OOD）验证集中的 22,275 张图像。对于每张图像，我们使用 GradCAM [52] 为从 [36] 获得的两个不同 ResNet-50 [27] 模型生成两张显著性图。每张显著性图包含 448×448 个像素，因为在输入 [36] 中的 ResNet-50 模型之前，*WILDS*图像（尺寸各异）会被调整为 448×448。第二组称为 *ImageNet*，包含 ImageNet 数据集 [49] 中的 1,331,167 张图像。我们同样使用 GradCAM [52] 为每张图像对应的两个不同 ResNet-50 [27] 模型生成两张显著性图，并将其用作 *ImageNet*的掩码。*ImageNet* 中的每个掩码包含 224×224 个像素，因为模型要求输入图像具有这一尺寸 [27]。这两组模型和数据集在图像（及掩码）数量以及掩码尺寸方面互为补充。

MaskSearch 配置。 除非另有说明，我们将b（像素值离散化的桶数）设为 16，适用于 *WILDS* 和 *ImageNet*；然后设置 $w_c = h_c = 64$（空间划分的单元格大小），分别用于 *WILDS* 和 $w_c = h_c = 28$ 用于 *ImageNet*，使两个数据集的未压缩索引大小约为压缩数据集大小的 5%：*ImageNet* 的未压缩索引大小为 6.5 GB，*WILDS* 为 88 MB。§4.4 讨论了更细粒度索引的影响。

基线方法。如 §1 和 §5 所述，目前缺乏针对我们目标查询的高效处理的系统支持。据我们所知，现有系统都无法减少处理查询所需的工作量，即从磁盘加载掩码并计算其覆盖概率函数值。因此，我们将 MaskSearch 与以下三种基线方法进行比较：(1) PostgreSQL 10。按照 §2 所述，掩码以浮点数二维数组的形式存储在一列中。覆盖概率函数通过以 C 编写的用户定义函数（UDF）实现，并编译为动态共享库。调用覆盖概率函数时，PostgreSQL 服务器会加载该函数。(2) TileDB 2.17.1 [43]，搭配 TileDB-Py 0.23.1。掩码以浮点数三维数组的形式存储，第一维是掩码 ID，第二维和第三维分别是掩码的高度和宽度。*WILDS* 和 *ImageNet* 的磁贴大小分别设为 448×448 和 224×224，因为我们发现，与较小的磁贴相比，这些大小能让 TileDB 达到最佳性能。(3) NumPy 1.21.6。掩码以 NumPy 数组的形式存储在磁盘上。覆盖概率函数以 Python 实现，并使用 NumPy 数组函数确保计算向量化。

机器配置。所有实验均在 AWS EC2 p3.2xlarge 实例上运行，该实例配备 Intel Xeon E5-2686 v4 处理器具有 8 个 vCPU 和 61 GiB 内存、配备 16 GiB 内存的 NVIDIA Tesla V100 GPU，以及为磁盘存储配置了 3000 IOPS 和 125 MiB/s 吞吐量的 EBS gp3 卷。我们在单节点环境中评估 MaskSearch，因为如今大多数数据科学家都使用单台机器 [14]。即使在多节点环境中，MaskSearch 仍能减少为回答查询而从磁盘（或通过网络）加载并处理的掩码数量，而这正是查询执行的主要成本。GPU 用于计算掩码。所有评估的方法均使用了全部 vCPU。

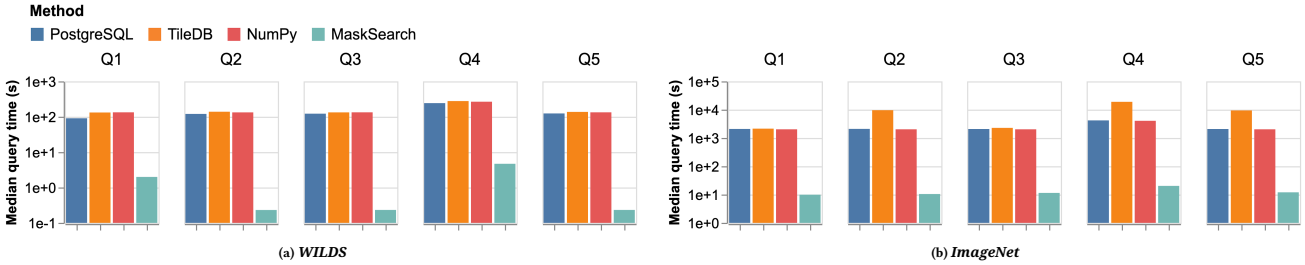
4.2 单个查询的性能

我们首先评估 MaskSearch 在 5 个单独查询上的性能，这些查询的设计灵感来自 §1 和 §2 中的应用场景：

- Q1（过滤，§1 中的场景 2）：使用 CP 上的过滤谓词选择掩码，所有掩码的 roi 均为常数。
- Q2（过滤，Q1 的一种变体）：使用 CP 上的过滤谓词选择掩码，不同掩码的 roi 各不相同。
- Q3（Top-K，§2 中示例 1 的一种变体）：选择 Top-k 个掩码，按 CP 排名，所有掩码的 roi 均为常数。
- Q4（聚合，§2 中示例 2 的一种变体）：对与不同模型相关联的掩码的 CP 值进行聚合，据此选择图像，并使用过滤谓词筛选聚合值。
- Q5（掩码聚合，§2 中的示例 2）：根据与不同模型关联的掩码计算聚合掩码，并依据该聚合掩码的覆盖概率值使用过滤谓词选择图像。

每个查询的具体参数如表 1 所示。对于 top-k 查询，k设置为 25，因为对于科学家而言，这是一个合理的待检查掩码数量。当 roi 设置为 object 时，roi 是由 YOLOv5 [32]生成的图像中前景对象的边界框。我们在执行基准测试查询之前，为所有掩码构建 CHI，并在每次查询执行前清除操作系统页面缓存。图 7 展示了每个查询运行 5 次的中位执行时间。此外，表 2 展示了每个系统在查询执行期间从磁盘加载的掩码数量。

如图 7 所示，在*WILDS*上，PostgreSQL、TileDB 和 NumPy 每次回答查询都需要约 2 分钟；在*ImageNet*上，它们每次回答查询则需要 30 多分钟。对这些查询进行性能分析后发现，从磁盘加载掩码占据了查询执行时间的大部分。所有基线方法都受制于同一性能瓶颈：它们都会从磁盘加载所有掩码，并对其进行处理以生成查询结果。Q4 所需时间明显长于其他查询，这是因为它需要为每张图像加载两个掩码，以便对掩码的 CP 值进行聚合。对于*ImageNet*上的 Q2、Q4 和 Q5，TileDB 的速度比另外两种基线方法慢。这是因为这些查询中的感兴趣区域（ROI）因掩码而异，TileDB 必须依次从磁盘加载掩码（而不是一次从多个掩码中切出相同的 ROI），导致磁盘读取带宽利用率不理想。在 PostgreSQL 和

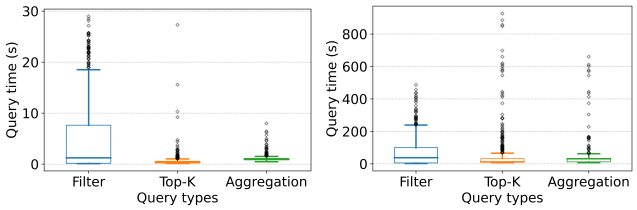


(a) *WILDS* (b) *ImageNet*

图 7：基于动机和相关工作的端到端单个查询执行时间。对于这两个数据集，MaskSearch 的索引大小为原始压缩数据集大小的~ 5%。请注意，y 轴采用对数刻度。

表 2：查询执行期间加载的掩码数量。PG =PostgreSQL，TDB =TileDB，NP =NumPy。

数据集	方法	Q1	Q2	Q3	Q4	Q5
WILDS	MaskSearch	407	40	32	874	48
	PG & TDB & NP	22,275	22,275	22,275	44,550	22,275
ImageNet	MaskSearch	2696	3849	2943	1494	2768
	PG & TDB & NP	1,331,167	1,331,167	1,331,167	662,334	1,331,167



(a) *WILDS* (b) *ImageNet*

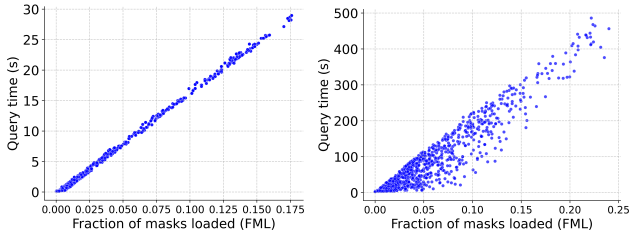
图 8：MaskSearch 在不同查询类型下的查询时间。MaskSearch 的索引大小： ~ 5%**数据集大小。**

NumPy 执行所有查询以及 TileDB 执行其他查询时，我们观察到磁盘读取带宽得到了充分利用，达到 125 MiB/s，即为我们的 EBS 卷配置的磁盘读取带宽。这证实，查询执行时间主要由从磁盘加载掩码所需的时间决定。因此，任何无法减少执行期间从磁盘加载的掩码数量的系统，其查询时间至多只能与 NumPy 和 PostgreSQL 相当。此外，虽然更快的 EBS 卷可以提升基线方法的性能，但 MaskSearch 仍可通过减少查询执行期间的掩码加载量胜过它们。

MaskSearch 在 *WILDS* 上执行每个查询的时间均少于 5 秒，在 *ImageNet* 上则少于 20 秒；与基线方法相比，查询时间最多可提升两个数量级。这一显著的性能差异归因于 MaskSearch 加载的掩码大幅减少（如表 2 所示），因为其过滤-验证框架能够避免从磁盘加载那些确定满足查询谓词或确定无法满足查询谓词的掩码。在 *ImageNet* 上，尽管 Q4 加载的掩码数量更少，MaskSearch 对 Q4 的查询时间仍比其他查询更长。这种差异源于 MaskSearch 针对 Q4 执行了额外的计算（比其他查询进行更多的 2× 边界计算），因为 Q4 包含聚合操作。

4.3 不同查询类型下的性能

在本实验中，我们评估 MaskSearch 对三类具有不同参数的查询的性能。我们仅展示



(a) ***WILDS*****, Pearson 的** $r = 0.99$ **(b)** ***ImageNet*****, Pearson 的** $r = 0.96$

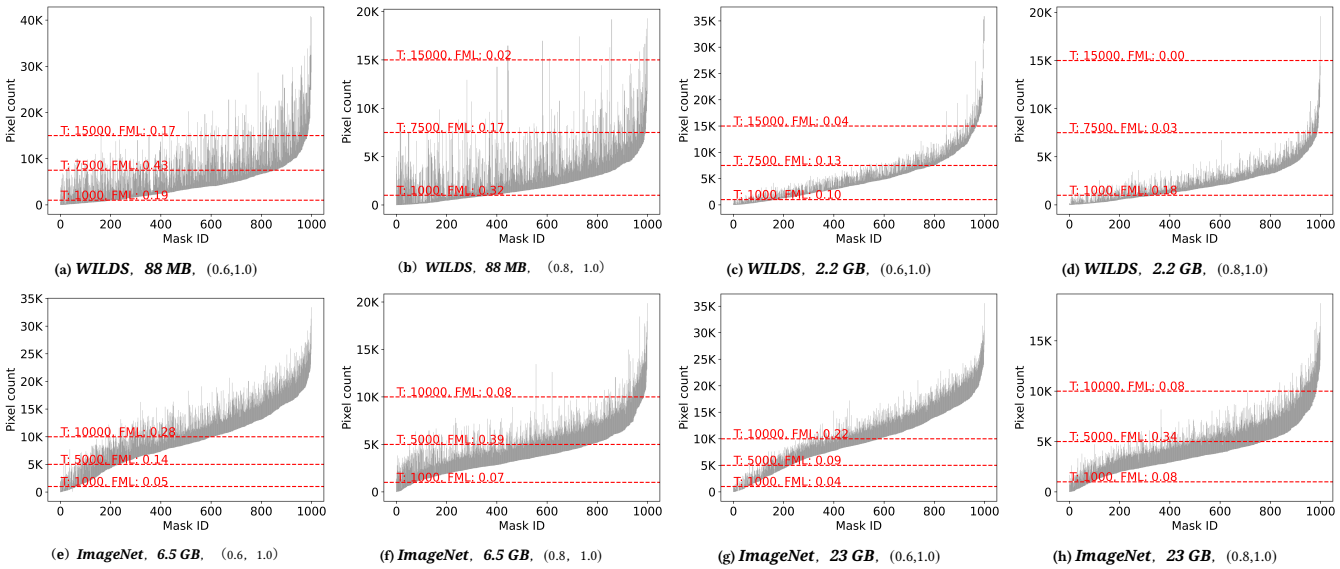
图 9：某个查询的端到端查询时间与掩码加载比例（FML）之间的关系。

MaskSearch 的执行时间，因为对于每种查询类型，无论具体查询参数如何，基线方法的执行时间都与 §4.2 中同类型查询的执行时间相近。对于每个数据集和查询类型，我们使用随机参数生成 500 个查询，并通过 MaskSearch 执行这些查询：

- **过滤**：此查询类型包含带有过滤谓词 CP($mask, roi, (lv, uv)$) $> T$ 的掩码选择查询。对于每个查询，roi被设为 YOLOv 5 [32] 生成的某个掩码中的前景对象边界框。lv和 uv从 $[0.1, ..., 0.9]$ 中随机选取，且 uv始终大于 lv。计数阈值T从 $[0, 1, ..., $总像素数$]$中随机选取。
- **Top-K**：此查询类型返回按 CP($mask, roi, (lv, uv)$) 排序的掩码。对于每个查询，roi被随机生成为掩码内的任意矩形。此 roi 针对每个查询仅生成一次，并在所有掩码中保持不变。k 设为 25。查询结果的排序方式（即 ORDER BY ... DESC 或 ASC）针对每个查询随机选择。
- **聚合**：此类查询按每张图像关联的多个掩码的 mean(CP($mask, roi, (lv, uv)$)) 对图像进行排名并返回。每张图像关联两个掩码，它们由 GradCAM 基于不同的模型生成。k设为 25。每次查询时，随机选择 roi、lv、uv 以及查询结果的顺序。

图 8 展示了在 *WILDS* 和 *ImageNet*上每种查询类型的查询执行时间分布。该图显示了这些时间的中位数、最小值、最大值和四分位距（IQR）。须线表示离群值，即与中位数的距离超过 IQR 的 1.5 倍的值。

MaskSearch 展示了其在所有查询类型和不同查询参数下优异的查询执行性能。即使考虑最坏情况下的执行时间（即异常值），MaskSearch 的表现仍会显著优于基线方法，



(a) ***WILDS***，***88 MB***，(0.6,1.0)　(b) ***WILDS***，***88 MB***，(0.8，1.0)　(c) ***WILDS***，***2.2 GB***，(0.6,1.0)　(d) ***WILDS***，***2.2 GB***，(0.8,1.0)

(e) ***ImageNet***，***6.5 GB***，(0.6，1.0)　(f) ***ImageNet***，***6.5 GB***，(0.8，1.0)　(g) ***ImageNet***，***23 GB***，(0.6,1.0)　(h) ***ImageNet***，***23 GB***，(0.8,1.0)

图 10：由 MaskSearch 计算得到的 CP($mask$, roi, (lv, uv)) 上下界分布。每个子图表示一组（数据集、索引大小、(lv, uv)）组合所对应的分布，该组合显示为各子图的标题。每条竖直线段表示单个掩码的 CP($mask$, roi, (lv, uv)) 上下界。对于每个掩码，roi 是前景对象边界框。我们在每个子图中展示了随机抽取的 1000 个掩码的上下界分布。x 轴表示按下界排序的掩码。水平虚线表示计数阈值 T 的示例。对于谓词 CP($mask$, roi, (lv, uv)) $> T$，FML 表示 MaskSearch 加载的掩码比例。对于每个计数阈值 T，FML 等于与由 T 定义的水平虚线相交的竖直线段比例。请注意，y 轴的刻度范围不同。

因为无论查询参数如何，基线方法仍需从磁盘加载所有掩码并进行处理。

此外，我们发现，MaskSearch 的查询执行时间在不同查询类型之间并未表现出很强的相关性。我们注意到，*Filter* 查询类型的第 75 百分位执行时间比另外两种查询类型更长。这是因为，对于其他查询类型，MaskSearch 会将边界（CP 的边界）与当前 Top-k 集合（k=25）的 CP 值进行比较。与在*Filter* 查询类型中将边界与固定计数阈值 T 进行比较相比，这一过程通常能够更高效地过滤掩码。例如，在*WILDS*上，在查询时间的第 75 百分位处，查询执行期间 MaskSearch 过滤阶段剪枝的掩码数量为：，184（*Filter* 查询类型）、22，106（*Top-K*）以及 21，677（*Aggregation*）。

相反，我们观察到，在同一查询类型中，具有不同参数的查询之间的执行时间差异往往更加显著。事实上，正如我们将在§4.4中进一步讨论的，对于给定数据集，MaskSearch 的查询执行时间主要由掩码加载比例（FML）决定，即在查询执行期间从磁盘加载并用于计算其 CP 值的掩码所占的比例。同一查询类型内执行时间的差异，主要源于每个查询的 FML 不同。例如，对于*Filter* 查询类型在*WILDS*上的查询，FML 在第 25、第 50 和第 75 百分位处分别为 0。002、0。012 和 0。049。

4.4 MaskSearch 的查询时间分析

在本节中，我们通过分析 MaskSearch 在每个数据集上执行的 1500 个 过滤查询（定义见 §4.3），探究影响 MaskSearch 查询执行时间的因素。

结合图 9，我们首先发现，对于给定的数据集，MaskSearch 的查询执行时间与每个查询的掩码加载比例（FML）成正比。查询的 FML 定义为从磁盘加载并用于计算实际 CP 值的掩码数量与数据集中掩码总数之比。查询时间与 FML 之间的皮尔逊相关系数在 *WILDS* 上为 0。99，在 *ImageNet*上为 0。96。这再次证实，查询执行时间主要由从磁盘加载掩码并计算其 CP 值所决定，FML 越高，表示从磁盘加载的掩码越多。

现在我们已经确立了查询执行时间与 FML 之间的关系，接下来探究影响 FML 的因素，包括查询参数（感兴趣区域 roi、像素值范围（lv，uv）、计数阈值 T）、掩码中的数据（$mask$）以及索引粒度（索引大小）。对于 MaskSearch，FML 是指在过滤验证框架中，既未被过滤阶段剪枝，也未被直接加入结果集的掩码所占的比例。FML 对应于过滤阶段步骤 2 中的情况 *3*；对于属于此情况的每个掩码，MaskSearch 计算出的覆盖概率（CP）下界 $\underline{\theta}$不大于计数阈值 T，其覆盖概率（CP）上界 $\bar{\theta}$大于 T，即 $\underline{\theta} \leq T < \bar{\theta}$。

图 10 展示了 MaskSearch 针对两个数据集以及采用不同参数的查询所计算边界的分布，这些查询来自所分析的 1500 个过滤查询。每个子图展示了不同（数据集、索引大小、(lv，uv）组合下的边界分布。所有子图中的 roi 均为前景对象边界框。每个子图中的（垂直）线段表示 MaskSearch 为从数据集中随机

采样的 1000 个掩码计算出的边界。每条红色水平虚线表示一个示例计数阈值 T。这样，每个子图都展示了边界与 FML 之间的关系：对于每个计数阈值 T，FML 等于与由 T 定义的红色虚线相交的线段所占的比例。

在每个子图中，由于与红色虚线相交的线段比例发生变化，不同的计数阈值 T 会导致相同数据集、索引大小和查询参数对应不同的 FML。

比较具有相同 roi 和 $(lv,\ uv)$、但属于不同数据集的子图可知，由于掩码 roi 中的像素值分布不同，不同的掩码集合会使相同查询参数对应不同的 FML。类似地，更改 roi 本质上会改变查询所针对的掩码集合，从而导致不同的 FML。对于具有相同数据集和roi、但采用不同 $(lv,\ uv)$ 配置的子图，即使计数阈值相同 T，其边界分布和 FML 也会有所不同。

此外，具有相同数据集和 $(lv,\ uv)$、但索引大小不同的子图，其边界分布和 FML 也各不相同。较大的索引大小可提供更细粒度的索引、更紧密的边界（图中的垂直线段更短），并使相同查询参数对应的 FML 更低。例如，比较图 10 (a) 和 (c) 可见，对于带有 $(lv,\ uv) = (0.6,\ 1.0)$ 的 $WILDS$，MaskSearch 计算出的边界在索引大小较大时更紧密。因此，对于相同的计数阈值 T，索引大小较大时的 FML 更低。

总之，掩码中的数据、感兴趣区域 roi、像素值范围（lv, uv）以及索引大小决定了 MaskSearch 计算出的边界分布。计数阈值 T根据边界分布定义 FML，而 FML 决定了 MaskSearch 的查询执行时间。索引粒度体现了索引大小与查询时间之间的权衡，具体取决于用户应用的需求和可用资源。

4.5 多查询工作负载性能

本节将评估 MaskSearch 在采用和未采用增量索引技术（§3.6）的多查询工作负载下的表现，该技术可缓解 MaskSearch 潜在的启动开销。我们生成工作负载，以模拟用户的探索和分析过程；这些用户希望找出满足给定谓词的一组掩码。

我们模拟这样的工作负载：用户首先查询属于特定类别的图像子集中的掩码，然后逐步探索与其他类别相关的掩码。例如，为了识别具有虚假相关性（??）的图像，用户可以先查看混淆矩阵，并找出假阳性率较高的类别。随后，用户可以发起查询，检索被预测为这些类别的图像，以识别可能存在的虚假相关性。用户可能会针对这些掩码发起多个查询，因为可以使用不同的查询参数（例如，roi、lv、uv、T）来检索并排列具有不同属性的掩码，例如聚焦于前景对象的掩码和聚焦于背景的掩码。分析返回的掩码后，用户可以继续探索其他类别的掩码，并重复这一过程。

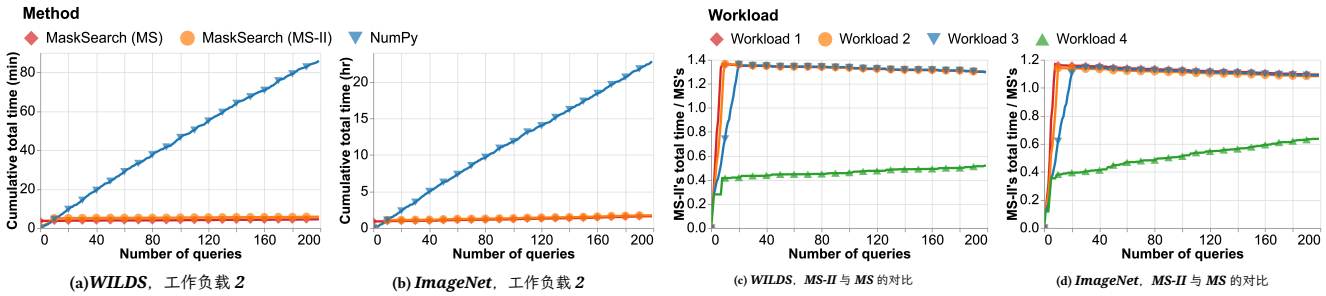
为考虑这种行为，我们针对每个数据集生成四种不同的工作负载，每种工作负载均包含 200 个过滤 查询，其查询参数按照 §4.3 中描述的方法随机生成。每种工作负载都关联一个参数 p_{seen}，用于表示查询此前已定位目标的可能性。同一工作负载中的掩码。随机化的查询参数和 p_{seen} 旨在模拟用户行为，即针对同一组掩码发起多个具有不同参数的查询，以检索具有不同属性的掩码。此外，工作负载中的每个查询都针对相应完整数据集中某个特定子集的掩码（例如，属于被预测为特定类别的图像的掩码）。设 N表示数据集中的掩码总数。每个查询所针对的掩码数量 n从 $[0.1 \cdot N, 0.2 \cdot N, 0.3 \cdot N]$中随机选取。随后，按照以下方式生成目标掩码集合：我们进行无放回抽样，选取 n 个掩码，其中 p_{seen}%为目标掩码，$(1-p_{seen})$%为未见掩码。请注意，当剩余未见掩码的数量少于 $n \cdot (1-p_{seen})$时，我们会将所有未见掩码纳入当前查询，并在该工作负载的剩余查询中仅对已见掩码进行抽样。

这些工作负载分别标记为工作负载 1、2、3 和 4，其对应的 p_{seen} 值依次为 0.2、0.5、0.8 和 1.0。这些概率表示数据集探索程度的不同水平，其中工作负载 1 的探索程度最高，工作负载 4 的最低。通过评估 MaskSearch 在这些不同工作负载上的性能，我们旨在衡量其在各种数据集探索场景下的有效性。

图 11 展示了 MaskSearch 在这四种工作负载下对 $WILDS$ 和$ImageNet$的性能。我们将采用和不采用增量索引的 MaskSearch 与 NumPy 进行比较；NumPy 代表现有方法，每次查询都必须从磁盘加载并处理所有掩码。图中，MS-II 指采用增量索引的 MaskSearch，MS 指未采用增量索引的 MaskSearch。我们测量每种方法累计耗时，即索引构建耗时与查询执行耗时之和。图 11 展示了结果。请注意，在所有子图中，对于 MS，未采用增量索引时首次构建索引的耗时计入第 0 次查询。

图 11(a) 和 (b) 展示了工作负载 2 的累计总耗时。由于 MS 和 NumPy 在所有工作负载上的性能趋势相似，因此未展示其他工作负载的结果。MS 的累计总耗时增长缓慢，因为它使用过滤验证查询处理框架高效地执行所有查询。不过，由于需要提前为数据集中的所有掩码构建索引，因此会产生启动开销。相比之下，NumPy 没有启动开销，但由于不会减少每个查询所需的工作量，其累计耗时迅速增长。尽管如此，得益于过滤验证查询处理框架和 CHI 技术，MS 的索引构建成本很快就能通过多个查询得到摊销。在两个数据集上，MS 在大约执行 10 个查询后便优于 NumPy。MS-II 则在 MS 和 NumPy 之间取得了良好平衡，既消除了启动开销，又实现了与 MS 相当的查询执行时间。

图 11 (c) 和 (d) 展示了两个数据集上所有工作负载中 MS-II 与 MS 的累计总时间比率。我们首先讨论工作负载 1、2 和 3 的结果。对于两个数据集，我们观察到，工作负载 1 、 2 和 3 的该比率在开始时迅速增长，并在大约 10 到 20 个查询后达到峰值，然后逐渐下降。最初快速增长的原因是，对于最初几个查询，MS-II 必须在没有索引帮助的情况下回答针对未见过的掩码的查询，这与.....类似。



(a) ***WILDS***，工作负载 *2*　(b) ***ImageNet***，工作负载 *2*　(c) ***WILDS***，***MS-II*** 与 ***MS*** 的对比　(d) ***ImageNet***，***MS-II*** 与 ***MS*** 的对比

图 11：多查询工作负载的累计总时间，包括索引构建时间和查询时间。MS-II 和 MS 分别指启用和未启用增量索引的 MaskSearch。(a) 和 (b) 展示了工作负载 2 中 MS、MS-II 和 NumPy 的总时间；(c) 和 (d) 展示了所有工作负载中 MS-II 的累计总时间与 MS 的累计总时间之比。MS 的索引大小为对应数据集的 ~ 5% 。MS-II 使用与 MS 相同的索引配置，以增量方式构建索引。

NumPy 的行为，并为这些掩码构建索引。在各个工作负载中，工作负载 1 的这一比率增长最快，因为它的初始值最低 p_{seen}，因此，在最初的几次查询中，目标掩码中包含更多尚未访问过的掩码，从而迫使 MS-II 为更多掩码构建索引。随后，该比率在大约 10 到 20 次查询时达到峰值，因为此时 MS-II 已为数据集中的所有掩码构建了索引，后续查询无需构建索引，即可通过过滤验证框架执行。峰值比率超过 1.0，这是因为在首次将目标掩码纳入查询执行时，MS-II 必须从磁盘加载这些掩码，并计算其 CP 值。相比之下，MS 在所有查询中均使用所有目标掩码的预构建索引，因此累计总耗时更短。达到峰值后，该比率逐渐下降，因为 MS-II 的累计总耗时与 MS 的累计总耗时以相近的速度增长。

对于工作负载 4，在两个数据集上，MS-II 都无法完成为所有掩码构建索引，因为该工作负载中的所有查询最终仅会涉及数据集中 30% 的掩码（WILDS 中为 6683 个，ImageNet 中为 399,351 个）。因此，在初期快速增长后，累计总时间比率趋于平稳。该比率始终不会达到 1.0，因为 MS 为从未被查询涉及的掩码构建索引所花费的时间，无法在各查询之间摊销。

最后，我们注意到，用户通常会在查询之间暂停，以查看结果。因此，MaskSearch 可以利用这段时间计算索引，从而降低用户感知到的延迟。

5 相关工作

机器学习任务中的图像掩码。 掩码在机器学习中被广泛用于标注图像内容，例如显著性图 [52, 54, 55, 64] 和分割图 [26, 35, 48]。从业者将其用于各种应用，包括识别恶意攻击样本 [58, 62, 63]、检测分布外样本 [29]、监控模型错误 [1, 2, 34], 以及执行交通和零售分析 [16, 17]。这些应用推动了 MaskSearch 的设计，并可利用 MaskSearch 高效的查询执行，快速检索满足所需属性的样本。**面向机器学习工作负载和查询的数据系统。** 为了更好地支持机器学习工作负载和查询，已有许多系统被提出 [4, 9, 19, 21, 25, 40, 45, 57, 61]。MaskSearch 与支持机器学习模型检查、解释和调试的系统有关 [24, 39, 51, 56, 60]。其中，DeepEverest [24] 与 MaskSearch 最为接近。它旨在根据神经表示高效检索样本，帮助用户更好地理解神经网络行为。虽然 MaskSearch 同样专注于高效检索样本，但它针对的是基于掩码属性而非神经表示的查询。

图像数据库与查询。许多系统和技术都支持对图像数据库进行高效查询 [5, 7, 20, 47, 50]。然而，这些方法并未针对我们的目标查询进行优化。例如，VDMS [47] 专注于根据元数据检索图像，而 DeepLake [23] 支持基于内容的查询，但不支持基于像素聚合的查询。像 SciDB [10] 这样的数组数据库旨在处理多维稠密数组，但无法高效地搜索大量数组。与 MaskSearch 不同，这些现有系统无法减少执行目标查询所需的工作量。此外，§2.2 中讨论的现有多维索引不适用于掩码这类稠密数据，也无法满足查询中针对掩码的特定感兴趣区域需求。

6 结论

我们介绍了 MaskSearch，这是一种能够加速基于掩码属性检索示例的查询的系统。通过采用一种新颖的索引技术和高效的过滤验证执行框架，MaskSearch 大幅减少了查询执行期间必须从磁盘加载的掩码数量。MaskSearch 的体积约为数据集大小的 5%，却能将单个查询的速度提升两个数量级，并且在各种多查询工作负载上始终优于现有方法。

参考文献

[1] [无日期。]。《Meerkat 与通往基础模型之路：作为可靠的软件抽象》。https://hazyresearch.stanford.edu/blog/2023-03-01-meerkat。访问日期：2023-04-20。[2] [无日期。]。《特斯拉的数据引擎，以及我们应该从中学到什么》。https://www.braincreators.com/insights/teslas-data-engine-and-what-we-should-all-learn-from-it。访问日期：2023-04-20。[3] Michael R. Anderson、Michael J. Cafarella、Germán Ros 和 Thomas F. Wenisch。2018。基于物理表示的视觉分析数据库谓词优化。CoRR abs/1806.04226（2018）。arXiv:1806.04226 http://arxiv.org/abs/1806.04226[4] Yuki Asada、Victor Fu、Apurva Gandhi、Advitya Gemawat、Lihao Zhang、Dong He、Vivek Gupta、Ehi Nosakhare、Dalitso Banda、Rathijit Sen 和 Matteo Interlandi。2022。共享张量秘诀：数据库如何利用机器学习生态系统。Proc. VLDB Endow. 15, 12（2022 年 8 月），3598–3601。https://doi.org/10.14778/3554821.3554853[5] Doug Beaver、Sanjeev Kumar、Harry C Li、Jason Sobel、Peter Vajgel 等。2010。在大海捞针：Facebook 的照片存储系统。载于 OSDI，第 10 卷。1–8。[6] Deblina Bhattacharjee、Martin Everaert、Mathieu Salzmann 和 Sabine Süsstrunk。2022。估计漫画领域中的图像深度。载于 IEEE/CVF 冬季计算机视觉应用会议论文集（WACV）。2070–2079。[7] Avinash N Bhute 和 BB Meshram。2014。基于内容的图像索引与检索。arXiv 预印本 arXiv:1401.1742（2014）。[8] Alceu Bissoto、Michel Fornaciali、Eduardo Valle 和 Sandra Avila。2019。（解）构建皮肤病变数据集中的偏差。载于 IEEE/CVF 计算机视觉与模式识别会议研讨会论文集。0–0。[9] Matthias Boehm、Iulian Antonov、Sebastian Baunsgaard、Mark Dokter、Robert Ginthör、Kevin Innerebner、Florijan Klezin、Stefanie Lindstaedt、Arnab Phani、Benjamin Rath 等。2019。SystemDS：面向端到端数据科学生命周期的声明式机器学习系统。arXiv 预印本 arXiv:1909.02976（2019）。[10] Paul G. Brown。2010。SciDB 概述：大规模数组存储、处理与分析。载于 2010 年 ACM SIGMOD 数据管理国际会议论文集（美国印第安纳州印第安纳波利斯）（SIGMOD '10）。计算机协会，纽约州纽约市，美国，963–968。https://doi.org/10.1145/1807167.1807271 [11] Z. Cao、G. Hidalgo Martinez、T. Simon、S. Wei 和 Y. A. Sheikh。2019。OpenPose：利用部件亲和场进行实时多人二维姿态估计。IEEE 模式分析与机器智能汇刊（2019）。[12] Bernard Chazelle。1990。正交范围搜索的下界：I. 报告情形。J. ACM 37, 2（1990 年 4 月），200–212。https://doi.org/10.1145/77600.77614[13] Bernard Chazelle。1990。正交范围搜索的下界：I. 报告情形。J. ACM 37, 2（1990 年 4 月），200–212。https://doi.org/10.1145/77600.77614[14] Sam Cohan。2021。高效交付机器学习产品：单节点机器学习工作流。https://medium.com/udemy-engineering/delivering-ai-ml-products-efficiently-the-single-node-machine-learning-workflow-bad1389410af[15] Abhishek Das、Harsh Agrawal、C. Lawrence Zitnick、Devi Parikh 和 Dhruv Batra。2016。视觉问答中的人类注意力：人类和深度网络关注的是相同区域吗？arXiv: 1606 03556 [ ]。[16] 2023 . cs.CV DataFromSky。《零售业 - DataFromSky》。https://datafromsky.com/retail/ [17] DataFromSky。2023。《交通监控 - DataFromSky》。https://datafromsky.com/traffic-monitoring/ [18] Alex J DeGrave、Joseph D Janizek 和 Su-In Lee。2021。用于放射影像的 AI 检测 COVID-19 时选择了捷径而非信号。Nature Machine Intelligence（2021）3, 7，610–619。[19] Francesco Del Buono、Matteo Paganelli、Paolo Sottovia、Matteo Interlandi 和 Francesco Guerra。2021。使用 MASQ 将机器学习预测流水线转换为 SQL。载于 2021 年国际数据管理会议论文集（线上活动，中国）（SIGMOD '21）。计算机协会，纽约州纽约市，美国，2696 2700 . 10.1145 3448016.3452771 –。https://doi.org// M. Flickner、[20] H. Sawhney、W. Niblack、J. Ashley、Qian Huang、B. Dom、M. Gorkani、J. Hafner、D. Lee、D. Petkovic、D. Steele 和 P. Yanker。1995。按图像和视频内容查询：QBIC 系统。Computer 28, 9（1995），23–32。https://doi . 10.1109 2.410146 org// Gharib Gharibi、Vijay Walunj、Rakan Alanazi、[21] Sirisha Rella 和 Yugyung Lee。2019。深度学习实验的自动化管理。载于第三届端到端机器学习数据管理国际研讨会论文集。1–4。[22] Rıza Alp Güler、Natalia Neverova 和 Iasonas Kokkinos。2018。DensePose：野外场景中的稠密人体姿态估计。载于 IEEE 计算机视觉与模式识别会议论文集。7297–7306。[23] Sasun Hambardzumyan、Abhinav Tuli、Levon Ghukasyan、Fariz Rahman、Hrant Topchyan、David Isayan、Mark McQuade、Mikayel Harutyunyan、Tatevik Hakobyan、Ivo Stranic 和 Davit Buniatyan。2022。Deep Lake：面向深度学习的湖仓。arXiv:2209.10785 [cs.DC]

[24] Dong He、Maureen Daum、Walter Cai 和 Magdalena Balazinska。2021。Deep-Everest：加速用于深度神经网络解释的声明式 Top-K 查询。《VLDB 纪要》15卷1期（2021年9月），98–111。https://doi.org/10.14778/3485450.3485460[25] Dong He、Supun C Nakandala、Dalitso Banda、Rathijit Sen、Karla Saur、Kwanghyun Park、Carlo Curino、Jesús Camacho-Rodríguez、Konstantinos Karanasos 和 Matteo Interlandi。2022。张量计算运行时上的查询处理。《VLDB 纪要》15卷11期（2022年9月），2811–2825。https://doi.org/10.14778/3551793.3551833[26] Kaiming He、Georgia Gkioxari、Piotr Dollár 和 Ross Girshick。2018。Mask R-CNN。arXiv: 1703 06870 [ ]。cs.CV Kaiming He、Xiangyu Zhang、Shaoqing [27] Ren 和 Jian Sun。2016。用于图像识别的深度残差学习。载于《IEEE计算机视觉与模式识别会议论文集》。770–778。[28] Sungsoo Ray Hong、Jessica Hullman 和 Enrico Bertini。2020。模型可解释性中的人为因素：行业实践、挑战与需求。《ACM人机交互论文集》4卷，CSCW1（2020），1–26。[29] Julia Hornauer 和 Vasileios Belagiannis。2022。基于热图的分布外检测。arXiv:2211.08115 [cs.CV][30] Kevin Hsieh、Ganesh Ananthanarayanan、Peter Bodik、Shivaram Venkataraman、Paramvir Bahl、Matthai Philipose、Phillip B. Gibbons 和 Onur Mutlu。2018。Focus：以低延迟和低成本查询大型视频数据集。载于第13届USENIX操作系统设计与实现研讨会（OSDI 18）。USENIX协会，美国加利福尼亚州卡尔斯巴德，269–286。https://www.usenix.org/conference/osdi18/presentation/hsieh [31] Yu Jiang、Guoliang Li、Jianhua Feng 和 Wen-Syan Li。2014。字符串相似性连接：实验评估。《VLDB纪要》7卷8期（2014年4月），625–636。https://doi . 10.14778 2732296.2732299 org/ / [32] 2020 5 Glenn Jocher。 。Ultralytics 推出的 YOLOv。https://doi.org/10.5281/zenodo.3908559 [33] Daniel Kang、John Emmons、Firas Abuzaid、Peter Bailis 和 Matei Zaharia。2017。NoScope：大规模优化视频上的神经网络查询。《VLDB纪要》10卷11期（2017年8月），1586–1597。https://doi.org/10.14778/3137628.3137664[34] DANIEL KANG、JOHN GUIBAS、PETER BAILIS、TATSUNORI HASHIMOTO、YI SUN 和 MATEI ZAHARIA。 [n. d.]。面向机器学习分析及其他领域的数据管理。([n. d.])。[35] Alexander Kirillov、Eric Mintun、Nikhila Ravi、Hanzi Mao、Chloe Rolland、Laura Gustafson、Tete Xiao、Spencer Whitehead、Alexander C. Berg、Wan-Yen Lo、Piotr Dollár 和 Ross Girshick。2023。Segment Anything。arXiv:2304.02643 [cs.CV][36] Pang Wei Koh、Shiori Sagawa、Henrik Marklund、Sang Michael Xie、Marvin Zhang、Akshay Balsubramani、Weihua Hu、Michihiro Yasunaga、Richard Lanas Phillips、Sara Beery、Jure Leskovec、Anshul Kundaje、Emma Pierson、Sergey Levine、Chelsea Finn 和 Percy Liang。2020。WILDS：分布内外变化基准。CoRR abs/2012.07421（2020）。arXiv:2012.07421 https://arxiv . 2012.07421 org/abs/ Yao Lu、[37] Aakanksha Chowdhery、Srikanth Kandula 和 Surajit Chaudhuri。2018。使用概率谓词加速机器学习推理。载于《2018年国际数据管理会议论文集》（美国德克萨斯州休斯顿）（SIGMOD '18）。计算机协会，美国纽约州纽约，1493 1508 . 10.1145 3183713.3183751 –。https://doi org/ / Willi [38] Mann、Nikolaus Augsten 和 Panagiotis Bouros。2016。集合相似性连接技术的实证评估。《VLDB纪要》9卷9期（2016年5月），636 647 . 10.14778 2947618.2947620 –。https://doi org/ / Parmita Mehta、[39] Stephen Portillo、Magdalena Balazinska 和 Andrew Connolly。2020。迈向深度学习模型诊断的采样方法。载于《2020年IEEE第36届国际数据工程会议（ICDE）》。IEEE，1910–1913。[40] Hui Miao、Ang Li、Larry S Davis 和 Amol Deshpande。2017。Modelhub：深度学习生命周期管理。载于《2017年IEEE第33届国际数据工程会议（ICDE）》。IEEE，1393–1394。[41] Yifei Ming、Hang Yin 和 Yixuan Li。2021。伪相关对分布外检测的影响。arXiv: 2109 05642 [ ]。cs.LG Luke [42] Oakden-Rayner、Jared Dunnmon、Gustavo Carneiro 和 Christopher Ré。2020。隐藏的分层导致医学图像机器学习中具有临床意义的失败。载于《ACM健康、推理与学习会议论文集》。151–159。[43] Stavros Papadopoulos、Kushal Datta、Samuel Madden 和 Timothy Mattson。2016。TileDB数组数据存储管理器。《VLDB学报》10卷4期（2016），349–360。[44] Vaishakh Patil、Christos Sakaridis、Alex Liniger 和 Luc Van Gool。2022。P3Depth：具有分段平面先验的单目深度估计。载于《IEEE/CVF计算机视觉与模式识别会议论文集（CVPR）》。[45] Arnab Phani、Benjamin Rath 和 Matthias Boehm。2021。LIMA：机器学习系统中的细粒度溯源追踪与复用。载于《2021年国际数据管理会议论文集》。1426–1439。[46] Gregory Plumb、Marco Tulio Ribeiro 和 Ameet Tal walkar。2022。利用解释发现并修复伪模式。arXiv: 2106 02112 [ ]。cs.LG Luis Remis [47] 和 Chaunté W. Lacewell。 。使用VDMS为1亿张图像建立索引并进行搜索。《VLDB纪要》14卷12期（2021年7月），3240–3252。https://doi.org/ 2021

10.14778/3476311.3476381[48] Olaf Ronneberger、Philipp Fischer 和 Thomas Brox。2015。U-Net：用于生物医学图像分割的卷积网络。arXiv:1505.04597 [cs.CV][49] Olga Russakovsky、Jia Deng、Hao Su、Jonathan Krause、Sanjeev Satheesh、Sean Ma、Zhiheng Huang、Andrej Karpathy、Aditya Khosla、Michael Bernstein、Alexander C. Berg 和 Li Fei-Fei。2015。ImageNet大规模视觉识别挑战。《国际计算机视觉期刊》（*IJCV*）115卷3期（2015），211–252。https://doi.org/10.1007/s11263-015-0816-y [50] Raimondo Schettini、Gianluigi Ciocca、Silvia Zuffi、Istituto Tecnologie、Informatiche Multimediali 和 Consilio Ricerche。2001。图像数据库中彩色图像索引与检索方法综述。（2001年2月）。[51] Thibault Sellam、Kevin Lin、Ian Huang、Michelle Yang、Carl Vondrick 和 Eugene Wu。2019。Deepbase：神经网络深度检查。载于《*2019*年国际数据管理会议论文集》。1117–1134。[52] Ramprasaath R. Selvaraju、Michael Cogswell、Abhishek Das、Ramakrishna Vedantam、Devi Parikh 和 Dhruv Batra。2017。Grad-CAM：通过基于梯度的定位对深度网络进行可视化解释。载于《*2017*年*IEEE*国际计算机视觉会议（*ICCV*）》。618–626。https://doi.org/10.1109/ICCV.2017.74[53] Sahil Singla、Eric Wallace、Shi Feng 和 Soheil Feizi。2019。理解高阶损失近似与特征对深度学习解释的影响。arXiv: 1902 00407 [ ] 。[54] cs.LG Daniel Smilkov、Nikhil Thorat、Been Kim、Fernanda Viégas 和 Martin Wattenberg。2017。Smoothgrad：通过添加噪声去除噪声。*arXiv*预印本 *arXiv:1706.03825*（2017）。[55] Mukund Sundararajan、Ankur Taly 和 Qiqi Yan。2017。深度网络的公理化归因。载于《国际机器学习会议》。PMLR，3319–3328。[56] Manasi Vartak、Joana M F. da Trindade、Samuel Madden 和 Matei Zaharia。2018。Mistique：一种用于存储和查询模型中间结果的系统诊断性能。载于*2018*年数据管理国际会议论文集。1285–1300。[57] Manasi Vartak、Harihar Subramanyam、Wei-En Lee、Srinidhi Viswanathan、Saadiyah Husnoo、Samuel Madden 和 Matei Zaharia。2016。ModelDB：一个用于机器学习模型管理的系统。载于人在回路数据分析研讨会。1–3。[58] Shen Wang 和 Yuxin Gong。2022。基于显著性图特征的对抗样本检测。*Applied Intelligence*（2022），1–14。[59] Julia K Winkler、Christine Fink、Ferdinand Toberer、Alexander Enk、Teresa Dein- lein、Rainer Hofmann-Wellenhof、Luc Thomas、Aimilios Lallas、Andreas Blum、Wilhelm Stolz 等。2019。黑色素瘤识别中，手术皮肤标记与深度学习卷积神经网络诊断性能之间的关联。*JAMA Dermatology* 155，10（2019），1135–1141。[60] Weiyuan Wu、Lampros Flokas、Eugene Wu 和 Jiannan Wang。2020。面向查询2.0的投诉驱动训练数据调试。载于*2020*年*ACM SIGMOD*数据管理国际会议。1317–1334。[61] Doris Xin、Litian Ma、Jialin Liu、Stephen Macke、Shuchen Song 和 Aditya Parameswaran。2018。加速人在回路机器学习：挑战与机遇。载于端到端机器学习数据管理第二届研讨会论文集。1–4。[62] Dengpan Ye、Chuanxi Chen、Changrui Liu、Hao Wang 和 Shunzhi Jiang。2020。利用显著性图进行针对对抗攻击的检测防御。arXiv: 2009 02738 [ ] 。cs.LG Chiliang Zhang、Zhimou Yang 和 Zuochang Ye。2018。利用显著性检测对抗扰动。载于信息技术：物联网与智慧城市第六届国际会议论文集。25–30。[64] B. Zhou、A. Khosla、Lapedriza。A.、A. Oliva 和 A. Torralba。2016。学习用于判别性定位的深层特征。*CVPR*（2016）。[63]