arXiv:2601.08976v1 [cs.LG] 13 Jan 2026

# 数据流上的持续公平性

Subhodeep Ghosh
sg2646@njit.edu
NJIT
USA

Zhihui Du
z
hihui.du@njit.edu
NJIT 美国

Angela Bonifati
angela.bonifati@univ-lyon1.fr
里昂第一大学
法国

Manish Kumar
manishk@iitrpr.ac.in
IIT Ropar
印度

David A. Bader
bader@njit.edu
NJIT
USA

Senjuti Basu Roy
senjutib@njit.edu
NJIT
USA

## 摘要

我们研究了在数据流的窗口中实现连续群体公平性的问题。我们提出了一种新颖的公平性模型，可在每个滑动窗口内以更细的粒度（称为块）确保群体公平性。当窗口大小较大时，这种表述尤其有用，因为此时更适合以更细的粒度来实现公平性。在这一框架下，我们解决了两个关键挑战：高效监控每个滑动窗口是否满足块级群体公平性；以及在公平性遭到破坏时，尽可能有效地重新排序当前窗口。为实现实时监控，我们设计了基于草图的数据结构，以极低的开销维护属性分布。我们还针对重排序任务开发了最优且高效的算法，并提供了严谨的理论保证。我们在四种真实世界流式场景中的评估结果表明，该方法具有实际效果。根据系统参数不同，我们实现了毫秒级处理速度，平均吞吐量约为每秒 30，000 次查询。在某些情况下，数据流重排序算法可将块级群体公平性提高多达 95%；在不同数据集上，平均提高 50–60%。定性研究进一步凸显了块级公平性相较于窗口级公平性的优势。

**ACM 引用格式：**
Subhodeep Ghosh、Zhihui Du、Angela Bonifati、Manish Kumar、David A. Bader 和 Senjuti Basu Roy。2026。数据流上的持续公平性。载于。 ACM，美国纽约州纽约市，14 页。https://doi.org/10.1145/nnnnnnn. nnnnnnn

## 1 引言

许多现实世界中的应用程序天生会生成数据流，而非静态数据集——例如金融市场行情、网络监控和交通管理中的性能指标、服务器日志、内容推荐引擎，以及网络分析和个性化系统中的用户点击流 [3, 14]。由于数据流具有连续且不断演变的特性，对其进行查询需要使用连续查询，即持续执行并随着新数据到达而返回更新结果的查询。流处理中的一个基本概念是滑动窗口，它能够对实时连续流动的数据进行分析。

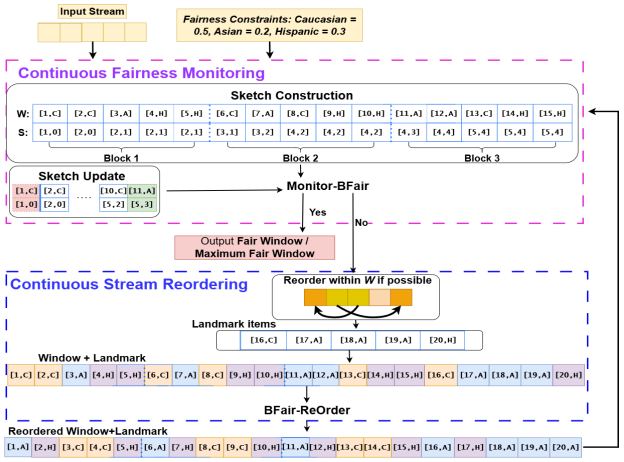



**图1：所提框架由两个紧密耦合的组件构成。对于第一个窗口，它会构建一个紧凑的公平性草图 来汇总块级计数；对于每个后续窗口，该草图都会随着窗口滑动而增量更新。该草图用于执行连续公平性监控，从而能够高效验证每个窗口内的块级群体公平性约束。当窗口违反公平性时，框架会首先尝试连续流重排序，仅使用当前窗口中的项目。如果不存在可行的重排序方案，它就会读取一组额外的地标项，并对窗口和这些地标项共同进行重排序。重排序的目标是最大化当前窗口以及由地标项引发的额外窗口中的唯一公平块数量。**

与此同时，算法决策中的公平性 [7]已成为数据管理中的关键议题，尤其是在招聘、借贷和执法等高风险领域 [13, 22, 30, 32, 34] 。

大量研究聚焦于群体公平性，旨在确保算法产生的结果不会系统性地使特定人口群体处于不利地位。然而，这方面的大多数研究都假设可以访问静态数据集，并且只在固定输入上施加一次公平性约束。尽管这些流式场景在现实应用中广泛存在，但对流式场景中公平性的研究仍远未充分。

允许免费制作本作品的全部或部分数字版或纸质版副本，供个人或课堂使用，但不得为营利或商业利益而制作或分发副本，且所有副本必须附有本声明和首页上的完整引文。本作品中归他人而非作者所有的组件，其版权必须得到尊重。注明出处后，允许摘录。若要以其他方式复制、再版、发布到服务器或分发至列表，则须事先获得明确许可和/或支付费用。请向 permissions@acm.org 申请许可。*Conference'17，2017*年7月，美国华盛顿特区 © 2026 版权归所有者/作者所有。出版权已授权给 ACM。ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

**应用动机——在线内容推荐**

在新闻或视频流媒体平台中，系统会实时持续推荐内容，项目由不同族群（如白人、亚洲人、西班牙裔）的创作者提供。系统以分页形式展示这些内容，每个推荐滑动窗口都分布在多个页面上。

即使整体窗口包含来自所有群体的均衡内容，页面级分布仍可能出现偏斜——例如，前几页可能主要展示白人创作者，而后续页面的代表性则更加均衡或多元。

由于用户的注意力通常会随着页面递减而迅速下降——大多数互动集中在前一两页——这种排序偏差会导致可见性和曝光度不均。因此，仅确保整个窗口内的比例代表性并不足够；还必须在页面内部及页面之间落实公平性，同时考虑内容的呈现顺序和注意力衰减模式。

**贡献**。 本文首次研究了流式数据上的群体公平性。我们提出了一种新颖的公平性模型，将每个滑动窗口划分为多个块，并在块级别落实群体公平性，以满足不同受保护属性值的比例要求（图1包含三个这样的块）。这些块由应用程序（例如，页面）自然定义，为传统公平性概念提供了一种局部且灵活的替代方案；传统概念仅在窗口级别或基于前缀运作。当应用程序将每个窗口限制为单个块时，我们的公式自然会简化为经典的窗口级群体公平性模型。

为支持这一框架，我们研究两个相互关联的问题：**连续公平性监控**和**连续流重排序问题**。监控模块会持续检查每个滑动窗口中的各个块是否满足公平性约束（见图1中的连续公平性监控）。检测到违规时，系统会首先尝试对当前窗口中的项目重新排序，以恢复公平性。如果仅靠重排序无法解决问题，决策过程便会暂停，并读取少量额外项目——称为地标位——从而有效延长当前数据流并创建新的滑动窗口。从概念上说，这些地标代表应用程序在输出结果前愿意容忍的有限"缓冲时间"（见图1中的连续流重排序）。随后，框架会对当前窗口和地标位一并进行重排序。我们将此任务形式化为连续流重排序问题，旨在最大化当前窗口和新生成的滑动窗口中不同公平块的数量。重排序完成后，**连续公平性监控**会输出公平窗口。

我们定义了一种**基于草图的**数据结构，用于对每个窗口进行预处理，并提出 Monitor-BFair 算法来回答连续公平性监控问题。该草图紧凑地编码滑动窗口中每个块的受保护属性分布，从而能够精确、高效地进行公平性评估。该草图仅需线性空间，支持快速增量更新，并避免在窗口滑动时进行完全重新计算。我们为其正确性、运行时间、更新效率和空间复杂度提供了理论保证，展示了其在高吞吐量数据流中的可扩展性和低延迟性能。

我们针对**高效且最优的**算法 BFair-ReOrder，解决**连续流重排序**问题，该算法结合了 - 当前窗口和地标项，以最大化唯一公平块的数量（若两个块至少有一个不同的项目，则认为它们是唯一的）。每个公平块都必须满足指定的群体代表性约束；我们的目标不仅是最大化当前窗口中的此类块的数量，还要最大化由地标位置定义的未来窗口中的此类块的数量。该算法的核心思想是构造内容的一种特定排列，称为同构流（详见第 4 节），使其在各块中持续满足公平性要求 - 。我们通过理论分析证明了其最优性，并分析了计算开销。

我们通过模拟流式环境，使用四个大规模真实世界数据集开展了广泛实验。实验结果与理论分析一致，证实了我们方案的最优性。所提框架可在毫秒以内处理查询，平均吞吐量达到每秒 30,000 次查询（视参数而定），同时内存占用极低且更新效率高。由于现有研究均未涉及我们的问题设定，我们设计了适当的基线方法。所提框架的性能比这些方法高出数个数量级。总而言之，本文的贡献如下：

- **块级公平性模型**。 我们提出了一种公平性模型，将每个滑动窗口划分为多个块，以实现局部群体公平曝光。该模型推广了窗口级公平性，并为基于前缀的概念提供了一种灵活的替代方案。
- **连续公平性监控与数据流重排序**。 我们通过基于草图的算法 Monitor-BFair，将连续公平性监控 问题形式化，并提出连续流重排序问题，由我们的最优算法 BFair-ReOrder 求解。该算法构造同构数据流，以最大化唯一公平块的数量。
- **理论保证**。 我们对正确性、最优性、空间复杂度和时间复杂度进行了形式化分析。B-FairQP 保证线性空间复杂度和快速增量更新，而 Block-FairReOrder 已被证明是最优且高效的。
- **实验验证**。 在四个大型真实世界数据集上，我们的框架实现了亚毫秒级运行时间，吞吐量最高可达每秒 30,000 次查询，且内存使用量极低，性能比基线方法高出数个数量级。

## 2 数据模型与框架

我们首先介绍数据模型，随后介绍所提出的框架，并以此形式化所研究的问题。我们先介绍一个基于引言中所讨论的动机性应用程序的示例。

*示例 2.1*。 一个滑动窗口包含 15 个项目，分为 3 个块（页面）——每个项目都包含由具有受保护属性"种族"的个体生成的内容。不失一般性，假设有 3 个不同的族群，如图 2 所示。表 1 展示了两个代表性的公平性约束。

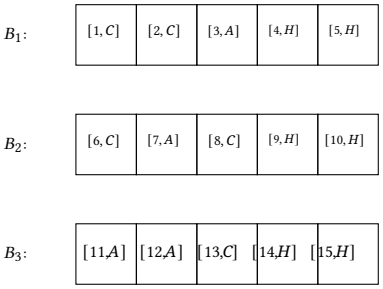



**图 2：由 3 个块组成的包含 15个项目（内容）的滑动窗口，内容来自三个族群 [*C*（白人）、*A*（亚洲人）、*H*（西班牙裔）]。**

| 属性 | 值 | C1 | C2 |
|---|---|---|---|
| 族群 y | *C* | 0.3 | 0.5 |
| | *A* | 0.3 | 0.2 |
| | *H* | 0.4 | 0.3 |

**表1：两个公平性约束。**

## 2.1 数据模型

**表示公平性的属性**。受保护属性 A 有 $\ell$ 个可能的值。根据图 1，种族 就是这类属性之一，有三个可能的值：*C*、*A*、*H*。

**基于计数的滑动窗口**。流处理中，基于计数的滑动窗口会对最近固定数量的数据项进行计算，并按指定数量的数据项向前移动，这一数量称为滑动步长。我们用$W$表示包含 $|W|$ 个项目的滑动窗口。本质上，该窗口会保留数据流中最新的$|W|$ 个元素，并在每当有新项目到达时更新为一个新窗口。随着数据持续流入，我们将第 $y$ 个这样的窗口记为$W_y$。除非另有说明，默认滑动步长为 1。

Let

$$W_y = \{ t_y,\ t_{y+1},\ \ldots,\ t_{y+|W|-1} \}$$

表示第 $y$个滑动窗口，该窗口包含数据流中连续的 $|W|$ 个项目。

例如， $W_1 =\{t_1、\ldots\ldots、t_{|W|}\}$、 $W_2 =\{t_2、\ldots\ldots、t_{|W|+1}\}$，依此类推。

根据示例 2.1，每个滑动窗口 $W$包含 15 个项目。窗口 $W_1$ 包含项目 $t_1$、 $\ldots t_{15}$，而窗口 $W_2$ 包含项目 $t_2$、 $\ldots t_{16}$，依此类推。

**数据项**。滑动窗口中的数据项 $t$ $W$是一个有序对 $t =< i,\ p >$，其中 $i$ 表示其在数据流中的顺序（键），而$p$ 表示受保护属性 A 的任意特定值，即 $p \in \ell$。

以示例 2.1 为例，数据流中的第三个视频由一位种族为亚洲人（*A*）的贡献者创作。

**窗口中的块**。每个窗口$W$ 又进一步划分为$k$ 个大小相等的子区间，称为块，块大小定义为

$$s = \frac{|W|}{k}.$$

窗口$W_y$ 中的每个块 $B_q$ 都恰好包含该窗口中的 $s$ 个连续项目：

$$B_q = [\, t_{y+(q-1)s},\, t_{y+(q-1)s+1},\, \ldots,\, t_{y+qs-1} \,],$$
$$q = 1, 2, \ldots, k.$$

因此，窗口中的每个项目都恰好属于该窗口中的一个块。

以示例 2.1 为例，每个滑动窗口包含 $k = 3$ 个块，每个块有 $15/3 = 5$ 个项目。对于 $W_1$，块 $B_1$、 $B_2$、 $B_3$ 分别从项目 1、6 和 11 开始；对于 $W_2$，块 $B_1$、 $B_2$、 $B_3$ 分别从项目 2、7 和 12 开始。图 2 展示了第一个窗口中的 $\{B_1, B_2, B_3\}$。对于任意项目 $t_j \in W$, 其块索引由下式给出

$$\text{block}(t_j) = \left\lceil \frac{j}{s} \right\rceil.$$

块大小 $s$（或等价地，每个窗口中的块数 $k$）由目标应用程序的语义决定。在数据流或推荐场景中，每个块通常对应于所需公平性或表示平衡的最小粒度，例如显示项目的页面。块粒度为公平性敏感度与计算效率之间的权衡提供了一个自然的调节旋钮。

**块的前缀和后缀**。对于窗口$W$中从索引 $i$开始的块 $B_q$

$$B_q = [t_i, t_{i+1}, \ldots, t_{i+s-1}], \quad \text{for } y \le i \le y + |W| - s.$$

其长度为 $j$ 的前缀为：

$$\text{Prefix}_j(B_q) = [t_i, t_{i+1}, \ldots, t_{i+j-1}], \quad \text{for } 1 \le j \le s.$$

其长度为 $j$ 的后缀为：

$$\text{Suffix}_j(B_q) = [t_{i+s-j}, t_{i+s-j+1}, \ldots, t_{i+s-1}], \quad \text{for } 1 \le j \le s.$$

例如，第一个块 $B_1$的长度为 3 的前缀是项目 $[t_1, t_2, t_3]$，长度为 2 的后缀是项目 $[t_4, t_5]$。

**地标项**。给定当前窗口 $W$ 地标项由紧随窗口存储的 $|X|$ 个额外元素组成，从位置 $|W| + 1$ 开始。图3展示了其中 5 个额外项目。

地标大小由应用程序定义，是另一个可调节的设计参数。在许多流式场景中，等待片刻以获取当前窗口之外的少量额外项目是可以接受的，因为系统通常会采用少量缓冲，在不明显影响延迟的情况下改善公平性或多样性。如果不允许延迟，则立即使用当前窗口进行重排序，以现有数据实现尽可能高的公平性。

**唯一块**。一个块由 $s$ 个连续项目组成，从项目 $t$ 开始，到项目 $t + s - 1$ 结束。不失一般性，如果两个块各自都至少包含一个对方没有的项目，则认为它们是唯一的 。例如，当前窗口中的第一个块与后续（滑动后的）窗口中的第一个块是唯一的，因为滑动后，前者包含项目 $[t_1, \ldots, t_5]$，而后者包含项目 $[t_2, \ldots, t_6]$。

**不相交块**。彼此不重叠的块称为不相交块。如果这些块中的每一个都分别满足给定的公平性标准，则称为不相交公平块。例如，块 $B_1, B_2, B_3$ 完全不相交。

**群体公平性约束**。对于具有 $\ell$ 个不同值的属性 A，每个 $p \in \cup_{i=1}^{l} A_i$, $f(p)$表示其所需的比例约束，使得 $\Sigma_{p=1}^{\ell} f(p) = 1$。

表1展示了针对种族属性的两种不同的群体公平性约束。

**公平滑动窗口**。给定一个滑动窗口$W$以及针对受保护属性 A 的每个值 $p$ 的期望比例函数 $f$，如果对于$W$中的每个块，属性值为 $p$ 的项目数量都在范围 $[\lfloor f \times s \rfloor, \lceil f \times s \rceil]$ 内（对所有 $p$ 均如此），则该窗口被视为公平。否则，该窗口被视为不公平。使用向下取整和向上取整运算，是因为 $f(p) \times s$ 可能不是整数，而这些界限可确保比例被取整为最接近的可行整数值。

根据表1中列出的约束*1*，每个块都需要包含 $[\lfloor .3\times 5 \rfloor, \lceil .3\times 5 \rceil]$ 个值为$C$和$A$的数据项（[1,2] 个数据项）以及 $[\lfloor .4\times 5 \rfloor, \lceil .4\times 5 \rceil]$，即 2 个受保护属性值为$H$的数据项。根据这一要求，图 2 所示的窗口是公平的。

此公平性定义与统计均等或人口统计均等 [5], 的概念一致，并针对项目顺序至关重要的情形作了适当调整。它也可视为 $p$ 公平性 [34], 的一种宽松变体，后者对排序列表的每个前缀都施加比例约束。最后，根据应用程序的需求，可以在定义中加入近似因子 $\epsilon$，使得窗口中的每个块都满足

$$[\lfloor \epsilon f(p) \times s \rfloor, \lceil \epsilon f(p) \times s \rceil] \quad \forall p.$$

## 2.2 所提框架

给定每个滑动窗口和指定的公平性约束后，框架（参见图1）会持续监控各个窗口。首次读取窗口时会构建草图。此后，每出现一个新的滑动步长，都会更新草图。构建或更新草图后，会将其送去进行公平性监控。只有在完成重排序后才会重新构建草图，因为重排序会改变项目的分布。

根据第2.1节中**公平滑动窗口**的定义，如果当前窗口不公平，则窗口中至少有一个块违反公平性约束。由于块级公平性取决于项目的排列方式，因此仅凭项目的排列不同，相同的项目集合也可能形成公平或不公平的窗口。因此，框架首先检查窗口中是否包含足够数量的数据项，使其能够通过重新排序来满足约束。具体做法是统计对应于每个受保护属性值$p$的实例数量，并验证该数量是否达到整个窗口所要求的比例。具体而言，如果满足以下条件：

$$\forall p \in \cup_{i=1}^{l} \mathcal{A}_i, \quad \sum_{i=1}^{|W|} \mathbb{1}[\{t_i(A) = p\} \geq (k \cdot \lfloor f(p) \cdot s \rfloor)]$$

基本上，对于每个属性值 $p \in \cup_{i=1}^{l} A_i$，它会检查$p$在窗口 $W$ 中出现的总次数（即满足 $t_i(A) = p$ 的项目 $i$的数量）是否至少为 $k \cdot \lfloor f(p) \cdot s \rfloor$，其中$f(p)$表示$p$的比例，$s$表示块大小，$k$表示块数。在这种情况下，可以对当前窗口中的项目进行重排序，以满足公平性约束。如果条件成立，就可以轻松地在当前窗口内进行内部重排序。然而，更具挑战性的情况是条件不成立——这正是本文重点探讨的情形。

在后一种情况下，框架会暂停，并额外读取 $|X|$ 个地标项。每个地标项对应一个滑动步长，合计产生 $|X|$ 个额外的滑动窗口。随后，框架会同时考虑当前窗口中的项目$W$以及地标项 X，并进行重排序。

理想情况下，当前窗口和 $|X|$ 个滑动窗口都应满足公平性约束。然而，即使纳入 $|X|$ 个地标项，合并后的数据项集合仍可能缺少某些受保护属性值的足够实例。

在这种情况下，框架会在 $W \cup X$ 内执行特殊排列，旨在最大化当前窗口和 $|X|$ 个滑动窗口中满足公平性约束的窗口总数。

完成排列后，监控过程会恢复；根据需要重复这两个步骤，直到处理完整个数据流。

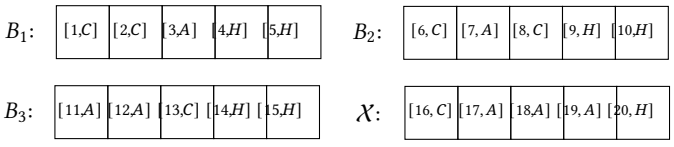



**图3：五个地标项 $\mathcal{X}$**

| 符号 | 说明 |
|---|---|
| $W$ | 宽度为 $\|W\|$ 的窗口 |
| $B$ | 宽度为 $s$的窗口中的块 |
| $S$ | 草图 |
| $k$ | 窗口中的块数 |
| $\mathcal{A}$ | 基数为 $\ell$的受保护属性 |
| $p$ | A 的一个值 |
| $\mathcal{X}$ | 地标项 |
| $\mathcal{A}^i$ | 项目 $i$的受保护属性值 |

**表 2：关键符号**

## 2.3 问题定义

问题 1. **连续公平性监控**对于数据流中的每个滑动窗口$W$以及给定的公平性约束，若窗口中的每个块 $B \in W$ 都满足该约束，则该窗口被视为公平；否则，被视为不公平。

受保护属性值 $p$在一个块中的计数必须落在区间 $\lfloor f(p) \cdot s \rfloor, \lceil f(p) \cdot s \rceil$ 内，该块才被视为相对于 $p$满足公平性，其中 $f(p)$是针对 $p$的公平性约束，$s$是块大小。例如，如果 $p = A$、$f$ $p$ $s = 5$ $p = A$ ( ) $= 0.3$，且，那么一个块中具有 的项目计数必须在范围 $[1, 2]$内。如果每个受保护属性值都满足这一条件，则该块是公平的。当且仅当窗口中的所有块都公平时，该窗口才是公平的。

给定图 2 中的窗口以及表1中的约束*1*，窗口$W$是公平的。然而，对于表1中的约束*2*，窗口 $W$是不公平的。

问题 2。**使用地标项进行连续流重排序** 给定一个窗口$W$和额外的 $|\mathcal{X}|$ 地标项，对于给定的公平性约束，确定数据项的重排序方式，使$W \cup \mathcal{X}$中的唯一公平块总数，以及由地标项生成的滑动窗口中的唯一公平块总数最大化。

根据表1中的约束*2*，图 2 的窗口 $W$ 中所需的最少 C实例数计算为 $\lfloor 0.5 \times 5 \rfloor \times 3 = 6$。由于$W$仅包含 5 个$C$实例，因此读取了另外 5 个用 X 表示的地标项（如图3所示）。这额外的 5 个项目产生了以下 5 个滑动窗口：

$W_2$： $[t_2, t_{16}]$， $W_3$： $[t_3, t_{17}]$， $W_4$： $[t_4, t_{18}]$， $W_5$： $[t_5, t_{19}]$， $W_6$： $[t_6, t_{20}]$

$W_1$图3包含块 $B_1 = [t_1, \ldots, t_5]$、$B_2 = [t_6, \ldots, t_{10}]$和 $B_3 = [t_{11}, \ldots, t_{15}]$，窗口 $W_2$包含 $B_1 = [t_2, \ldots, t_6]$、$B_2 = [t_7, \ldots, t_{11}]$ $B_3 = [t_{12}, \ldots, t_{16}]$ 和 。这两个窗口中的所有块都是唯一的，因为每个块都至少包含一个其他块中没有的项目。$W_1$中的块也彼此不相交，因为它们没有共同项目。我们只统计唯一块，因为同一组项目可能出现在多个窗口中（例如，$[t_6, \ldots, t_{10}]$在 $W_1$ 中显示为 $B_2$，而在 $W_6$中显示为 $B_1$），且不应重复计算这些重复项。

每个窗口分为 3 个大小为 5 的块。因此，从 $t_1$ 到 $t_{16}$的每个索引处都有一个唯一块，总计 16 个唯一块。这些块之所以唯一，是因为每个块至少有一个项目与其他块不同。理想情况下，应在$W$中 20 个可用项目和地标项之间进行重新排序，在可行时实现完全公平。当无法实现完全公平时，目标是通过最大化当前窗口和滑动窗口中的唯一 公平块数量，生成最大公平排列。

## 3 连续公平性监控

本节介绍我们用于持续监控每个滑动窗口中的公平性约束的技术方案$W$。

(1) **草图构建：**首先计算当前窗口$W$的受保护属性 A 的频数分布，并将此信息存储在一种轻量级数据结构中，我们称之为前向草图。(2) **公平性检查：**给定公平性约束后，利用草图判断窗口是否公平。(3) **草图更新：**随着窗口向前滑动，更新草图以反映变化，从而实现持续监控。

### 3.1 预处理

总体而言，草图 [11, 19, 35] 能够对数据分布进行紧凑的摘要。在我们的场景中，目标是捕获滑动窗口内受保护属性值的分布。

定义 *3.1*（前向草图）。宽度为 $|W|$ 的前向草图FSketch $S$是一种与窗口$W$关联的数据结构，其中索引为 $i$ 的每个条目都存储一个长度为（$\ell - 1$）的向量，表示从窗口起始位置到位置 $i$ 为止（$\ell - 1$）个受保护属性值的累积频率。向量的第 $j$ 个条目对应于受保护属性 A 的第 $j$ 个值。由于受保护属性 A 有 $\ell$ 个不同的值，因此可以通过从 $i$（截至该位置所考虑的项目总数）中减去已存储频数之和，推断出索引为 $i$ 处剩余受保护属性值的频数。

请看图 4 中的草图 $S$，其中展示了运行示例的块 $B_1$ 和 $B_2$。向量的第一位用于存储$C$的累积频数，第二位则用于存储$A$的累积频数。例如，$S$的第 4 个条目包含 $[2, 1]$, ，表示从窗口起始位置到第 4 个位置，$C$ 的总频数为 2，$A$ 的总频数为 1。由于 A 有三个不同的值，可以通过从位置 $i$处$C$ 和$A$ 的累积频数中分别减去每个 $i$，推断出$H$ 的频数。例如，从起始位置到位置 $i$的$H$ 累积频数因此为 $4 - (2 + 1) = 1$。

*3.1.1* 草图构造算法。草图 $S$的每个条目 $i$ 都存储一个长度为（$\ell - 1$）的向量，表示从窗口起点到位置 $i$为止，前（$\ell - 1$）个受保护属性值的累积计数。对于 $i > 1$，索引 $i$ 处的草图会复制前一个索引（$i - 1$）处的计数，然后将与第 $j$个受保护属性值对应的计数加一，前提是项目 $t_i$ 具有 $\mathrm{A}^i = p_j$。如果遇到的值没有显式存储在向量中，则不执行任何操作。此构造确保每个草图条目都维护窗口中截至该位置的 $\ell - 1$ 受保护属性值的累积和。由此，始终可以计算出未存储的受保护属性值。算法1的伪代码如下。

**算法 1**前向草图构造 - FSketch

**输入：** 大小为 $|W|$的窗口 $W = \{t_1, t_2, \ldots\ldots, t_{|W|}\}$
**需要：** 受保护属性值集合 $\mathrm{A} = \{p_1, p_2, \ldots, p_\ell\}$

```
1: 将草图 S 初始化为由 |W| 个 | 向量组成，每个向量的长度为 (ℓ − 1)，所有条目均设为 0
2: 对于 i = 1 到 |W| | do
3:     如果 i > 1，则
4:         S[i] ← S[i − 1]        ▷ 从上一条记录复制计数
5:     结束 if
6:     令 p ← A^{t_i}             ▷ 项目的受保护属性值 t_i
7:     对于 j = 1 到 (ℓ − 1) 执行
8:         如果 p = p_j 则
9:             S[i][j] ← S[i][j] + 1
10:            跳出
11:        结束 if
12:    结束 for                   ▷ 如果 p = p_ℓ，则不进行更新
13: 结束 for
14: 返回 S
```

定理3.2。 算法 构造草图 （算法*1*）的运行时间为 $O(|W| \times \ell)$，空间复杂度为 $O(|W| \times \ell)$。

证明。根据算法 1 可知，草图中的每个条目都需要 $\mathrm{O}(\ell)$ 的计算时间和 $\mathrm{O}(\ell)$ 的存储空间，因为它

维护了 $\ell-1$ 个受保护属性值的信息。由于外层循环遍历大小为 $|W|$ 的整个窗口，因此该算法的总体时间和空间复杂度为 $O(|W| \times \ell)$。□

*3.1.2* 草图更新。 首次创建时，草图从第一个窗口的第一个索引开始，追踪 $(\ell-1)$ 个受保护属性值并进行初始化。这确保最早的数据项会先被记录。更新操作可确保草图准确反映最近的窗口数据。窗口$W$滑动时，草图会将所有条目向左移动一个位置。随后，根据倒数第二个条目以及新加入项目的受保护属性值，更新草图中的最后一个条目。这样，草图便与当前窗口保持同步，并持续提供受保护属性随时间变化的最新精简摘要。

再次考虑图 4 中的草图。窗口每滑动一个项目，第 11 个项目便会进入。之前草图中的所有条目（参见图 5）现在都向左移动一个位置，草图的最后一个条目也随之更新。

引理 3.3。 草图更新需要 $O(\ell)$ 时间。

证明。移除草图的第一个元素只需常数时间。在末尾追加一个新向量，其摊销成本为 $O(\ell)$ 时间。□

| $S$: | [1, 0] | [2, 0] | [2, 1] | [2, 1] | **[2,1]** | [3, 1] | [3, 2] | [4, 2] | [4, 2] | **[4,2]** |
|---|---|---|---|---|---|---|---|---|---|---|

**图 4：图 2 中块 $B_1$ 与 $B_2$ 的草图**

| $W$: | [2, C] | [3, A] | [4, H] | [5,H] | [6, C] | [7, A] | [8, C] | [9, A] | [10, H] | [11, A] |
|---|---|---|---|---|---|---|---|---|---|---|
| $S$: | [2, 0] | [2, 1] | [2, 1] | [2, 1] | [3, 1] | [3, 2] | [4, 2] | [4, 2] | [4, 2] | [4, 3] |

**图 5：通过一次项目滑动步长维护草图**

## 3.2 公平性检查

接下来，我们将讨论 Monitor-BFair 算法，该算法用于检查窗口是否公平。该算法的创新之处在于，它最多只需检查当前草图中的块数 $k$ 个条目 $S$，并且始终能返回正确答案。在某些情况下，它可以提前终止，尤其是在已经发现违反公平性约束时。具体来说，它只检查当前草图中 $S$ 代表每个块最后一个条目的那些条目。

为评估大小为 $s$的第一个块，Monitor-BFair 算法首先检查草图 $S$ 的第 $s$ 个条目，其中包含前 $\ell-1$ 个受保护属性值的累计计数。最后一个（隐含的）属性的计数通过用项目数减去已存储计数之和来计算。通过检查所有这些 $\ell-1$ 计数以及推断出的第 $\ell$ 个属性的计数是否满足指定要求，来验证公平性约束。

对于后续的每个块，算法都会计算两个草图向量之差：具体而言，就是前一个块和当前块的最后一个条目之差。该差值给出当前块中每个受保护属性值的频数。对所有块重复此过程，直到某个块违反公平性约束，导致算法提前终止并返回 否；或者所有块都通过检查，此时 Monitor-BFair 返回 是。算法 2 给出了伪代码。

考虑图 4 和表1中的约束1。窗口$W$包含 10 个元素，Monitor-BFair 将其划分为两个块，$B_1$，$B_2$，每块大小为 $s=5$。算法首先检查第一个块末尾处的草图，即索引 5 处的草图，其中保存了累积向量 $[2,1]$。这表明在$B_1$中，有 2 个 $C$ 实例和 1 个$A$实例，这意味着剩余的 2 个项目对应于$H$。该块满足第一条公平性约束。接下来，算法继续检查第二个块末尾处的草图，即索引 10 处的草图。它通过从索引 10 处的草图中减去索引 5 处的草图，计算出$B_2$ 的频数向量。所得向量表明，$B_2$ 包含 2 个$C$实例、1 个$A$实例和 2 个$H$实例。这也满足指定的公平性要求。该过程会对全部三个块继续进行。最后，算法得出结论：该窗口是公平的。

另一方面，考虑图 4 和表1中的约束2。前两个块会被视为公平，但第三个块不会，因此该窗口不公平。

**算法 2**Monitor-BFair：公平性检查算法

**要求：** 草图 $S$ 包含 $|W|$ 个条目，索引范围为 1 至 $|W|$;
**要求：** 块大小为 $s$，块数为 $k$;
**要求：** 每个 $p \in \{1、\ldots\ldots、\ell\}$ 的公平性比例为 $f(p)$
**确保：** **公平**当所有块都满足公平性约束时；**不公平**否-则 对于$b \leftarrow 1$ 至 $k$

$curr \leftarrow b \cdot s$
**如果** $b = 1$ **那么**
  $countVec \leftarrow S[curr]$
**否则**
  $prev \leftarrow (b-1) \cdot s$
  $countVec \leftarrow S[curr] - S[prev]$
**结束如果**
$total \leftarrow s$
$lastCount \leftarrow total - \sum_{j=1}^{\ell-1} countVec[j]$
构造 $fullVec[1 \ldots \ell] \leftarrow (countVec[1 \ldots \ell - 1], lastCount)$
**对于** $p \leftarrow 1$ 到 $\ell$ **执行**
  $minReq \leftarrow \lfloor f(p) \cdot s \rfloor$
  $maxReq \leftarrow \lceil f(p) \cdot s \rceil$
  **如果** $fullVec[p] < minReq$ **或** $fullVec[p] > maxReq$，**则**
    **返回不公平**
  **如果结束**
**结束循环**
**返回公平**

引理 3.4. Monitor-BFair 的最坏情况运行时间为 $O(k \times \ell)$

证明。在最坏情况下，Monitor-BFair 需要处理所有 $k$ 个块，每个块需要 $O(\ell)$ 时间。因此，总共需要 $O(k \times \ell)$ 时间。□

引理3.5. *Monitor-BFair* 的最佳情况运行时间为 $\Omega(\ell)$

证明。在最佳情况下，Monitor-BFair 会在第一个块中发现公平性违规，需要 $\Omega(\ell)$ 时间将该块声明为不公平并终止。□

引理3.6. 算法Monitor-BFair是最优的。

证明见 [1] 中的技术报告。

## 4 连续流重排序

再次考虑公平性**约束2**，这里有三个受保护属性值：$C$，$A$，$H$。具体规定了两种不同的公平性要求：

**情况1：** 要求为 $[2, 1, 2]$,，这意味着每个块必须包含 2 个值为$C$的元素、1 个值为$A$的元素，以及 2 个值为$H$的元素。
**情况2：** 要求为 $[3, 1, 1]$。

如果一个块满足上述任一标准，就视为公平块。由于原始块 $B_3$ 不满足任一要求，因此又读取了 5 个地标项（图3）。更新后的输入流（包括这些地标项）现在由 6 个值为 $C$的元素、7 个值为 $A$的元素和 7 个值为 $H$的元素组成。我们提出了一种算法 BFair-ReOrder，用于重新排列这 20 个元素，以最大化唯一公平块的数量。

设 $n = |W| + |\mathbb{X}|$和 $InStm = [ib_1, ib_2, \ldots, ib_n]$为所考虑的数据流，其中包含 $|W|$个常规项目和 $|\mathbb{X}|$个地标项。目标是将这些元素排列成一个新序列 $OutStm = [ob_1, ob_2, \ldots, ob_n]$，以最大化唯一公平块的数量。用 $fb(Stm)$表示数据流 $Stm$中的唯一公平块数量。于是，该问题可表述为最大化 $OutStm = \arg\max_{Stm} fb(Stm)$。

### 4.1 BFair-ReOrder算法

BFair-ReOrder算法包含一个外层循环，该循环遍历每种有效的受保护属性计数组合，并运行一个子程序（MaxReOrder 子程序），以重新排序并据此创建数量最多的唯一公平块。对于**约束2**，$[C, A, H]$有两种这样的组合：$[2, 1, 2]$ 和 $[3, 1, 1]$,。算法会针对每种组合分别运行该子程序，并在最后输出唯一公平块数量最多的结果。

我们首先介绍 MaxReOrder 子程序设计中用到的关键概念。

定义 *4.1*（同构块与同构块计数）。若一组不相交块中的每个块都包含相同的有序受保护属性值序列，则称这些块为同构。数据流中此类块的总数称为同构块计数（*IBC*）。对于给定的公平性约束，如果其中一个同构块满足公平性条件，则该集合中的所有块都被视为公平块。

假设输入流由 $\ell$ 个不同属性值组成。令 $v_j \geq 1$表示公平性约束要求的值为 $p \in \cup_{i=1}^{\ell} \mathrm{A}_i$的元素数量，使得 $\sum_{j=1}^{\ell} v_j = s$. 令 $V_j$为输入流中值为 $p$的元素总数。那么，满足给定公平性要求的同构块数量（IBC）可计算如下：

$$\mathrm{IBC} = \min\left\{\mathrm{IBC}_j \mid 1 \leq j \leq \ell\right\}, \quad \text{where } \mathrm{IBC}_j = \left\lfloor \frac{V_j}{v_j} \right\rfloor.$$

例如，在**约束2**中，**情况1：**

$$\left\lfloor \frac{6}{2} \right\rfloor = 3, \quad \left\lfloor \frac{7}{1} \right\rfloor = 7, \quad \left\lfloor \frac{7}{2} \right\rfloor = 3.$$

因此，公平同构块的总数为：

$$\mathrm{IBC} = \min(3, 7, 3) = 3.$$

对于**情况2：**，IBC $= \min(2,\ 7,\ 7) = 2$

定义*4.2*（同构流和扩展同构流）。如果一个数据流完全由同构块组成，即每个块都遵循相同的固定组成模式，则称其为同构流。如果在该数据流后面扩展一个由该块模式构成的前缀，则所得数据流称为扩展同构流。这一附加部分称为扩展前缀（*EP*），其长度称为扩展前缀长度（*EPL*）。

给定一个长度为 $n$的数据流，以及指定块大小为 $s$的公平性要求，如果 $IBC \cdot s = n$，则该数据流可以被完全划分为$IBC$个同构块，或者重新排序为同构流。

如果 $IBC \cdot s < n$，剩余的 $n - IBC \cdot s$ 个项目无法组成一个完整的同构块。这些剩余元素可以构成一个扩展前缀扩展前缀。扩展前缀的长度为：

$$EPL = \sum_{i=1}^{\ell} EP_i, \quad \text{where } EP_i = \min\left(v_i,\ V_i - IBC \cdot v_i\right)$$

这里，$\ell$是受保护属性中不同值（例如人口群体）的数量；$v_i$是每个公平块所需的群体 $i$ 数量（根据公平性要求）；$V_i$是整个数据流中群体 $i$ 的元素总数；$EP_i$是分配到扩展前缀的群体 $i$ 元素数量；$IBC$是完整同构块的数量。

注意：$EPL < s$，因为扩展前缀本身无法构成一个完整块。

考虑针对 $C$、$A$ 和 $H$ 的公平性要求 $[2, 1, 2]$，其中 $C = 6, A = 7, H = 7$。

块大小为 $s = 5$，且由于 $IBC = 3$，我们计算：$EP_1$ =min( 2 6 − 3 2 = 0 $EP_2$ = 1 7 − 3 1 = 1 $EP_3$ = 2 7 − ,· ) , 3 2 = 1 $EPL = EP_1 + EP_2 + EP_3 = 0 + 1 + 1 = 2$ min(,· ) ,min(,· ) ，结果为：。因此，扩展前缀为 $EP = [A, H]$（任意排列均有效）。要完成同构块模式，请使用剩余计数 $v_i - EP_i$:

$$\begin{aligned} C:&\quad 2 - 0 = 2 &&\Rightarrow [C, C] \\ A:&\quad 1 - 1 = 0 &&\Rightarrow [] \\ H:&\quad 2 - 1 = 1 &&\Rightarrow [H] \end{aligned}$$

一种有效的同构块模式是：$[A, H, C, C, H]$。现在我们可以构造扩展同构流：

$$[A, H, C, C, H,\ A, H, C, C, C, H,\ A, H, C, C, H,\ A, H]$$

这由三个完整同构块和一个长度为 2 的前缀组成，实现了最优块级公平性。剩余的 3 个 $A$可以追加到末尾，补全数据流。

在某些情况下，可能需要涉及多个公平性约束情况，才能使数据流达到最大公平性。例如：考虑数据流中项目的分布为 $[C, A, H]$: $[7, 8, 5]$。对于所需模式 $[2, 1, 2]$，我们得到同构块数量 $_{Case1} = 2$。剩余的$[3, 6, 1]$个项目可以按照情况2的模式 $[3, 1, 1]$ 排列，这同样满足公平性约束。

得到所需模式的同构块数量

$= 2$ $[2, 1, 2]$ $C =$后，对于 ,,，我们构

$7$ $A = 8$ $H = 5$ $EP$ $EP_1 =$ $2$ $7 - 2$ $2 = 2$

建：$\min(,\cdot)$ ,$EP_2 =$ $1$ $8 - 2$ $1 = 1$ $EP_3 =$ $2$ $5 - 2$ $2 = 1$

$\min(,\cdot)$ 和 $\min(,\cdot)$，由此得到 $EPL = 2 + 1 + 1 = 4$。因此，$EP = [C, C, H, A]$（目前任意排列均有效）。

现在，对于剩余项目 $R = [3, 6, 1]$，我们计算案例模式 的同构块数量

$min$ $1, 6, 1$ $= 1$ $2$ $[3, 1, 1]$ $EP_R$ $EP_{R1} = ()$。我们构建：

$\min(3, 3 - 1 \cdot 3) = 0$, $EP_{R2} = \min(1, 6 - 1 \cdot 1) = 1$ 和 $EP_{R3} = \min(1, 1 - 1 \cdot 1) = 0$。因此，$EPL_R = 0 + 1 + 0 = 1$ 和 $EP_R = [A]$。

因此，所有块能够构成有前缀模式 $[A]$ 其前缀遵循模式 $[A, C, C, H]$（只有 A 的位置固定，其余部分仍可采用任意排列）。因此，情况1之后的前两个同构块为：$[A, C, C, H, H]$, $[A, C, C, H, H]$。情况2之后的第三个块为：$[A, C, C, H, C]$。因此，完整的同构流为：

$$[A, C, C, H, H, A, C, C, H, H, A, C, C, H, C, A]$$

剩余的 4 个 $A$可以追加到数据流末尾，以完成该数据流。

总而言之，MaxReOrder 子程序（参见子程序3）首先计算 $IBC$，即能够满足给定公平性要求的同构块数量。如果没有块能够满足该约束，算法就直接返回原始数据流。

如果 $IBC \cdot s = n$，其中 $s$是块大小，$n$是数据流总长度，那么整个输出流可以恰好由 $IBC$ 个同构块构成，从而形成一个同构流。在这种情况下，算法直接返回构建好的同构流。

如果 $IBC \cdot s < n$，那么剩余元素——即无法放入完整同构块的元素——可用于构成一个扩展前缀$EP$。扩展流通过将 $IBC$ 同构块与该扩展前缀连接起来构造而成。根据给定约束，该数据流包含可能的最大数量的公平块（见引理 4.3）。所有不属于同构块或扩展前缀的剩余元素都会追加到数据流末尾，以补全数据流。

## 4.2 最优性与运行时间分析

引理4.3（同构流与扩展同构流的最优性）。

证明。设数据流长度为 $n$，块大小为 $s$。该数据流中大小为 $s$的唯一块数量为 $n - s + 1$。如果数据流中的所有这些块都是最优的——也就是说，它包含可能的最大数量的唯一公平块——我们就称该数据流为公平的。

一个同构流 由 IBC $= \begin{matrix} n \\ s \end{matrix}$个彼此不相交、–完全相同且大小为 $s$ 的公平块组成。由于每个重叠块（即任意 $s$个连续元素组成的块）都由这些相同公平块的后缀和前缀构成，因此每个这样的块也满足公平性条件。因此，所有 $n - s + 1$ 个唯一块都是公平的，该数据流实现了最优性。

**算法 3 MaxReOrder 子程序 1：输入：长度为 $|W| + X$ 的输入流 $InStm$，公平性约束 $[v_1, \ldots, v_\ell]$ 2：输出：重新排序后的数据流 $OutStm$ 3：计算公平同构块的数量 $IBC$ 4：如果 $IBC < 1$ 则 5：返回 $InStm$ 6：结束如果 7：如果 $IBC \cdot s = n$ 则 8：根据给定的公平性约束构造 $IBC$ 个公平同构块 9：由 $IBC$ 个公平同构块形成$OutStm \leftarrow$ 同构流 10：返回 $OutStm$ 11：结束如果 12：构建扩展前缀 $EP$ 13：如果 $EP$ 不属于任何新的公平块 14：使用 $v_i - EP_i$补全同构块模式 15：通过串接 $IBC$个公平同构块和 $EP$构造扩展流 16：$OutStm \leftarrow$扩展流 + 剩余元素 17：返回 $OutStm$ 18：否则 19：根据剩余部分和新的约束计算 $IBC_R$ 20：根据剩余部分和新的约束构建扩展前缀 $EP_R$ 21：补全 $IBC$、$IBC_R$ 同构块模式 22：$OutStm \leftarrow IBC + IBC_R$ 个公平同构块 + 扩展前缀 $EP_R$+最终剩余部分 23：返回 $OutStm$ 24：结束如果**

设想两个连续的同构块：

$IB1 = [< 1, A >, < 2, C >, < 3, C >, < 4, H >, < 5, H >]$,

$IB2 = [< 6, A >, < 7, C >, < 8, C >, < 9, H >, < 10, H >]$

$U = [< 3, C >, < 4, H >, < 5, H >, < 6, A >$ 重叠块 $, < 7, C >]$ 1 2

恰好由 IB 的后缀和 IB 的前缀组成。由于 IB1 和 IB2 都满足所需的公平性计数，由这些同构块的后缀和前缀组成的任何块——包括 U——也必定是公平的。

扩展同构流 包含一个经过精心构造的前缀 $EP$，用于追加这些同构块。该前缀经过设计，使得 $EP$与最后一个同构块之间的任何重叠块都是公平的。由于同构部分中的所有块也都是公平的，因此整个数据流包含 $n - s + 1$ 个公平块。因此，扩展同构流也是最优的。

□

定理 4.4（BFair-ReOrder算法是最优的 L).

证明。若 $IBC \cdot s = n$，则该数据流是同构流，结论由引理 4.3 得出。

如果 $IBC \cdot s < n$，扩展流包含 $IBC \cdot s + EPL$ 个元素，这些元素构成 $IBC \cdot s + EPL - s + 1 = (IBC - 1) \cdot s + EPL + 1$ 个公平块。根据构造方法，剩余元素

无法再完成一个公平块，也不会与任何公平块重叠。因此，这就是可能的公平块最大数量。MaxReOrder 子程序会针对每种有效的受保护属性值组合求出该最优值，而 BFair-ReOrder算法会检查所有可能的组合，并输出使其最大化的组合。□

**运行时间**。MaxReOrder 子程序的运行时间至多为 O( $n$ $n$ = $|W|+|$ $|$ ) ( X )。在最坏情况下，MaxReOrder 需要运行 $2^{\ell}$ 次。因此，BFair-ReOrder算法的运行时间为 O($n \times 2^{\ell}$)。空间复杂度主要由队列大小决定，为 O($n$)。

## 5 实验评估

我们展示了侧重于三个方面的实验分析：(a) 公平性模型在块级别确保群体公平性的有效性，(b) 公平性检查和重排序方案的有效性，以及 (c) 预处理框架的有效性和成本。代码和数据可在 [1] 获取。

### 5.1 实验设置

**流式环境**。使用 Confluent Kafka Python (CKP) 模拟流式环境，它是 Apache Kafka 的官方 Python 客户端。单个代理开放一个外部监听器，以接收来自主机的生产者发送的数据；该生产者以可控速率将 JSON 格式的数据集记录流式传输到 Kafka主题。

基于 CKP 的消费者轮询代理，在内存缓冲区中维护最新的 $|W|$ 条消息，并构建一个用于预处理和公平性检查的滑动窗口 $W$。

**硬件和软件**。所有实验均在 HP-Omen 16-n0xxx（Windows 11 Home v10.0.26100，Python 3.9.13）上进行，该设备配备 8 核 AMD Ryzen 7 6800H CPU 和 16 GB 内存。单节点 Kafka集群（v3.7.0）和 ZooKeeper（均来自 Confluent Platform 7.8.0）运行于 Docker（v4.41.2）中，以模拟流式环境。结果取 10 次运行的平均值。

**数据集**。实验使用了4个真实世界数据集（更多详情见表3）。

**医院** [8]：包含印度一家三级护理医院的10,000多条患者入院记录（2017年4月至2019年3月），按入院日期排序。“AGE”属性按分位数分箱为五个区间。

**Movies** [16]：包含来自200K+ 位用户的33M+ 条评分，涉及87,585部电影。我们使用连接后的电影和评分表，将类型映射为2类和3类“情绪”（暗/亮/中性）。

**股票** [27]：追踪截至2020年4月的纳斯达克每日价格。苹果公司的价格变化按开盘价与收盘价之间的百分比差计算，并分为五个等量区间。交易量分为2类和3类。

**推文** [12]：包含1,600万条推文（2009年4月至5月）。我们应用情感分析[25]，将推文归入五种基于极性的情绪类别之一。

**基线方法**。我们注意到，目前没有相关工作可以调整用于解决本文研究的两个问题（更多细节见第 *6* 节）。因此，我们设计了自己的基线方法。

**反向草图**或*BSketch*。此方法旨在与第 3.1 节讨论的窗口预处理技术进行比较，该技术称为 FSketch。我们将其称为*BSketch*，因为它从反向开始构建草图。在索引 $i$ 处，它会捕获从第 $i$个位置到窗口末尾的 $\ell - 1$ 受保护属性值的累积计数。自然地，对于每个新窗口，都必须从头重建草图，因为草图中的所有项目都需要重新更新（而不仅仅是像前向草图那样移位）。**BSketchQP** 一种用于监控连续公平性的方法，使用*BSketch*进行预处理。此方法旨在与第 3.2 节提出的 Monitor-BFair 进行比较。

**BruteForce**会枚举所有可能的重排序，并选择唯一公平块数量最多的方案，以便与第 4 节讨论的 BFair-ReOrder 进行比较。显然，*Brute- Force* 的计算开销很大，完全无法扩展。

**度量指标**。对于公平性分析，我们测量算法生成的公平块占原始数据流中公平块的百分比。随后，我们开展一项小规模定性研究，基于用户满意度比较块级公平性与窗口级公平性。我们还评估 BFair-ReOrder 的最优性。我们测量平均公平性检查时间、90% 尾延迟（即 90% 的检查在此时间内完成，其余 10% 耗时更长）、吞吐量（单位时间内处理的公平性约束数量）以及内存消耗。对于预处理分析，我们报告草图构建与更新时间、内存消耗，以及预处理的 90% 尾延迟。我们还分析各个组件的运行时间。

**公平性约束和参数**。我们视情况改变窗口大小 $|W|$、块大小 $s$、受保护属性的基数 $\ell$、地标大小 |X|。公平性约束根据相应数据集中受保护属性值的比例生成。表3讨论了所用的受保护属性及其各自的 $\ell$；每次使用一个受保护属性。默认参数值为 $|W|$=1000、 $s$=25、 |X|=100 和 $\ell = 5$。

| 所用数据集 | 受保护属性($\ell$) | 记录 |
|---|---|---|
| 医院 | 性别(2)、结果(3)、年龄(5) | 10,101 |
| Twitter | 情感(5) | 16,000,000 |
| MovieLens | 评分(5)、情绪(2,3) | 33,832,161 |
| 股票 | % 价格变化(5)，交易量交易(2,3) | 9,908 |

**表3：数据集与统计信息**

**结果总结**。我们的实验分析揭示了几项关键发现。首先，我们的定性研究表明，相较于仅在整体窗口上启用群体公平性，用户一致更偏好我们提出的公平性模型。其次，实证结果与我们针对预处理、重排序和公平性检查中所使用算法的理论保证一致。第三，就效率而言，我们的框架比所有设计的基线方法高出 2–3 个数量级。第四，在应用 BFair-ReOrder 时，该框架能够有效地在块级别实施公平性，大幅提高公平块的比例。此外，无论在时间还是内存方面，公平性监控的开销都极小：Monitor-BFair 可在不到一毫秒的时间内回答查询，而 BFair-ReOrder 随输入流大小呈线性扩展。总体而言，该框架在广泛的参数范围内都具备准确性和可扩展性。

| 数据集 | 参数设置 | 之前重新排序 | 之后重新排序 |
|---|---|---|---|
| 医院 ($s$=25, $\|\mathbb{X}\|$ = 100, $l$ = 5) | $\|W\|$ = 200 | 0% | 62% |
| | $\|W\|$ = 1000 | 8% | 95% |
| | $\|W\|$ = 2000 | 1% | 93% |
| 推文 ($\|W\|$=1000, $\|\mathbb{X}\|$ = 500, $l$ = 5) | $s$ = 25 | 0% | 32% |
| | $s$ = 100 | 0% | 87% |
| | $s$ = 250 | 0% | 96% |
| Movies ($\|W\|$=1000, $s$ = 250, $\|\mathbb{X}\|$ = 500) | $l$ = 2 | 2% | 98% |
| | $l$ = 3 | 0% | 99% |
| | $l$ = 5 | 0% | 21% |

**表 4：BFair-ReOrder实现的公平性**

## 5.2 公平性分析与定性研究

*5.2.1* 公平性分析。 我们将提出的重排序算法 BFair-ReOrder 与 BruteForce 进行比较，以通过实证评估我们的理论主张。我们观察到，BFair-ReOrder 返回的唯一公平块数量与 BruteForce 相同，从而验证了其最优性。更多结果请参见技术报告 [1]。

我们改变不同数据集的窗口大小 $|W|$、块大小 $s$ 和基数 $\ell$，并展示应用 BFair-ReOrder 于原始数据流后变得公平的唯一块所占的百分比。表 4 展示了一组具有代表性的结果。总体而言，BFair-ReOrder 显著提升了数据流的公平性。通常，随着窗口大小 $|W|$ 以及块大小 $s$ 的增加，公平性会大幅提升；这是直观的，因为对于相同的约束，更大的窗口或块大小会提高更多块满足公平性要求的可能性。另一方面，随着基数增加，群体公平性要求变得更加复杂，从而导致公平块所占的百分比降低。

在图 6 中，我们改变 $|\mathbb{X}|$，并测量应用 BFair-ReOrder 后唯一公平块所占的百分比。结果表明，随着 $|\mathbb{X}|$ 的增加，唯一公平块所占的百分比也会增加——这是直观的，因为随着地标项增加，满足公平性约束的可能性也会提高，从而提升总体百分比。

*5.2.2* 定性研究。 我们邀请 10 名研究生开展定性研究，比较两种在线内容推荐应用中，块级公平性与窗口级公平性下用户感知的推荐质量——评分范围为 1 分（满意度最低）至 5 分（满意度最高）：**(1) 歌曲播放列表**，包含来自三种类型（流行音乐、电子音乐和乡村音乐）的 15 首曲目，分布在三页（块）中；**(2) 机器学习内容**，包含涵盖三种类型（教程、项目和讲解视频）的 15 个 YouTube 视频，分布在三页（块）中。

对于每项任务，系统会以随机顺序呈现两种推荐排序——一种确保逐页（块级）代表性，另一种确保整体（窗口级）代表性；参与者不会获知所使用的模型。对于每个应用，参与者需回答三个问题：（1）您对模型 A 生成的列表的平均满意度如何？（2）您对模型 B 生成的列表的平均满意度如何？（3）总体而言，您更喜欢模型 A 还是模型 B 生成的列表？

表 5 总结了参与者对两种应用中的块级与窗口级公平性模型进行比较后给出的定性反馈。总体而言，参与者表示，对块级模型所产生的推荐的满意度明显更高（平均满意度超过 4.2 分），而对窗口级模型所产生推荐的满意度约为 3.0 分。绝大多数用户（约 89%）在歌曲播放列表和机器学习内容场景中都更偏好块级排序。这些结果表明，在块级别实施公平性，能够带来用户认为更公平、也更令人满意的推荐，尤其是在数据流场景中。

| 应用程序 | 平均满意度（块） | 平均满意度（窗口） | % 偏好 块 | 偏好比例 窗口 |
|---|---|---|---|---|
| Song 播放列表 | 4.22 | 3.00 | 89% | 11% |
| ML Content | 4.33 | 3.00 | 89% | 11% |

**表 5：定性研究结果：块与窗口级公平性对比**

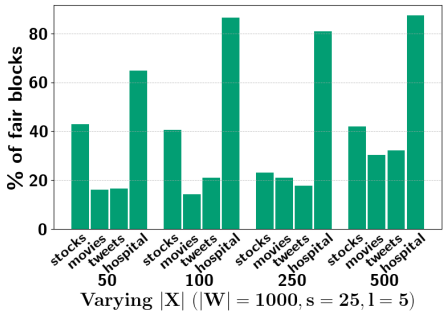


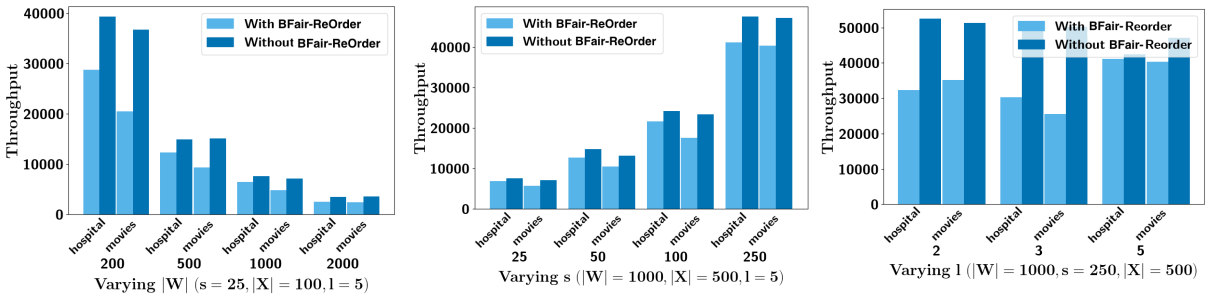

**图 6：地标对公平性的影响**

## 5.3 公平性算法的运行时间 hms

我们首先展示 Monitor-BFair 和 BFair-ReOrder 在改变相关参数时的效果。在第 5.3.4 节中，我们分别将实现的基线方法与我们的方案进行比较，结果表明，基线方法的运行时间比我们的方案慢 2 – 3 个数量级。

*5.3.1* 吞吐量。 在这些实验中，我们通过改变 $|W|$、$s$ 和 $\ell$，测量 Monitor-BFair 的吞吐量（每秒处理量），并考虑两种情况：一种是需要使用 BFair-ReOrder 对涉及地标项的内容重新排序，另一种则不需要。图 7a 展示了这些结果，表明我们的算法具有很高的吞吐量（某些情况下高达 40，000）。正如预期，在需要重新排序时，Monitor-BFair 的处理时间更长，导致吞吐量较低，反之亦然。我们还观察到 $|W|$ 与吞吐量之间存在反向关系。这是因为，随着窗口大小增加 $|W|$，需要处理的块数量也随之增加，从而耗时更长。

另一方面，在图 7b 中，我们观察到 $s$ 与吞吐量之间存在直接的正比关系。增大 $s$ 会在固定 $|W|$ 的情况下减少块的数量，从而缩短公平性监控所需的时间并提高吞吐量。




**(a) 不同窗口大小** **(b) 不同块大小** **(c) 不同基数**

**图 7：使用和不使用BFair-ReOrder时的吞吐量**

$\ell$ 的影响不太明显（图 7c）。虽然更高的基数可能会增加运行时间，但均匀分布的约束通常会产生更多公平块，从而减少重排序工作。因此，在 $\ell$ 处于中等水平时，总运行时间保持稳定；在没有 BFair-ReOrder 的情况下，吞吐量也几乎保持不变。

*5.3.2* 尾延迟。我们报告了 Monitor-BFair 在 5,000 个窗口中的第90百分位数尾延迟，包括使用和不使用算法 BFair-ReOrder 的情况，如图 8 所示。

如图 8a 所示，随着 $|W|$ 增大，尾延迟也会增加。例如，当 $W = 2000$时，处理单个公平性约束的时间偶尔会超过 10 ms。不过，需要注意的是，这代表最慢的 10% 情况，并不反映典型的运行时间。图 8b 展示了随着 $s$ 增加，尾延迟按预期下降，这与此前的观察结果一致。同样，图 8c 展示了增大 $\ell$ 所产生的影响。由于不同 A 中值的分布存在差异，在这种情况下，尾延迟没有呈现出明确的趋势。

将 BFair-ReOrder 集成到 Monitor-BFair 后，尾延迟显著上升，因为重排序会带来大量开销。不进行重排序时，延迟极低。

*5.3.3* 运行时间与内存消耗。即使需要重新排序，平均运行时间通常也仅为几分之一毫秒。Monitor-BFair 的内存消耗可以忽略不计，与后文所示的草图内存消耗一致。详细结果见我们的技术报告 [1]。

*5.3.4* 与基线相比的运行时间。如图 9 所示，我们在 Hospital 数据集上针对不同的 $|W|$、$s$ 和 $|X|$ 值，对比了 BFair-ReOrder 与 BruteForce 的运行时间。由于 BruteForce 算法效率极低，在尝试对窗口大小和地标大小超过 24 个项目，或基数大于 3 的情况进行实验时，流式环境会自动终止进程（因轮询超过 5 分钟而关闭连接）。

图 9a 展示了 BruteForce 的低效性：当 $|W| \le 16$ 时，其运行时间与 BFair-ReOrder 相当，但在 $|W| = 20$ 时会飙升至 10 s 以上，证实了其具有指数复杂度。BruteForce 穷举搜索所有重新排序，因此对 $s$ 高度敏感（图 9b）；较大的 $s$ 会减少块的数量，从而提高尽早实现公平性的可能性。增加 $|X|$ 同样会延长运行时间，但增长趋势较为平缓。总体而言，BFair-ReOrder 快逾三个数量级速度更快。BSketchQP 的表现始终不及 Monitor-BFair，因此为简洁起见，省略了相关结果。

## 5.4 预处理分析

我们介绍所提方案的预处理分析，并将在本节后面介绍与适当基线方法的比较。

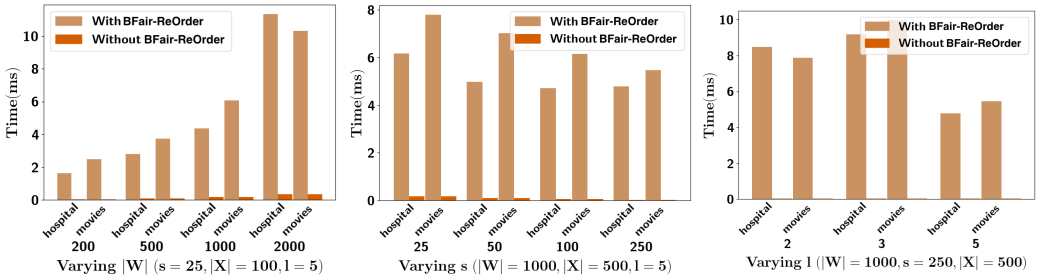
*5.4.1* 草图构建与更新。 我们在图10中报告了股票和Movies数据集上5,000个窗口的预处理时间，并展示了使用和不使用BFair-ReOrder时草图构建与更新的时间分解（请注意，如果使用BFair-ReOrder，则需要从头开始构建草图）。不使用BFair-ReOrder时，草图构建时间可以忽略不计，因为草图只需构建一次，随后在其余窗口中进行更新。

如图10a所示，由于草图构建成本更高，预处理时间会随 $|W|$ 增长，而更新时间保持稳定。改变 $s$ 或 $\ell$ 的影响很小（图10b），表明该方法对块大小和基数具有稳健性。有时，不使用BFair-ReOrder时的预处理速度反而更慢，因为通过地标项进行批量内存访问（固定 $|X| = 500$）相比于为每个窗口加载新记录，可以减少开销。

*5.4.2* 内存需求。 图11展示了在5,000个窗口的预处理过程中内存使用量的平均峰值，反映了在不同参数下所使用的最大内存（包括后台进程）。

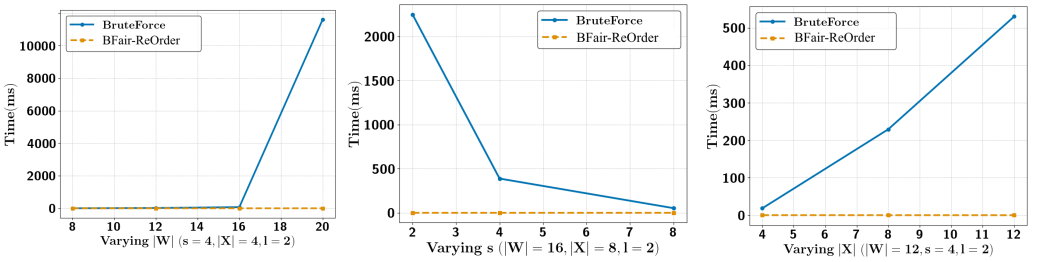
正如预期，FSketch 的内存消耗会随着 $|W|$ 增加而增加（图11a），这是因为草图变大了。随着 $s$ 增大，内存消耗会减少（图11b），因为较大的块减少了 BFair-ReOrder 过程中所需的配置数量，从而降低了内存使用量。同样，内存使用量会随着 $|X|$ 增加，这并非由草图大小导致，而是因为在 BFair-ReOrder 中对更长的数据流进行重排序会带来更大的后台内存开销。这些实验印证了我们的理论分析，表明所提出的 FSketch 占用空间小，且内存使用量受窗口大小限制。

*5.4.3* 与基线方法比较。 我们比较了*BSketch* 和*FSketch* 在股票数据集上的预处理时间（图12）。*FSketch* 始终更快，通常快出几个数量级；随着窗口大小增大，差距也随之扩大（图12a），这是因为*BSketch* 会从头开始重建，而*FSketch* 则会增量更新。块大小和基数几乎没有影响（图12b–12c），只有因平台级波动而产生的细微差异。



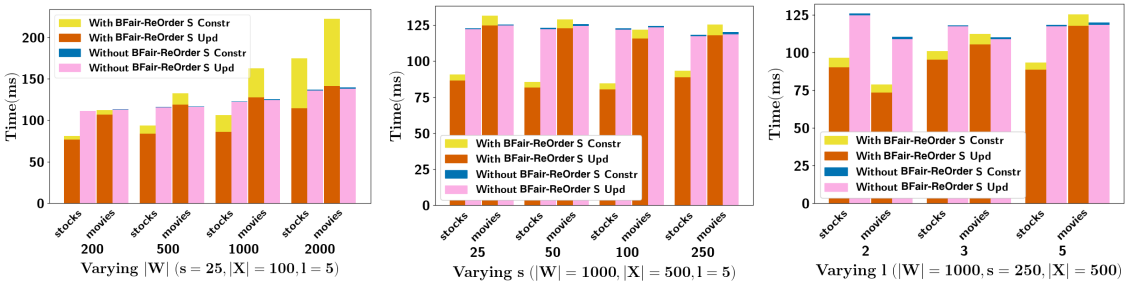

(a) 不同窗口大小 (b) 不同块大小 (c) 不同基数

图 8：5000个窗口的运行时间尾延迟第90百分位



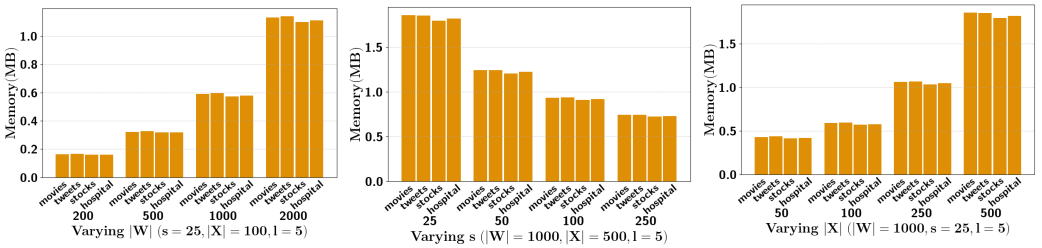

(a) 改变窗口大小 (b) 改变块大小 (c) 改变地标项

图 9： BruteForce 与 BFair-ReOrder 运行时间




(a) 改变窗口大小 (b) 改变块大小 (c) 基数变化

图10：5000个窗口的预处理总运行时间

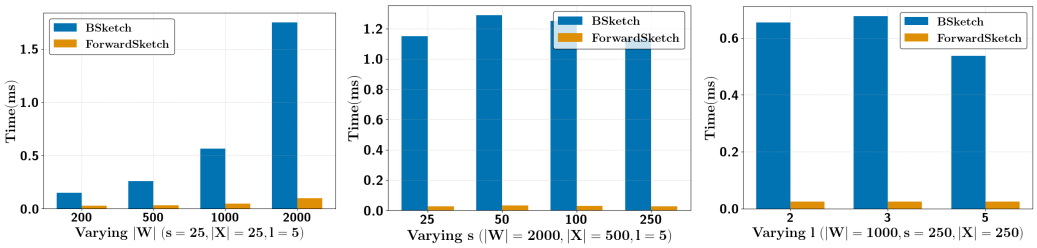



(a) 窗口大小变化 (b) 块大小变化 (c) 地标项变化

图11：5000个窗口的平均峰值内存消耗

## 5.5 组件运行时间

我们在所有四个数据集上，针对不同的 $|W|$、$s$和 $|X|$，评估四个核心组件——草图构建、草图更新、BFair-ReOrder 和 Monitor-BFair——的平均运行时间。BFair-ReOrder（图13）的耗时最高，这符合预期；其次是草图构建，而草图更新和 Monitor-BFair 始终保持轻量级。草图构建随 $|W|$ 线性扩展，但基本不受 $s$ 和 $|X|$ 的影响。相比之下，BFair-ReOrder




**(a) 窗口大小变化** **(b) 块大小变化** **(c) 基数变化**

**图12：**反向草图 与前向草图**的预处理时间**

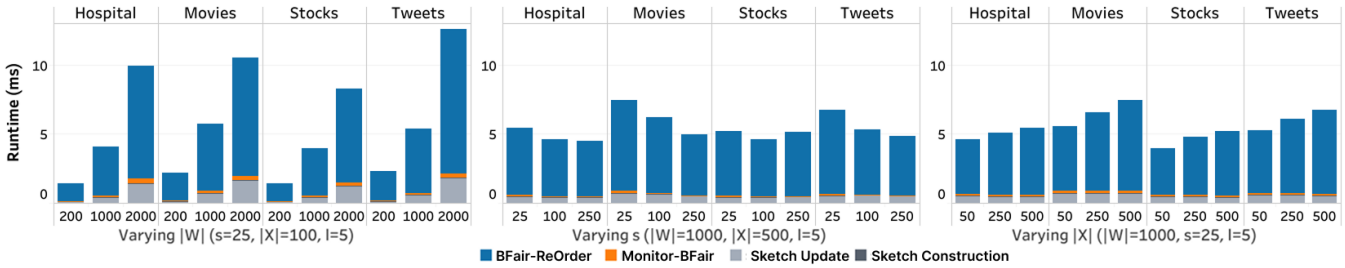



**图13：各组件运行时间**

随 $|W|$ 和 $|X|$ 均呈线性扩展，且运行时间随着 $s$ 增加而略有下降。

## 6 相关工作

据我们所知，目前尚无相关工作研究我们提出的问题。

**群体公平性**。群体公平性最初在分类领域得到研究，旨在确保由受保护属性（例如种族、性别、年龄）定义的人口群体受到公平对待，通常通过人口 或统计均等 [17, 21]来衡量。在数据管理领域，群体公平性已在排序和推荐中得到广泛研究 [18, 22, 30, 32, 34], ，其中，*top-k* 均等 [22]和*P*-公平性 [34] 等公平性概念确保结果或前缀中的群体代表性均衡。将这些保障扩展到多个 受保护属性仍然面临计算上的挑战 [9, 20] ，因为相交子群体的数量会呈指数级增长。

我们将P-公平性 的概念推广到每个块（而非每个前缀），并开始研究其在流式场景中的应用。

**数据流**。数据流研究已从高效处理逐步转向在持续更新下维护数据质量。CQL [2], 、C-SPARQL [4], 和窗口连接算法 [15, 24, 26] 等基础系统解决了可扩展且节省内存的数据流处理问题。后续研究侧重于一致性和约束管理，否定约束 [10] 支持逻辑验证，而近期的框架 [28, 29] 则支持实时监控和自适应模式实施。已有研究 [23,31] 还提出自动修复技术，以在流式更新期间维持约束满足。

相比之下，我们着重于在窗口的每个块内施加公平性约束，只允许重新排列数据，而不修改或填补值。因此，现有的数据质量与修复方法并不直接适用于我们的场景。

**数据流公平性**。近年来，数据流研究开始更加关注公平性。Baumeister 等人 [6] 通过估计实时数据流中的概率，检查人口统计均等和均等化赔率等公平性度量。Wang 等人 [33]提出了流公平采样（$FS^2$），一种在考虑公平性和概念漂移的同时平衡类别分布的方法，用于处理演化数据流中的公平性。他们还提出了公平性绑定效用（*FBU*），这是一种用于权衡公平性与准确率的度量。

相比之下，我们提出了一种框架，通过实时监控公平性并智能地重新排序窗口，最大限度地提升可实现的公平性水平，从而在数据流中持续保障群体公平性。

## 7 结论

我们率先研究了数据流中的群体公平性，通过在滑动窗口内进行块级评估，以更细的粒度保障公平性。与传统的窗口级方法相比，这种局部化方法能够更准确、更灵活地捕捉公平性违规。我们提出了一个全面的计算框架，其中包括高效的基于草图的监控结构和最优的数据流重排序算法，二者均旨在支持实时的持续公平性监控与重排序。这些贡献共同助力在高吞吐量流式系统中，以可扩展、低延迟且有理有据的方式保障群体公平性，并为动态且对公平性要求严格的应用提供了有力的理论保证和实用价值。

我们目前正在研究如何扩展所提出的框架，以同时处理多个受保护属性（例如性别、种族和年龄）。虽然我们认识到这个问题本身在计算上难以处理，但初步研究结果表明，重排序问题尤其具有很大挑战，因为当前解决方案无法推广到满足多个属性下的公平性。

# 参考文献

[1] 2025. 匿名 Git。https://anonymous.4open.science/r/ProposesdFramework- B66B。匿名 GitHub 仓库。[2] Arvind Arasu、Shivnath Babu 和 Jennifer Widom。2006。CQL 连续查询语言：语义基础与查询执行。VLDB J. 15, 2 (2006), 121–142。[3] Shivnath Babu 和 Jennifer Widom。2001。数据流上的连续查询。ACM Sigmod Record 30, 3 (2001), 109–120。[4] Davide Francesco Barbieri、Daniele Braga、Stefano Ceri、Emanuele Della Valle 和 Michael Grossniklaus。2009。C-SPARQL：用于连续查询的 SPARQL。载于 WWW。ACM，1061–1062。[5] Solon Barocas、Moritz Hardt 和 Arvind Narayanan。2019。公平性与机器学习：局限与机遇。fairmlbook. org。[6] Jan Baumeister、Bernd Finkbeiner、Frederik Scheerer、Julian Siber 和 Tobias Wagenpfeil。2025。基于流的算法公平性监测。载于 TACAS (1)（《计算机科学讲义》，第 15696 卷）。施普林格，60–81。[7] Reuben Binns。2020。论个体公平性与群体公平性之间表面上的冲突。载于 2020 年公平性、问责制与透明度会议论文集（西班牙巴塞罗那）。计算机协会，美国纽约州纽约市，514–524。https://doi.org/10.1145/3351095.3372864[8] Sandeep Chandra Bollepalli、Ashish Kumar Sahani、Naved Aslam、Bishav Mohan、Kanchan Kulkarni、Abhishek Goyal、Bhupinder Singh、Gurbhej Singh、Ankit Mittal、Rohit Tandon、S hibba Takkar Chhabra、Gurpreet S. Wander 和 Antonis A. Armoundas。2022。优化后的机器学习模型可准确预测入院至心脏科病区后的住院结局。Diagnostics 12, 2 (2022)。https://doi.org/ 10 3390 12020241 。/diagnostics [9] Felix S Campbell、Alon Silberstein、Julia Stoyanovich 和 Yuval Moskovitch。2024。多样化 Top-k 选择的查询细化。ACM 数据管理论文集 2, 3 (2024), 1–27。[10] Xu Chu、Ihab F. Ilyas 和 Paolo Papotti。2013。发现否定约束。Proc. VLDB Endow. 6, 13 (2013), 1498–1509。[11] Mayur Datar、Aristides Gionis、Piotr Indyk 和 Rajeev Motwani。2002。维护滑动窗口上的流统计。SIAM J. Comput. 31, 6 (2002), 1794–1813。[12] FERNO2。2020。training.1600000.processed.noemoticon.csv。https://www.kaggle.com/datasets/ferno2/training1600000processednoemoticoncsv。[13] David García-Soriano 和 Francesco Bonchi。2021。最大最小公平排名：群体公平性约束下的个体公平性。载于第 27 届 ACM SIGKDD 知识发现与数据挖掘会议论文集（线上会议，新加坡）。计算机协会，美国纽约州纽约市，436–446。https://doi.org/ 10 1145 3447548 3467349 。/ 。Lukasz Golab 和 M Tamer Özsu。 。[14] 2003 处理数据流连续查询中的滑动窗口多重连接。载于 2003 年 VLDB 会议论文集。爱思唯尔，500–511。[15] Lukasz Golab 和 M. Tamer Özsu。2003。处理数据流连续查询中的滑动窗口多重连接。载于 VLDB。摩根·考夫曼，500–511。[16] F. Maxwell Harper 和 Joseph A. Konstan。2015。MovieLens 数据集：历史与背景。ACM Trans. Interact. Intell. Syst. 5, 4, Article 19 (Dec. 2015), 19 页面。https://doi.org/ 10 1145 2827872 。/ Corinna Hertweck、Christoph Heitz 和 [17] Michele Loi。2021。统计均等的道德正当性。载于 2021 年 ACM 公平性、问责制与透明度会议论文集。747–757。[18] Nyi Nyi Htun 等。2021。群体音乐推荐系统中的公平性感知。载于 IUI。302–306。[19] Piotr Indyk。2006。稳定分布、伪随机生成器、嵌入与数据流计算。J. ACM 53, 3 (2006), 307–323。[20] Md Mouinul Islam 等。2022。通过偏好替换满足复杂的 top-k 公平性约束。PVLDB (2022)。[21] Zhimeng Jiang、Xiaotian Han、Chao Fan、Fan Yang、Ali Mostafavi 和 Xia Hu。2022。群体公平性的广义人口统计均等。载于国际学习表征会议。[22] Caitlin Kuhlman 和 Elke Rundensteiner。2020。实现公平共识的排名聚合算法。VLDB 基金会论文集 13, 12 (2020)。[23] Samuele Langhi、Angela Bonifati 和 Riccardo Tommasini。2025。使用不一致性注释评估连续查询。(2025)。[24] Hyo-Sang Lim、Jae-Gil Lee、Min-Jae Lee、Kyu-Young Whang 和 Il-Yeol Song。2006。利用数据与查询的对偶性处理数据流中的连续查询。载于 SIGMOD 会议。ACM，313–324。[25] Steven Loria。2018。textblob 文档。版本 0.15 2 (2018)。[26] Kamesh Munagala、Utkarsh Srivastava 和 Jennifer Widom。2007。优化带有共享高开销筛选器的连续查询。载于 PODS。ACM，215–224。[27] Oleh Onyshchak。2020。股票市场数据集。https://doi.org/10.34740/KAGGLE/ DSV/1054465[28] Vasileios Papastergios 和 Anastasios Gounaris。2025。Stream DaQ：流优先数据质量监测。CoRR abs/2506.06147 (2025)。[29] Eduardo H. M. Pena、Fábio Porto 和 Felix Naumann。2024。在动态数据集中发现否定约束。载于 ICDE。电气电子工程师学会，3546–3558。[30] Evaggelia Pitoura 等人。2022。排名与推荐中的公平性：综述。*VLDB*期刊 (2022)，1–28。[31] Shaoxu Song、Aoqian Zhang、Jianmin Wang 和 Philip S. Yu。2015。SCREEN：速度约束下的流数据清洗。载于*SIGMOD* 会议。ACM，827–841。[32] Julia Stoyanovich、Bill Howe 和 Hosagrahar Visvesvaraya Jagadish。2020。负责任的数据管理。*VLDB* 基金会论文集 13，12（2020）。[33] Zichong Wang、Nripsuta Saxena、Tongjia Yu、Sneha Karki、Tyler Zetty、Israat Haque、Shan Zhou、Dukka Kc、Ian Stockwell、Xuyu Wang、Albert Bifet 和 Wenbin Zhang。2023。防止演化数据流中的歧视性决策。载于*FAccT*。ACM，149–159。[34] Dong Wei 等人。2022。具有比例公平性的排名聚合。载于*SIGMOD*。[35] Li Yan、Nerissa Xu、Guozhong Li、Sourav S. Bhowmick、Byron Choi 和 Jianliang Xu。2022。SENSOR：面向时间序列数据的基于草图的可视化查询界面的数据驱动构建。*Proc. VLDB Endow.* 15，12（2022），3650–3653。