Alignment-free Genomic Analysis via a Big Data Spark Platform

摘要

尽管大数据技术在计算生物学中越来越受欢迎，但尚未有团队为这些任务开发大数据平台。因此，本文为无比对基因组分析搭建了一个可扩展、高效且可扩展的 Spark 平台。该平台实现了无比对基因组分析中常用的一些算法，支持数据密集型和计算密集型任务，并且对用户友好，非 Spark 专业用户也能轻松扩展。本人在原有项目的基础上，实现了 Hellinger 距离方法，使基因组之间的距离能够归一化，更加直观地评估多个基因组之间的相似性。

关键词： Alignment-free；Genomic Analysis；Big Data；Spark

1 引言

1.1 选题背景

对于许多基因组、宏基因组和表观基因组的多序列比对任务，免对齐相似性比较算法（Alignment-free algorithm，简称 AF 算法）是比较常用的解决方案。对于数据密集型应用，AF 算法的计算是一个大数据领域的问题。最近的文献表明，开发快速、可扩展的平台来计算 AF 算法是一项亟待实现的任务。然而，尽管大数据技术在计算生物学中越来越受欢迎，但尚未有团队为这些任务开发大数据平台。

1.2 选题依据

基因组分析是生物信息学一个热门且重要的研究领域。随着高通量测序技术的发展，生成的基因组数据量巨大，对其进行有效分析具有重要意义。传统的基因组分析方法主要依赖序列对齐算法（alignment），但对齐算法随数据量增大，会大大增加计算成本。为了解决该问题，可采用免对齐算法（alignment-free）进行基因组分析，这会减少计算量，提高效率。Spark 是一个流行的大数据分析平台，利用 Spark 处理大规模基因组数据可以发挥其在分布式计算等方面的优势，提升基因组分析的速度和效率。结合免对齐算法和 Spark 大数据平台进行基因组分析，既考虑了数据规模增长的计算挑战，又探索了方法学创新，具有一定的学术价值和实用意义。

1.3 选题意义

该论文提供了第一个支持 AF 算法评估的 Spark 平台 FADE。该平台高效且可扩展，并保证了在可用工作负载和计算资源方面依然具有良好的可扩展性。该平台实现了无比对基因组分析中常用的一些算法，支持数据密集型和计算密集型任务，并且对用户友好，非 Spark 专业用户也能轻松扩展。该研究具有一定的前瞻性，也切合当前生物信息学和大数据技术快速发展的态势，对推动相关领域技术发展具有正面作用。

2 相关工作

在介绍本课题内容相关工作前，需要大致了解对齐算法和免对齐算法的含义和区别，从而更好地理解后续工作。

2.1 对齐算法（AB 算法）

对齐算法（AB 算法）是将两个或多个 DNA/蛋白质序列按照一定规则进行匹配和比较，找出它们的相似和不同之处的一组算法。Needleman-Wunsch 比对算法 [2] 是最经典的对齐算法之一，于 1970 年提出，使用最大匹配来比对两种蛋白质的序列，其本质上是两个字符串的比对算法。

Needleman-Wunsch 比对算法的步骤大致如下：

1. 构建评分矩阵：以两个序列为矩阵的行列，矩阵中的每个元素对应序列中相应位置的符号 (碱基或氨基酸) 之间的匹配或错配评分。
2. 填充评分矩阵：从矩阵的两个角逐步向内填充分数。当前元素的分数为左元素、上元素和左上元素的最高分数，加上当前元素之间的匹配或错配评分。
3. 回溯路径：从矩阵右下角开始，根据最大分数回溯找到一条对齐路径。
4. 输出结果：根据回溯路径，形成最优全局对齐的结果。

对齐算法也存在一些缺点：该算法仅对同源序列的比对能提供更好的结果，适用于相对较小的序列，并且该类算法的时间复杂度和空间复杂度都比较高。对于多序列比对或者长序列比对，该算法会变成 NP-hard 问题 [4]。

2.2 免对齐算法（AF 算法）

免对齐算法（AF 算法）是直接在序列上提取各种数值特征，然后对这些数字特征进行比较，无需进行传统意义上的序列比对，从而加快分析速度。常用特征有 k-mer、单词频数等。受限于篇幅，本文将侧重于 k-mer 方法。

K-mer 方法是对于集合中的每个序列，将其中包含的长度为 k 的连续子串（即 k-mers，获取方法类似滑动窗口）及其出现频率进行计数，然后转化为一组向量。最后通过计算每对向量之间距离/相似度来对序列进行比较 [5]。

本处展示当 k 取 2 时，用 **AGCATC** 、**AGTATC** 两个序列作比对的示例。

首先根据滑动窗口方法，获取 **AGCATA** 的所有 2-mers： **AG**、**GC**、**CA**、**AT**、**TA**。再获取 **AGTATA** 的所有 2-mers： **AG**、**GT**、**TA**、**AT**、**TA**。

再将结果分别转换为向量，向量中每一项的值表示该 2-mer 出现的次数，具体如表 1、表 2所示。

表 1. AGCATC 对应的 2-mer 向量

AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
0	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0

表 2. AGTATA 对应的 2-mer 向量

AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
0	0	1	1	0	0	0	0	0	0	0	1	2	0	0	0

然后根据向量距离/相似度算法来计算两个向量的距离/相似度，此处使用欧拉距离计算。计算公式如公式 1所示：

$$Euclidean(\mathbf{P}, \mathbf{Q}) = \sqrt{\sum_{i=1}^{n}(\mathbf{P}_i - \mathbf{Q}_i)^2} \tag{1}$$

计算出来的相似度为 2。

2.3 比较

对齐算法和免对齐算法各有各的优势：

1. 对齐算法通过建立显性的序列匹配对应关系来比较序列，而免对齐算法则依赖于隐性的编码信息。
2. 对齐算法比较复杂，需要高计算成本，而免对齐算法简单、快速。
3. 对齐算法比较全面和直观，可以找到两个序列之间相似的区域，免对齐更注重整体统计特征。
4. 对齐算法主要应用在小规模序列比较，而免对齐更适合大规模基因组序列分析。

3 本文方法

3.1 本文方法概述

本文实现了第一个针对高效计算 AF 算法的 Spark 平台 FADE，它具有良好的可扩展性。假设要处理的数据集由 n 条序列组成，那么输出是一个 $n \times n$ 的矩阵，每一项 (i, j) 则对应序列 i 和序列 j 经过所选的 AF 函数计算出来的相似度。如图 1所示，FADE 实现免对齐算法需要经过五个步骤：

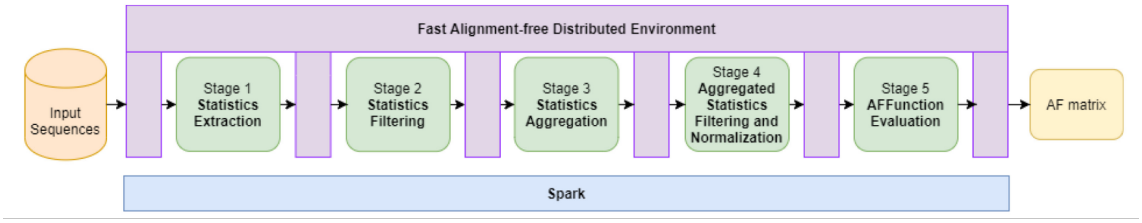

图 1. 快速计算 AF 算法的基本流水线的逻辑结构

1. 收集部分统计数据：需要从各个序列中收集统计数据，例如从序列中提取 k-mers。
2. 基于特征的统计过滤：用户可能需要从其统计数据中排除部分特征，例如包含”N”字符的 k-mers。
3. 统计结果聚合：在前两个阶段期间，由不同计算节点收集但来自相同输入序列的部分统计信息，都会自动在同一节点上进行聚合。例如获取序列中所有 k-mers 的统计数据。
4. 基于值的聚合统计数据过滤和标准化：根据条件过滤聚合后的统计数据，例如排除低频 k-mers。
5. AF 矩阵计算：给每对序列使用 AF 算法并获取评分，然后相应地填充 AF 矩阵。

输出的 AF 矩阵会以分布式数据结构进行编码，其内容可以以 csv 格式保存在文件中，或用来做进一步的分析。

3.2 聚合策略

FADE 提供了 3 种不同的聚合策略，这些策略会以特定的方式，将被划分的数据聚合到一起，分别如下：

1. 全部聚合。当提取和处理整体较小的统计数据时，该策略具有非常好的执行效率，因为在管道期间提取的所有统计数据（不论是部分数据还是聚合后的数据）都在分布式系统的单个节点上维护和处理。对于每个统计数据评估的部分 AF 函数也会发生同样的情况。一方面，这意味着除了第一阶段之外，不会发生分布式计算。另一方面，该策略可以避免在处理数据之前，将数据传输到分布式系统的节点所需的数据传输开销。
2. 不聚合。当从非常大的输入数据中提取和处理统计数据时，该策略具有非常好的可扩展性，因为在管道期间提取的每个统计数据（不论是部分数据还是聚合后的数据）都作为独立的数据对象进行管理。对于每个统计数据评估的部分 AF 函数也会发生同样的情况。唯一的聚合发生在管道末端，这是因为部分 AF 函数最后需要整合所有的结果，并进行最终计算。
3. 部分聚合。当从大量输入数据中提取和处理统计信息时，该策略可以在效率和可扩展性之间实现良好的权衡，因为在管道期间提取的所有统计数据（不论是部分数据还是聚合后的数据）都被划分到容器中。对于每个统计数据评估的部分 AF 函数也会发生同样的情况。因此，每个节点只需处理较少数量的数据记录，而不是数量大得多的单个数据记录。

3.3 相似度计算方法

此处介绍一些常用的相似度计算方法 [1]。给定一组序列集合 $S = s_1, \ldots, s_n$ ，集合中的序列 s 关于 k-mer 的直方图 h_s（可理解为向量）的定义如公式 2所示：

$$h_s = \langle c(w_1), c(w_2), \ldots, c(w_{|k|}) \rangle \tag{2}$$

其中 $c(w_i)$ 是指在序列 s 中单词 w_i（也就是第 i 个 k-mer）的出现频率，而 K 是指所有可能出现的 k-mer 的数量。

限于篇幅，此处仅介绍闵可夫斯基距离系列、匹配度/不匹配度系列、卡方距离和 Canberra 距离。

3.3.1 闵可夫斯基距离系列

给定两个序列 s 和 t 及其相关统计量 h_s 和 h_t，欧拉距离可表示为公式 3：

$$\text{Euclidean}\,(h_s, h_t) = \sqrt{\sum_{w \in K} (h_s(w) - h_t(w))^2} \tag{3}$$

欧拉距离的另一个被广泛使用的变种是曼哈顿距离，可表示为公式 4：

$$\text{Manhattan}\,(h_s, h_t) = \sum_{w \in K} |h_s(w) - h_t(w)| \tag{4}$$

该系列还有一种距离公式，称作切比雪夫距离，可表示为公式 5：

$$\text{Chebyshev}\,(h_s, h_t) = \max_{w \in K} |h_s(w) - h_t(w)| \tag{5}$$

3.3.2 匹配度/不匹配度系列

令 h_s^* 和 h_t^* 分别为 h_s 和 h_t 中具有非零项 k-mers 的集合：

Jaccard 指数的计算一直是信息检索界和生物信息学界备受关注的对象，它的定义如下，可表示为公式 6：

$$\text{Jaccard}\,(h_s, h_t) = \frac{(h_s^* \cap h_t^*)}{(h_s^* \cup h_t^*)} \tag{6}$$

3.3.3 卡方距离

卡方距离的定义如公式 7所示：

$$\chi^2\,(h_s, h_t) = \sum_{w \in K} \frac{(h_s(w) - h_t(w))^2}{(h_s(w) + h_t(w))} \tag{7}$$

3.3.4 Canberra 距离

Canberra 距离结合了曼哈顿距离和卡方距离，定义公式 8所示：

$$\text{Canberra}\,(h_s, h_t) = \sum_{w \in K} \frac{|h_s(w) - h_t(w)|}{(h_s(w) + h_t(w))} \tag{8}$$

4 复现细节

4.1 与已有开源代码对比

本项目的复现代码参考了https://github.com/fpalini/fade，在该源码的基础上实现了新的相似度计算方法。该开源代码虽然实现了很多 AF 相似度计算方法，但是这些方法均没有实现结果的归一化，于是本人实现了 Hellinger 距离方法。本人为了实现该方法，额外实现了 NormalizedValue 类，用于辅助计算统计结果的归一化过程。

4.2 实验环境搭建

本人的开发设备为 MacBook Air M1，芯片为 Apple M1，内存为 8GB，系统为 macOS Ventura 13.2.1。

软件环境为 Apache Spark 2.3.3。主要使用 Java 8 作为开发语言，部分使用了 Scala 2.10.7，并安装了 mllib、fastdoop、fastutil 等生物信息学领域的常用库。项目在 IntelliJ IDEA 2022.1.3 上开发。

4.3 使用说明与参数分析

本项目包含一个可执行的 jar 文件 **fade-1.0.0-all.jar**，可以从命令行运行。该文件使用标准 Apache Spark Spark-submit 命令和以下语法来运行：

```
spark-submit fade-1.0.0-all.jar [conf-file]
```

如果未指定 conf-file，程序将在工作目录中查找 fade.conf 文件。FADE 主要有两个操作任务：

1. 距离/相似度评估：输入基因组序列集合，根据提供的 AF 函数集合，计算出一组 AF 评估矩阵。
2. 蒙特卡洛模拟：针对给定的基因组序列集合，对一组输入 AF 函数运行统计显着性测试。

本文只介绍距离/相似度评估。 3将介绍 conf-file 的常用参数：

表 3. conf-file 常用参数表

参数	描述
task	任务类型。distance 表示距离/相似度评估，simulation 表示蒙特卡洛模拟。
local	是否本地运行。true 表示 Apache Spark 以本地模式运行，false 表示运行在集群上。默认为 true。
input	基因组数据集所在路径。
output	存放结果的文件路径。
extractor	需要使用的提取器（Extractor）名字。
aggregator	需要使用的聚集器（Aggregator）名字。
evaluator	需要使用的评估器（Evaluator）名字。
strategy	聚集策略。可选 no_aggregation，partial_aggregation（默认）或者 total_aggregation。
k	要收集的统计数据的长度（以核苷酸为单位）。
slices	并行任务的数量，至少应与 Spark 工作线程的数量相匹配，默认值为 128。

4.4 创新点

Hellinger 距离在生物信息学研究中应用广泛，但在基于 AF 分析的工具中较少见。本人实现了这个方法在 AF 序列分析领域中的新应用，该方法具有以下优点 [3]：

1. 灵敏度高：Hellinger 距离更加灵敏，即使两个序列有细微差异也能反映出来，更能体现出微小变异。
2. 更稳健：与欧拉距离等其他度量相比，Hellinger 距离对离群值和噪声数据更具稳健性。
3. 易于计算：Hellinger 距离计算方法简单，与序列长度无关，计算速度快。
4. 独立性好：Hellinger 距离判断两个概率分布的相似性时，不需要假设它们之间相互独立，因此应用范围更广。
5. 结果一致性：基于 Hellinger 距离构建的基因组距离矩阵，其多维尺度分析结果与其他矩阵方法高度一致。
6. 效果优异：在基因组距离比较、分类、聚类等方面，Hellinger 距离效果明显优于其他算法，可以更准确判断序列间的进化关系。

Hellinger 距离的计算公式如公式 9所示：

$$\text{Hellinger}\left(h_s, h_t\right) = \sqrt{\frac{1}{2}\sum_{w \in K}\left(\sqrt{h_s(w)} - \sqrt{h_t(w)}\right)^2} \tag{9}$$

要实现该方法，需要按照以下步骤执行。首先创建 NormalizedValue 类，该类实现了 Value 接口，用于存放后续计算的中间值。该类本质上是对 double 类型的 count 变量进行了封装，重写了一些常用方法。

```java
public class NormalizedValue implements Value {
    public final double count;
    public NormalizedValue(double count) {this.count = count;}
    @Override
    public String toString() {return "" + count;}
    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof NormalizedValue)) return false;
        return ((NormalizedValue)obj).count == count;
    }
    @Override
    public int hashCode() {return Double.hashCode(count);}
}
```

然后需要编写 Hellinger 类，该类继承了 AFFunctionEvaluatorByStatistic 类，后者的作用是封装 AF 函数计算中间过程所需要的变量。Hellinger 类需要重写三个方法：

1. evaluatePartialAFValue。该方法需要两个 Value 类型的参数 s1 和 s2，返回值类型也是 Value。该方法的作用是对两个序列的直方图中相同项的值进行处理，并返回中间值。例如，对于 Hellinger 距离的计算，首先需要计算 $(\sqrt{h_s(w)} - \sqrt{h_t(w)})^2$ 部分，而这正是 evaluatePartialAFValue 的职责。

2. combinePartialAFValues。该方法需要两个 AFValue 类型的参数 d1 和 d2，返回值类型也是 AFValue。该方法的作用是对上一步处理后的中间值聚合起来，再进行下一步操作。例如，对于 Hellinger 距离的计算，需要把上一步的中间值累加起来，即对应公式的求和部分。

3. finalizeAFValue。该方法需要一个 AFValue 类型的参数 d，返回值类型也是 AFValue。该方法的作用是对上一步处理后的中间值进行最终的操作。例如，对于 Hellinger 距离的计算，需要把上一步累加的中间值先除以 2，再开平方。

具体的实现代码如下：

```java
public class Hellinger extends AFFunctionEvaluatorByStatistic {
    public Hellinger(Configuration conf) {super(conf);}
    @Override
    public AFValue evaluatePartialAFValue(Value s1, Value s2) {
        double count1 =
            ((NormalizedValue)s1).count * 1.0 / getCount1();
        double count2 =
            ((NormalizedValue)s2).count * 1.0 / getCount2();
        return new AFValue(Math.pow(Math.sqrt(count1) -
            Math.sqrt(count2), 2));
```

```
    }
    @Override
    public AFValue combinePartialAFValues(AFValue d1, AFValue d2) {
        return new AFValue(d1.value + d2.value);
    }
    @Override
    public AFValue finalizeAFValue(AFValue d) {
        return new AFValue(Math.sqrt(d.value / 2));
    }
}
```

最后，需要在 quickstart.conf 中修改 evaluator 的参数，本文中选择了欧拉距离、调和平均值、卡方距离和 Hellinger 距离。

```
evaluator=fade.affunction.Euclidean,
          fade.affunction.HarmonicMean,
          fade.affunction.ChiSquare,
          fade.affunction.Hellinger
```

5 实验结果分析

本次实验选择了以下三组数据集：

1. mito：总大小为 422KB，包含 25 种线粒体的基因序列，平均每个基因序列的大小为 16.88KB，平均长度约为 8643 bp（即 8643 个字符）。
2. unassembled-ecoli-coverage-5：总大小为 780.4MB，包含 29 种大肠杆菌的基因序列，平均每个基因序列的大小为 26.9MB，平均长度约为 1.41×10^7 bp（即 1.41×10^7 个字符）。
3. assembled-plants：总大小为 4.8GB，包含 14 种植物的基因序列，平均每个基因序列的大小为 351MB，平均长度约为 3.68×10^8 bp（即 3.68×10^8 个字符）。

每个数据集均使用 k-mer 方法来处理序列，k 取 5。并行任务数量取 32，策略均同时使用 no_aggregation、partial_aggregation、total_aggregation，提取器使用 KmerExtractorByBin，聚集器使用 KmerAggregatorByBin。

AF 相似度计算方法使用了欧拉函数、调和平均数、卡方距离和 Hellinger 距离。

在本开发环境（详见 4.2）中，运行时间如表 4所示：

表 4. 各数据集在不同策略下的执行时间（单位：s）

数据集	No aggregation	Partial aggregation	Total aggregation
小	6	6	6
中	23	25	32
大	155	147	173

注：小 = mito，中 = unassembled-ecoli-coverage-5，大 = assembled-plants

对于采用了 partial_aggregation 策略的 assembled-plants 数据集，计算产生了 $C_{14}^2 = 91$ 组比较结果，依据 Hellinger 距离升序排序，前 10 组数据如图 2所示：

	SEQ1	SEQ2	euclidean	harmonicmean	chisquare	hellinger ▲ 1
1	camaldule	grandis	941778.0520207508	6.4766941237370 11E8	202851.25259833067	0.004196877196982231
2	rubella	thalian	823004.2548261825	1.240502379771 2643E8	608846.0457471655	0.016876790859784056
3	lyrata	parvulum	3892634.9085168517	1.4090490938646698E8	1.6291551227065977E7	0.018184521480776608
4	parvulum	thalian	410051.7941284979	1.1655291731164221E8	247869.37671552575	0.018438830741008053
5	parvulum	rubella	924666.6142507796	1.215010652098529E8	1140603.5802941846	0.01900722369718696
6	lyrata	thalian	3765575.842750216	1.4427464747291544E8	1.411866305416915E7	0.022180383776069453
7	lyrata	rubella	3041601.352603263	1.519339204270378E8	9589147.145924414	0.02266887707322453
8	rapa	rubella	3222450.8460406344	1.517597590737284E8	9461207.85254315	0.023428916762517394
9	parvulum	rapa	4078144.051892478	1.4068908099879548E8	1.624694600240905E7	0.02473989600361656
10	clementin	sinensis	924645.7698854194	2.9764078855857027E8	918953.8828594775	0.027041979450335433

图 2. 采用 partial_aggregation 策略的 assembled-plants 数据集计算结果

可见 camaldule 和 grandis 在该指标下的距离最近，基因相似度最高。

并且由于 Hellinger 距离实现了归一化，取值范围不受序列长度的影响，更能清晰地反映两个序列之间的相似性。

6 总结与展望

本项目搭建了一个可扩展、高效且可扩展的 Spark 平台，并实现了多种 AF 序列相似度计算方法。但原有代码在计算相似度结果时未进行归一化处理。考虑到不同序列长度会对距离计算产生影响，本人在项目基础上新增实现了 Hellinger 距离算法，并辅助设计了 NormalizedValue 类，用于归一化处理统计结果，从而提高了不同序列间相似度结果的可比性。该优化使项目能够产生更加准确、可解释性更好的序列比较结果。经测试，Hellinger 距离实现了归一化，取值范围不受序列长度的影响，更能清晰地反映两个序列之间的相似性。

本人的工作暂且仅实现了 AF 序列分析中的 Hellinger 距离算法，但 AF 方法中还存在其他计算序列距离的算法，。后续可继续拓展和优化实现新的 AF 距离算法，构建更全面的 AF 分析工具包。

另外，目前工作主要关注算法实现本身，未深入考察各算法在不同实际问题下的效果对比。后续可基于更多实际数据集，系统地评估各 AF 距离算法的优缺点，为算法选择和效果改进提供指导。

此外，AF 分析主要应用于序列的无监督比较和分类。未来可进一步探索结合 AF 分析与机器学习、深度学习方法，应用于病毒变异检测、基因表达预测等实际问题，拓展 AF 在有监督模型中的应用。

综上，本项目仅是 AF 技术研究的起步，未来还可从算法实现、效果评估、实际应用等多个方面进行拓展，以发掘 AF 分析的更大潜力。

参考文献

[1] Brian B Luczak, Benjamin T James, and Hani Z Girgis. A survey and evaluations of histogram-based statistics in alignment-free sequence comparison. *Briefings in bioinformatics*, 20(4):1222–1237, 2019.

[2] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[3] Aideen C Roddy, Anna Jurek-Loughrey, Jose Souza, Alan Gilmore, Paul G O ' Reilly, Alexey Stupnikov, David Gonzalez de Castro, Kevin M Prise, Manuel Salto-Tellez, and Darragh G McArt. Nuqa: estimating cancer spatial and temporal heterogeneity and evolution through alignment-free methods. *Molecular Biology and Evolution*, 36(12):2883–2889, 2019.

[4] Machbah Uddin, Mohammad Khairul Islam, Md Rakib Hassan, Farah Jahan, and Joong Hwan Baek. A fast and efficient algorithm for dna sequence similarity identification. *Complex & Intelligent Systems*, 9(2):1265–1280, 2023.

[5] Md Sayeed Iftekhar Yousuf, Machbah Uddin, Mohammad Khairul Islam, Md Rakib Hassan, Aysha Siddika Ratna, and Farah Jahan. Dna matching using k-mer derived spatial features. In *2023 International Conference on Next-Generation Computing, IoT and Machine Learning (NCIM)*, pages 1–6. IEEE, 2023.