OpenWoud：无厂商锁定的可扩展自适应随机森林加速器

Ewout van Nimwegen*、Elijah Seth Cishugi*、Hendrik Folmer*、Marco Ottavi*† 和 Kuan-Hsun Chen‡,
*特文特大学，荷兰； †罗马第二大学，意大利； ‡帕德博恩大学，德国
*{e.w.vannimwegen, e.s.cishugi, h.h.folmer, m.ottavi}@utwente.nl, ‡kuan-hsun.chen@uni-paderborn.de

arXiv:2611.32662v1 [cs.AR] 24 Nov 2026

摘要——在线机器学习（ML）能够基于演化数据流实现实时决策。这些动态模型会持续学习，以适应新概念；随着旧概念失效，模型也随之调整，但其性能往往受吞吐量限制。自适应随机森林（ARF）是适用于这一场景的先进模型，它天然具备并行性，因此非常适合硬件加速。然而，先前基于现场可编程门阵列（FPGA）的加速器受厂商限制，并依赖资源消耗巨大的浮点（FP）运算，制约了其可扩展性和可移植性。我们提出了*OpenWoud*，一种适用于 FPGA 的不受厂商限制的 ARF 框架。我们的设计采用灵活的定点（FxP）算术库取代浮点运算算子，并为训练和漂移检测引入完全透明、经手动调度的数据流。这使得精细的时间/面积权衡成为可能，并确保跨平台行为可预测。对合成数据集和真实数据集进行的全面评估表明，与先前的 FPGA 加速器相比，OpenWoud 在将资源利用率降低一半的同时，保持或提升了准确率（最高达 +12.89%）。与图形处理器（GPU）（0.5–1.5× 加速比）和 FPGA 基线（0.2–0.5×）相比，OpenWoud 的性能也具有竞争力，尽管后者基于理想化仿真。尤为重要的是，OpenWoud 支持真正逐样本推理，而 GPU 则需要批量插入。

索引词—**自适应随机森林、基于在线数据流的学习、概念漂移、ARF硬件加速器、现场可编程门阵列**

I. 引言

卫星干扰检测和网络入侵检测是需要对导致通信中断的事件立即采取行动的应用 [1] [2]。这些都是演化数据流的示例，其中环境发生部分或完全变化，使先前学习的模型失效。这种现象称为概念漂移。传统的基于批处理的离线机器学习模型，例如卷积神经网络（CNN） [3] 或 Transformer [4]，是为静态数据集设计的，数据发生变化时必须从头开始重新训练。这会造成实时系统无法承受的长时间延迟。

基于在线流的机器学习提供了一种动态的替代方案。模型通过一次处理一个样本来持续学习；对于每个样本，模型都会返回预测结果并利用该样本进行训练，从而为下一次预测提供更新后的模型。一种用于此分类任务的流行先进算法是自适应随机森林（ARF） [5] [6] [7]。它由一组快速决策树组成，这些决策树彼此独立地学习，并且从直观上看，需要采用并行架构来提高吞吐量。因此，ARF 非常适合移植到硬件加速器上 [8] [9]。然而，现有的基于 FPGA 的加速器面临两个主要限制。首先，它们被锁定于 Xilinx FPGA，阻碍了跨厂商部署。这一点限制了其应用，因为该设计无法利用其他可编程逻辑 fabric 技术——例如，Microsemi 等厂商提供基于闪存的现场可编程门阵列（相对于 Xilinx 采用的基于 SRAM 的技术），能够以相对较低的成本提供抗辐射器件，从而提高航空航天应用（例如卫星）中的容错能力。其次，它们依赖数据流中的浮点算术，这会消耗大量 FPGA 资源，从而限制可扩展性。

为此，我们提出了 *OpenWoud*，这是一个面向现场可编程门阵列的不受厂商限制的可扩展 ARF 框架。OpenWoud 使用灵活的定点数替代方案，将训练和漂移检测器数据流中的复杂浮点算术（除法、平方根、自然对数）替换为灵活的定点数运算，从而实现细粒度的时间/面积权衡。该设计在两个层面上具备可扩展性：在局部层面，通过参数化的算术运算实现；在全局层面，通过超参数调优实现（例如树数量、深度和类别数量）。我们的定量评估（基准测试）表明，OpenWoud 能够保持或提高准确率（最高提升 12.89%），实现具有竞争力的吞吐量，并将面积占用减少一半。本工作的贡献有三点：

- 我们提出了一种不受厂商限制的自适应随机森林加速器，在不影响其精度的情况下提升了易用性。
- 一种灵活的开源定点算术库，可对精度和资源利用率进行细粒度控制。
- 我们提出了一种可扩展的训练和漂移检测数据流架构，该架构完全透明，并采用

手动推导且可调节的任务调度。

本文其余部分的组织结构如下：第 II 节介绍自适应随机森林和 FPGA 加速，第 II-D 节则介绍现有的加速器。第 III 节描述 OpenWoud 框架、其灵活的定点算术库和可扩展数据流。随后，第 IV 节详细介绍模型之间的实现和超参数对齐。第 V 节给出全面的评估。最后，第 VI 节总结全文。

II. 背景

本节介绍自适应随机森林及相关概念，以便将其高效映射到 FPGA 硬件加速器上。

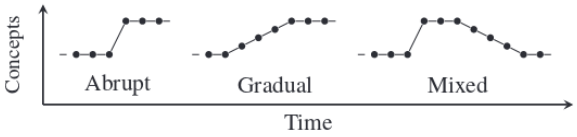


图 1：概念漂移的类型 [9]。

A. 自适应随机森林

自适应随机森林 [5]是一种最先进的算法，能够从演化数据流中高效学习并处理概念漂移。该算法由超快速决策树（VFDTs）集成和一个多数投票器组成。超快速决策树使用Hoeffding界拆分节点，以确定<style id='2'>何时（在时间上）进行拆分，并利用信息增益决定应选择<style id='4'>哪个属性，从而高效地从无限数据流中学习 [10]。然而，单棵超快速决策树也容易出现不稳定并产生偏差。自适应随机森林遵循随机森林（RF） [11] 的方法来解决这一问题：结合多样化弱学习器集成的知识（让每棵树在属性的随机子集上进行训练），掩盖存在偏差的树，使整体误差降低。具体而言，采用另外两种机制来扩展随机森林，以适应演化数据流：

(1) 在线装袋 [12]：通过使用由泊松分布生成的权重进行有效的重新采样，在各棵树之间引入随机性（多样性）。

(2) 漂移检测器：为每棵树配备概念漂移检测机制，以感知并应对环境变化。发生变化时，性能较差的树会被替换为另一棵仅使用新概念训练的树。

B. 概念漂移检测

在自适应随机森林中，概念漂移检测通常采用自适应窗口（ADWIN） [13] [14]算法实现，该算法可检测三种漂移类型（见图 1），即突变、渐变和 混合 漂移。突变 漂移会立即发生，而突变 漂移发生得较慢（分布于数万个样本中，例如 25k - 50k）。混合 漂移同时包含突变和渐变模式。每棵树的错误率都存储在其各自的动态滑动窗口中。系统会定期进行统计切分检验，将滑动窗口分为两个子窗口，并检验两者的数据分布是否存在显著差异。检测到变化后，较旧的部分会被丢弃，使检测器能够适应新概念。

C. FPGA 加速

自适应随机森林集成中的各棵树彼此独立运行，因此具备内在的并行性。现场可编程门阵列非常适合用于自适应随机森林的硬件加速，因为它们支持真正的逐样本处理、能够利用细粒度并行性，并且可以高效处理流式数据；相比之下，图形处理器通常仅限于基于批次的样本插入。然而，训练和漂移检测组件包含设计复杂度较高的算法：

- Hoeffding bound (train): $\epsilon = \sqrt{\frac{R^2 \cdot \ln(1/\delta)}{2 \cdot n}}$
- Information gain (train): $G_{WA} = \sum_b^B \frac{T_b}{T} \cdot (1 - \sum_{c=1}^C P_c^2)$
- Cutting point (drift): $\epsilon_{cut} = \sqrt{\frac{2}{m} \cdot \sigma_W^2 \cdot \ln \frac{2}{\delta'} + \frac{2}{3m} \ln \frac{2}{\delta'}}$

FPGA架构不包含用于除法、平方根和自然对数等复杂算术运算的硬件资源，而仅支持加法和乘法。对这些运算进行朴素实现通常会消耗过多资源或导致较高延迟。因此，必须谨慎设计算术原语和整体算法架构。

D. 现有技术

先前的工作包括基于 Gomes 等人提出的、面向中央处理器的原始 ARF 模型构建的图形处理器和 FPGA 加速器 [5]；这些工作构成了评估本研究时所采用的基线。

a) ARF（图形处理器基线） *[8]*：该方法采用混合 GPU–CPU架构加速 ARF，即*GSARF*，报告称其相较于 CPU基线实现了 $138\times$ 倍加速；不过，这一改进仅适用于基于批次的样本插入。通过数据集预处理，将属性值替换为布尔值，从而降低推理复杂度。此外，通过近似计算信息增益来提高吞吐量。引入候选树，以减少在重复概念发生概念漂移恢复期间的训练开销。

b)ARF [FPGA基线] [9]：作为首个报告的基于 FPGA的 ARF，该设计继承了*GSARF* 的两项优化，其性能优于图形处理器方案 1.6–7×（通过仿真评估），同时逐个处理流中的样本。然而，该框架被限制为只能使用 Xilinx FPGA。该设计使用 Vitis-HLS工具开发，目标平台为 Alveo U55。训练数据流中的算术运算和漂移检测控制路径均通过专有浮点库实现，后者是 ARF 中主要的性能瓶颈。由此产生的浮点架构采用编译器生成的任务调度，其透明度有限，且只能通过编译指令进行粗粒度控制；这些指令往往会被 Vitis 忽略或应用不一致 [15], [16]。

III. 架构设计

本节讨论*OpenWoud*[1]，面向现场可编程门阵列的自适应随机森林硬件加速器设计。该设计旨在克服 Ridder 等人提出的 FPGA 基线所存在的供应商依赖（仅限部署于 Xilinx 现场可编程门阵列）和有限可扩展性（源于基于浮点数的算术运算，以及由编译器生成且不透明的数据流）。我们的设计引入了：(i) 无供应商依赖的自适应随机森林框架；(ii) 灵活的定点数算术库；以及 (iii) 用于训练和漂移检测的可扩展数据流，并采用人工推导、完全透明的任务调度。 [9]

A. 高层概述

OpenWoud 分三个阶段处理演化数据流：(S1)管理、(S2)推理以及 (S3)训练/漂移检测/投票。高层组件（图 2）

[1]https://anonymous.4open.science/r/OpenWoud-866E

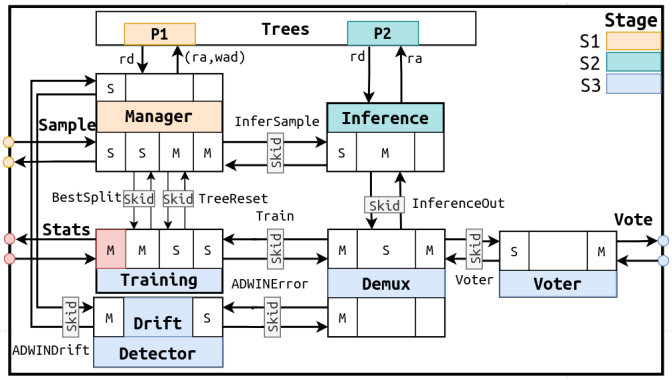


图 2：OpenWoud 高层概览。协议定义为 AXIS 从机：S、主机：M，Advanced eXtensibleInterface（AXI）以红色突出显示。

通过高级可扩展接口流式传输（AXIS）接口连接，形成双向通信流，使各阶段能够并发运行。

S1：管理器 集中控制森林：将输入样本分发到所有树中，插入分裂，在检测到概念漂移时重置树，并跟踪每棵树的状态（活动/非活动）和角色（前景/背景）。

S2： 接下来，**推理引擎** 从根节点到叶节点遍历每棵树，采用流水线架构以实现森林的并行处理。通过推理解复用器，叶端点被传递给训练引擎，预测结果则转发给多数投票器，而错误率发送至漂移检测器。

S3： 最后阶段包括并行运行的训练、漂移检测和投票：

- **训练引擎：** 根据叶节点统计信息（分支计数器、分支类别计数器和总计数器）计算分裂，并采用基于定点数的数据流架构处理信息增益和 Hoeffding界。分裂结果将反馈给管理器。
- **漂移检测器：** 为每棵树独立实现 ADWIN2 [13]，以感知概念漂移。对基于直方图的滑动窗口应用采用基于定点数的数据流处理切分点，从而高效跟踪误差。漂移警告/错误会反馈给管理器。
- **投票器：** 跟踪前景树的投票结果，并将多数类发送到输出流。

树节点存储在本地双端口块RAM（BRAM）中，而叶节点统计信息则通过多个 AXI接口写入主存。Skid缓冲器将反压引入的关键路径解耦，从而改善时序。与依赖 Vitis-HLS 的FPGA基线不同，OpenWoud 使用开源硬件描述语言（HDL）Clash 实现 [17]。与会插入时序并自动生成任务调度的高级综合（HLS）不同，HDL 需要显式定义时序 [18]。Clash编译器生成通用 Verilog，确保与不同FPGA平台兼容，从而支持部署到 Lattice 和 Microsemi 目标平台。

B. 灵活算术库

高效映射复杂的算术运算（除法、平方根、自然对数）是构建可扩展训练和漂移检测数据流的关键。FPGA基线 [9] 依赖 Vitis-HLS 提供的静态浮点算术运算，灵活性有限。为解决这一问题，我们基于现有算法，使用 Clash 开发了一个开源定点数（FxP）库。其主要特点包括：(i) 动态（且可扩展）的分辨率；(ii) 多种针对时间和面积优化的配置。

我们开展了全面的设计空间探索（DSE）：考虑了 8 种除法、11 种平方根和 7 种自然对数的现有基于定点数且针对 FPGA 优化的实现。我们的 DSE 最终筛选出迭代式快速方法；由于整个数据路径所需的分辨率尚不明确（对于 FPGA基线中的浮点数并不相关），这些方法既具备可扩展性，又能实现高性能。我们的数据流使用牛顿-拉夫逊[19] [20]方法实现除法和平方根，这是一种以乘法为主且收敛速度快的算法。而自然对数则采用广义双曲坐标旋转数字计算机（GH-CORDIC） [21][22], 实现，仅需移位/加法操作。该库提供两种可配置模式：多周期模式针对面积进行优化，流水线模式针对性能进行优化，从而实现时间/面积权衡。为最大限度提高效率，我们应用了 FPGA优化：

- **高效归一化：** 采用单周期前导一检测器 [19]进行归一化，减少迭代次数。
- **硬件高效数据类型**：归一化操作数使用 Q2.15，以匹配来自 Xilinx 和 Microsemi 的特定硬件 18×25 和 18×18数字信号处理（DSP）单元。与基于浮点的实现相比，硬件乘法器的使用量减少了 $4\times$。
- **迭代调优：** 对于 32 位输出精度，平方根/除法/自然对数分别在 4/6/8 次迭代后收敛，从而确保相比 FPGA基线 [9]具有更高的准确率。

C. 数据流

训练和漂移检测数据流均经过明确规定，以确保充分的透明度和灵活性，而不是由编译器生成调度方案。我们将信息增益、Hoeffding界和切分点重新表述为 FPGA优化变体，例如移除常数项，并将除法移至求和运算之外。

- **训练数据流（图3）：** 由三条数据路径组成，属性子集中的叶节点统计信息通过*play*分发到这些路径。Hoeffding界（图3上部）针对每个属性子集运行一次，并进行了面积优化（多周期）；信息增益（中部）处理所有属性，并针对吞吐量进行了优化（流水线）。通过微调分支类别计数器的分辨率，对查找表（LUT）/触发器（FF）的使用进行了优化。此外，还通过以下方式优化数字信号处理器的使用：(i) 复用中间路径处理左、右两个分支，因为乘加累加器（MAC）的数字信号处理器数量随类别数量线性增长；(ii) 在乘加累加器（MAC）和除法之前使用位移进行预缩放。

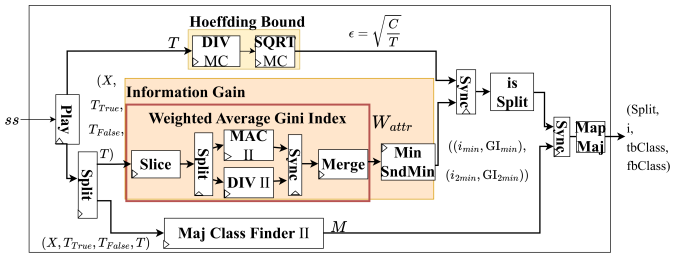


图 3：训练数据流架构。算术运算配置：多周期（MC）和流水线（II）。

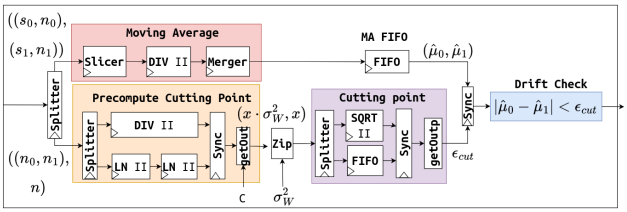


图 4：ADWIN2 切分检查数据流架构。

- **漂移检测数据流（图4）：** 采用基于定点数的流水线架构实现ADWIN2切分检查，以并行处理一组切分点，从而感知概念漂移。两条路径消耗相同数量的输入，因此均配置为追求性能。

最终得到一种可调节的数据流架构，其数据通路中的分辨率灵活可变，可在保持精度的同时降低资源消耗。此外，模块化结构便于替换算术算法（例如优化除法运算），或通过数据流分析推导出改进的任务调度，从而直观地提升性能。

IV. 实现

为验证 OpenWoud 的厂商无关性、可扩展性和性能，我们将自适应随机森林映射到来自不同厂商（Xilinx/AMD 和 Microsemi）的两款现场可编程门阵列上，并将其与三个基线模型进行比较：*River* （中央处理器） [5]，*GSARF*（图形处理器） [8] 和*Vitis* （现场可编程门阵列） [9]。本节介绍硬件目标平台和超参数对齐，以确保比较公平且可复现。

A. 硬件目标平台

OpenWoud针对低端 Xilinx/AMD Kria KV260 片上系统（SoC）进行综合（Vivado 2022.2），用于目标平台推理，并针对 Microsemi PolarFire 片上系统（Libero 2022.1）进行跨供应商验证。我们使用面向中央处理器的 Python 框架 River 作为准确率测量的基线。River 未列出性能指标，因为中央处理器的执行速度比 GSARF 等硬件加速器慢两个数量级（加速比为 138 倍）。GSARF 在 NVIDIA GeForce GTX 1650 图形处理器上实现，而 Vitis（FPGA基线）在 Vitis-HLS 仿真中进行评估，目标是配备高带宽存储器 2（HBM2）主存的高端 Alveo U55。两款 Xilinx 片上系统均配备四个

		常规			漂移检测器			训练引擎			
数据集	自适应随机森林	目标	T	D	δ_W	δ_E	GP	W	M	δ	τ	GP	SS	λ
AGR	River	CPU	64	9	0.001	0.00001	32	1000	5	0.05	0.05	16	10	1
	GSARF	GPU	96	9	0.001	0.00001	32	10	5	0.002 –	16	10	1	
	Vitis	现场可编程门阵列	64	9	0.001	0.00001	32	5	5	0.05	0.05	16	10	1
	OpenWoud	现场可编程门阵列	64	9	0.001	0.00001	32	1000	5	0.05	0.05	16	10	1
LED	River	CPU	64	9	0.001	0.00001	32	1000	5	0.05	0.05	16	5	1
	GSARF	GPU	96	9	0.001	0.00001	32	10	5	0.002 –	16	5	1	
	Vitis	现场可编程门阵列	64	9	0.001	0.00001	32	5	5	0.05	0.05	16	5	1
	OpenWoud	现场可编程门阵列	64	9	0.001	0.00001	32	1000	5	0.05	0.05	16	5	1
COV	River	CPU	64	9	0.001	0.00001	32	1000	5	0.05	0.05	16	7	1
	GSARF	GPU	96	9	0.001	0.00001	32	10	5	0.002 –	16	7	1	
	Vitis	现场可编程门阵列	64	9	0.001	0.00001	32	5	5	0.05	0.05	16	7	1
	OpenWoud	现场可编程门阵列	64	9	0.001	0.00001	32	1000	5	0.05	0.05	16	7	1

表 I：各数据集和模型的 ARF 超参数。

现场可编程门阵列互连与主存之间的高性能总线（HPB），可绕过中央管理单元，实现高吞吐量突发传输。不过，HBM2 支持完全并行的读写操作，而双倍数据速率（ DDR）-4 会在内部对来自不同通道的请求进行排队。中央处理器基线和硬件加速器均采用二进制属性优化，因为后者仅接受二进制属性。

B. 超参数

加速器的超参数针对各个数据集进行了微调/校准（表 I）。在本研究中，所考虑的模型限定为 $T = 64$ 棵树，深度为 $D = 9$，因为最小目标平台（Kria）的查找表（LUT）数量和块 RAM（BRAM）深度分别有限。请注意，该设计也可扩展至更大型的现场可编程门阵列。尽管尚无系统性研究确定自适应随机森林的最佳树数量，Oshiro等人 [23] 指出，随机森林的最佳树数量在 64 至 128 棵之间，因此我们采用其下限。漂移检测器超参数：δ_W 和 δ_E 是概念漂移的警告/错误阈值；为提高吞吐量，会通过宽限期（GP）定期执行切分检查； W 是应用切分点算法所需的最小滑动窗口大小； M 表示启用压缩所需的最小桶数，桶大小为 2^i 。训练超参数： δ 是 Hoeffding 界的置信度阈值，平局阈值 τ 确保在上界 o f 周期数，GP 支持训练，属性子集大小为 SS，泊松分布为 λ。

GSARF 还具有额外的 候选树类型。为进行公平比较，前景树的数量需要保持一致，因此 GSARF 总共包含 96 棵树，使其在概念漂移恢复期间相较于其他模型具有优势。OpenWoud 避免了信息增益近似（不同于 GSARF/Vitis），从而实现更高的精度。

五、评估

本节以基准测试的形式，对 OpenWoud 进行了全面的定量评估。

A. 实验设置

为评估 OpenWoud，我们开发了一套基础设施，用于将各种演化数据流应用于自适应随机森林。

数据集	Type	样本	属性	类别
AGRAWAL	合成	1,000,000	82	2
LED	合成	1,000,000	24	10
COVTYPE	Real	581,012	44	7
KDD99	Real	494,020	24	23

表 II：合成数据集和真实数据集。

数据搬运器 直接内存访问（DMA）控制器通过 AXI HPB 接口，在自适应随机森林与主存之间批量搬运样本和投票结果，从而最大限度地减少软件开销和额外的设计复杂性（例如由外部流生成器引起的复杂性）。其读写通道通过 HP1 和 HP2 上的智能互连进行多路复用，而训练引擎使用 HP0 和 HP3 更新叶节点统计信息。片上系统的硬处理器用于：配置 DMA 传输、处理样本/投票结果以及轮询完成标志。所有 FPGA 实验均以 100MHz 的系统时钟运行。

数据集基准测试采用了两个合成数据集：Agrawal 等人 [24] 提出的 AGRAWAL 和 Breiman 等人 [11] 提出的 LED（每个数据集都包含突变、渐变和混合漂移）；以及两个真实数据集：Blackard 等人 [25],提出的森林覆盖类型（COVTYPE），以及 Hettich 等人[26]提出的用于网络入侵检测的 KDD Cup 1999 （ KDD-99 ）。这四个数据集使我们能够研究概念漂移恢复和真实场景下的分类性能，并与 FPGA基线进行对比。属性预处理通过 Wu 等人[2] 提供的转换脚本完成。

B. 面积占用

与 FPGA基线相比，OpenWoud 平均仅需使用 52% 的查找表（LUT）、24% 的触发器（FF）、77% 的数字信号处理器（DSP）和 40 % 的块 RAM（BRAM）资源，同时无需使用 Ultra RAM（URAM）（一种 Xilinx 专用构建模块）。这使其能够部署在更小型的 Xilinx 现场可编程门阵列上，而 FPGA基线无法适配这些器件（Kria 上的 4 个数据集中有 3 个无法适配，见表 III）；同时也能部署在非 Xilinx 器件上（PolarFire 上的所有数据集，见表 IV）。尤为重要的是，这种跨厂商可移植性使 OpenWoud 能够利用不同系列的内置可编程逻辑资源特性，包括广泛应用于航空航天领域的抗辐射平台，从而提高容错能力。

进一步分析资源分解可以看出，查找表和块 RAM（BRAM）是利用率最高的构建模块。查找表主要用于训练和漂移检测组件，以实现叶节点统计信息的多路复用逻辑和滑动窗口索引；而树结构和直方图存储则占用了大部分块 RAM。

此外，与 FPGA基线相比，OpenWoud 具有更强的可扩展性。在调整树数量/深度、属性和类别等超参数时，资源使用量会以可预测的方式增长。由于 FPGA基线依赖 HLS 编译指示以及由编译器驱动且不透明的调度，因此对其进行微调不够直观。

[2]https://github.com/ingako/gsarf

C. 准确率

在所有数据集上，OpenWoud 均保持或提高了精度（表 V）；与 River（中央处理器）/ GSARF（图形处理器）/ Vitis（现场可编程门阵列）相比，平均准确率差异分别如下：

- AGRAWAL（合成）：-0.97%/+6.3%/+8.45%
- LED（合成）：-7.11%/-4.56%/+6.55%
- COVTYPE（真实）：-2.93%/+12.89%/+7.31%

这一改进源于快速且精确的定点数据流，避免了信息增益近似（不同于 GSARF/Vitis）。因此，Open Woud 在保持中央处理器基线准确率的同时，仍能受益于硬件加速。

我们在合成 AGRAWAL 和 LED 数据集上评估自适应随机森林的概念漂移恢复行为，涵盖突变漂移（w1）、渐变漂移（w25k）和混合漂移（图 5 和图 6）。

在 AGRAWAL 上，OpenWoud 始终比图形处理器和现场可编程门阵列基线实现更快且更稳定的恢复。当新概念出现时，所有模型的准确率都会骤降，而 OpenWoud 能更快恢复到漂移前的水平。

在 LED 上，图形处理器在重复出现的概念上表现出优势，其候选树机制能够带来更快的恢复。相比之下，OpenWoud 的准确率下降幅度略深，但仍能稳定恢复。而现场可编程门阵列基线在渐变概念漂移下则完全崩溃。

总体而言，这些结果表明，OpenWoud 能够稳健地恢复概念漂移，同时保持其独立于厂商的特性和逐样本处理能力。

D. 推理时间

我们测量 OpenWoud 的端到端推理时间（表 V），从 DMA 批量传输启动信号开始，直到硬处理器收到 DMA 完成中断。与图形处理器（0.5 - 1.5× 加速比）和 FPGA 基线（0.2 - 0.5×）相比，OpenWoud 实现了具有竞争力的性能。

需要特别指出的是，报告中的FPGA基线并非通过硬件测量得出，而是通过仿真获得的 [9],活动平均值

SET	模型	6LUT (117120)	DFF (234240)	DSP (1248)	BRAM (144)	URAM (64)
AGR	OpenWoud	87046	24761	143	112	0
	Vitis	180974	109229	191	404	16
LED	OpenWoud	83803	26709	155	112	0
	Vitis	159825	120439	192	211	16
COV	OpenWoud	107322	29349	150	112	0
	Vitis	189813	140074	196	288	16
KDD	OpenWoud	149885	54378	140	72	0
	Vitis	231094	278763	94	64	8

表 III：OpenWoud 在 Kria 上的资源利用率报告。

SET	模型	Fab 4LUT (254k)	Fab DFF (254k)	Intf 4LUT (254k)	Intf DFF (254k)	Math (784)	微型静态随机存取存储器 (2.4k)	大型静态随机存取存储器 (812)
AGR	T64	13,691	29,704	14,292	14,292	297	128	170
LED	T64	123,572	31 ,424	14,616	14,616	297	128	179
COV	T64	151,092	35,622	14,508	14,588	297	128	176
KDD	T32	224,700	50 ,886	11,244	11,244	169	64	192

TABLE IV：OpenWoud 资源利用率报告 P olarFire。

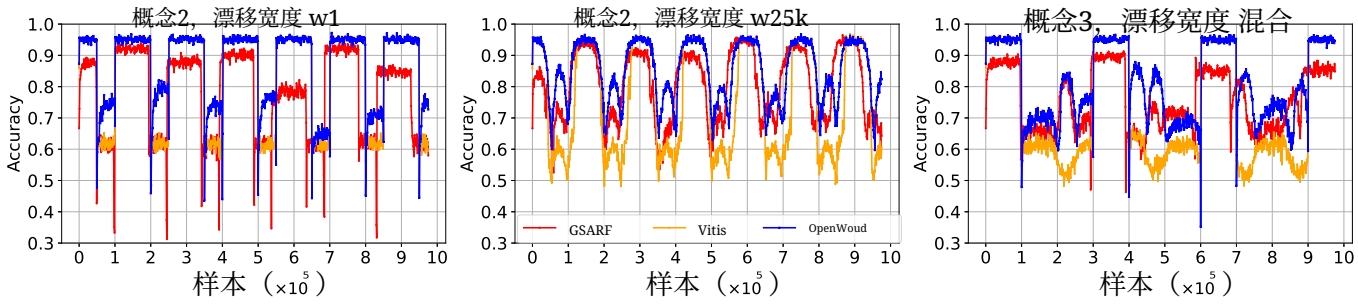


图 5：AGRAWAL（合成）漂移恢复对比：GSARF（图形处理器） [8], Vitis（图形处理器） [9] 和 OpenWoud（现场可编程门阵列）。

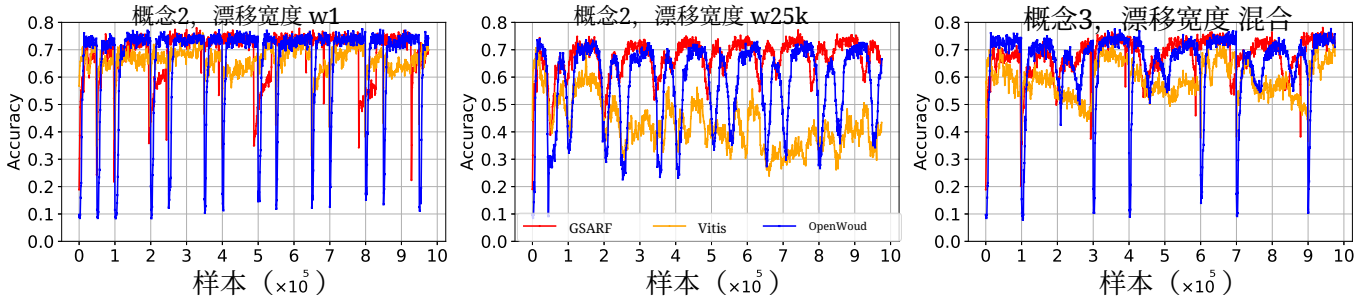


图 6：LED （合成）漂移恢复对比：GSARF（图形处理器） [8], Vitis（图形处理器） [9] 和 OpenWoud（现场可编程门阵列）。

数据集	#概念	漂移宽度	准确率 [%]				推理时间			OpenWoud 加速比	
			River（中央处理器）	GSARF（图形处理器）	Vitis（现场可编程门阵列）	OpenWoud（现场可编程门阵列）	GSARF	Vitis	OpenWoud	对比 GSARF	对比 Vitis
AGRAWAL	2	1	86.45 ± 0.26	79.04 ± 1.84	83.21 ± 0.06	86.69 ± 0.46	16.11s	9.1s	25.9s	0.6	0.4
	2	25k	83.87 ± 0.21	80.17 ± 1.08	75.67 ± 0.24	85.24 ± 0.46	22.66s	9.33s	24.26秒	0.9	0.4
	2	50k	81.28 ± 0.14	77.62 ± 0.74	68.93 ± 0.36	75.23 ± 0.53	22.99秒	9.33秒	20.34秒	1.1	0.5
	3	1	87.31 ± 0.24	74.4 ± 3.07	77.62 ± 0.34	88.99 ± 0.22	16.43秒	9.12秒	25.92秒	0.6	0.4
	3	25k	81.87 ± 0.21	73.57 ± 1.76	69.56 ± 0.27	80.55 ± 0.37	22.78秒	9.34秒	24.76秒	0.9	0.4
	3	混合	83.69 ± 0.21	76.03 ± 1.22	72.98 ± 0.1	81.95 ± 0.25	19.76秒	9.24秒	25.63秒	0.8	0.4
LED	2	1	72.87 ± 0.04	69.6 ± 1.26	65.51 ± 1.63	67.13 ± 0.25	42.99s	9.87s	42.78s	1.0	0.2
	2	25k	67.82 ± 0.05	67.32 ± 0.4	47.8 ± 2.89	58.23 ± 0.41	47.51秒	9.52秒	42.07秒	1.1	0.2
	2	50k	64.06 ± 0.15	64.15 ± 0.65	42.81 ± 3.05	57.68 ± 0.42	48.06秒	9.37秒	31.49秒	1.5	0.3
	3	1	73.16 ± 0.06	67.49 ± 0.77	65.38 ± 1.54	66.62 ± 0.22	41.34秒	9.12秒	42.95秒	1.0	0.2
	3	25k	69.54 ± 0.05	66.51 ± 0.37	55.9 ± 1.65	61.88 ± 0.21	49.37秒	9.45秒	42.15秒	1.2	0.2
	3	混合	71.55 ± 0.06	68.64 ± 0.53	59.65 ± 1.16	65.45 ± 0.17	47.34秒	9.78秒	42.3秒	1.1	0.2
COVTYPE	-	-	72.8	56.98	62.56	69.87	12.71秒	6.78秒	25.24秒	0.5	0.3

表 V：OpenWoud 性能和精度基准测试。

前景树数量进行分析推导所得。此外，该计算假设主存不会对请求进行排队；对于HBM2而言，这一假设成立，但DDR-4（Kria上可用）无法在多个接口之间实现真正的并行读写行为。此外，图形处理器需要基于批次的样本插入，而FPGA模型能够持续进行真正的逐样本推理，这对于实时应用中的演化数据流低延迟至关重要。

VI. 结论

我们介绍了 OpenWoud，这是一款不受厂商限制的 FPGA 自适应随机森林加速器。通过引入灵活的定点算术库，以及完全透明、手动调度的数据流架构，OpenWoud 克服了 FPGA基线的可扩展性限制。

我们的定量评估（基准测试）表明，与现有硬件加速器相比，OpenWoud 能够保持或提高准确率（最高提高 12.89%）。它保持了具有竞争力的吞吐量，实现了真正的逐样本处理，并具备稳健的漂移恢复能力。重要的是，在调整超参数时，OpenWoud 能够实现可预测的扩展，避免了基于高层次综合的基线中反直觉的 pragma 调优和不透明的任务调度。这些结果表明，OpenWoud 为实时在线机器学习提供了一个实用且可移植的框架。

通过成功映射到 Xilinx 和 Microsemi 的现场可编程门阵列，OpenWoud 验证了其跨厂商可移植性。这一能力使其能够利用厂商特有的可编程互连特性，例如 Microsemi 的抗辐射闪存解决方案，因此该框架尤其适用于对容错能力要求极高的航空航天研究。

展望未来，仍有若干研究方向有待探索。首先，通过在基于闪存的平台上开展抗辐射加固活动，研究自适应随机森林在航空航天应用中的容错能力。其次，探索自适应随机森林的能效；由于现场可编程门阵列加速器支持细粒度时钟门控，这一方向尤其值得期待。第三，推导数据流中时间/面积权衡的帕累托最优分辨率，并通过数据流分析进一步优化任务调度。

参考文献

[1] A. Boumeftah、O. B. Yahia、J.-F. Frigon、G. Falco 和 G. K. Kurt，《在轨干扰自适应检测：保障 GEO 卫星链路》，2025 年 1 月，arXiv:2411.16588 [eess]。[在线]。获取地址：http://arxiv.org/abs/2411 16588 。S. Salim、N. Moustafa 和 M. Reisslein，《卫星通信系统网络安全：空间段、地面段和链路段综述》，IEEE Communications Surveys & Tutorials，第 27 卷，第 1 期，第 372–425 页，2025 年 2 月。[在线]。获取地址：https://ieeexplore.ieee.org/abstract/document/10546924 [3] Z. Li、F. Liu、W. Yang、S. Peng 和 J. Zhou，《卷积神经网络综述：分析、应用与展望》，IEEE Transactions on Neural Networks and Learning Systems，第 33 卷，第 12 期，第 6999–7019 页，2022 年 12 月。[在线]。获取地址：https://ieeexplore.ieee.org/document/9451544 [4] P. Agarwal、A. A. Rahman、P.-L. St-Charles、S. J. D. Prince 和 S. E. Kahou，《强化学习中的 Transformer：综述》，2023 年 7 月，arXiv:2307.05979 [cs]。[在线]。获取地址：http://arxiv.org/abs/2307 05979 。H. M. Gomes、A. Bifet、J. Read、J. P. Barddal、F. Enembreck、B. Pfharinger、G. Holmes 和 T. Abdessalem，《用于演化数据流分类的自适应随机森林》，Machine Learning，第 106 卷，第 9 期，第 1469–1495 页，2017 年 10 月。[在线]。获取地址：https://doi.org/10 1007 10994 017 5642 8 。/s - - - W. Zhang、A. Bifet、X. Zhang、J. C. Weiss 和 W. Nejdl，《FARF：公平自适应随机森林分类器》，载于《知识发现与数据挖掘进展》，K. Karlapalem、H. Cheng、N. Ramakrishnan、R. K. Agrawal、P. K. Reddy、J. Srivastava 和 T. Chakraborty 编。Cham：Springer International Publishing，2021 年，第 245–256 页。[7] A. O. AlQabbany 和 A. M. Azmi，《衡量自适应随机森林处理大数据流中概念漂移的效果》，Entropy，第 23 卷，第 7 期，第 859 页，2021 年 7 月，期号：7，出版商：Multidisciplinary Digital Publishing Institute。[在线]。获取地址：https://www.mdpi.com/1099 4300 23 7 859 - / / / O. Wu、Y. Sing Koh 和 G. Russello，《用于演化数据流的基于 GPU 的状态自适应随机森林》，载于《2020 年国际神经网络联合会议》（IJCNN），2020 年 7 月，第 1–8 页，iSSN：2161-4407。[在线]。获取地址：https://ieeexplore.ieee.org/document/9207333 [9] F. Ridder、K.-H. Chen 和 N. Alachiotis，《使用自适应随机森林加速演化数据流的实时分类》，载于《2023 年国际现场可编程技术会议》（ICFPT），2023 年 12 月，第 232–237 页，iSSN：2837-0449。[在线]。获取地址：https://ieeexplore.ieee.org/abstract/document/10416144 [10] G. Hulten、L. Spencer 和 P. Domingos，《挖掘时变数据流》，第七届 ACM SIGKDD 知识发现与数据挖掘国际会议论文集，2001 年 7 月。[ ] 10 1145 502512 502529 在线。获取地址：https://dl.acm.org/doi/ 。 / 。L. Breiman，《随机森林》，Machine Learning，第 1 期，第 5–32 页，2001 年 10 月。[在线]。获取地址：https://doi.org/10.1023/A:1010933404324 [12] N. Oza 和 S. Russell，《在线装袋与提升》，人工智能与统计论文集，2001 年 1 月。[在线]。获取地址：https://www.researchgate.net/publication/2453583 Online Bagging and Boosting [13] A. Bifet 和 R. Gavald`a，《通过自适应窗口从时变数据中学习》，载于《2007 年 SIAM 数据挖掘国际会议论文集》。工业与应用数学学会，2007 年 4 月，第 443–448 页。[在线]。获取地址：https://epubs.siam.org/doi/10.1137/1.9781611972771.42 [14] P. M. Grulich、S. Breß、R. Saitenmacher、T. Rabl、J. Traub 和 V. Markl，《使用并行自适应窗口对数据流中的概念漂移进行可扩展检测》，2018 年。[在线]。获取地址：https: //www.researchgate.net/publication/325217048 Scalable Detection of Concept Drifts on Data Streams with Parallel Adaptive Windowing [15] H. Folmer，《使用 SDF-AP、Template Haskell、QuasiQuotes 和广义代数数据类型，通过层次化输入规范生成电路的高层次综合》，2025 年 4 月。[在线]。获取地址：https://research.utwente.nl/en/publications/high-level-synthesis-using-sdf-ap-template-haskell-quasiquotes-an [16] H. Folmer、R. d. Groote 和 M. Bekooij，《基于 Template Haskell 和 SDF-AP 的数字电路高层次综合》，2025 年 4 月，iSSN：0302-9743、1611-3349，arXiv:2504.07585 [cs]。[在线]。获取地址：http://arxiv.org/abs/2504 07585 。C. Baaij，《Clash：从 Haskell 到硬件：Christiaan Baaij 的作品 | PDF | 硬件描述语言 | Vhdl》，2009 年。[在线]。获取地址：https://www.scribd.com/document/50588123/scriptie-C-Baaij [18] S. Lahti 和 T. D. H¨am¨al¨ainen，《面向现场可编程门阵列的高层次综合——硬件工程师视角》，IEEE Access，第 13 卷，第 28 574–28 593 页，2025 年。[在线]。获取地址：https://ieeexplore.ieee.org/abstract/document/10878985 [19] E. Libessart、M. Arzel、C. Lahuec 和 F. Andriulli，《一种适用于定点平方根倒数运算算子的无缩放牛顿-拉夫逊流水线实现》，载于《2017 年第 15 届 IEEE 国际新型电路与系统会议》（NEWCAS），2017 年 6 月，第 157–160 页。[在线]。获取地址：https://ieeexplore.ieee.org/document/8010129/ ?arnumber=8010129 [20] P. K. Pandey、D. Singh 和 R. Chandel，《使用牛顿-拉夫逊除法算法的定点除法器》，载于《第五届微电子、计算与通信系统国际会议论文集》，V. Nath 和 J. K. Mandal 编。新加坡：Springer，2021 年，第 225–234 页。[在线]。获取地址：https://www.researchgate.net/publication/354484213 Fixed-Point Divider Using Newton Raphson Division Algorithm [21] L. Bangqiang、H. Ling 和 Y. Xiao，《面向随机小数点位置数据的 FPGA 上 N进制对数实现》，载于《2013 年第 9 届 IEEE 信号处理及其应用国际研讨会》，2013 年 3 月，第 17–20 页。[在线]。获取地址：https://ieeexplore.ieee.org/document/6530006 /?arnumber D. Llamocca 和 C. Agurto，《基于扩展双曲CORDIC算法的自然对数定点实现》，2006 年 3 月。[在线]。获取地址：https://www.researchgate.net/publication/230668515 A fixed-point implementation of the natural logarithm based on a expanded hyperbolic CORDIC algorithm [23] T. M. Oshiro、P. S. Perez 和 J. A. Baranauskas，《随机森林中需要多少棵树？》，载于《模式识别中的机器学习与数据挖掘》，P. Perner 编。柏林、海德堡：Springer，2012 年，第 154–168 页。[在线]。获取地址：https://link.springer.com/chapter/10.1007/978-3-642-31537-4 13 [24] R. Agrawal、T. Imielinski 和 A. Swami，《数据库挖掘：性能视角》，IEEE Transactions on Knowledge and Data Engineering，第 5 卷，第 6 期，第 914–925 页，1993 年 12 月。[在线]。获取地址：https://ieeexplore.ieee.org/document/250074/ [25] J. Blackard，《Covertype》，1998 年 7 月。[在线]。获取地址：https: //archive.ics.uci.edu/dataset/31/covertype [26] S. Hettich，《KDD CUP 1999》，1999 年 10 月。[在线]。获取地址：https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html