开放空调 ：一种基于静态随机存取存储器的开源近似存内计算编译器

Yiqi Zhou[1], JunHaoMa[1], Xingyang Li[2], Yule Sheng[1],Yue Yuan[1],YikaiWang[1],Bochang Wang[1],YihengWu[1], Shan Shen[1,*], Wei Xing[3,*], Daying Sun[1,*], Li Li[1] 和 Zhiqiang Xiao[4]1南京理工大学，南京，*210094*，中国[2]北京航空航天大学，北京，*100191*，中国[3]谢菲尔德大学，谢菲尔德，*S10 2TN*，英国[4]中国电子科技集团公司第五十八研究所，无锡，*214035*，中国

arXiv:2601.11292v1 [cs.AR] 16 Jan 2026

摘要——**数据密集型人工智能工作负载的兴起加剧了“内存墙”瓶颈。采用 SRAM 的数字存内计算（DCiM）提供了一种可扩展的解决方案，但其庞大的设计空间使手动设计难以实践，因此亟需自动化编译器。近似计算利用人工智能应用的误差容忍度，可显著节省能耗，这带来了重要机遇。然而，现有数字存内计算编译器侧重于精确算术，未能利用这一优化机会。本文介绍 OpenACM，这是首个面向基于 SRAM 的近似数字存内计算架构的开源精度感知编译器。OpenACM 弥合了应用误差容忍度与硬件自动化之间的差距。其核心贡献是集成了一个精度可配置乘法器库（包括精确、可调近似和对数型乘法器），使设计人员能够细粒度地权衡精度与能耗。该编译器可自动生成数字存内计算架构，并将晶体管级可定制 SRAM 宏与变异感知表征集成到一个基于 OpenROAD 和 FreePDK45 库的完整开源物理设计流程中。这确保了完全可复现性和易用性，并消除了对专有工具的依赖。针对代表性卷积神经网络（CNN）的实验结果表明，OpenACM 在应用精度损失可忽略不计的情况下，最多可节省 64% 的能耗。该框架可通过OpenACM:URL。**

I. 引言

数据密集型人工智能（AI）应用的增长，加剧了传统冯·诺依曼架构中的“内存墙”瓶颈。存储器与处理单元之间持续的数据传输，严重限制了性能和能效 [1]。存内计算（CiM）通过直接在存储器阵列内执行计算来应对这一挑战，从而最大限度地减少数据移动。

在各种存内计算方法中，数字存内计算（DCiM）通过将逻辑电路集成到标准 SRAM 阵列中，已成为一种可扩展且实用的解决方案 [2], [3]。然而，DCiM 的架构灵活性带来了广阔的设计空间，涵盖位单元设计、算术电路和阵列组织等方面的选择。手动探索这一设计空间耗时极长，使全面的设计空间探索（DSE）无法满足人工智能应用所要求的快速硬件迭代需求。这一挑战要求借助先进的编译器，将从高级规格到物理布局的设计流程自动化。

与此同时，近似计算为提升能效提供了一种强有力的技术。许多人工智能应用天然能够容忍轻微的计算误差 [4], [5]。通过有策略地在硬件中引入可控近似，设计人员可以用应用级精度的微小损失，换取功耗和面积的大幅降低。

现有数字存内计算编译器主要专注于精确方法 算术 [6]–[10]，缺乏对近似计算的支持，因此无法充分发挥容错型应用所带来的节能潜力。此外，当前工具无法提供精度可配置的数字存内计算架构，以适应不同应用特定的误差容忍度。许多编译器依赖专有的电子设计自动化工具链，这进一步限制了可复现性，并阻碍了社区更广泛的协同开发 [11]。

为解决这些局限，本文介绍 **OpenACM**，这是一款开源的基于静态随机存取存储器的近似存内计算编译器。OpenACM 是首个将精度感知近似计算系统性集成到全自动数字存内计算设计流程中的开源框架。借助灵活的位宽和精度可配置能力，OpenACM 提供了一个统一平台，可在不同的应用特定精度约束下探索节能型存内计算电路。我们的主要贡献如下：

- **具备精度可配置乘法器的近似数字存内计算编译器：** OpenACM 集成了一个乘法器库，其中包含三类可根据应用精度约束选择的乘法器：(i) 精确的基于4:2压缩器的乘法器；(ii) 精度可调的近似型基于4:2压缩器的乘法器；以及 (iii) 对数近似乘法器。这使得用户能够在统一的流程中细致权衡精度与能耗。

- **完整集成开源生态系统：** 后端（物理）设计流程采用 OpenROAD 数字设计流程 [12] 和 FreePDK45 [13] 实现，以确保完全可复现，且不依赖

本研究得到中央高校基本科研业务费（项目编号：30924012004 和 30925010605 ）以及集成电路与微系统国家重点实验室（项目编号：JCYQ2310803-1）的资助。

* 通讯作者。

专有依赖。SRAM 宏采用 FakeRAM2.0 风格 [14] 模板，以实现宏抽象和物理设计（PD）的无缝集成 [14]。

- **变异感知 SRAM 分析：** OpenACM 建立在 OpenYield [15]，之上，支持对 SRAM 宏进行晶体管级定制，并集成基于重要性抽样的蒙特卡洛（MC）仿真，以加速工艺、电压、温度（PVT）变化下的库表征，同时自动进行晶体管尺寸优化。

本文其余部分的组织结构如下：第 II 节回顾相关背景及相关数字存内计算编译器；第 III 节介绍 OpenACM 框架及其核心组件；第 IV 节详细介绍基于 OpenROAD 的物理设计流程；第 V 节报告布局后结果；第 VI 节总结全文并讨论未来发展方向。OpenACM 正在持续开发中，若干关键的未来改进将在第 VI 节中讨论。

II. 背景

A. 容错应用

本研究的基本前提建立在一个重要机遇之上：神经网络具有固有的误差容忍能力。与传统高性能计算不同，深度神经网络是统计模型，能够从嘈杂的真实世界数据中学习识别模式。它们的稳健性类似于人脑在光线昏暗的照片中辨认熟悉面孔的能力；要得出正确结果，并不需要完美、高保真的输入。这种韧性也直接体现在硬件层面。支撑网络推理的数百万次乘累加运算，并非每次都要求达到完美的数学精度。电路中引入少量计算误差，往往几乎不会降低最终应用精度。这为我们提供了绝佳机会，可以用不必要的精度换取能耗和硅面积的大幅节省 [16], [17]。

B. 数字存内计算编译器

自动化数字存内计算编译器 [2], [6]–[10] 显著提升了设计效率。然而，目前尚无工具支持可配置近似或全面的良率感知 SRAM 分析。 表 I 着重介绍了它们在精度、开放性和分析深度方面的主要区别。如表所示，AutoDCIM [6]开创了自动化先河，但它是闭源的。后续工作，如 ARCTIC [7], SynDCIM [8], SEGA-DCIM [9], 和 DAMIL-DCIM [10] 推进了特定方面

例如多精度支持和物理布局等方面，但它们仍是专有工具，且仅支持精确算术。OpenC² [2] 打破了开源壁垒，但仅提供基本的 SRAM 分析，且不具备近似功能。

OpenACM 的独特之处在于，它是首个系统性地将精度可配置近似与高级变异感知 SRAM 分析相结合的开源编译器。通过弥合精确计算工具与人工智能应用误差容忍度之间的差距，它为下一代节能设计提供了全面的解决方案。

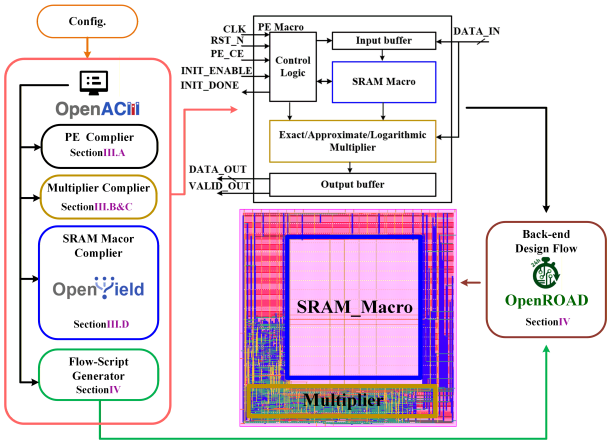

图 1：OpenACM 支持的端到端开源工作流程，由其核心硬件组件和电子设计自动化工具链构成。该流程集成了一个精度可配置乘法器库、一个可定制 SRAM 宏生成器，以及一个用于 PD 各步骤的脚本生成器。

III.: 开放空调编译器

本节概述 OpenACM 的流程，首先介绍整体架构，随后介绍编译器组件。

A. 总体架构

如图 1所示，OpenACM 是一个端到端框架，可根据架构规范和乘法器配置生成 DCiM 宏，超越了仅依靠位宽缩放进行精度控制的传统 DCiM 编译器。

OpenACM 编译器由四个主要组件组成：1) **处理单元（PE）编译器**为静态随机存取存储器和乘法器生成控制逻辑，以及相关的输入/输出缓冲区。PE 首先使用所需数据初始化静态随机存取存储器，然后将输入数据与存储的值相乘，生成最终结果。

2) **乘法器编译器**提供：(i) 任意位宽的精确乘法器；(ii) 使用可配置精度的近似乘法器，并采用近似 4-2 压缩器来优化部分积归约树；以及

表 I：自动化 DCiM 编译器概览：一项缺失的能力。

编译器	精度	打开？	静态随机存取存储器 模拟	乘数
AutoDCIM [6]	定点	✗	基本	加法树
ARCTIC [7]	整数/浮点 with BIST	✗	有限	加法树
SynDCIM [8]	多种精度 sion	✗	基本	加法树，采用4-2 压缩机
SEGA-DCIM [9]	整数/浮点精度	✗	有限	加法树
DAMIL-DCIM [10]	数字	✗	基本	加法树
OpenC² [2]	精度 数字 精度	✓	基本	加法树
OpenACM（我们提出的）	可配置 近似	✓	高级	精确方法 / 近似 4-2 压缩机 / 对数

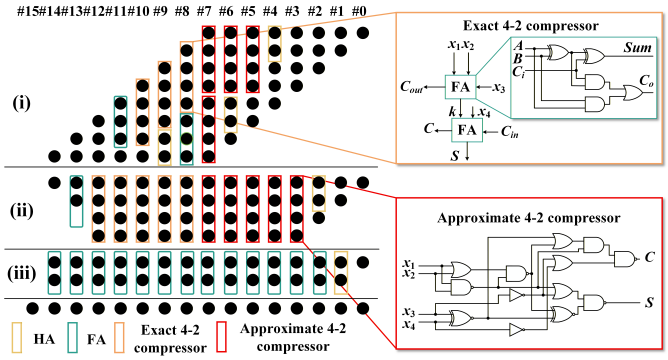

图 2：所提出的 8 位近似乘法器框图。它由三个阶段组成：(i) 部分积生成，(ii) 在选定的低位列上采用精确或近似 4-2压缩器的可配置归约树，以及 (iii) 最终的进位传播加法器。

(iii) 对数乘法器，以进一步提升能效，尤其适用于大位宽设计。3) **SRAM 宏编译器**可生成任意尺寸的 6T SRAM阵列，以及所需的控制电路和读写电路。4) **流程脚本生成器**可生成 OpenROAD 所需的后端设计脚本，并借助 OpenROAD 完成物理设计流程。该编译器最终交付经过物理实现的存内计算宏及签核文件，使设计人员无需依赖专有版图生成工具，即可满足特定应用对精度和能效的要求。

B. 近似乘法器

现有数字存内计算编译器主要采用不同位宽的定点表示，只能提供粗粒度的精度控制。然而，不同应用的误差容忍度各异，因此需要更精细的调整。为此，我们引入了近似计算技术，在给定位宽下利用近似压缩器，实现更精准的精度调节。在 SynDCIM [8], 中，有研究提出使用精确4-2压缩器来优化加法器树，但其有效性仍受限于严格的精确计算要求。相比之下，OpenACM 集成了基于近似4-2压缩器的近似乘法器，可在精度与功耗、性能和面积之间灵活权衡，并大幅拓展了数字存内计算超越精确计算的设计空间。对于基于4-2压缩器的近似乘法器，其电路结构通常如图 2所示，其中展示了一个8位乘法器。在初始阶段，黑点表示由与门对两个输入操作数进行运算所生成的部分积（PP）。这些部分积经过两个中间阶段的压缩，最终在最后阶段形成两行部分积。随后将这两行相加，得到乘法器的输出。在这三个阶段中，采用了包括半加器、全加器和4-2压缩器在内的各种组合逻辑电路来压缩部分积。为在保持精度可接受的同时优化资源消耗，通常会在部分积的低8位中应用近似4-2压缩器，具体而言是在第 #0 至 #7 列，如图 2中的红框所示。该框架具有完全可扩展性，支持任意位宽的近似乘法器。设计人员可以针对特定精度要求定制近似4-2压缩器，也可以采用广泛使用的近似4-2设计 [18]–[23]。此外，OpenACM 还支持生成任意位宽的精确乘法器，使用户能够灵活选择最符合不同应用精度要求的乘法器。

C. 对数乘法器

尽管近似 4-2压缩器乘法器提高了面积和功耗效率，但对于误差容忍度较高的应用，还可以进一步放宽精度要求。为拓展基于压缩器的近似方法之外的设计空间，OpenACM 集成了一种节能的对数乘法器（LM），可自然地提供更细粒度的精度与功耗、性能和面积权衡。

传统对数乘法器源自 Mitchell 算法（MA） [24],，该算法在对数域中近似执行乘法。对于操作数 N，可表示为 $N = 2^k(1 + x)$，其中 k 表示最高有效位“1”的位置，x 表示小数部分。此表达式也可改写为 $x \cdot 2^k = N - 2^k$。因此，两个操作数 A 和 B 的乘积可写为：

$$\begin{aligned} A \times B =& 2^{k_1+k_2} + (A - 2^{k_1})2^{k_2} + (B - 2^{k_2})2^{k_1} \\ &+ (A - 2^{k_1})(B - 2^{k_2}) \end{aligned} \tag{1}$$

其中，k_1 和 k_2 分别表示 A和 B的最高位1位置，($A-2^{k_1}$)($B-2^{k_2}$)被视为误差部分（EP），其余部分则视为近似部分（AP）。对等式两边取对数后，AP 仅通过移位和加法操作即可计算，而 EP 通常会被忽略或近似处理。忽略 EP 会引入显著误差；因此，我们提出一种无需加法器的动态补偿策略来处理 EP 项。

对于一个 n 位乘法器，与最高位1位置 k 相关的最大舍入误差（RE）可表示为：

$$\mathrm{RE}_k = (2^{k+1} - 2^k)/2 = 2^{k-1}, \quad k \in \{1, 2, \ldots, n-2\}. \tag{2}$$

最坏情况误差（WCE）取决于 EP 中选择哪个操作数进行舍入，其中 $Q_1 = A - 2^{k_1}$ 和 $Q_2 = B - 2^{k_2}$。若对较小的操作数进行舍入，则 WCE 为 $4^{n-2} - 2^{n-3}$；若对较大的操作数进行舍入，则 WCE 降至 $3 \cdot 4^{n-3}$。为使 WCE 最小化，所提出的算法会动态选择并缩放 EP 中较大的操作数，以生成补偿值。

此外，为了在不增加硬件开销的情况下实现误差补偿，所提出的补偿算法会在一个独特的范围内生成补偿值。由于操作数 Q_1 和 Q_2 是由移位后的值去除其最高位的 1 得到的，因此补偿值 round(Q_1)Q_2 的范围严格小于 $2^{k_1+k_2}$。这保证了

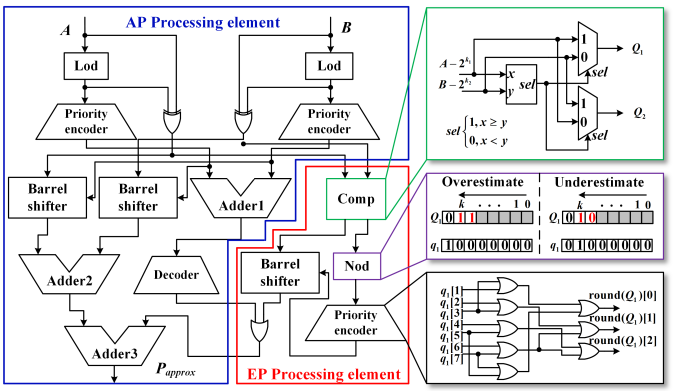

图3：所提出的8位对数乘法器框图。它通过以下步骤近似实现乘法：(i) 使用最高位1检测器和小型查找表进行二进制对数编码，(ii) 在对数域中进行加法，以及 (iii) 通过桶形移位器和查找表进行反对数解码，并可选择进行误差补偿。

将补偿值加到 $2^{k_1+k_2}$时不会产生进位。因此，可以用按位或门替代全加器，而不影响正确性。最终的近似乘积可表示为：

$$P_{approx} = 2^{k_1+k_2}|\text{round}(Q_1)Q_2+(A-2^{k_1})2^{k_2}+(B-2^{k_2})2^{k_1} \tag{3}$$

最后，图3展示了所提出的对数乘法器。在AP处理单元中，实现了两个最高位1检测器（LoD）和两个优先编码器，以确定两个操作数的最高位1位置； k_1 和 k_2 分别表示两个操作数的最高位1位置。使用异或门去除两个操作数中的最高位1，得到 $A-2^{k_1}$和 $B-2^{k_2}$；加法器1将 k_1 和 k_2 相加，并将加法器1的结果解码为 $2^{k_1+k_2}$。两个桶形移位器分别将$A-2^{k_1}$ 和 $B-2^{k_2}$ 移位 k_2 位和 k_1 位。加法器2将$(A-2^{k_1})2^{k_2}$ 和$(B-2^{k_2})2^{k_1}$相加。在EP处理单元中，对 $A-2^{k_1}$ 和 $B-2^{k_2}$ 的乘积进行估算。比较器比较两个操作数，并将较大的操作数高估为 2^{k+1} 或低估为 2^k。随后，通过移位较小的操作数即可得到补偿后的结果。或门将 $2^{k_1+k_2}$ 的部分和与EP处理单元的结果合并，再由加法器3组合中间值，得到近似乘积。该架构只需按比例扩展各模块的位宽，即可轻松适配不同位宽。

D. SRAM 宏

SRAM 宏为存内计算操作提供存储器基底。如图 4所示，OpenACM 采用紧凑的分 bank、分子阵列 6T 设计，配备分层字线（WL）解码器/驱动器、预充电（PRE）、写驱动器、可选列多路复用器和差分灵敏放大器（SA）。读操作会对 BL/BLB 进行预充电，拉高 WL，并通过 SA 锁存；写操作则在 WL 拉高时驱动 BL/BLB。该结构刻意保持精简，以便简化时序收敛，并支持跨越

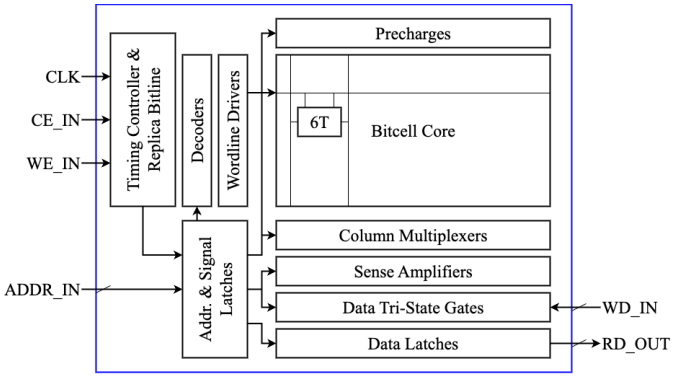

图 4：OpenACM 静态随机存取存储器宏：采用分层字线、预充电、写驱动器、列多路复用器和差分灵敏放大器的分组式、子阵列化 6T 阵列。

各 bank/子阵列的平铺布局。我们重点介绍 SRAM 宏实现中的以下特性：

1) 感知蒙特卡洛与良率的静态随机存取存储器表征（来自 OpenYield [15]）：OpenACM 运行蒙特卡洛 SPICE，考虑工艺变化（局部失配），并内置良率分析算法，以提取访问延迟、静态噪声容限（读/写/保持）以及动态功耗和漏电功耗的分布；为提高效率，它采用重要性抽样（IS）方法对稀有事件区域进行偏置，在保持目标良率下精度的同时，大幅减少所需仿真次数并加快 LIB 表征。2) 该宏支持编译器可见的细粒度可调参数，以实现协同优化：行/列、字宽、存储体/子阵列数量、列多路复用比和时序控制（如 SAE、预充电）。这些参数可直接用于结合近似乘法器开展设计空间探索，以进行精度-PPA协同设计。

3) 流程就绪性与可复现性：抽象视图和版图占用与开源流程（Open- ROAD）以及 FakeRAM 风格的 [14] 抽象宏相匹配，可在布局布线过程中实现黑盒集成；详见第 IV 节 中的流程详情。

此外，通过遵循 FakeRAM2.0 风格的抽象存储器接口，我们的静态随机存取存储器宏可以无缝集成到其他已使用 FakeRAM 宏的开源项目中（例如 OpenROAD 的 tinyRocket 教程 [25]，该教程使用 fakeram45 256x16/fakeram7 256x32），从而在更广泛的 SoC 流程中实现即插即用替换。

IV. 设计流程：OpenROAD 工具链

如图 5所示，OpenACM 通过标准数字设计流程实现电路。OpenACM 借助 Python 脚本实现电路自动生成，并支持对 SRAM 行/列数量、乘法器位宽、基于 4-2压缩器的近似乘法器以及对数乘法器等参数进行配置。对于基于压缩器的近似乘法器，用户还可以进一步指定压缩器类型以及不同近似压缩器的组合策略。一旦配置

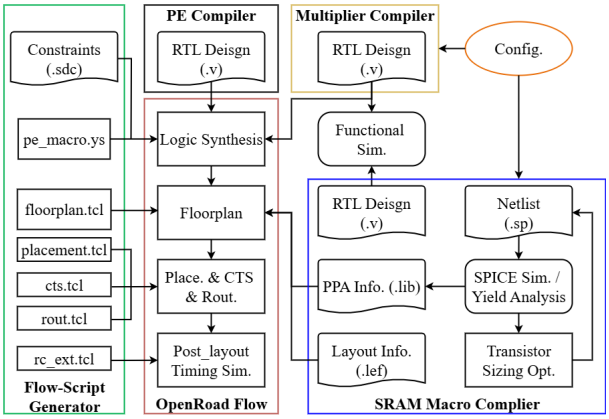

图 5：OpenACM 的总体设计流程，该流程构建于 OpenROAD 和 OpenYield 之上。

参数定义完成后，工具链将生成：(i) SRAM 行为模型；(ii) 用于物理集成的 LEF 抽象视图；(iii) 根据表征结果生成的 LIB 时序/功耗/面积视图；(iv) 用于处理单元控制逻辑和乘法器的寄存器传输级代码；以及 (v) OpenROAD 完成后端物理设计所需的脚本。需要注意的是，当前版本不会生成精确的 SRAM 布局；在布局布线过程中，SRAM 将作为黑盒硬宏进行集成。由于我们希望支持晶体管尺寸定制，SRAM 宏的 GDSII 生成仍在开发中，将留待后续工作完成。OpenROAD 实现标准单元逻辑，并围绕 SRAM 抽象视图执行顶层集成，同时使用 OpenSTA 进行静态时序分析。详细布线完成后，将对综合后的逻辑和互连（不包括 SRAM 内部结构）执行寄生参数提取（SPEF）和 SDF 生成，以进行布局后时序仿真，确保设计满足规定的功能和性能要求。

V. 实验

我们通过一组针对三个关键研究问题的实验，评估 OpenACM 的有效性。所有存内计算宏均使用 FreePDK 45 [13] 技术生成，并通过 OpenROAD 流程 [12], 从综合实现到布局。功耗和能量根据布局后开关活动仿真结果推导，而静态随机存取存储器的行为则使用 Xyce [26] 电路仿真进行表征。

A. PPA 比较

为了进行公平比较，需要注意的是，本文未报告每瓦每秒万亿次运算，因为该指标主要反映架构级吞吐量，且受互连与调度的显著影响，因此不适用于处理单元宏级评估；在这一评估中，SRAM 访问与控制是影响性能的主要因素。相反，我们给出了面积、延迟和功耗，这些指标直接体现了处理单元的电路效率，并为系统级扩展提供了更可靠的依据。表 II 展示了 OpenACM 在 100 MHz 时钟和 0.5 pF 负载下生成的 SRAM-乘法器系统的布局后结果。编译器会自动组装 SRAM 阵列、控制单元和三个算术核心：

表 II：在 100 MHz 和 0.5pF 输出负载下，OpenACM 生成的静态随机存取存储器乘法器系统的布局后性能。

SRAM	乘法器类型	延迟（ns）	面积（μm^2）			功耗（W）
			逻辑电路	SRAM	P&R	
16 × 8（8位宽度）	OpenC²	5.22	1431	7052	8483	2.82E-04
	精确方法	5.22	1079		8131	2.45E-04
	Log-our	5.22	1173		8225	2.82E-04
	Appro4-2	5.22	**939**		**7991**	**2.11E-04**
32 × 16（16位宽度）	OpenC²	5.24	4842	16910	21752	1.15E-03
	精确方法	5.24	3568		20478	1.08E-03
	Log-our	5.24	**2402**		**19312**	**6.15E-04**
	Appro4-2	5.24	2633		19543	7.58E-04
64 × 32（32 位宽度）	OpenC²	5.24	19734	48642	68376	7.00E-03
	精确方法	5.24	10132		58774	4.03E-03
	Log-our	5.24	**4960**		**53602**	**1.45E-03**
	Appro4-2	5.24	9331		57973	3.36E-03

一种使用精确4-2压缩器的精确乘法器、一种基于近似4-2压缩器的近似设计（Appro4-2），以及所提出的对数乘法器（Log-our）。由于 OpenACM 支持任意组合近似4-2压缩器，我们采用广泛使用且被大量引用的 Yang 1 [22]作为代表性示例，以展示近似存内计算编译器在能效方面的优势。为求完整，我们还纳入了一个来自 OpenC² 、基于加法树的代表性实现作为基线。

所有设计均采用相同的乘法工作负载进行评估，以确保功耗比较的公平性。结果表明，所有乘法器的关键延迟几乎保持不变（5.2 ns），这说明静态随机存取存储器主导了系统时序。近似设计实现了显著的面积缩减：对于 32×16，对数乘法器将逻辑面积减少了 33%；对于 64×32，减少了 51%。在能耗方面，4-2压缩器设计在 16×8 下最多可节省 14% 的功耗，更适合小规模场景；而对数型设计在更大规模下表现出色，在 64×32 中相比精确乘法器将功耗降低了近 64%，同时性能也优于 Appro4-2。相比之下，基于加法树的架构（OpenC² [2]）在所有配置下都表现出更高的面积和功耗。值得注意的是，在较小位宽（例如 8 位）下，对数乘法器的开销略高于基于 4-2压缩器的结构。这是因为对数和反对数估算模块在较短的数据通路中占据了相对较大的比例，从而引入了额外的逻辑复杂度和开关活动。相比之下，4-2压缩器网络在结构上更加紧凑，因此在低位宽下的面积和动态功耗方面均具有优势。随着位宽增加，对数模块的相对开销逐渐降低，使对数乘法器能够在中、大规模配置中实现显著的面积和能耗收益。

B. 精度约束应用

图像混合 [27]和边缘检测 [23] 用于评估 OpenACM 生成的近似乘法器的实际性能。在图像混合中，一个 8 位无符号乘法器逐像素处理两幅灰度图像，

表 III：不同近似乘法器在各种图像处理任务中的峰值信噪比比较。

处理任务	测试图像	乘法器类型		
		Appro4-2	Log-our	对数乘法器 [24]
图像混合	Lake 和 Mandril	67.19 分贝	32.01 分贝	26.08 分贝
	Jetplane 与 Boat	70.93 分贝	37.17 分贝	22.10 分贝
	摄影师与湖泊	69.81 分贝	43.22 分贝	24.82 分贝
边缘检测	Boat	66.21 分贝	46.43 分贝	38.77 分贝
	摄影师	67.55 分贝	45.61 分贝	38.37 分贝
	Jetplane	66.20 分贝	44.13 分贝	39.07 分贝

表 IV：近似乘法器对 ResNet-18 网络 Top-1 和 Top-5 分数的影响。

乘法器类型	Top-1	Top-5	NMED	MRED
精确方法	0.677	0.873	-	-
Appro4-2	0.668	0.880	1.70E-09	1.27E-10
Log-our	**0.680**	**0.870**	4.40E-03	1.55E-02
对数乘法器 [24]	0.610	0.842	2.79E-02	9.44E-02

并将结果缩放回 8 位。对于边缘检测，卷积和平方运算使用 16 位有符号近似乘法器，而平方根则通过精确方法计算。图像质量采用峰值信噪比（PSNR）衡量，并以精确乘法器作为基准。通常，低于 30分贝的 PSNR 表示存在可见的质量下降，而高于 40分贝则意味着图像质量几乎相同。

如表 III所示，Appro4-2 具有较高精度，可有效替代图像处理中的精确乘法器。与 LM 相比，Log-our 进一步提高了精度 [24]。在图像混合中，LM 的效果不理想，PSNR 通常低于 30 分贝；而 Log-our 始终超过这一阈值，因此适用于高能效应用。在边缘检测中，LM 的 PSNR 约为 30 分贝，而 Log-our 可达到 45 分贝，其质量与精确设计相当。总的来说，所提出的 OpenACM 架构既适用于对精度要求严格的场景，也适用于受能耗约束的场景，从而能够灵活地进行电路选择，以满足不同应用需求。

神经网络广泛应用于计算机视觉和图像分类领域，在前向推理过程中，近似乘法器往往可以替代精确乘法器。我们采用在 ILSVRC2012 数据集上预训练的 ResNet-18 [28] 模型作为基线，并测试了多种近似乘法器对该神经网络 Top-1 和 Top-5 准确率的影响。计算时，浮点权重和输入被量化为 32 位定点数。此外，我们还计算了归一化平均误差距离（NMED）和平均相对误差距离（MRED），作为近似乘法器的误差指标以供参考。

如表 IV所示，Appro4-2 和 Log-our 均显著降低了功耗，同时不影响分类准确率。具体而言，与精确乘法器相比，Appro4-2 可节省 17% 的功耗，而 Log-our 可节省 64%。得益于深度神经网络固有的误差容忍能力，这些功耗降低可在不

表 V：不同静态随机存取存储器规模下蒙特卡洛与 MNIS 良率分析方法的比较。

指标	蒙特卡洛			MNIS [29]			加速比
	P_f	FoM	#模拟次数	P_f	FoM	#模拟。	
16 × 2	1.6E-4	0.1	55,600	3.2E-4	0.05	2,985	18×
32 × 2	6.4E-2	0.17	22,900	1.7E-2	0.15	2,260	10×
64 × 2	3.9E-3	0.05	41,500	1.5E-3	0.03	4,260	9.7×

损失推理质量的情况下实现。值得注意的是，尽管 Appro4-2 引入的绝对误差较小，但其单侧分布会导致累积结果出现系统性偏差。相比之下，Log-our 产生的双向误差类似于零均值扰动，可起到噪声正则化的作用，并增强泛化能力。因此，Log-our 的 Top-1 准确率略高于精确乘法器，而 Appro4-2 仅在 Top-5 方面略有提升。传统对数乘法器的误差较大，导致准确率大幅下降，进一步验证了所提出的 Log-our 的有效性。

C. 基于重要性抽样的良率分析

表 V 比较了蒙特卡洛 (MC) 与均值平移重要性抽样 (MNIS) 的良率估计结果 [29]。为进一步加快电路级仿真，我们使用仅含两列的精简 SRAM 阵列 (N×2)，同时保留原始阵列的完整字线寄生效应。这既保留了驱动器所见的字线 RC 负载，又减少了仿真的位线列数，从而大幅缩短运行时间。表中所示的优值 (FoM) 定义为 $\mathrm{std}(P_f)/P_f$，而 $\mathrm{std}(P_f)$ 表示估计失效率的标准差 P_f。如表所示，MC 需要进行多得多的仿真（例如，对于 64×2 情形需要 41,500 次），而 MNIS 仅需显著更少的运行次数（4,260 次），便可达到相当的精度，加速比约为 9。7×。对于较小的电路，也观察到类似趋势：在18×情形下，MNIS 的加速比约为 16×2，而在 10×情形下为 32×2。此外，随着电路规模增大，MNIS 仍表现出更优的可扩展性和稳健性，进一步证明了其有效性。总体而言，这些结果凸显了基于重要性抽样的方法在良率估计方面的优势，并表明其有望在大规模电路特性表征中取代传统的 MC 方法。

VI. 结论

在本研究中，我们介绍了 OpenACM，这是首个面向基于SRAM的近似数字存内计算架构、具备精度感知能力的开源编译器。尽管 OpenACM 奠定了坚实基础，仍有若干关键扩展有待完成。近期的优先事项包括完成定制 SRAM宏的自动布局生成器，以实现完全物理感知的流程；扩展前端和乘法器库，以支持原生浮点运算。我们还计划开发自动化设计空间探索引擎，联合优化乘法器选择、精度和阵列组织，从而实现应用特定效率。我们相信，OpenACM 将继续支持并加速能效计算领域的研究。

参考文献

[1] G. Armeniakos、G. Zervakis、D. Soudris 和 J. Henkel，《深度神经网络加速器的硬件近似技术：综述》，*ACM*计算综述，第55卷，第4期，第1–36页，2022年。[2] T. Dong、S. Li、Y. Zuo、H. Jiang、Y. Ma 和 S. Huang，《Openc 2：面向数字存内计算宏单元的开源端到端硬件编译器开发框架》，载于*2025*年欧洲设计、自动化与测试会议（*DATE*）。IEEE，2025年，第1–2页。[3] K. Yoshioka、S. Ando、S. Miyagi、Y.-C. Chen 和 W. Zhang，《基于SRAM的存内计算电路综述》，《日本应用物理学杂志》，2024年。[4] C.-Y. Chen、J. Choi、K. Gopalakrishnan、V. Srinivasan 和 S. Venkatara-mani，《利用近似计算加速深度学习》，载于*2018*年欧洲设计、自动化与测试会议暨展览会（*DATE*）。IEEE，2018年，第821–826页。[5] Q. Zhang、T. Wang、Y. Tian、F. Yuan 和 Q. Xu，《Approxann：一种面向人工神经网络的近似计算框架》，载于*2015*年欧洲设计、自动化与测试会议暨展览会（*DATE*）。IEEE，2015年，第701–706页。[6] J. Chen、F. Tu、K. Shao、F. Tian、X. Huo、C.-Y. Tsui 和 K.-T. Cheng，《AutoDCIM：一种自动化数字存内计算编译器》，载于*2023*年第*60*届*ACM/IEEE*设计自动化会议（*DAC*）。IEEE，2023年，第1–6页。[7] H. Zhang、H. Zhu、S. He、M. Li、C. Wang、X. Xiong、H. Tian、X. Zeng 和 C. Chen，《ARCTIC：具备参数化整数/浮点精度和内建自测试功能的敏捷稳健存内计算编译器》，载于*2024*年欧洲设计、自动化与测试会议暨展览会（*DATE*）。IEEE，2024年，第1–6页。[8] K. Shao、F. Tian、X. Wang、J. Zheng、J. Chen、J. He、H. Wu、J. Chen、X. Guan、Y. Deng 等，《SynDCIM：一种具备多规格导向子电路综合功能的性能感知数字存内计算编译器》，载于*2025*年欧洲设计、自动化与测试会议（*DATE*）。IEEE，2025年，第1–7页。[9] H. Diao、H. Zhang、J. Song、H. Luo、Y. Lin、R. Wang、Y. Wang 和 X. Tang，《SEGA-DCIM：一种由设计空间探索引导、支持多精度的自动数字存内计算编译器》，载于*2025*年欧洲设计、自动化与测试会议（*DATE*）。IEEE，2025年，第1–7页。[10] C. Wang、K. Hu、F. Yang、K. Zhu 和 X. Zeng，《DAMIL-DCIM：一种采用数据流感知布局规划和基于混合整数线性规划的详细布局的数字存内计算布局综合框架》，载于*2025*年欧洲设计、自动化与测试会议（*DATE*）。IEEE，2025年，第1–7页。[11] AnySilicon，《开源EDA工具终极指南》，2024年，可访问：https://anysilicon.com/the-ultimate-guide-to-open-source-eda-tools/。[12] T. Ajayi、V. A. Chhabria、M. Fogaça、S. Hashemi、A. Hosny、A. B. Kahng、M. Kim、J. Lee、U. Mallappa、M. Neseem 等，《迈向开源数字流程：OpenROAD项目的首批经验》，载于第*56*届（*2019*年），2019年，第1–4页，年度设计自动化会议，可访问：https://openroad.readthedocs.io/。[13] 北卡罗来纳州立大学，《FreePDK45：开源45纳米工艺设计套件》，2024年，可访问：https://www.eda.ncsu.edu/freepdk/freepdk45/。[14] ABKGroup，《FakeRAM2.0：用于物理设计研究的伪存储器编译器》，2024年，可访问：https://github.com/ABKGroup/FakeRAM2.0。[15] S. Shen、X. Li、Z. Liu、Y. Wang、Y. Wu、J. Ma、Y. Sun 和 W. W. Xing，《OpenYield：开源SRAM良率分析与优化基准套件》，*arXiv*预印本 *arXiv:2508.04106*，2025年。[16] S. Venkataramani、S. Chakradhar、K. Roy 和 A. Raghunathan，《近似计算与提升计算效率的探索》，载于第*52*届6，2015年，第1届年度设计自动化会议，第–页。[17] V. Mrazek、Z. Vasicek、L. Sekanina、M. A. Hanif 和 M. Shafique，《深度神经网络加速器的硬件近似技术：综述》，*ACM*电子系统设计自动化学报（*TODAES*），第27卷，第5期，第1–34页，2022年。[18] O. Akbari、M. Kamal、A. Afzali-Kusha 和 M. Pedram，《用于动态精度可配置乘法器的双质量4:2压缩器》，*IEEE*超大规模集成（*VLSI*）系统汇刊，第25卷，第4期，第1352–1361页，2017年。[19] M. Ha 和 S. Lee，《采用近似4–2压缩器和错误恢复模块的乘法器》，*IEEE*嵌入式系统快报，第10卷，第1期，第6–9页，2018年。

[20] T. Kong 和 S. Li，“面向高精度乘法器的近似4–2压缩器设计与分析”，*IEEE*超大规模集成（*VLSI*）系统汇刊，第29卷，第10期，第1771–1781页，2021年。[21] A. Momeni、J. Han、P. Montuschi 和 F. Lombardi，“用于乘法的近似压缩器设计与分析”，*IEEE*计算机汇刊，第64卷，第4期，第984–994页，2015年。[22] Z. Yang、J. Han 和 F. Lombardi，“面向误差容忍乘法器设计的近似压缩器”，载于*2015*年*IEEE*超大规模集成与纳米技术系统缺陷与容错国际研讨会（*DFTS*），2015年，第183–186页。[23] A. G. M. Strollo、E. Napoli、D. De Caro、N. Petra 和 G. D. Meo，“用于低功耗近似乘法器的近似4-2压缩器比较与扩展”，*IEEE*电路与系统汇刊*I*：正规论文，第67卷，第9期，第3021–3034页，2020年。[24] J. N. Mitchell，“使用二进对数进行计算机乘法与除法”，*IRE*电子计算机汇刊，第4期，第512–517页，2009年。[25] OpenROAD项目，“OpenROAD：面向计算机体系结构师的开源ASIC设计（MICRO 2022教程代码库）”，GitHub代码库，2022年，可访问：https://github.com/The-OpenROAD-Project/micro2022tutorial。[26] E. Keiter、J. Verley、T. Russo、R. Hoekstra、H. Thornquist、T. Takhtaganov、R. Schiek、T. Mei、R. Pawlowski、K. Santarelli 等，“Xyce（TM）并行电子仿真器v. 7.6”，桑迪亚国家实验室（SNL），新墨西哥州阿尔伯克基和加利福尼亚州利弗莫尔……，技术报告，2013年。[27] F. Sabetzadeh、M. H. Moaiyeri 和 M. Ahmadinejad，“一种用于超高效近似图像乘法的基于多数逻辑的非精确乘法器”，*IEEE*电路与系统汇刊*I*：正规论文，第66卷，第11期，第4200–4208页，2019年。[28] K. He、X. Zhang、S. Ren 和 J. Sun，“面向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，2016年，第770–778页。[29] L. Dolecek、M. Qazi、D. Shah 和 A. Chandrakasan，“突破仿真瓶颈：通过范数最小化评估静态随机存取存储器”，载于计算机辅助设计国际会议（*ICCAD*）论文集。IEEE，2008年，第322–329页。