XBTorch：用于基于交叉阵列的深度学习加速器建模与协同设计的统一框架

Osama Yousuf，[1、2、3] [1,2] [3] Andreu L. Glasmann、Martin Lueker-Boden、Sina Najmaei 和 Gina C. Adam*[1]乔治·华盛顿大学[2]*DEVCOM*陆军研究实验室[3]西部数据研究院

（*电子邮件：GinaAdam@gwu.edu）

（日期：2026年1月13日）

新兴存储器技术作为深度学习应用中克服传统计算架构局限的一种潜在途径，受到广泛关注。这些技术基于电导可调且具有非易失性的纳米级器件，能够直接在内存中进行计算，因此有望大幅降低能耗和延迟，相较于传统冯·诺依曼系统尤为如此。本文介绍了 XBTorch（CrossBarTorch 的简称），这是一种新型仿真框架，可与 PyTorch 无缝集成，并提供专用工具，以准确、高效地对基于新兴存储器技术的交叉阵列系统进行建模。通过对硬件感知训练与推理进行详细比较和案例研究，我们展示了 XBTorch 如何为器件级建模、跨层协同设计和推理时容错等关键研究领域提供统一接口。示例研究采用铁电场效应晶体管（FeFET）模型，但该框架不依赖特定技术，还支持其他新兴存储器，例如电阻式随机存取存储器（ReRAM），并允许用户自定义器件模型。代码已公开发布于：https://github.com/ADAM-Lab-GW/xbtorch。

arXiv:2601.07086v1 [cs.ET] 11 Jan 2026

I. 引言

人工智能应用的指数级增长推动了神经网络架构日益复杂化，其计算需求也不断攀升。基于冯·诺依曼架构的传统计算系统面临能效和性能方面的根本性限制，这是由众所周知的“存储墙”造成的：处理单元与内存单元之间的数据传输成为主要瓶颈[1]。这一限制促使人们开始探索能够突破这些约束的新型计算范式。

存内计算是备受关注的一种计算范式[2,3]。这类系统通过直接在内存中执行计算，减少数据移动并提升效率，以克服传统冯·诺依曼架构的局限。忆阻器是实现存内计算的关键器件。凭借非易失性、低能耗以及具有历史依赖性的电导，忆阻器已成为颇具前景的器件[4]。这些特性使忆阻器非常适合加速神经网络运算，尤其是乘累加运算（MAC），这是深度学习的基础运算。通过直接在内存中执行乘累加运算，基于忆阻器的系统与传统中央处理器相比，能耗最高可降低五个数量级[5]。

多种新兴存储器技术已展现出用于神经形态计算的潜力，包括阻变随机存取存储器（ReRAM）[6]，磁性隧道结（MTJ）[7]，铁电场效应晶体管（FeFET）[8]，和相变存储器（PCM）[9]。尽管这些技术各具优势，例如能效、可扩展性或数据保持特性，但目前尚无一种忆阻器件成为神经形态硬件的明确首选。要确定最佳技术，还需进一步研究耐久性、变异性和制造可扩展性等因素[10]。

无论底层器件技术如何，忆阻神经网络都容易受到器件物理因素导致的噪声和非理想特性的影响，因此其性能会逊于全数字实现。这些挑战在训练与推理工作负载中均有体现，因此亟须在算法层面取得进展。现有关于忆阻神经网络算法研究的文献大致可分为三类。第一类聚焦于器件建模，研究人员开发高效且准确的仿真技术，用于大规模忆阻器阵列，以刻画器件级缺陷及其对神经网络行为的影响[11–13]。第二类探索硬件感知训练方法，将忆阻器件的非理想特性纳入训练过程，从而提升鲁棒性和适应性[14–17]。第三类关注推理时容错，旨在通过纠错技术和自适应计算策略，缓解器件变异性和系统级噪声导致的性能下降[18–20]。

然而，这些领域的现有研究往往针对特定技术或较为分散，缺乏能够适用于多种忆阻器技术及其独特特性的通用框架。这种碎片化阻碍了比较研究，也减缓了构建稳健、可扩展的忆阻神经网络的进程。为弥补这一不足，我们推出了 XBTorch，这是一个模块化仿真框架，旨在促进针对不同忆阻技术的算法研究。XBTorch 与 PyTorch 无缝集成，使用户无需对现有工作流程进行大幅修改，即可开展器件建模、硬件感知训练和容错研究。核心库完全采用 Python 实现，并构建于

PyTorch 之上，复用其原生向量化操作和 CUDA 加速能力，以兼容标准机器学习工作流程，并尽可能降低仿真和训练过程中的额外开销。

II. 相关工作

针对新兴存储器技术，已有众多基于交叉开关阵列的深度神经网络模拟器，可提供模拟训练和推理工作负载所需的重要功能。与我们的工作较为接近的有 NeuroSim[11]，IBM 的模拟硬件加速工具包（AIHWKit）[21]，和 CrossSim[22]。接下来的各节将简要比较这些工具与 XBTorch。

A. NeuroSim

NeuroSim 擅长系统级性能估算，可对能耗、计算延迟和面积效率进行详细建模。然而，它不支持硬件感知训练和容错技术，而这些技术对于提升基于新兴存储器的神经系统的鲁棒性至关重要。另一方面，XBTorch 实现了这些先进功能，集成了用于硬件感知训练与推理的现有方法和新方法。这一设计使研究人员能够在统一的深度学习框架内探索算法—硬件协同优化策略。不过，XBTorch 有意牺牲详细的系统级性能估算，以实现增强的建模与协同设计功能，转而专注于新兴存储器技术的算法和器件交互方面。

B. AIHWKit

IBM AIHWKit 在模拟存内计算的多个方面提供了全面支持，表现突出。然而，它没有内置的容错机制实现，因此研究人员若想研究这些技术，就必须对代码库进行大量修改。其依赖 CUDA 后端这一特点进一步加剧了这一挑战：虽然 CUDA 后端有助于加速大规模工作负载，但也限制了可定制性，并要求使用者具备 GPU 编程专业知识。相比之下，XBTorch 采用高层级的基于 Python 的架构，确保算法修改易于进行且简单直观。此外，XBTorch 原生集成了文献中最先进的容错技术，便于开展比较研究，而无需进行大幅修改。值得注意的是，近期推出的衍生版本 AIHWKit-lightning[23]，通过简化设计提升了硬件感知训练效率，这对于模拟拥有数十亿参数的大规模模型尤为重要。例如，它仅在前向传播期间应用权重噪声注入，从而优化了这一过程，这使得大型语言模型的训练速度比基础版 AIHWKIT 模拟器快一个数量级。由于训练速度更快，在大规模硬件感知训练研究中，与 XBTorch 相比，AIHWKIT-lightning 是更合适的选择。我们计划在未来版本中集成用于训练和服务大模型的优化库。

C. CrossSim

CrossSim 的独特之处在于，它允许模拟核心作为 NumPy 数组的即插即用替代品，从而能够无缝模拟模拟硬件部署，并将应用范围扩展到深度学习之外，例如通用信号处理和线性系统求解。不过，它的一项显著局限是仅支持全连接层和卷积神经网络层，这限制了其在现代深度学习架构中的适用性。相比之下，XBTorch 专为神经网络应用而设计，充分利用了 PyTorch 的模块化特性。它对 PyTorch 层进行封装，同时将每个权重视为非理想忆阻器件，从而无论底层层类型如何，都能在推理和反向传播过程中无缝建模非理想因素。

除了上述关键差异外，XBTorch 还提供了多项功能，进一步增强其实用性。它支持对网络量进行任意量化，便于研究权重、激活值、梯度和误差信号的自定义数值格式。此外，它还包含针对各种忆阻器件和系统经硬件校准的预设，以及用于评估不同器件模型保真度的专用模块。

下一节将更详细地介绍 XBTorch 的设计理念和针对各项功能的具体实现。

III. XBTORCH

图 1a 展示了 XBTorch 主要功能的概览。XBTorch 的设计理念是，有效的忆阻神经网络模拟器应同时具备准确模拟硬件约束的能力，以及支持通用算法开发的灵活性。代码库具有高度模块化的特点，并尽可能确保所有操作都与 PyTorch 对应操作保持统一。因此，熟悉 PyTorch 的用户能够快速理解并上手试用这一熟悉的应用程序接口，如图 1b 所示。下文将详细介绍具体功能和模块。

A. 器件模型

devices 模块实现了两种器件模型：解析型和 表格型。解析模型基于显式的数学公式，可明确指定器件级噪声源，例如循环间变异性、器件间变异性和非线性；而

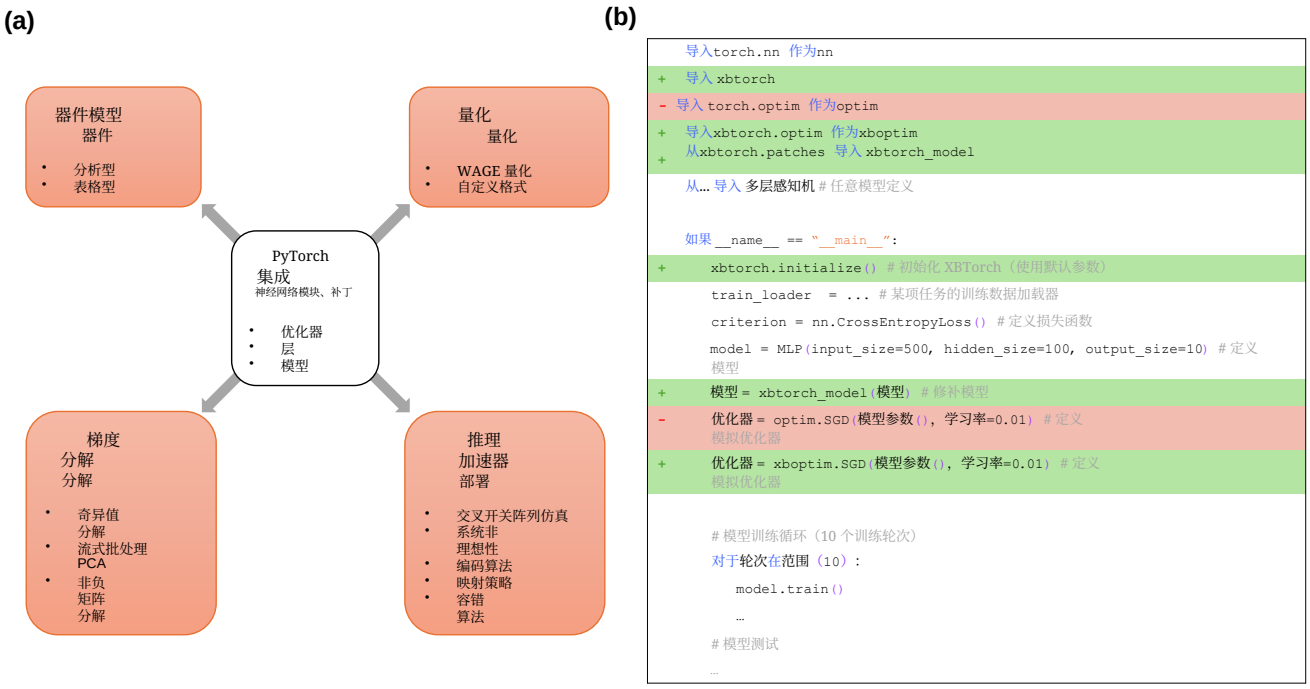

图 1. XBTorch 的功能及示例脚本对比。(a) XBTorch 的功能（以**粗体**显示）、对应的库模块（以等宽字体显示）以及重要关键功能列表。(b) 将用于训练简单神经网络的 PyTorch 脚本转换为 XBTorch 脚本的代码差异。XBTorch 只需对初始化和模型修补进行少量修改。

表格模型则基于预先填充的跳转表或查找表，以隐式方式捕捉这些噪声源。我们为这两种类型都提供了若干预设。这些开箱即用的模型无需任何定制即可使用，但高级用户仍可根据需要对各个预设进行参数化。所有预设均基于文献中报告的实验示例，或根据某种真实开关分布合成。解析模型预设的示例包括 NeuroSim[11]，中针对 ReRAM 器件的真实和理想器件模型；表格模型预设的示例则包括先前研究[17,24]中针对 ReRAM 和铁电场效应晶体管器件的基于查找表的模型。此外，我们还提供了真实解析型器件预设的表格型等效模型，作为参考。我们在公共代码仓库中比较了这两种模型随网络深度变化时更新神经网络权重所需的总时间。

devices 模块还实现了用于创建自定义器件模型的函数。用户只需提供初始电导，以及施加电压脉冲后电导变化的数据集，即可借助先进的基于克里金法的插值方法[25]，轻松创建具有任意响应特性的自定义器件模型。自定义模型可以加载并用作现有预设的直接替代项，从而例如研究自定义器件的网络训练性能。我们还在公共代码仓库中提供了有关使用和自定义器件模型的详细示例。

使用 XBTorch 器件模型进行权重更新的过程可概括如下：每个可训练参数 ter θ_{ij} 会转换为等效的器件电导 G_{ij}，而损失函数 $\nabla_\theta L_{ij}$ 对应的反向传播梯度则会转换为整数脉冲数，具体如下：

$$G_{ij} = \left(\frac{\theta_{ij} - W_{\min}}{W_{\max} - W_{\min}}\right)(G_{\max} - G_{\min}) + G_{\min}, \quad (1)$$

$$p_{ij} = \boldsymbol{round}(p_{max} \cdot \nabla_\theta L_{ij}), \quad (2)$$

其中，p_{max} 是缩放参数，定义为将器件完全置位或复位所需的脉冲总数；$\boldsymbol{round}(\cdot)$ 表示随机舍入。随后，通过器件模型将 p_{ij} 个脉冲施加到初始电导为 G_{ij} 的器件上，以更新电导；最后再将其转换回权重域 θ_{ij}，用于下一轮次（使用式 1 的逆运算）。

B. 硬件感知训练

所有采用 XBTorch 器件模型的训练实验都具有硬件感知特性，因为权重更新带有噪声，其整体噪声与所建模底层器件中的开关噪声相对应。为了对系统级非理想因素进行建模，这些带噪更新可以

与其他噪声注入技术结合使用，这些技术可以是显式的（例如，向网络量显式注入高斯噪声），也可以是隐式的（例如，对网络量进行量化、将网络梯度分解为低秩近似等）。XBTorch 器件模型可与此类注入技术结合使用，以实现和研究更先进的硬件感知训练方案。

XBTorch 提供了一种直接实现此类技术的方法，开箱即用地支持 WAGE 量化[26] 以及多种梯度分解方法（详见第 III C 节）。WAGE 量化是一种量化感知训练范式，其中所有网络量，即权重（W）、激活值（A）、梯度（G）和误差（E），都被离散化为低位宽整数。从忆阻神经网络的角度来看，2-8-8-8 WAGE 配置尤为重要，因为它能够生成易于映射到新兴存储器件阵列的三值权重网络（即使器件的可调性有限），并且能够抵抗系统级噪声源[17]。

激活值和误差的原语由量化模块处理，而权重和梯度的原语则通过封装 PyTorch 优化器，由补丁模块处理。这种分离是 PyTorch 中前向传播和反向传播分别进行这一机制的自然结果。XBTorch 保持了简单且可扩展的接口，供用户实现自己的量化函数。它还集成了 QTorch [27]，，以及 HuggingFace 的 transformers[28] 和 EleutherAI 的 lm-eval评测工具[29], 等常用的大语言模型（LLM）服务与评测工具，从而支持为所有网络量指定完全自定义的数值格式，促进忆阻神经网络量化方案的研究，并在模拟噪声源下实现可扩展、标准化的大语言模型基准测试。

在 XBTorch 中，我们选择在权重更新期间通过器件模型准确模拟器件级噪声源（如第三节A所述）。这与 NeuroSim 等经典模拟器中的器件模型行为类似，但不同于 AIHWKIT-lightning 等现代模拟器中的噪声注入操作；后者会在更高层次上近似权重更新噪声，以缩短训练时的模拟时间。因此，XBTorch 的硬件感知训练结果能够更具代表性地反映硬件实现中可达到的网络性能，尤其是对于参数过度冗余程度不高的小规模神经网络实验。我们通过确保硬件感知训练期间的器件模型为无状态而非有状态 （详见第三节E）来缩短模拟时间。

C. 梯度分解方法

训练忆阻神经网络需要大量内存和计算资源，主要是因为它涉及生成并平均在训练数据批次上通过外积构建的大型梯度矩阵。低秩近似通过压缩梯度信息，为减轻这一负担提供了有效策略，从而进一步减少整体数据占用。给定权重矩阵 θ，损失函数L 在批次B 上的全秩梯度可以近似为一个秩较低的k 的矩阵 $\nabla_\theta^{(k,B)} l$，使得：

$$\nabla_\theta^{(k,B)} l = \Delta \cdot \Sigma \cdot X^T, \tag{3}$$

这种精简在分布式计算环境中尤为有利，有助于降低通信开销和能耗[30]。对于忆阻神经网络而言，这也很重要，因为压缩梯度需要更少的器件写入；鉴于器件的写入限制，这能带来显著益处[31]。即使在更传统的 CMOS 系统中，此类压缩技术也能在训练过程中显著节省能耗[32]。

XBTorch 通过分解模块实现易于扩展的基本功能，抽象掉网络层面的细节，从而便于研究梯度分解方法（用户只需提供如何分解梯度矩阵的细节，其余工作均由 XBTorch 管理）。XBTorch 实现了流式批处理主成分分析（SBPCA）、非负矩阵分解（NMF）以及基于奇异值分解（SVD）的压缩技术，用于梯度分解。

若与硬件感知训练结合使用，操作顺序如下：误差量化、梯度量化、梯度压缩，以及通过器件模型和压缩梯度进行电导更新。此操作顺序体现了一种模拟阵列与数字协处理器相连接的硬件架构：通过模数转换器（ADC）读取电流测量值，由协处理器计算低秩梯度更新，再通过数模转换器（DAC）将更新应用回阵列。

D. 损失景观

损失景观[33] 提供了一种可视化给定神经网络在其参数受到扰动时表现的方式。

$$f(\alpha,\beta) = L\left(\vec{\theta}_i + \alpha\frac{\vec{\delta}}{\|\vec{\delta}\|} + \beta\frac{\vec{\eta}}{\|\vec{\eta}\|}\right), \tag{4}$$

其中，$\vec{\theta}_i$ 只是对应于训练轮次 i 后（未进行修改的）网络参数的列向量；$\vec{\delta}$和 $\vec{\eta}$ 表示方向向量（其维度与 θ_i相近），这些向量既可以随机选择，也可以通过某种结构化方法选择；α、β 是缩放参数，用于决定施加于网络的整体扰动程度。对于 α 和 β的某些取值，将扰动施加到网络参数上，然后在整个训练集上计算损失函数 L 。为了可视化完整的损失景观，可以在 α和 β取值构成的网格上计算公式 4 中的量。

损失景观还可用于可视化参数优化轨迹。然而，由于已知神经网络的优化轨迹位于维度极低的空间中，因此需要有意义的方向向量 $\vec{\delta}$和 $\vec{\eta}$。[33] 中的作者表明，如果将 $\vec{\delta}$和 $\vec{\eta}$ 设为参数演化矩阵的前两个主成分，则可以有效利用优化轨迹，该矩阵定义为

$$\boldsymbol{M} = [\vec{\theta}_0 - \vec{\theta}_n, \ldots, \vec{\theta}_{n-1} - \vec{\theta}_n]. \tag{5}$$

确定主成分后，可通过简单的点积运算将 $\boldsymbol{M}$ 投影到这些主成分张成的子空间中，从而直接有效地可视化网络参数的优化轨迹。

以这种方式可视化损失景观和优化轨迹，能够深入了解忆阻神经网络的行为与性能。这些系统固有的非理想因素——例如器件级漂移、读取噪声和编程变异性——可有效建模为参数扰动，类似于式（4）中所述的扰动。此外，这类可视化有助于洞察网络优化的有效维度，因此可用于分析器件—网络交互。它们还能用于比较不同器件模型或硬件感知训练策略，尤其是在噪声鲁棒性和参数扰动方面。

XBTorch 损失模块提供了一系列工具，可简化忆阻神经网络损失景观和参数优化轨迹的计算与可视化。公开代码库中还包含一个详细的示例笔记本。

E. 硬件感知推理

部署模块负责处理与硬件感知推理相关的所有事务。该模块适用于以下场景：用户希望评估一个预训练网络（无论是否经过硬件感知训练）在部署到具有器件级和系统级噪声源的真实硬件系统后，其推理性能将如何表现。该模块维护底层交叉阵列的有状态表示，我们将其称为底层推理加速器。有状态表示意味着推理加速器始终维护完整的交叉阵列。该加速器由交叉阵列的尺寸参数化。如果需要将预训练网络映射到加速器上进行硬件感知推理，用户还必须指定权重编码 算法，用于描述如何将软件权重编码为器件电导；指定电导映射算法，用于规定应将编码后的电导映射到加速器的何处；以及指定输出轮询 参数，用于规定应如何汇集来自（可能是多个）映射后的层实例的输出电流。由于采用了这种表示方式，如果加速器中的器件数量不足以映射某个神经网络，就可能耗尽可用器件来得到网络解。这种有状态表示不同于 XBTorch 的器件建模功能中的无状态表示，后者在 III A 节中有所介绍；在该表示中，无论网络参数的数量是多少，都假定器件始终可用。

有状态表示支持研究映射算法、编码算法，以及二者的组合，以用于面向硬件的容错算法。大多数模拟器在硬件感知推理中采用无状态表示。尽管这种方法有助于缩短仿真时间，但它存在根本性局限，因为无法用于研究推理时的容错算法。与构建用于训练的忆阻神经网络模拟器相比，对这类算法的研究在一定程度上彼此割裂。XBTorch 在硬件感知推理过程中维持有状态表示，提供了一个统一接口，能够有效衔接这些研究领域。XBTorch 原生实现了具有内部容错的映射算法[20], 层集成平均算法[18], 和委员会机算法[19] ，以支持探索性研究或比较研究。

除了用于层级映射和推理的有状态交叉开关阵列仿真外，XBTorch 还支持对大型语言模型（大语言模型）进行硬件感知评估。由于大语言模型参数数量通常达到数十亿级别，为每个参数维护一个完整的有状态加速器实例会消耗过多内存，因此 XBTorch 以无状态模式实现大语言模型的硬件感知评估：XBTorch 不会为每个模型参数实例化持久化的交叉阵列分配，而是按需流式处理权重到电导的编码，并在每次推理时执行仿真读取和含噪乘加计算。这种无状态方法降低了峰值内存使用量，同时在推理循环中保留了关键的模拟非理想性（读写噪声、模数转换器/数模转换器量化、电导重叠等），从而能够以可控的资源成本，真实地评估大语言模型在交叉开关阵列加速器上的行为。为便于采用标准基准测试工作流，XBTorch 提供了与常见的大语言模型服务和评测工具的适配器（例如，HuggingFace 的 transformers[28] 可用于灵活且注重内存效率的推理，EleutherAI 的 lm-eval评测工具[29] 可用于标准化任务评估）。

与器件模块一样，我们在公共代码仓库中提供了加速器预设，并附有关于硬件感知推理的详细示例，从而确保用户和器件研究人员能够将其扩展到自定义平台。我们还提供了一个关于上述推理时容错方案的比较示例。

IV. 结果

为了展示 XBTorch 的功能以及可以开展的算法研究类型，我们以一个简单的多层感知机网络为例，进行案例研究；该网络用于对 MNIST 数据集中的手写数字图像进行分类。该网络架构和所有算法超参数的更多详情

见附录 B。公共代码仓库中提供了详细的示例笔记本和指南，可用于复现所有报告的结果。

A. 硬件感知训练

XBTorch 的硬件感知训练功能可用于比较原始软件网络与器件网络的训练性能，也可用于比较采用不同器件模型训练的两个器件网络。损失景观可视化也有助于进行这些比较，因为它们可用于评估不同硬件感知训练方案的鲁棒性，或一般情况下的器件—网络交互。

图 2 比较了软件基线网络与采用低变异性表格化 FeFET 模型的器件网络的硬件感知训练结果，并采用了 2-8-8-8 设置的 WAGE 量化。该硬件感知训练方案与[17]，中所述方案类似，并能生成鲁棒的三值权重网络，即使器件可调性受到严格限制，也能轻松部署到新兴存储器件阵列中。完整的超参数列表见附录 B

可以看出，软件网络的性能始终优于硬件感知网络。这直接源于硬件感知训练范式中隐式和显式噪声注入操作的程度（含噪权重更新、网络量化量）。这些噪声源本质上起到了正则化作用。通过仅扰动第一个全连接层来计算相应的损失景观，因为该层规模最大，因此与后续层相比，很可能在训练期间积累了更多信息。除底层器件模型外，所有随机性来源均保持不变。值得注意的是，与软件基线相比，硬件感知网络的轨迹位于更高维的空间中：在硬件感知情况下，两个主成分仅能解释 $\approx 60\%$ 的方差，而软件情况下为 80%。这表明器件级和系统级噪声源会向反向传播梯度中注入噪声，使器件网络更难训练。优化轨迹显示，这两个网络收敛到了截然不同的局部极小值。由于除底层器件模型外的所有随机性来源均保持不变，我们得出结论，这直接由器件级开关行为所致。

B. 梯度分解方法

图 3a 展示了一个低秩梯度分解示例的可视化结果，该分解将梯度拆分为左右奇异向量矩阵，如前文公式 3 所述。对于一个 $m \times n$ 梯度，秩为 k 的近似需要存储 $O(k(m+n))$ 个元素。因此，对于大型矩阵，当 $k << min(m,n)$ 时，可以节省大量内存。

我们先使用 SBPCA，再使用 NMF，以逐渐增大的秩重新训练相同的网络架构。与图 2 相比，我们选择了变异性更高的 FeFET 器件模型，且不采用 WAGE 量化。这些选择仅为展示 XBTorch 所提供的多种选项。完整的超参数列表见附录 B。两种算法的网络训练结果分别汇总于图 3b 和图 3c。

我们发现，这两种算法的性能差异很大，尤其是在低秩情况下。这是因为它们生成的分解各具特点，从而导致器件—网络交互存在差异。SBPCA 具有振荡特性，因为即使对于严格为正的梯度矩阵，它也能生成含负值的分解。相比之下，NMF 受非负约束的限制，始终生成加性特征。在秩为 1 和 2 时，SBPCA 的表现优于 NMF，因为在训练时间相近的情况下，它能将网络训练到更高的准确率；但在秩为 4 及以上时，两者的表现趋于相近。尽管存在这一差距，人们仍可能选择使用 NMF 而非 SBPCA，因为其加性特性可以减少逐秩更新之间的重叠；对于非线性、保持能力受限的器件，这可能有所裨益，因为振荡更新会造成严重的不利影响。XBTorch 提供了一个统一接口，用于研究此类分解算法如何与器件特性及神经网络性能相互作用。

C. 硬件感知推理与容错方案

在进行硬件感知推理时，XBTorch 允许用户定义模拟交叉阵列的整体尺寸。还可以指定会引入噪声的系统级参数，例如 ADC/DAC 位精度或卡滞百分比。根据这些设置，XBTorch 使用第三节E中所述的加速器原语，模拟具有状态的交叉阵列表示。随后，可以采用选定的权重（权重到电导）编码算法和交叉阵列（电导到器件）映射算法，将预训练的解写入加速器实例。

为展示这些功能，我们使用图 2 中的硬件感知训练方案，在尺寸为 2500×2500 的模拟交叉阵列上演示硬件感知推理。我们采用差分权重编码方案，将每个权重矩阵 θ 编码为两个电导矩阵，G^{pos} 和 G^{neg}，使得 $\theta \propto (G^{pos} - G^{neg})$；同时采用随机交叉阵列映射算法，将电导矩阵映射到模拟交叉阵列上随机且互不冲突的位置。调节器件时，会模拟写入噪声；用户可以从多种分布中进行选择，以匹配所需的器件物理特性。最后，在推理过程中，归一化数据集输入会转换为电压，并按指定的 V_{read} 电压进行缩放。随后，通过执行显式含噪读出，并对输入电压与器件电导进行向量-矩阵乘法，将这些电压施加到两个差分映射电导矩阵的输入线上。由此生成与差分映射电导相对应的差分输出电流，

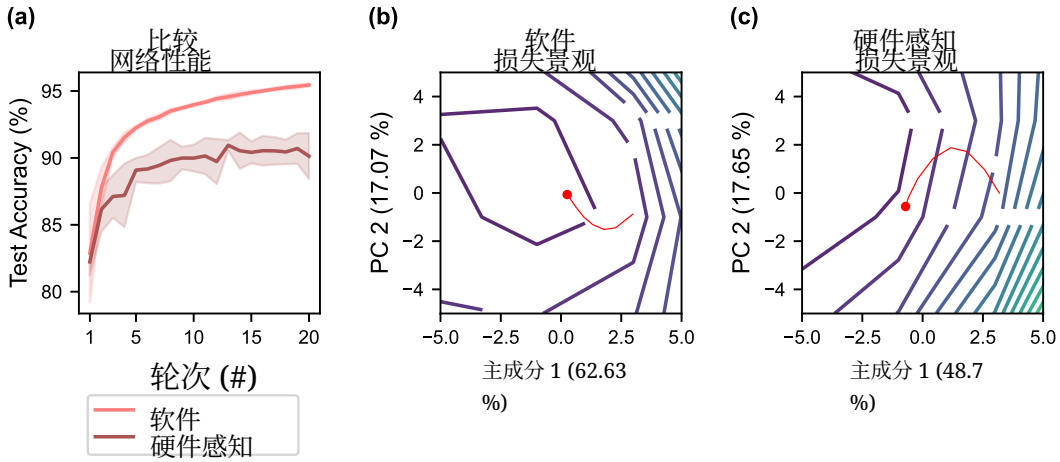

图 2. 比较使用 XBTorch 中的低精度硬件感知训练在 MNIST 上训练的多层感知机网络，与使用原生 PyTorch 训练的全精度软件基线。（a）10 次独立运行的网络测试准确率平均值。软件基线网络（b）和硬件感知网络（c）的损失景观。损失景观坐标轴对应于参数演化矩阵 M 上的主成分，括号中给出了所解释的方差。优化轨迹以红线表示，并终止于圆形标记处。硬件感知网络采用 2-8-8-8 WAGE 配置进行训练，该配置生成三值权重网络，并使用表格型铁电场效应晶体管器件模型，其 $V_{gs} = 0.9\,V$ 和 1% 变异性（见图 A.1）。

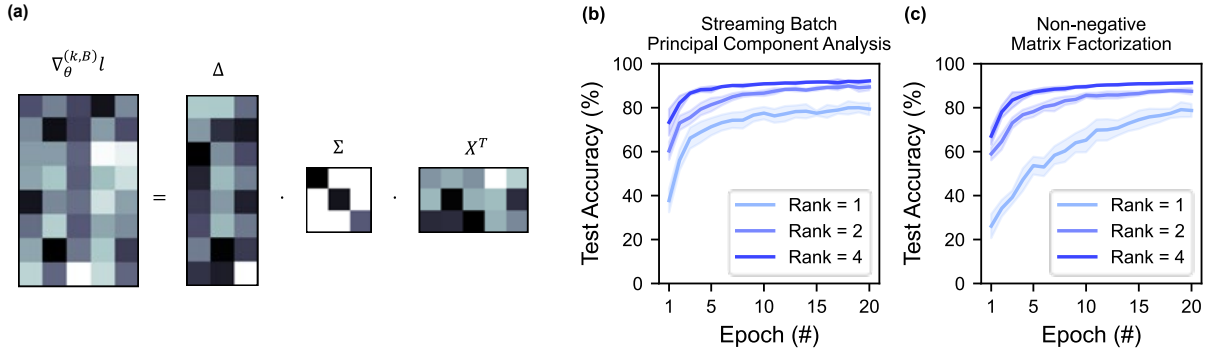

图 3. 使用 XBTorch 展示梯度分解方法。（a）根据公式 3，将假设的梯度矩阵分解为低秩 k 表示的示例。当k 较小时，与全秩梯度矩阵相比，分解形式的内存开销更低。在不同秩下，SBPCA（b）和 NMF 算法（c）的网络测试准确率为 10 次独立运行的平均值。

与此类电导相对应。与写入阶段一样，也会模拟读取噪声；用户可以指定噪声水平，以模拟含噪的模数转换器/数模转换器读出。根据差分电流，通过计算二者之差并进行适当缩放，得到最终输出。此操作以及任何非线性激活函数都不包含显式噪声源。因此，XBTorch 假设这些操作是在独立于由忆阻器交叉阵列组成的模拟加速器的数字协处理器中执行的。完整的超参数列表见附录 B

图 4a 显示了写入操作后器件电导的读出结果，即电导图。两层以不同颜色标示，并以插图形式展示了一个放大的区块，以便清晰查看。图 4b 显示了概率分布第一网络层中 G^{pos} 个器件。电导状态之间的重叠，直接源于模拟加速器读写操作期间的噪声源。最后，图 4c 展示了该预训练解的硬件感知推理性能随 ADC/DAC 位精度变化的情况（保持所有其他噪声源不变）。不出所料，随着位精度提高，推理性能也随之提升。我们还提供了基线准确率估计作为参考；该估计对应于完全理想加速器（即不存在噪声源）上的硬件感知推理性能。即使 ADC/DAC 位精度达到最高，硬件感知推理性能仍落后于这一理想基线，这是因为其他噪声源的存在导致了图 4b 中可见的电导重叠。

供参考，本实验所用超参数列表见附录 B。

容错算法是 XBTorch 的另一项关键特性。有效的容错算法能够在存在严重非理想因素时提升忆阻神经网络的推理性能，这些因素既包括器件级因素（如固定故障），也包括系统级因素（如 ADC/DAC限制）。XBTorch 中的有状态表示以及对编码和映射阶段的分离，使用户能够在任何现有的硬件感知推理应用之上无缝指定容错算法。图 5a 展示了使用容错算法时，各个组件如何汇入 XBTorch 硬件感知推理引擎。用户既可以选择预先实现的容错算法，也可以借助精简的预设设计方法轻松实现并使用自定义算法。

为展示这一特性，我们以层集成平均（LEA）[18] 作为容错算法，展示相关结果。在 LEA 中，需要指定一个冗余参数，用于决定每个编码后的电导矩阵在加速器交叉开关阵列上的映射次数。推理过程中，每层的两个差分编码电导矩阵所产生的电流，会在冗余映射之间取平均值。随后，这些平均电流将用于后续计算。平均处理能够降低神经网络层映射到含噪器件交叉开关阵列时产生的映射误差，进而提升网络的整体性能。

图 5b 展示了 LEA 的硬件感知推理性能如何随冗余度（每个电导矩阵的映射次数）和卡滞百分比（2500×2500 中固定高电平或固定低电平的器件所占比例）的增加而变化。在任意给定的冗余级别下，固定故障的比例越高，网络的推理性能就越低。这是意料之中的，因为卡滞器件增多会直接加大网络的映射误差。在固定的固定故障比例下考察冗余度逐渐增加时，容错算法的优势便显现出来。在固定故障最严重的情况下，即 20% 时，LEA 可将网络性能从 $< 40\%$ 提升至 $> 80\%$，与不存在固定故障时的情况相比，性能差距不到 5%。图 5c 展示了一个电导图，与图 4a 类似，对应冗余级别为 6、卡滞百分比为 20% 的情况。如图所示，每一层的所有冗余映射均以有状态的方式表示，从而能够准确模拟真实系统中的忆阻神经网络推理。我们认为，这是开展推理时容错算法研究的重要前提。

D. 语言模型评估

我们在一组标准自然语言基准测试上评估了硬件感知、无状态的大语言模型推理流程。表 I 报告了 TriLM模型系列[34] （性能出色的三值权重网络）中两个不同规模模型在不同 ADC/DAC 定点精度下的任务级准确率。生成这些结果所用的所有加速器和推理运行参数（器件电导范围、读写噪声水平、读取电压等）均列于附录 B。实验表明，大语言模型任务性能对 ADC/DAC 精度十分敏感，并凸显了 XBTorch 所捕获的模拟非理想性可能如何导致端到端任务性能下降，以及容错能力或更高位数 ADC/DAC 硬件所带来的潜在提升空间。另一个值得关注的现象是，在所有评估任务中，较大规模的模型相比小规模模型，对模拟噪声的抵抗力天然更强。得益于无状态表示以及与 transformers 和 lm-eval 的原生集成，研究人员可以轻松替换模型、评估任务，以及底层加速器参数，并使用自定义测量数据，以节省内存的方式获得准确的性能估算。附录 C 简要介绍了表 I 中使用的评估任务。

V. 结论

我们介绍了 XBTorch，这是一个旨在简化忆阻神经网络算法研究流程的框架。XBTorch 致力于弥合该领域现存的差距，并促进更广泛的合作。尽管该框架提供了一套连贯的功能，但仍存在一些局限，尤其是在硬件感知的大模型训练方面。为此，我们正积极致力于开发用于训练的快速噪声近似方法，将其作为第一步。我们希望 XBTorch 能成为一项由社区推动的工作，促进忆阻神经网络算法研究的发展，并推动整个神经形态计算领域的进步。

致谢

Osama Yousuf 的工作由美国陆军研究实验室赞助，并根据合作协议编号 W911NF-25-2-0073 开展。本文所述观点和结论仅代表作者的意见，不应被视为美国陆军研究实验室或美国政府明示或暗示的官方政策。尽管本文包含版权声明，美国政府仍有权为政府用途复制和分发本文的再印本。

数据可用性声明

本研究所有研究结果相关的数据，均可通过 XBTorch 代码库中提供的示例生成：
https://github.com/ADAM-Lab-GW/xbtorch。

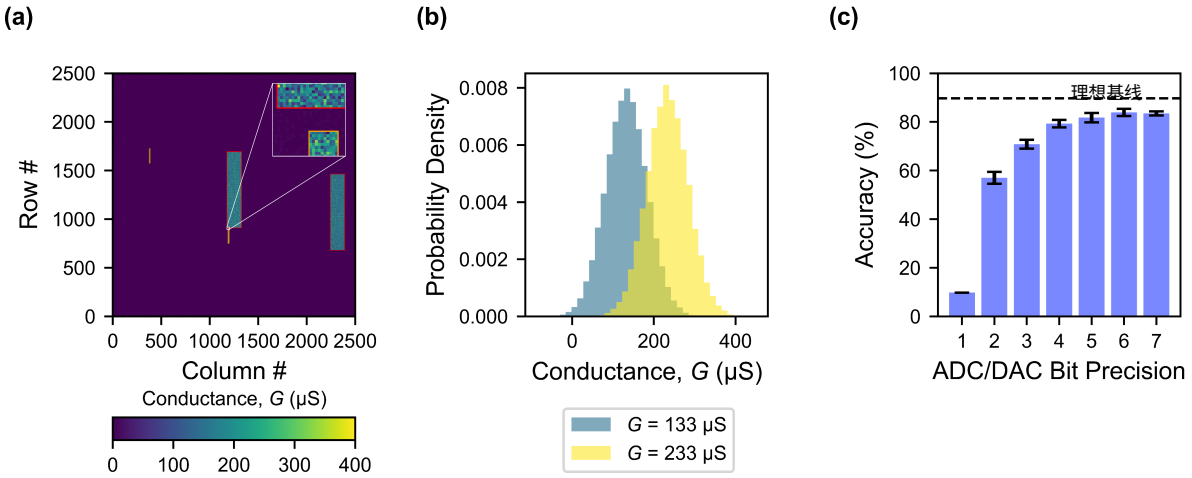

图 4. 使用 XBTorch 进行硬件感知推理。(a) 模拟的 2500×2500 交叉阵列具有真实噪声，显示了写入训练后的解（来自图 2）后的电导图。每个权重矩阵 θ 均采用差分编码，分别映射至G^{pos} 和G^{neg}，并放置在随机且互不重叠的位置。红色/橙色方框标记第一层/第二层网络层；深色/浅色分别表示G^{pos}/G^{neg}。插图显示了一个放大区域。读出期间，未使用的器件会被禁用。(b) 第一层G^{pos} 的实际电导分布。(c) 推理时测试准确率随 ADC/DAC 位精度的变化。虚线表示不存在噪声源时的理想基线。柱高为 10 次独立迭代的平均值。误差线表示 3 个标准差。

模型	ADC/DAC 位数	ARC-Easy (10-shot)	ARC-Challenge (10-shot)	BoolQ (0-shot)	PIQA (0-shot)	Winogrande (5-shot)	平均值 ↑	标准误
TriLM-830M	4	25.93	21.84	41.16	53.21	53.20	39.07	0.50
	8	53.87	22.95	61.59	65.94	53.04	51.48	0.51
	12	56.31	24.49	60.58	68.23	53.75	52.67	0.51
	16	56.23	24.57	60.49	68.23	54.06	52.72	0.51
TriLM-3.9B	4	25.97	22.61	42.26	51.14	52.33	38.86	0.51
	8	61.91	30.12	61.68	67.79	57.06	55.71	0.52
	12	68.18	34.98	67.61	73.83	64.96	61.91	0.50
	16	67.80	34.90	67.52	73.94	65.04	61.84	0.50

表 I. 使用 XBTorch 推理加速器，在不同 ADC/DAC 位精度下 TriLM 模型的性能。报告的数据为多个基准测试上的准确率（%），最后两列分别显示平均性能及其标准误差。

附录 A：器件模型预设

XBTorch 提供多种器件模型预设。例如，使用铁电场效应晶体管器件的表格模型时，可将多种器件间变异性水平和栅源电压V_{gs} 值指定为参数。随着这些参数变化，器件的整体开关噪声也会随之变化，进而影响神经网络级性能。图 A.1 展示了可用 FeFET 器件模型在$V_{gs} = 0.9V$ 条件下，随着变异性水平增加，其置位阶段和复位阶段对应的表格数组。

附录 B：超参数

对于所有非语言模型结果，我们使用了一个简单的两层感知器网络，其中包含 784 个输入神经元和 150 个隐藏神经元，以及 10 个输出神经元。我们使用交叉熵作为损失函数。输入数据集经过转换后具有零均值和单位方差。在训练过程中，数据以大小为 4096 的随机打乱小批次输入网络。各项实验的超参数列于下表中。

附录 C：语言模型评估数据集

以下任务用于构建表 I 中报告的语言模型评估。对于每项任务，我们都使用了通过 lm-eval 提供的开源实现。

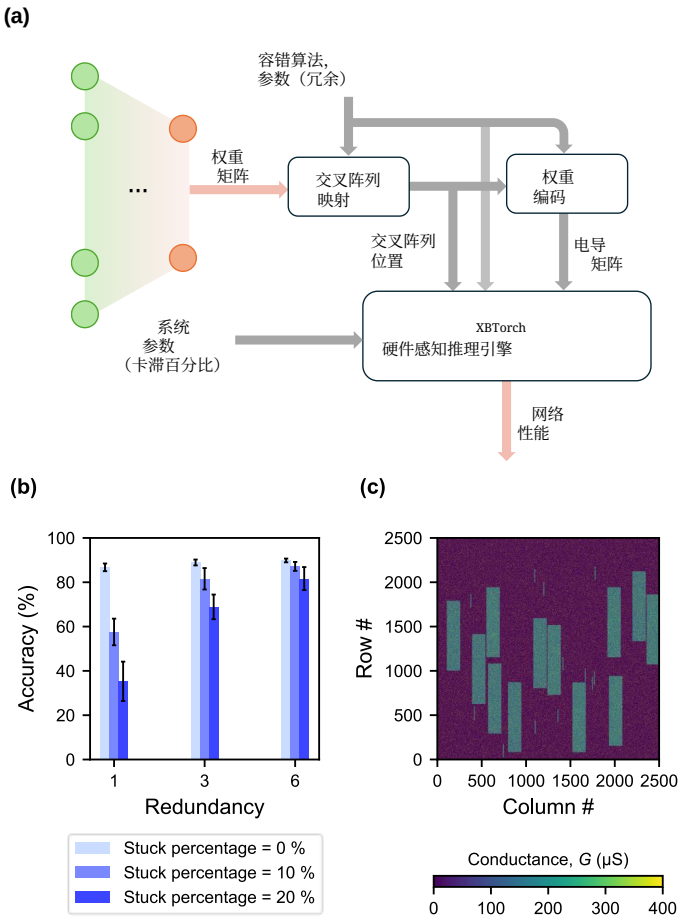

图 5. XBTorch 中层集成平均的示例。(a) 使用 XBTorch 将单个网络层映射到模拟交叉阵列的整体流程。可以指定系统参数（例如交叉阵列的总尺寸以及其中卡死器件的比例），以评估网络在真实噪声源下的性能。(b) 网络推理准确率随映射冗余和卡死器件比例的变化。在层集成平均中，推理时会对冗余映射的输出逐层取平均。柱高表示 10 次独立迭代的平均值，误差线表示一个标准差。(c) 当冗余级别为 6 且 20% 的器件卡死时，对网络解进行编码和映射后得到的电导图。

超参数	值
学习率（软件）	0.1
器件模型	表格型铁电场效应晶体管：$V_{gs} = 0.9V$，var = 1%
学习率（硬件感知）	4.76
WAGE 位宽	2, 8, 8, 8
WAGE 舍入	最近邻
权重范围	(-1, +1)
优化器	xboptim.SGD

表 B.1. 图 2 中结果所用的超参数。

超参数	值
学习率（所有秩）	0.1
器件模型	表格型铁电场效应晶体管：$V_{gs} = 0.9\,V$，5% 变异性
权重范围	(-1, +1)
优化器	xboptim.SGD

表 B.2. 图 3 中结果的超参数。

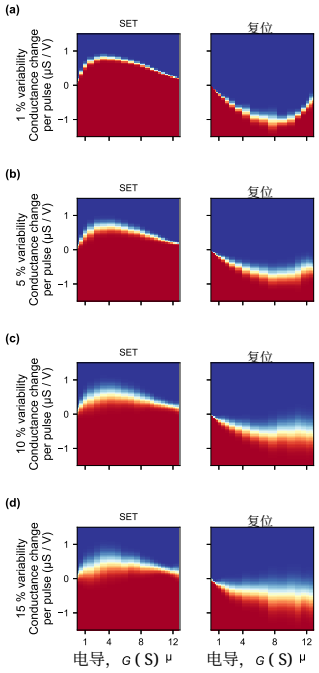

图 A.1. 铁电场效应晶体管表格化器件模型预设在$V_{gs} = 0.9V$ 条件下以及变异性逐渐增大时的累积分布函数 (CDF) 数组。

超参数	值
加速器支持的器件数量（行、列）	(2500, 2500)
最小器件电导（μS）	133
最大器件电导（μS）	233
读取电压（V）	0.3
注入加速器的均匀读取噪声（μS）	10
零均值高斯写噪声（μS）	50
将权重编码为器件电导的方案	差分
将器件电导矩阵映射到加速器位置的映射方案	随机
ADC/DAC 精度（映射层输入/输出的定点精度）	8
卡滞百分比（%）	0.0
容错算法	None
层冗余映射时的输出聚合方案	N/A

表 B.3. 图 4 结果的超参数。

超参数	值
ADC/DAC 精度（定点精度）	变量
卡滞百分比 (%)	变量
容错算法	LEA
层进行冗余映射时聚合输出的方案	平均

表 B.4. 图 5 中结果的超参数。其他参数与图 4 保持相同。

超参数	值
加速器支持的器件数量（行数、列数）	(∞, ∞)
注入加速器的均匀读取噪声（μS）	0
零均值高斯写噪声（μS）	0
ADC/DAC 精度（定点精度）	变量
卡滞百分比 (%)	-
容错算法	-

表 B.5. 表 I 中结果的超参数。其他参数与图 4 相同。加速器以无状态模式运行。

1. ARC（AI2推理挑战）— ARC-Easy / ARC-Challenge

ARC AI2[35] 是一组小学阶段的多项选择科学问题；该数据集分为 Easy 集和 Challenge 集（Challenge 集包含简单检索和词共现方法无法回答的问题）。我们按照标准任务划分，对 ARC-Easy 和 ARC-Challenge 子集的多项选择准确率进行评估。

2. BoolQ

BoolQ[36] 是一项自然产生的是非问答任务，每个样例都包含一个问题和一段支持段落；模型必须预测布尔值（是/否）答案。我们使用标准的 BoolQ 数据集划分，将 BoolQ 作为二元分类准确率任务进行评估。

3. PIQA（物理交互问答）

PIQA[37] 是一个二选一的多项选择数据集，用于测试物理常识推理（选项对应合理的物理交互或解决方案）。我们测量 PIQA 测试题目的选择准确率。

4. Winogrande

WinoGrande[38] 是一个大型的 Winograd 模式代词消解基准，旨在衡量常识推理能力，同时减少数据集偏差；示例采用二选一的方式进行评估（哪个候选项适合填入代词）。我们报告 WinoGrande 评估划分上的准确率。

参考文献

[1]X. Zou、S. Xu、X. Chen、L. Yan 和 Y. Han，“突破冯·诺依曼瓶颈：架构级存内处理技术”，《中国科学：信息科学》**64**，160404（2021）。[2]R. Yu、Z. Wang、Q. Liu、B. Gao、Z. Hao、T. Guo、S. Ding、J. Zhang、Q. Qin、D. Wu、等，“一个基于忆阻器、采用软硬件协同开发的全栈存内计算系统”，《自然·通讯》**16**，2123 2025( )。[3]X. Huang、C. Liu、Y.-G. Jiang 和 P. Zhou，“通过存内计算突破存储墙”，《中国物理 B》**29**，078504（2020）。[4]K. Sun、J. Chen 和 X. Yan，“忆阻器的未来：材料工程与神经网络”，《先进功能材料》**31**，2006773（2021）。[5]T. M. Taha、R. Hasan、C. Yakopcic 和 M. R. McLean，“探索专用多核神经处理器的设计空间”，载于*2013* 年国际联合神经网络会议（*IJCNN*）（IEEE，2013），第 1–8 页。[6]S. Yu，“利用新兴非易失性存储器实现神经启发式计算”，《IEEE 汇刊》**106**，260–285（2018）。[7]J. Grollier、D. Querlioz 和 M. D. Stiles，“用于仿生计算的自旋电子纳米器件”，《IEEE 汇刊》**104**，2024–2039（2016）。[8]M. Jerry, P.-Y. Chen, J. Zhang, P. Sharma, K. Ni, S. Yu, 和 S. Datta，“用于加速深度神经网络训练的铁电场效应晶体管模拟突触”，载于2017年IEEE国际电子器件会议（IEDM）（IEEE，2017年），第6–2页。[9]A. Sebastian, M. Le Gallo, 和 E. Eleftheriou，“计算相变存储器：超越冯·诺依曼计算”，《物理学杂志D：应用物理》52，443002（2019年）。[10] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola，等，“使用非易失性存储器的神经形态计算”，《物理学进展：X》2，89–124（2017年）。[11]X. Peng, S. Huang, Y. Luo, X. Sun, 和 S. Yu，“Dnn+ neurosim：用于存内计算加速器的端到端基准测试框架，支持多种器件技术”，载于2019年IEEE国际电子器件会议（IEDM）（IEEE，2019年），第32–5页。[12]Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, 和 H.-S. P. Wong，“经实验验证的金属氧化物阻变式随机存取存储器紧凑模型”，《IEEE电子器件汇刊》63，1884–1892（2016年）。[13]S. Chatterjee, S. Kumar, A. Gaidhane, C. K. Dabhi, Y. S. Chauhan, 和 H. Amrouch，“面向存储器和逻辑应用的铁电全耗尽绝缘体上硅场效应晶体管建模”，《固态电子学》200，108554（2023年）。[14]T. Gokmen 和 W. Haensch，“在阻变器件阵列上训练神经网络的算法”，《神经科学前沿》14，103（2020年）。[15] W. A. Borders, A. Madhavan, M. W. Daniels, V. Georgiou, M. Lueker-Boden, T. S. Santos, P. M. Braganca, M. D. Stiles, J. J. McClelland, 和 B. D. Hoskins，“面向集成磁性隧道结阵列的测量驱动神经网络训练”，《物理评论应用》21，054028（2024年）。[16]F. Cai, S. Kumar, T. Van Vaerenbergh, X. Sheng, R. Liu, C. Li, Z. Liu, M. Foltin, S. Yu, Q. Xia，等，“利用忆阻器霍普菲尔德神经网络中的本征噪声实现高能效组合优化”，《自然·电子学》3，409–418（2020年）。[17]O. Yousuf, A. L. Glasmann, A. L. Mazzoni, S. Najmaei, 和 G. C. Adam，“面向铁电场效应晶体管加速器的鲁棒硬件感知神经网络”，《IEEE纳米技术汇刊》（2025年）。[18]O. Yousuf, B. D. Hoskins, K. Ramu, M. Fream, W. A. Borders, A. Madhavan, M. W. Daniels, A. Dienstfrey, J. J. McClelland, M. Lueker-Boden，等，“忆阻神经网络的容错层集成平均”，《自然·通讯》16，1250（2025年）。[19]D. Joksas, P. Freitas, Z. Chai, W. H. Ng, M. Buckwell, C. Li, W. Zhang, Q. Xia, A. Kenyon, 和 A. Mehonic，“委员会机——处理基于忆阻器的神经网络中非理想因素的通用方法”，《自然·通讯》11，4273（2020年）。[20]W. Huangfu, L. Xia, M. Cheng, X. Yin, T. Tang, B. Li, K. Chakrabarty, Y. Xie, Y. Wang, 和 H. Yang，“面向计算的阻变随机存取存储器计算系统容错方案”，载于2017年第22届亚洲及南太平洋设计自动化会议（ASP-DAC）（IEEE，2017年），第794–799页。[21]M. J. Rasch, D. Moreda, T. Gokmen, M. Le Gallo, F. Carta, C. Goldberg, K. El Maghraoui, A. Sebastian, 和 V. Narayanan，“用于模拟模拟交叉阵列上训练与推理的灵活快速PyTorch工具包”，载于2021年IEEE第三届人工智能电路与系统国际会议（AICAS）（IEEE，2021年），第1–4页。[22]B. Feinberg, T. P. Xiao, C. J. Brinker, C. H. Bennett, M. J. Marinella, 和 S. Agarwal，“CrossSim：模拟存内计算的精度仿真”，GitHub（2017年）。[23]J. Büchel, W. A. Simon, C. Lammie, G. Acampa, K. El Maghraoui, M. Le Gallo, 和 A. Sebastian，“AIHWKIT-lightning：用于模拟存内计算的可扩展硬件感知训练工具包”，载于NeurIPS 2024年新计算范式下的机器学习研讨会（2024年）。[24]O. Yousuf, I. Hossen, M. W. Daniels, M. Lueker-Boden, A. Dienstfrey, 和 G. C. Adam，“基于ReRAM的神经网络仿真中的器件建模偏差”，《IEEE电路与系统新兴及精选主题期刊》13，382–394（2023年）。[25]I. Hossen, M. A. Anders, L. Wang, 和 G. C. Adam，“采用克里金插值的数据驱动阻变随机存取存储器器件模型”，《科学报告》12，5963（2022年）。[26]S. Wu, G. Li, F. Chen, 和 L. Shi，“ 深度神经网络中的整数训练与推理 ”，arXiv预印本 arXiv:1802.04680（2018年）。[27]T. Zhang, Z. Lin, G. Yang, 和 C. D. Sa，“QPyTorch：低精度算术仿真框架”（2019年），arXiv:1910.04540 [cs.LG]。

[28]T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz，等，“HuggingFace的transformers：最先进的自然语言处理”，arXiv预印本 arXiv:1910.03771（2019年）。[29]L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac’h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, 和 A. Zou，“语言模型评估工具”（2024年）。[30]T. Vogels, S. P. Karimireddy, 和 M. Jaggi，“PowerSGD：面向分布式优化的实用低秩梯度压缩”，《神经信息处理系统进展》**32**（2019年）。[31]B. D. Hoskins, M. W. Daniels, S. Huang, A. Madhavan, G. C. Adam, N. Zhitenev, J. J. McClelland, 和 M. D. Stiles，“面向硬件神经网络的流式批处理特征更新”，《神经科学前沿》**13**，793（2019年）。[32]S. Huang, B. D. Hoskins, M. W. Daniels, M. D. Stiles, 和 G. C. Adam，“用于深度存内阵列内存高效训练的低秩梯度下降”，《ACM计算系统新兴技术期刊》**19**，1–24（2023年）。[33]H. Li、Z. Xu、G. Taylor、C. Studer 和 T. Goldstein，《神经网络损失景观的可视化》，神经信息处理系统进展**31**（2018年）。[34]A. Kaushal、T. Vaidhya、A. K. Mondal、T. Pandey、A. Bhagat 和 I. Rish，《Spectra：大规模预训练三值语言模型的惊人有效性》，arXiv预印本 arXiv:2407.12327（2024年）。[35]P. Clark、I. Cowhey、O. Etzioni、T. Khot、A. Sabharwal、C. Schoenick 和 O. Tafjord，《你认为自己已经解决了问答问题？试试 ARC，即 AI2推理挑战》，arXiv预印本 arXiv:1803.05457（2018年）。[36]C. Clark、K. Lee、M.-W. Chang、T. Kwiatkowski、M. Collins 和 K. Toutanova，《BoolQ：探索自然语言是非问题出人意料的难度》，arXiv预印本 arXiv:1905.10044（2019年）。[37]Y. Bisk、R. Zellers、J. Gao、Y. Choi、等，《PIQA：用自然语言进行物理常识推理》，载于人工智能促进协会人工智能会议论文集，第34卷（2020年），第7432–7439页。[38]K. Sakaguchi、R. L. Bras、C. Bhagavatula 和 Y. Choi，《Winogrande：大规模对抗性温诺格拉德图式挑战》，ACM通讯**64**，99–106（2021年）。