可信人工智能加速器计算框架的设计

任轩乐

阿里巴巴达摩院计算技术实验室

摘要

可信计算正成为一种保护人工智能（AI）计算的有效方法。特别是基于可信 CPU 的方案，用户可以将他们的敏感数据和模型上传到 CPU 信任区域，并在其中运行人工智能任务，同时保证数据和模型的机密性和完整性。由于 AI 加速器已经广泛用于提高人工智能计算的性能，因此人们期望有一个通用的框架，能够在这些 AI 加速器上实现可信计算。为了设计这样的计算框架，我们提出了一个快速保护隐私的人工智能计算框架，它包含一个未修改的可信 CPU 和一个定制的 AI 加速器。我们仔细地利用密码学原语来实现信任建立和数据保护，并且我们在加速器上增加了一个安全接口和一个密码引擎来支持这些密码学原语。作为一个案例研究，我们基于通用张量加速器 VTA（一种开源 AI 加速器）展示了我们的框架。评估结果表明，我们提出的计算框架以较小的设计代价和适度的性能开销实现了隐私保护的人工智能计算。

1 引言

隐私已经成为一个阻碍人工智能（ML）应用的大问题。在许多现实的机器学习应用中，数据和模型是由不同所有方拥有的。这样的机器学习任务只有在这些方将输入数据和模型放在一起时才能完成。然而，考虑到数据安全和隐私，许多所有方并不愿意共享他们的数据/模型。为了实现隐私保护的机器学习，可信计算是一种有前途的方法。当不同的所有方可以将他们的敏感数据/模型上传到信任区域 —— 一个由最近的可信 CPU（例如，Intel 的 SGX 启用的 CPU）提供的隔离的安全环境。在信任区域内，所有的代码和数据都受到 CPU 硬件的保护，不受信任区域外部环境的读写影响。因此，只有在信任区域内运行的可信软件才能使用敏感数据和模型进行计算。

然而，CPU 的计算能力对于最近的大规模人工智能模型（如深度神经网络）来说是不够的。为了解决这个问题，学术界和工业界都设计了各种专用的 AI 加速器用于现代机器学习任务，如 Eyeriss [1]、TPUs [2]、VTA [3] 等。虽然 AI 加速器可以帮助处理这样的工作负载，但是如何在使用 AI 加速器的同时保护隐私仍然是一个开放的问题。例如，最先进的研究 [4,5] 在 GPU 上实现了可

信执行环境，但它们都依赖于 GPU 具有可信设备内存这样一个假设。然而，这个假设对于许多 AI 加速器来说并不合适，因为它们没有集成的可信设备内存。因此，目前还缺乏一个通用的框架来在 AI 加速器上实现可信计算。

为了解决这个问题，在本文中，我们提出了一个基于 AI 加速器的可信计算框架，用于快速保护隐私的机器学习。该框架由未修改的 CPU 和定制的 AI 加速器组成。具体来说，我们通过在 AI 加速器上增加一个安全接口和一个密码引擎来定制加速器；CPU、加速器的 IP 核、总线等都保持不变。为了确保机密性和完整性，我们首先利用新添加的密码学原语在 CPU 信任区域和 AI 加速器之间建立信任并交换一个对称密钥。然后，我们使用这个对称密钥来保护 CPU 信任区域和加速器之间的所有信道。由于密钥只由 CPU 信任区域和加速器持有，没有密钥的任何人都无法获取或篡改代码和数据。

作为一个案例研究，在第 4 节中，我们展示了我们的框架在一个开源 AI 加速器上的实现。它表明了我们的方法可以利用 AI 加速器的高计算能力，同时主要的硬件/软件设计（如 IP 核设计、指令集架构和编译器）不需要改变。因此，这个框架非常适合那些可以定制 AI 加速器但不能定制 CPU/GPU 的学术研究者和工业开发者。事实上，我们的框架已经应用在云计算的一个工业环境中。

总之，本文的贡献如下：

1、 这是第一个基于定制 AI 加速器的可信计算框架，用于快速保护人工智能算法的隐私；
2、 我们提出了在可信 CPU 和定制 AI 加速器之间建立信任和保护通信的方法。特别地，即使 AI 加速器没有可信设备内存，这些方法也是有效的；
3、 我们在 VTA（一个开源 AI 加速器）上展示了我们框架的实现，并用精确的寄存器传输级模拟评估了它的性能。

2 背景

2.1 保护隐私的机器学习

隐私是许多机器学习应用中的一个关键问题，如机器学习服务、联合学习、边缘推理等。一般来说，我们可以用以下公式来表达一个机器学习任务

$$\boldsymbol{result = f(model, data)}$$

当模型和数据是由两个或多个方拥有时，保护隐私的机器学习就是在不泄露他们的私有模型/数据的情况下，计算 $f(model, data)$。

为此，许多最近的研究将其视为一个安全多方计算（MPC）问题，并提出了基于密码学的解决方案。例如，同态加密、混淆电路和秘密共享被用于保护隐私的推理 [6,7] 和保护隐私的训练 [8,9]。然而，基于密码学的解决方案非常低效，因为同态加密和 MPC 协议有非常高的计算和通信复杂度。此外，这些解决方案的通用性很差，因为同态加密和 MPC 协议支持的操作是有限的。

另一种方法是利用可信计算来实现快速保护隐私的机器学习。最近，处理器制造商将可信计算组件集成到 CPU 中，以提供基于硬件的可信执行环境。基于可信 CPU（例如 Intel SGX），一些最近的解决方案 [10,11] 使得模型/数据拥有者可以通过加密的通信通道将他们的私有模型/数据上传到一个安全的信任区域，并保证只有经过验证的可信软件才能访问和解密模型和数据。在信任区域内，CPU 使用解密后的模型和数据计算 $f(model, data)$。因此，与基于密码学的解决方案相比，这些方法更具通用性，并且可以以更低的开销实现保护隐私的机器学习。

2.2 基于硬件加速器的可信计算

使用可信 CPU，性能受到 CPU 性能的限制，这对于现代机器学习任务（如训练深度神经网络）来说是不够的。为了提高性能，最近的研究提出了几种利用硬件加速器（主要是通用 GPU）并同时保护隐私的方法。

Graviton [4] 是一个涉及到一个未修改的可信 CPU 和一个修改过的可信 GPU 的可信执行环境框架。它通过定制 GPU 命令处理器，在多核 GPU 上提供安全的“上下文”。该设计基于这样一个假设：GPU 的封装内存（例如 HBM）是在信任边界内的。然而，许多 AI 加速器没有可信设备内存，所以这种设计不能直接适用于 AI 加速器。

HIX [5] 是一种保护用户软件和未修改商品 GPU 之间 I/O 路径的硬件/软件架构。为了实现这一点，HIX 改变了 CPU 的硬件架构，以 1) 为 GPU 驱动提供一个特定的信任区域，以及 2) 保护对 GPU 的 MMIO 访问。此外，它假设整个显卡（包括设备内存）和整个硬件系统（包括总线）都是可信的。因此，它只能提供针对特权软件的保护，但不能防止硬件攻击，如窃听。为了使用 AI 加速器，我们希望说服用户，我们的框架在软件和硬件方面都是安全的。

Slalom [12] 是一个框架，用于将矩阵乘法外包给不可信的硬件加速器，以加速深度神经网络（DNN）的计算，基于现有的可信 CPU。为了确保隐私和完整性，它利用了针对 DNN 定制的密

码外包协议。然而，正如作者所承认的，其中一个协议需要一个昂贵的预处理阶段，所以 Slalom 并不能有效地提高总运行时间方面的性能。由于协议的限制，Slalom 也不能支持隐私保护的训练。一般来说，它仍然受到 MPC 协议的限制。理想情况下，我们希望充分利用 AI 加速器进行多样化的机器学习计算。

3 方法

在本节中，我们将详细介绍我们提出的用于保护隐私的机器学习的可信计算框架，其中包括数据拥有者、模型拥有者和一个计算平台。在这种场景中，数据/模型拥有者希望使用计算平台（它可以是第三方，也可以是数据拥有者或模型拥有者之一提供的）来完成一个机器学习任务，如模型训练、分类、预测等，而不泄露他们的数据/模型。为了有效地实现这一点，我们的框架利用计算平台内的可信 CPU 和定制的 AI 加速器来帮助数据拥有者和模型拥有者将他们的数据/模型安全地放入 CPU 信任区域，然后将计算卸载到加速器上，如图 1 所示。

3.1 威胁模型

我们认为计算平台可能被敌手攻击，敌手可以控制整个软件/硬件系统。对于硬件，只有CPU封装和加速器芯片封装是可信的。所有其他的硬件都可能被攻击，包括内存、存储、外围设备等。我们认为敌手有以下能力：1）窃听系统总线和 PCIe 总线，2）通过恶意设备直接访问主内存，3）在 CPU 和加速器之间进行中间人攻击，4）窃听、篡改或直接访问加速器的设备内存（如果 AI 加速器有不可信的设备内存）。

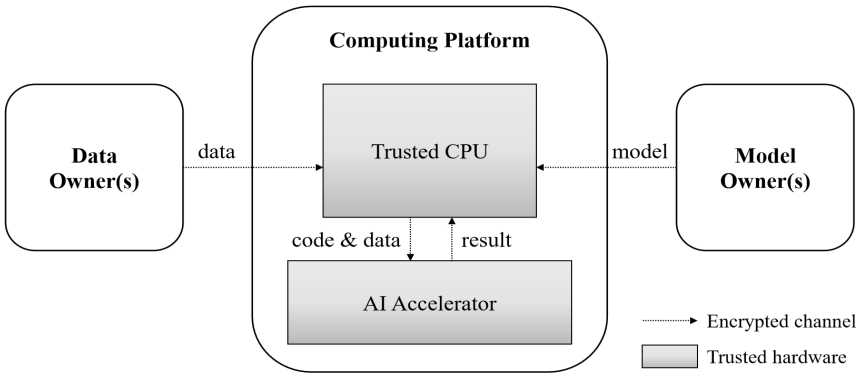

图 1: 用于保护隐私的机器学习的框架概述。

对于软件，除了在信任区域内运行的程序外，其他一切都是不可信的。恶意软件能够攻击操作系统或虚拟机监控器，在最高权限级别运行，所以敌手具有以下能力：1）直接访问主内存中除 PRM（处理器保留内存）区域外的任何部分，2）通过 MMIO 访问并向加速器发送命令，3）攻击驱动程序并调用或篡改驱动程序 API（假设驱动程序不在信任区域内运行）。

敌手的目标是窃取计算平台上的数据/模型。我们框架的目标是防止这种情况，并确保代码和数据的完整性。由于我们的框架包含了未修改的可信 CPU，所以这项工作无法防止可信 CPU 已有的漏洞，因此旁路攻击不在我们的范围之内。

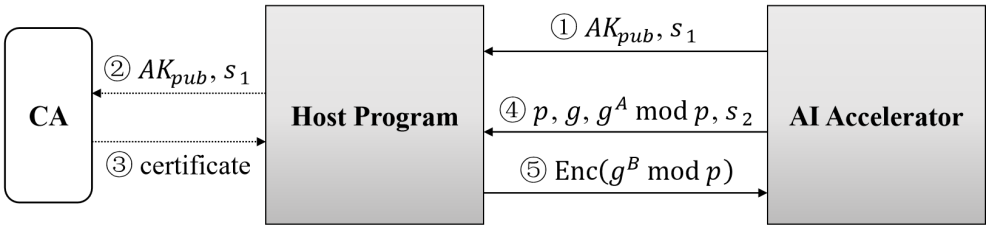

图 2: 主机程序和 AI 加速器之间的信任建立。

3.2 信任

为了在两方之间建立信任，我们采用远程证明 [13] 来建立数据所有者（或模型所有者）和 CPU 信任域之间的信任。因此，数据和模型所有者都可以信任 1) 信任域是由可信 CPU 提供的真正的安全环境，2) 经过证明的软件在这个信任域中正确运行，3) 他们的数据和模型已经安全完整地上传到了信任域。接下来，为了将可信边界从 CPU 信任域扩展到 AI 加速器，我们借用了远程证明的概念，并详细介绍了我们在图 2 中描述的方法。

3.2.1 认证

在认证步骤中，AI 加速器向在 CPU 信任域中运行的可信软件（在下面部分称为“主机程序”）证明它是一个可信的加速器。认证过程基于公钥密码学。具体来说，每个可信加速器在制造阶段都会被分配一个唯一的背书密钥对(EK_{pri}, EK_{pub})。私钥 EK_{pri} 被烧录到加速器硬件中，它只能为加速器所知。公钥 EK_{pub} 由制造商的证书颁发机构（CA）维护。然后，每次主机程序想要使用加速器时，加速器生成一对不同的认证密钥（AK_{pri}, AK_{pub}），然后将公钥 AK_{pub} 和一个 EK_{pri}-签名的签名 $s_1 = Sign(EK_{pri}, AK_{pub})$ 发送给主机程序。

主机程序应该验证 AK_{pub} 是否绑定到一个真正的可信加速器。为此，它将收到的 AK_{pub} 和 s_1 发送给制造商的 CA，然后 CA 使用 EK_{pub} 验证签名。如果 s_1 与 AK_{pub} 匹配，CA 就会颁发一个

证书并将其返回给主机程序。一旦主机程序收到证书，它就可以确认 AK_{pub} 确实是由可信加速器生成的。

3.2.2 密钥交换

密钥交换的目的是在 AI 加速器和主机程序之间建立一个共享的对称密钥。基于第一步建立的信任，我们使用 AK 作为临时 Diffie Hellman 密钥交换的签名密钥。具体来说，加速器生成一个素数 p、一个原根 g 和一个随机数 A，并将 p、g、$g^A \ mod \ p$ 和一个 AK_{pri}-签名的签名 $s_2 = Sign(AK_{pri}, p||g||g^A \ mod \ p)$ 传输给主机程序。在主机程序接收到它们并用 AK_{pub} 验证签名后，它生成一个随机数 B，然后将 Encrypt(AK_{pub},$g^B \ mod \ p$) 发送给加速器。最后，主机程序和加速器都可以计算 $g^{AB} \ mod \ p$ 作为他们的共享秘密，并可以通过一个密钥派生函数从共享秘密中派生出一个对称密钥 K。

3.3 保护

在共享对称密钥的基础上，我们提出了以下数据保护方法，以防止敌手在主机程序将计算卸载到 AI 加速器时获取数据/模型。

3.3.1 概述

首先，我们保护主机程序和加速器之间传输的代码和数据。一般来说，代码和数据是从主机程序的内存空间放置到“片外 DRAM”，这意味着加速器的设备内存或主内存的共享区域。然后，加速器从片外 DRAM 中获取代码，并在运行时访问片外内存。考虑到敌手可以读写片外 DRAM 设备内存，而且总线可能被攻破，我们不应该在这样不安全的区域中泄露代码和数据。形式上，我们定义以下原则：

原则 1 – 代码和数据在主机程序和片外 DRAM 之间传输时应该被加密和完整性保护。

原则 2 – 如果加速器没有可信的内存，代码和数据应该在片外 DRAM 中保持加密和完整性保护。

基于这些原则，我们采用认证加密，它是加密和消息认证的结合，以确保传输消息的机密性和完整性。具体来说，主机程序在 CPU 信任域中用对称密钥 K 加密代码和数据，并用 K 计算它们的消息认证码（MAC），然后将密文和 MAC 写入片外 DRAM。为了使用代码和数据，加速器解密它们并用 MAC 验证它们的完整性。对称地，在计算出结果后，加速器加密结果并计算其 MAC，然后将密文和 MAC 写入片外 DRAM。主机程序可以在解密和验证后得到结果。

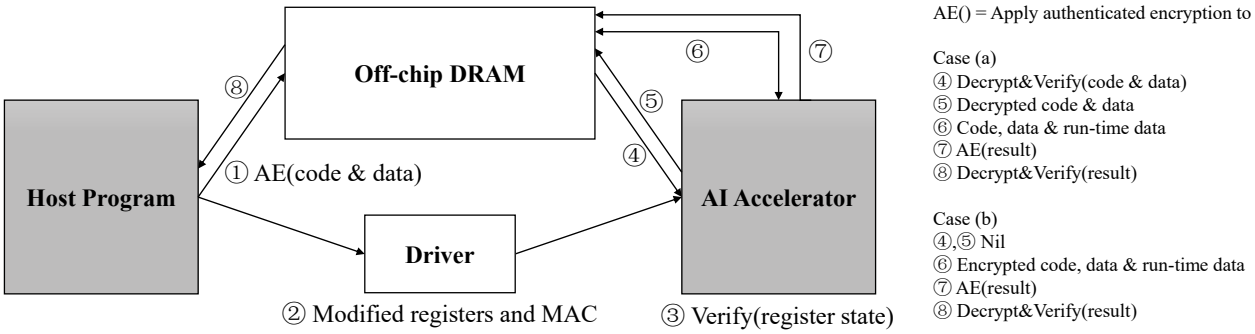

图 3: 保护方法（a）如果 AI 加速器有可信的设备内存（b）如果 AI 加速器没有可信的设备内存。

3.3.2 细节

为了实现效率和安全性，我们需要考虑两个问题：解密后的代码和数据放在哪里？如何保护存储在片外 DRAM 中的运行时数据？

如果 AI 加速器有自己的可信的封装内设备内存，答案很简单：它可以直接将解密后的代码和数据放在可信的设备内存中，并直接存储运行时数据，无需加密或完整性保护。如果没有，根据原则 2，加速器永远不能将未加密的消息放在片外 DRAM 中。将它们存储在加速器有限的片上存储中也是不切实际的。因此，加速器必须"按需"解密代码和数据。具体来说，每次加速器从片外 DRAM 中将一块代码或数据取到加速器的片上存储（例如 SRAM 缓冲区或寄存器）时，它取出相应的密文并在加速器芯片内解密它。为了实现这一点，我们需要一个基于计数器模式的认证加密方案，如 AES-GCM 和 AES-CCM。

另一个挑战是完整性保护。如果 AI 加速器没有可信的设备内存，因为敌手可以在运行时修改不可信的片外 DRAM 中的代码和数据。因此，加速器需要每次访问片外 DRAM 时验证完整性。然而，反复验证整个代码和数据是否与 MAC 匹配是极其低效的。为了解决这个问题，我们提出了以下方案：

1. 在主机程序对代码和数据进行认证加密之前，它将整个代码和数据分成 m 块；
2. 然后主机程序计算它们各自的密文和 MAC，并将这 m 块密文和 MAC 存储在片外 DRAM 中；
3. 在这种情况下，每次加速器访问片外 DRAM 时，它只取出相应的块，并验证它们是否与其 MAC 匹配。

m 的数量越大，表示粒度越细，导致计算开销更低（即较少的加密/解密量）。然而，它也导致内存消耗增大以及加速器的 DRAM 访问增加。因此，m 的值应该是计算和内存访问之间的权衡。

如果 AI 加速器没有可信的设备内存，运行时数据也应该被保护。具体来说，根据原则 2，在将运行时数据写入片外 DRAM 之前，应该对它们进行基于计数器模式的加密，并用前一段描述的方法保护它们的完整性。

3.3.3 寄存器状态

到目前为止，我们已经完成了对片外 DRAM 中的代码和数据的保护，以确保敌手不能学习或篡改它们。然而，存储在 AI 加速器的可编程寄存器（例如地址寄存器和控制寄存器）中的信息还没有得到保护。具体来说，如果主机程序想要使用 AI 加速器，它就调用加速器的驱动程序通过 MMIO 来写加速器的寄存器。在我们的威胁模型中，敌手能够通过 MMIO 直接访问寄存器。因此，我们有以下原则：

原则 3 – AI 加速器的可编程寄存器在主机程序和加速器之间的传输过程中应该被完整性保护。

基于这个原则，我们提出使用消息认证码（MAC）来防止敌手篡改寄存器。注意，内核模式驱动程序可能不在 CPU 信任区域内运行，所以我们让在信任区域内运行的运行时（也称为用户模式驱动程序）来维护寄存器状态。每次主机程序修改一个寄存器时，它也用对称密钥 K 计算整个寄存器状态的 MAC。在主机程序写入修改后的寄存器后，它将 MAC 写入加速器的一个特定寄存器。所有的写操作都是通过调用内核模式驱动程序完成的。由于只有主机程序和加速器知道 K，所以加速器可以验证寄存器状态的完整性，而驱动程序不能伪造 MAC。因此，即使驱动程序不在信任边界内（即不可信），这个问题也可以解决。总之，图 3 描述了我们的保护方法。

4 案例研究：VTA

在本节中，我们将展示使用我们的可信计算框架在定制的 AI 加速器上执行隐私保护机器学习任务的详细过程。具体来说，我们在开源的多功能张量加速器（VTA）[3] 上实现了所提出的框架，并评估了它对 VTA 性能的影响。

4.1 架构

参考图 1，我们在计算平台中部署了一个可信 CPU 和一个定制的 VTA，并且我们假设模型和数据已经安全地上传到了 CPU 信任域。然后，主机程序（在信任域中运行的可信和经过证明的软

件）用 TVM 深度学习编译栈 [14] 实时地将模型编译成 VTA 代码。为了支持 VTA 上的可信计算，我们对 VTA 的软件和硬件进行了修改。

软件的修改只涉及 VTA 运行时，而 VTA 的指令集架构或即时编译器在我们的框架中不需要改变。VTA 运行时（vta/src/runtime.cc）是主机程序的一部分，它充当着与不可信的片外 DRAM 和内核模式驱动程序的接口。具体来说，原始的 VTA 运行时负责在片外 DRAM 中分配一些缓冲区，将代码（即 VTA 中的“内核”）和数据放入相应的缓冲区，并调用驱动程序来启动内核。在我们的框架中，运行时被修改为它可以 1) 对代码和数据进行认证加密，以及 2) 维护寄存器状态以计算寄存器的 MAC。内存布局保持不变，因为基于计数器模式的加密不会改变消息的大小，但认证加密除了原始消息外还会产生额外的元数据（即 IV 和 MAC）。元数据被存储在片外 DRAM 中新分配的缓冲区中。

在硬件方面，我们没有改变 VTA 核心的设计。相反，在 VTA 核心和主机 CPU/DRAM 之间添加了一个安全层，包括一个安全接口和一个密码引擎（图 4）。安全接口将接收到的数据缓存在一个片上缓冲区（2 KB）中，将数据与密码引擎通信，然后将加密/解密后的数据发送到片外 DRAM 或 VTA 核心。密码引擎包含用于认证加密和信任建立的组件（如 AES-GCM、RSA 和 TRNG）。在这个实验中，我们实现了 AES-256 模块，它采用流水线结构，加密/解密 128 位明文/密文需要 29 个时钟周期，以及 GFM（Galois-Field Multiplication）模块，它对每个 128 位文本进行认证需要 8 个时钟周期。

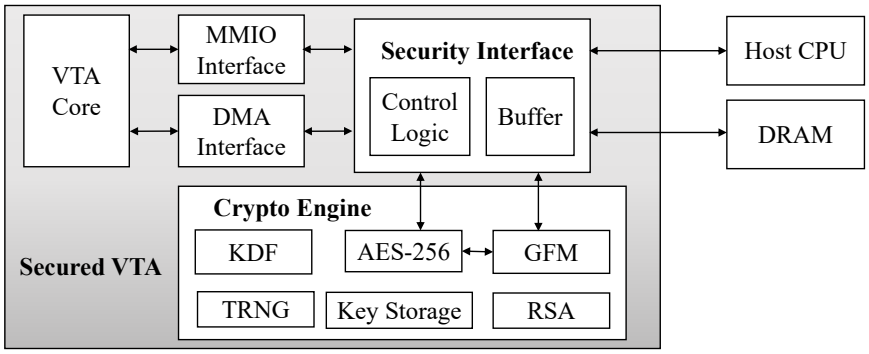

图 4: 通过添加安全接口和密码引擎来保护 VTA。

4.2 评估

我们基于 TSIM 构建了一个精确的周期模拟环境，它可以将寄存器传输级设计编译成 libvta$_{hw}$.soTV M 由于 VTA 主要是为计算机视觉任务设计的 [3]，我们选择了三个常用模型的前向

计算作为我们的基准，分别是 AlexNet [15]、ResNet-18 [16] 和多层感知器（MLP）。对于每个基准，我们评估了 VTA 的三种配置：1) 没有任何保护的原始 VTA（用 VTA 表示），2) 具有机密性保护的 VTA（用 VTA-AES-CTR 表示），3) 具有机密性和完整性保护的 VTA（用 VTA-AES-GCM 表示）。实验结果如图 5 所示。与原始 VTA 相比，添加保密性保护导致 1.009 倍~1.316 倍的减速，而添加保密性和完整性保护导致 1.079 倍~5.388 倍的减速。还注意到，不同基准测试的减速程度差异很大。我们将在下面进行讨论。

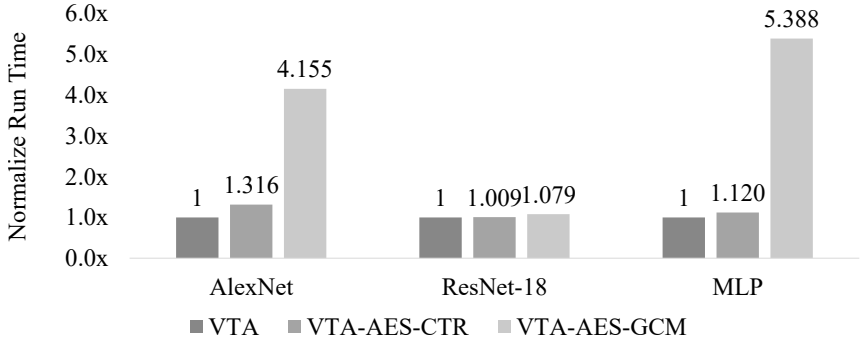

图 5: 在三种配置上执行深度神经网络计算。

表 1: AlexNet 各层的运行时间（以时钟周期为单位）和与原始 VTA 相比的减速比。

	VTA	VTA-AES-CTR	VTA-AES-GCM
Conv2 (64 – 192)	177,860,356	228,392,863 (1.28×)	676,785,137 (3.81×)
Conv3 (192 – 384)	92,822,991	130,856,866 (1.41×)	452,302,397 (4.87×)
Conv4 (384 – 256)	2,782,962	2,872,727 (1.03×)	2,988,247 (1.07×)
Conv5 (256 – 256)	1,879,117	1,969,399 (1.05×)	2,083,659 (1.11×)
FC1 (9216 – 4096)	5,418,983	6,016,817 (1.11×)	29,300,635 (5.40×)
FC2 (4096 – 4096)	2,412,609	2,682,866 (1.11×)	13,034,043 (5.40×)

原始 VTA 的总运行时间由计算和内存访问组成。相比之下，我们的保护方法还引入了加密、解密和消息认证，所有这些都与内存访问有关。因此，如果工作负载涉及大量内存访问，减速将会很明显。相反，少量的内存访问将导致轻微的减速。因此，性能开销的程度取决于工作负载的内存访问强度。

基于这个结论，我们可以指出，对于 VTA-AES-CTR（我们稍后再讨论 VTA-AES-GCM），不同基准测试的减速差异主要是由于 TVM 的不同编译优化级别造成的。截至撰写本文时，TVM 对 VTA 的模型编译仍在进行中。特别地，只有 ResNet-18 可以进行全模型优化，因此它可以被编译

成具有最低内存访问强度的代码。因此，运行 ResNet-18 只引入了 0.9% 的减速。相反，MLP 和 AlexNet 只能在层级上进行优化，而 AlexNet 中的一些层不能使用设备特定的优化进行编译。

以 AlexNet 为例。如表 1 所示，AlexNet 的第二和第三卷积层，它们没有被优化，在运行时间中占据了相当大的比例，并且对 VTA-AES-CTR 造成了高性能开销（1.28 倍 和 1.41 倍）。相比之下，第四和第五卷积层以及全连接层被优化了，在运行时间中占据了较小的比例，并且对 VTA-AES-CTR 造成了相对较小的开销（1.03 倍~1.11 倍）。

对于 VTA-AES-GCM，只有 ResNet-18 运行时开销低（1.079 倍）。我们认为，在 AlexNet 和 MLP 上运行时引起的显著高开销是由两个因素造成的。第一个是加密引擎的性能。具体地说，虽然我们的 GFM 模块的延迟比 AES-256 模块短，但 GFM 模块没有流水线化，所以它的吞吐量比 AES-256 模块和缓冲区低得多。第二个因素仍然是内存访问强度。没有完全优化的代码导致了频繁的数据移动，从而引入了更多的开销。此外，FC 层更多地依赖于内存，而卷积层相对来说更多地依赖于计算。因此，AlexNet 和 MLP 比 Resnet-18 更多地依赖于内存访问，并且它们对 VTA-AES-GCM 造成了高开销（4.155 倍和 5.388 倍）。

因此，如果编译器可以采用更高级别的优化，并且加速器配备了高性能加密引擎，则我们保护方法引入的性能开销可以显著降低，而这些在工业级 AI 加速器中已经实现了。

5 结论

本文提出了一个用于隐私保护机器学习的可信计算框架。关键的创新在于将定制的 AI 加速器纳入框架中，同时确保隐私和完整性。借助我们的方法，我们可以使用户安全地上传他们的敏感模型和数据，并在 AI 加速器内部计算结果。我们的案例研究表明，我们的框架在多功能张量加速器上是有效的，设计成本小，性能开销适中，并且在工业中使用这个框架具有较低的开销是有希望的。

参考文献

[1] Y.-H. Chen, J. S. Emer, and V. Sze, ``Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in International Symposium on Computer Architecture (ISCA), 2016, pp. 367--379.

[2] N. P. Jouppi, C. Young et al., ``In-Datacenter Performance Analysis of a Tensor Processing Unit,'' in International Symposium on Computer Architecture (ISCA), 2017, pp. 1--12.

[3] T. Moreau, T. Chen et al., ``A Hardware-Software Blueprint for Flexible Deep Learning Specialization,'' IEEE Micro, vol. 39, no. 5, pp. 8--16, 2019.

[4] S. Volos, K. Vaswani, and R. Bruno, ``Graviton: Trusted Execution Environments on GPUs,'' in USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2018, pp. 681--696.

[5] I. Jang, A. Tang et al., ``Heterogeneous Isolated Execution for Commodity GPUs,'' in International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2019, pp. 455--468.

[6] R. Gilad-Bachrach, N. Dowlin et al., ``CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy,'' in International Conference on Machine Learning (ICML), vol. 48, 2016, pp. 201--210. [7] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan,

[7] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, ``GAZELLE: A Low Latency Framework for Secure Neural Network Inference,'' in USENIX Security Symposium, 2018, pp. 1651--1669.

[8] P. Mohassel and Y. Zhang, ``SecureML: A System for Scalable Privacy-Preserving Machine Learning,'' in IEEE Symposium on Security and Privacy (S&P), 2017, pp. 19--38.

[9] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, ``Deepsecure: Scalable provably-secure deep learning,'' in Design Automation Conference (DAC), 2018, pp. 2:1--2:6.

[10] O. Ohrimenko, F. Schuster et al., ``Oblivious Multi-Party Machine Learning on Trusted Processors,'' in USENIX Security Symposium, 2016, pp. 619--636.

[11] T. Hunt, C. Song et al., ``Chiron: Privacy-preserving Machine Learning as a Service,'' CoRR, vol. abs/1803.05961, 2018.

[12] F. Tramèr and D. Boneh, ``Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware,'' in International Conference on Learning Representations (ICLR), 2019.

[13] V. Costan and S. Devadas, ``Intel SGX Explained,'' IACR Cryptology ePrint Archive, vol. 2016, p. 86, 2016.

[14] T. Chen, T. Moreau et al., ``TVM: An Automated End-to-End Optimizing Compiler for Deep Learning,'' in USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2018, pp. 578--594.

[15] A. Krizhevsky, ``One weird trick for parallelizing convolutional neural networks,'' CoRR, vol. abs/1404.5997, 2014.

[16] K. He, X. Zhang et al., ``Deep Residual Learning for Image Recognition,'' in Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770--778.