arXiv:2611.29067v1 [cs.CR] 21 Nov 2026

HiPerfCC-StateReuse：通过密文复用实现快速机密GPU数据移动

Haoxuan Xu[1] Shaoyu Wang[1] Minghao Lin[1] BeijieLiu[1]
Potung Yu[1] Yongqin Wang[1] VincentQuentin Ulitzsch[2]
Seo Jin Park[1] Murali Annavaram[1] Mengyuan Li[1]
[1]南加州大学 [2]麻省理工学院
xuhaoxua@usc.edu wangshao@usc.edu mlin7282@usc.edu beijieli@usc.edu
potungyu@usc.edu yongqin@usc.eduviniul@mit.edu seojinpa@usc.edu
annavara@usc.edu mli49061@usc.edu

摘要

机密 GPU 能够在设备上以接近原生的计算性能提供受保护的 GPU 计算，但数据移动仍然代价高昂。通过不可信主机内存在 CPU 与 GPU 可信执行环境之间传输应用数据或模型权重，需要进行大量 CPU 加密和解密，即使内容保持不变也是如此。现有软件技术会重叠或并行执行这些操作，但仍需承担按字节计算的 CPU 开销。复用已导出的密文本可避免这项开销。然而，GPU可信执行环境固件会强制使用方向专用加密密钥和每次传输都新生成的初始化向量（IV），从而阻止直接复用密文。

本文提出了密文重新绑定（ciphertext rebinding），这是一种纯软件原语，可在无需修改硬件的现有 GPU 可信执行环境（GPU TEEs）上实现重复的跨 GPU 密文复用，同时保持机密性、完整性和新鲜性。具体而言，密文重新绑定利用了 GPU TEE 的 AES-GCM 身份验证中依赖密文的组件与依赖初始化向量的组件之间的分离：在目标密钥下由 GPU 预先计算的摘要，可以使 CPU 无需处理有效载荷即可更新认证标签。经过这一修改，GPU 随后对保留密文的读取仍可通过由硬件固定、无法由软件修改的 GPU 复制引擎接口进行。随后，GPU 可执行额外的校正步骤，以补偿密钥和初始化向量的差异，从而恢复原始明文。通过将 GPU 导出的密文保留在不可信主机内存中，密文重新绑定避免了 CPU 对数据进行批量解密和重新加密，从而提高了 GPU 状态恢复与共享的有效带宽。我们仅通过修改驱动程序和库，在 *HiPerfCC-StateReuse* 中实现了这一原语，并结合批量提交、流水线 GPU 处理和复制引擎传输。我们的评估通过微基准测试衡量传输效率和 CPU 开销，并通过三个具有代表性的案例研究考察应用性能：模型切换、键值缓存卸载和跨实例缓存共享。

关键词

机密计算、GPU、密文重新绑定

1 引言

机密计算（CC）利用硬件强制实施的可信执行环境（TEE），保护敏感数据和计算免受特权基础设施软件的侵害。NVIDIA 机密 GPU 将这种保护扩展到 GPU 执行，在受保护的 GPU 内存中进行计算时，性能接近原生水平 [5]。然而，在 CPU 机密虚拟机（CVM）和机密 GPU 之间传输数据仍然代价高昂 [34, 51]。在现有的 GPU 机密计算硬件上，经过不可信共享主机缓冲区的传输由认证加密提供保护。CVM 在软件中加密从主机传往 GPU 的数据，并解密从 GPU 传往主机的数据；机密 GPU 复制引擎（CEs）则在硬件中执行相应的解密/加密操作 [5]。

即使 GPU 应用从主机内存恢复未发生变化的 GPU 状态，这些 CPU 加密工作仍会重复执行。例如，模型切换可能反复加载相同的权重，而分层键值（KV）缓存则会卸载并重复使用共享前缀的缓存状态 [10, 13, 48]。在机密 GPU 的传统受保护传输中，备份此类状态需要 CPU 将 GPU生成的密文解密到私有 CVM内存中。此后每次恢复，都需要 CPU 再次加密相同的有效载荷。尽管状态没有发生变化，CPU 仍必须在每次恢复时对每个字节重复进行解密和重新加密。

以往的方法通过在仿真中更改硬件或协议，或提高现有 CPU 加密工作的效率，来降低受保护传输开销。面向硬件的设计 包括 LÆGIS，它通过增加一个驻留于 HBM 的初始化向量（IV）存储区，使 CPU 预加密与 GPU 页迁移顺序解耦 [50]。TensorTEE 扩展 CPU 和 NPU 架构，使其可信执行环境（TEE）能够在不重新加密的情况下交换张量 [16]。Fastrack 则修改 GPU CC 协议，以建立直接的客户端–GPU 安全通道 [44]。这些面向硬件的设计需要进行架构或协议修改，因此无法直接部署在现有 GPU机密计算硬件上。相反，
它们代表了未来硬件和协议设计的潜在方向。相比之下，软件优化面向现有的 GPU机密计算硬件，无需架构或协议变更。其目标是通过隐藏 CPU 密码运算延迟或并行执行密码运算来降低受保护传输开销。PipeLLM 会预测哪些数据块将被恢复以及恢复顺序。CPU 使用预期的初始化向量对每个数据块进行加密，同时 GPU 计算 [43]。即使预测完全准确，CPU 也必须在每次恢复时重新加密完整的有效载荷。如果通道推进到了预测的初始化向量之后，PipeLLM 必须丢弃推测生成的密文，并可退回按需加密。Serialized Bridge 跨 CUDA上下文并行执行受保护传输，同时保留逐字节的 CPU 密码运算工作量 [52]。FLASH 保留

创建检查点时由 GPU生成的密文，但其恢复路径会在机密虚拟机中解密保存的状态，然后恢复传统的受保护传输 [23]。因此，当未改变的 GPU 状态被反复恢复时，这些软件技术并不能消除批量CPU密码运算。

实现高效的 GPU 状态复用。复用 GPU 生成的密文可以消除重复的 CPU 密码运算，但现有机密 GPU 传输协议不允许直接复用。机密 GPU 传输使用方向特定密钥和新 IV，因此，GPU到主机传输中的密文不能直接在主机到GPU通道上重放。其原始 AES-GCM认证标签在目标密钥和 IV 下无效。仅替换标签也不够：即使 CE 对保留密文进行身份验证，在新密钥和 IV 下解密也会得到不同的明文。因此，复用需要解决两个不同的问题：为目标传输验证保留密文，以及恢复原始明文，且两者都不能依赖批量CPU密码运算。本文探讨能否通过现有机密 GPU 硬件，以软件方式解决这两个问题。

我们提出了**密文重新绑定**，这是一种全新的纯软件原语，无需更改硬件，即可在现有 GPU 可信执行环境（TEE）上实现重复状态复用和跨 GPU 状态复用。密文重新绑定利用了AES-GCM认证中依赖密文的部分与依赖 IV 的部分相互分离这一特性 [9]。具体而言，GPU 使用可信GPU数据，基于目标通道的密钥预先计算依赖密文的认证摘要。对于后续每次主机到GPU传输，CPU 都将保留的摘要与依赖新 IV 的掩码结合起来，无需处理有效载荷即可更新认证标签。新标签使现有复制引擎能够在当前通道状态下对保留密文进行身份验证并解密。随后，GPU 内核利用计数器模式加密的异或结构，补偿原始 GPU到主机传输与当前主机到GPU传输之间的密钥流差异，从而恢复原始明文。通过将密文保留在不可信共享主机内存中，密文重新绑定避免了批量 CPU 解密和重新加密，从而提高 GPU状态恢复和共享的有效带宽。密文重新绑定还在 GPU机密计算的威胁模型下，保障 GPU 状态的机密性、完整性和新鲜性。我们基于标准AES-GCM界限和可信IV递进开展安全性分析，以论证这些保障；此外，还在 EasyCrypt 中通过机器验证证明，验证了校正的正确性，以及机密性和完整性归约 [7]。

我们仅通过修改驱动程序和库，在 *HiPerfCC-StateReuse* 中实现密文重新绑定。GPU 摘要计算和明文校正仍需处理有效载荷，传输也会产生提交与同步开销。*HiPerfCC-StateReuse*通过批量提交来摊销这些开销，并将 GPU处理与 CE 传输流水化，以重叠剩余工作。我们通过传输微基准测试和三个应用程序案例研究评估 *HiPerfCC-StateReuse*： 模型切换、KV缓存卸载和跨实例KV缓存共享。与多上下文 CC（一种在多个 CUDA上下文间并行执行传统受保护传输的基线方案）相比， [52], *HiPerfCC-StateReuse*在 H200 和 RTX PRO 6000 上的微基准测试中，将每 GiB 的 CPU 时间降低了 92–97%。在 H200 上，其有效传输带宽达到基线方案的 1.9–2.3× 倍，达到 54–57 GB/s，接近关闭 GPU机密计算（关闭 CC）时的带宽。

本文作出了三项贡献：

- 我们提出了密文重新绑定这一原语，它使身份验证和明文恢复能够适配目标通道的密钥以及每次传输都新生成的初始化向量，从而通过现有的 CE 接口实现密文的重复使用和跨 GPU 复用，无需进行大规模 CPU 解密或重新加密。我们在所述威胁模型下分析了其机密性、完整性和新鲜性。我们在抽象的 EasyCrypt 模型中对校正恒等式以及机密性和完整性归约进行了机器验证 [7]。

- 我们通过驱动程序和库修改实现了*HiPerfCC-StateReuse*，在无需修改 GPU 硬件或固件的现有机密 GPU 上，对提交进行批处理，并将 GPU处理与 CE 传输进行流水线处理。批处理和流水线处理相结合，使两个平台上的传输带宽相比不采用任何优化时提升了 23–60%。

- 我们在 H200 上将*HiPerfCC-StateReuse*与多上下文 CC 进行了评估。*HiPerfCC-StateReuse*使两个模型和两个引擎的完整模型恢复延迟降低了 38–59%。在采用 KV卸载时，其输出令牌吞吐量达到基线的 1.38× 倍（在峰值负载下对九组模型–工作负载组合取几何平均值）；在采用跨实例 KV 共享时，其吞吐量达到基线的 1.38× 倍。

系统范围。*HiPerfCC-StateReuse* 面向采用软件加密的主机反弹缓冲区进行 CPU–GPU传输的机密GPU部署。这包括已发布的 Hopper 和 Blackwell 直通模式，以及 Google Cloud A3 H100、Azure H100 和 VoltageGPU 的 RTXPRO 6000 机密虚拟机等云服务 [6, 11, 30, 53]。因此，*HiPerfCC-StateReuse*可以降低在各种现有机密GPU部署中恢复和共享 GPU 状态所需的 CPU 开销。可信I/O技术可通过将基于 TDISP 的可信设备分配与基于 PCIe IDE 的硬件加速内联加密相结合，加速 CPU–GPU通信 [27, 36, 37]。不过，这种方法需要兼容的 CPU、GPU、固件和驱动程序 [27],，而这些目前尚未公开商用。因此，TDISP 集成仍是未来工作的一个方向，尤其是随着 PCIe 6.0 平台在未来得到更广泛部署 [56]。

代码可用性。我们的代码可在 https://github.com/sept-usc/HiPerfCC-StateReuse。

2 密文重新绑定

密文重新绑定会保持 GPU 生成的密文不变，同时根据目标通道的密钥和每次传输新生成的初始化向量，调整身份验证和明文恢复。该原语结合标签更新与 GPU 明文校正，通过现有的复制引擎接口复用此密文。

2.1 威胁模型

我们遵循 CPU–GPU 机密计算的传统威胁模型 [15]。主机操作系统、虚拟机监控程序、共享主机内存和 CPU–GPU 互连均不受信任。控制这些基础设施的对手可以观察、修改、注入或重放共享内存中的数据以及互连上的流量。可信计算基包括 CPU/GPU安全硬件和固件

以及 CVM 内的所有软件，包括客户机操作系统、GPU 驱动程序、应用程序，以及我们的 *HiPerfCC-StateReuse* 库和驱动扩展。由这些可信软件启动的 GPU 内核同样可信。我们假设 CPU 和 GPU 环境已成功完成远程证明，并已安全建立其通信密钥。在这些假设下，*HiPerfCC-StateReuse* 旨在保护保留的 GPU 状态在共享主机内存中存储以及通过 CPU–GPU 互连传输期间的机密性、完整性和新鲜性。

访问模式泄漏不在我们的安全范围内，这与现有 CPU–GPU 机密计算平台已记录的局限性一致。NVIDIA 明确将 CPU 端侧信道防护交由底层 CPU 可信执行环境的威胁模型处理 [32],AMD SEV-SNP 不支持页级访问模式监控 [1],，而 Intel 也承认 TDX 中存在残余的内存访问模式泄漏 [19]。因此，*HiPerfCC-StateReuse* 不会尝试隐藏访问模式，包括状态传输的时序或对同一加密状态的重复导入。其他侧信道攻击，例如时序攻击和缓存攻击 [14, 21], ，以及耗尽资源或阻止 CVM 或 GPU 继续运行的拒绝服务（DoS）攻击 [26, 54], ，也不在本工作的范围内。这些排除项与 GPU 机密计算威胁模型一致 [15, 32]。

2.2 现有 GPU 可信执行环境的限制

现有 GPU 可信执行环境硬件会为不同方向强制使用专用加密密钥，并为每次传输使用新的初始化向量，从而阻止直接复用密文。

受保护传输。 CVM私有内存与受保护的 GPU 内存之间的传输会经过不受信任的共享主机缓冲区 [15]。对于主机到设备（H2D）传输，驱动程序会将来自 CVM私有内存的明文加密到共享缓冲区中。复制引擎通过 PCIe 读取密文，验证其认证标签，并将其解密到受保护的 GPU 内存中。对于设备到主机（D2H）传输，复制引擎会将加密的 GPU 数据写入共享缓冲区，而 CPU 则在 CVM

-

私有内存中验证并解密该密文。因此，以明文形式将 GPU 状态备份到 CVM私有内存，随后再恢复该状态，需要先由 CPU 解密，然后重新加密有效载荷。

方向性通道状态。 AES-GCM 要求在给定密钥下加密时初始化向量具有唯一性 [9]。H2D 和 D2H 使用不同的加密密钥，并维护独立的初始化向量序列。在每个方向内，驱动程序和复制引擎必须就每次传输所使用的密钥和初始化向量达成一致，分别记为 H2D 的 (K_{H2D}，IV $_{\mathrm{H2D}}$) 和 D2H 的 (K_{D2H}，$\mathrm{IV}_{\mathrm{D2H}}$)。驱动程序会在提交传输时跟踪每个通道的初始化向量序列，其中包括复制引擎完成传输后 CPU 解密所需的 D2H初始化向量。复制引擎通过固定的硬件接口 [15]，使用该通道的密钥和当前初始化向量对 H2D 数据进行身份验证和解密。在 H2D 路径上重复使用 D2H 密文，因此需要在 H2D 密钥下使用有效的认证标签和新的初始化向量。除此之外，还必须恢复原始明文，尽管复制引擎会使用不同的密钥和初始化向量解密所保留的密文。

2.3 标签更新与明文校正

密文重新绑定包含两个步骤：更新认证标签，使 CE 能够接受保留密文；并在 GPU 上修正 CE 的输出，以恢复原始明文。这两者

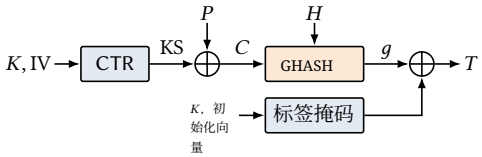

图 1：AES-GCM 数据依赖关系。此处省略 AAD 和长度编码。

个步骤利用了 AES-GCM 的结构，如图1所示，同时保持保留密文和 CE 接口不变。**AES-GCM 结构**。两个端点均使用 AES-GCM 进行认证加密 [9, 15]。我们针对一个独立认证的传输分段描述该原语。对于密钥 K、初始化向量初始化向量和明文 P，计数器模式加密会生成密文

$$C = P \oplus \mathrm{KS}(K, \mathrm{IV}), \tag{1}$$

其中，KS(K，初始化向量) 是由 AES 生成的密钥流。对于完整长度的 128 位标签，身份验证的形式为

$$T = g_K(A, C) \oplus E_K(J_0(\mathrm{IV})). \tag{2}$$

此处 A 是附加认证数据（AAD）， E_K 是一个数据块的 AES 加密结果，而 J_0（IV）是由 IV 派生的 GCM 初始计数器块。术语 g_K（A， C）表示在$H = E_K$（0^{128}）下计算的 GHASH 结果，其中包括规定的填充和长度编码。剩余项 E_K（J_0（IV））是依赖于 IV 的标签掩码。本文研究的批量数据传输使用空 AAD（$A = \epsilon$）。当上下文已明确其输入时，我们使用 KS 和 g 分别作为 KS（K, IV）和 g_K（A， C）的简写。

重新绑定认证。 对于固定的 K、 A和 C，公式（2）中的 GHASH 结果与 IV 无关。密文重新绑定利用了这种分离，通过预先计算一个认证摘要，即在目标主机到设备密钥下、依赖于密文的 GHASH 结果：

$$g_{\mathrm{H2D}} = g_{K_{\mathrm{H2D}}}(\epsilon, C). \tag{3}$$

尽管 C 是在 K_{D2H}下生成的，但其摘要必须使用 K_{H2D}，以匹配 CE 用于主机到设备认证的密钥。GPU 从受信任数据中预先计算该摘要，并将其存储在受保护的 GPU 内存中。只要 C和 K_{H2D} 保持不变，该摘要在多次主机到设备传输之间始终有效。在每次后续的主机到设备传输中，CPU 将保留的摘要与通道新鲜 IV 的标签掩码$_{\mathrm{H2D}}$组合：

$$T_{\mathrm{new}} = g_{\mathrm{H2D}} \oplus E_{K_{\mathrm{H2D}}}(J_0(\mathrm{IV}_{\mathrm{H2D}})). \tag{4}$$

因此，每次标签更新或重新标记，每个分段只需进行一次 AES 分组加密和一次 128 位异或，无需 CPU 处理有效载荷。

CE 解密后恢复明文。新标签允许复制引擎对保留的密文进行认证，但使用 H2D 密钥和初始化向量进行解密时，得到的是中间值，而不是原始明文。令 $\mathrm{KS}_{\mathrm{D2H}}$ 和 $\mathrm{KS}_{\mathrm{H2D}}$ 分别表示原始 D2H 密钥/初始化向量对和当前 H2D 密钥/初始化向量对对应的密钥流。对于存储的密文 $C = P \oplus \mathrm{KS}_{\mathrm{D2H}}$，复制引擎会将以下中间值写入受保护的 GPU 内存：

$$P' = C \oplus \mathrm{KS}_{\mathrm{H2D}} = P \oplus \mathrm{KS}_{\mathrm{D2H}} \oplus \mathrm{KS}_{\mathrm{H2D}}. \tag{5}$$

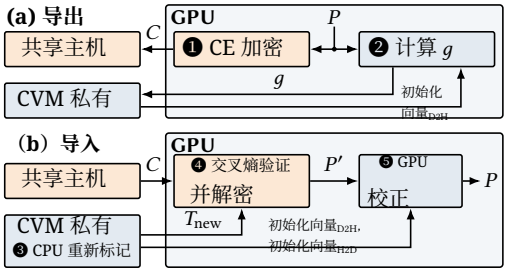

图2：密文导出与导入。此处 g 表示 g_{H2D}，并通过受保护的 D2H 路径返回。密钥提供和完成检查已省略。

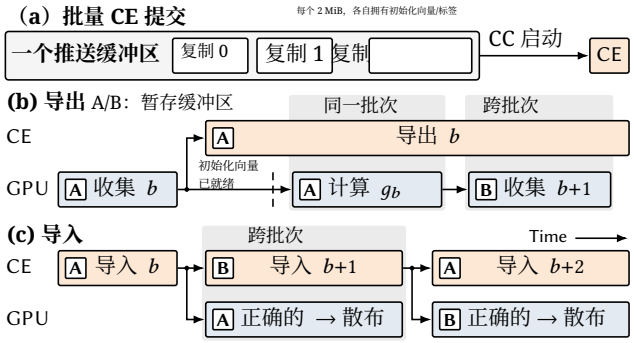

图3：批处理与流水线处理。阴影表示交叉熵/内核可能发生重叠。(b) 中的初始化向量标记表示将该批次记录的设备到主机初始化向量传送到 GPU。

GPU 内核利用计数器模式的 XOR 结构，抵消认证中间值中的两个密钥流：

$$P = P' \oplus \mathrm{KS}_{\mathrm{H2D}} \oplus \mathrm{KS}_{\mathrm{D2H}}. \tag{6}$$

2.4 密文导出与导入

导出会将 GPU 生成的密文存储在共享主机内存中，并准备可信元数据以供后续复用。每次导入都会使用这些元数据更新认证标签，并通过 CE 解密和 GPU 校正恢复原始明文。图2 展示了同一 GPU 上一个认证分段的操作。

可信状态。批量密文 C 存储在不可信共享主机内存中。对于每个导出的分段，库的 CVM 私有分段表会保留 g_{H2D}、原始初始化向量$_{\mathrm{D2H}}$，以及对源 D2H 密钥和目标 H2D 密钥的引用。该表将此状态与相应的导出分段关联起来。已建立的密钥仍保存在 CVM 和 GPU 内部的可信信道状态中。

密文导出。要导出的应用程序数据可能位于非连续 GPU 内存区域中。库首先将这些区域收集到一个受保护的 GPU 暂存缓冲区中，并将其组织为传输分段。提交 D2H 传输时，驱动程序会记录分配给它的初始化向量$_{\mathrm{D2H}}$。CE 会加密暂存的明文，并将 C 直接写入共享主机缓冲区（图❶ 中的2（a））。库通过受保护的 H2D 路径将记录的初始化向量发送到 GPU。GPU 内核使用与 CE 相同的暂存明文、D2H 密钥和记录的初始化向量，在内部重现 C，同时计算其在 K_{H2D}（❷）下的摘要，如公式（3）所定义。暂存明文必须保持不变，直到 CE 写入和摘要计算均完成，以确保摘要与 CE 写入的密文相对应。

128 位摘要通过传统的受保护的 D2H 路径返回到分段的 CVM 私有元数据中。只有在交叉熵写入和受保护的摘要回读均完成后，库才会将导出的分段提供导入。

密文导入。要导入一个分段，CPU 会从私有分段表中获取其摘要，以及导出时记录的 D2H 密钥引用和初始化向量。驱动程序会通过通道的正常推进过程分配一个新的 初始化向量$_{\mathrm{H2D}}$。CPU 使用此初始化向量和保留的 g_{H2D}，根据公式 T_{new}（4）（❸，见图 2（b））计算。CE 会读取保留的 C，验证新标签，并将数据解密到受保护的 GPU 暂存缓冲区中，得到 P'（❹）。库会将记录的 D2H 初始化向量和新分配的 H2D 初始化向量发送至 GPU，使用受保护的 H2D 路径。随后，GPU 内核根据公式（6）对 P' 进行处理，使用导出时的 D2H 密钥/初始化向量以及本次传输的 H2D 密钥/初始化向量，恢复原始明文（❺）。接着，散射内核会将恢复的明文从暂存区复制到应用程序的目标区域。只有成功完成 CE 认证、明文校正和散布后，导入才算完成。

因此，同一密文和可信元数据可支持重复导入，每次导入都使用新的初始化向量$_{\mathrm{H2D}}$ 并计算新的标签。

2.5 跨 GPU 的密文重新绑定

密文重新绑定还允许目标 GPU B 导入源 GPU A 在同一 CVM 信任域内导出的密文。保留密文 C 仍使用 A 的 D2H 密钥和初始化向量加密，而身份验证必须匹配 B 的 H2D 通道。

通过受保护的 CPU–GPU 传输，可信 CVM 向 A 提供 B 的 H2D GHASH 密钥材料，以便在导出期间计算$g_{\mathrm{H2D},\,B}$；并向 B 提供 A 的 D2H 加密密钥，用于明文校正。密钥材料保留在可信内存中，而 CVM 私有元数据将每个分段的$g_{\mathrm{H2D},\,B}$ 及其原始 D2H 初始化向量绑定到相应的源通道和目标通道。导入遵循第 2.4 节，使用 $g_{\mathrm{H2D},\,B}$ 和 B 新生成的 H2D IV 进行身份验证，并使用 A 导出时的 D2H 密钥/IV 以及 B 当前的 H2D 密钥/IV 进行校正。

为了供多个目标复用，导出会针对每个不同的目标 H2D 密钥分别准备一个摘要；若需要在本地重新导入，也会为 A自身的 H2D 密钥准备摘要。所有目标共用相同的已存储 C 和原始 D2H 初始化向量，而每个目标则使用与其对应的摘要，以及从自身 H2D 通道获取的新初始化向量。

2.6 批处理与流水线

上述操作避免了大量的 CPU 解密和重新加密，但仍会产生 CE 提交开销，以及用于收集、摘要计算、校正和散布的 GPU 工作。*HiPerfCC-StateReuse* 将复制命令批处理，以摊销提交开销，并将 CE 传输与这些 GPU 工作进行流水线处理。我们通过 C++/CUDA 库以及对 NVIDIA 统一虚拟内存（UVM）驱动程序的扩展来实现这些操作。

批量 CE 提交。我们的实现采用 2 MiB 的认证分段，与 UVM 的最大块大小一致 [28]。

UVM 驱动程序将 CE 复制命令编码到命令缓冲区或推送缓冲区中，以便提交到 CE 通道。在机密计算下，驱动程序会加密此缓冲区，受保护启动路径则会在执行前对其进行身份验证和解密 [15, 29]。因此，即使批量密文得到复用，每次提交仍需进行命令准备、受保护启动和完成跟踪。

该库会在一次驱动程序调用中提交一批分段，驱动程序则将这些复制命令分组为 CE 提交，如图3(a) 所示。这样可将驱动程序入口和受保护启动的开销分摊到多个分段上，同时每个分段仍保留各自的初始化向量和认证标签。

流水线导出。提交一批数据后，库会将其记录的 D2H 初始化向量发送到 GPU。随后，摘要计算可与该批次剩余的 CE 传输重叠进行（图 3(b)）。这两种操作读取的是同一份明文。双缓冲进一步使将下一批数据聚集到缓冲区 B 的操作与缓冲区 A 的 CE 导出重叠。只有在 CE 和 GPU 读取器均完成后，才能复用缓冲区。

流水线导入。导入使用独立的暂存存储 .

每批数据均按顺序执行 CE 身份验证和解密、校正及散布。每个 CE 在两个缓冲区之间交替使用，使数据导入缓冲区 B 与对前一批数据在缓冲区 A 中进行校正和散布重叠（图 3(c)）。只有在散布完成后才能复用缓冲区 A；只有驱动程序确认 CE 执行成功且 GPU 完成处理后，导入才算完成。

HiPerfCC-StateReuse 进一步利用 GPU 的多个复制引擎，将相互独立的导出分配到 D2H 通道，并将导入分配到 H2D 通道。每次导入都使用与所选通道对应的认证摘要和 H2D 密钥/初始化向量状态。

3 安全性分析

本节分析密文重新绑定原语的安全性。我们考察重复导入已导出密文期间的明文机密性、完整性和通道新鲜性。

3.1 机密性

性质。 控制共享主机内存和互连的对手，除了分段长度、传输的时间和顺序，以及某次导入复用了哪个存储的密文之外，无法获知任何明文信息。此保证涵盖共享内存中的存储和重复导入。

论证。 导入标签和验证结果可根据导出的密文和目标密钥生成，无需明文或源密钥。对手可以观察导出的密文及其原始 D2H 标签、为导入生成的标签，以及验证是否成功。密文由普通的 D2H 机密计算传输在新鲜初始化向量下生成，因此源通道的加密保护了其明文。每个导入标签都根据这一已公开的密文、该次导入的初始化向量和独立目标密钥计算得出。验证同样只取决于传输的密文和标签，以及目标通道状态。GPU 校正不会泄露额外信息：其中间值和恢复出的明文都保留在受保护的 GPU 内存中。

与普通 CC 传输的关系。在相同的源密钥使用量和允许观察的条件下，重新绑定继承了普通 D2H CC 传输的机密性界限，具体量化见第3.4节。重复导入不会在此界限中引入任何目的密钥项。

3.2 完整性

性质。 可信软件请求的导入操作要么失败，要么恢复其所指明的导出的明文，除非发生认证伪造，但其概率受到上界限制。

论证。 修改共享缓冲区无法使 CPU 授权修改后的密文，因为 CPU 会根据导出时保留的摘要生成每个导入标签。该摘要由与 CE 导出相同的受保护明文和设备到主机状态计算得出，与共享缓冲区的内容无关。

CE 使用与普通 H2D 传输相同的 GCM 认证方程（公式（2））检查生成的标签。无论最初使用哪个密钥加密数据，检查都取决于所传送的密文、目标密钥和当前初始化向量。每个新的初始化向量都会提供一个新的伪随机标签掩码，从而隐藏 GHASH 值，并防止观测到的标签暴露秘密哈希密钥。若要使不同的密文通过验证，对手必须选择与 GHASH 变化相匹配的标签调整。对于任何固定的密文修改和标签调整，该匹配仅对少量哈希密钥成立。这就是普通 GCM 认证中的伪造条件，其上界会随着消息长度和验证尝试次数的增加而增大 [9]。

除非此类伪造成功，否则复制引擎已验证预期密文的真实性。校正使用与该次导出绑定的源密钥和初始化向量，恢复原始明文（公式（6））。校正作用于受保护的 GPU 内存中的复制引擎输出，只有身份验证和校正均成功后，应用程序才会收到结果。**与普通机密计算传输的关系**。因此，重新绑定保留了普通主机到设备机密计算传输的保证：成功导入的数据与可信软件授权的内容一致。为现有密文添加新标签，是标准 GCM 加密之外的额外操作。将重新标记计入目标密钥的使用量后，公式（7）中的身份验证界限与 AES-GCM 的形式相同。

3.3 新鲜性

性质。每次导入仅接受经授权、适用于当前通道 IV 的密文—标签对；否则，伪造成功的概率不超过第3.4节给出的身份验证伪造概率。只要可信软件为每次传输签发新标签，对现有导出的重复导入仍然有效。

论证。CE 会根据当前可信信道状态验证每次导入，因此，从共享内存重放的 IV 无法选择较早的验证上下文。每次导入都会通过通道的正常分配路径获取一个新的 H2D IV，并与使用相同密钥的普通传输共享该路径。已分配的 IV 永不复用，即使传输失败也不例外。

在同一密钥下重复导入同一密文时，GHASH 摘要保持不变，但标签掩码会随初始化向量改变。使用 96 位 IV 和全长标签时，由于 AES 是一个置换，不同的初始化向量会产生不同的掩码，因此先前的标签会被拒绝。接受任何并非针对当前导入所生成的密文—标签对，正是第 3.2 节所分析的伪造行为。

与普通机密计算传输的关系。与普通机密计算传输一样，重放保护依赖于 GCM 认证和可信IV递进 [9]。重新绑定保留了这一规则：密文可以保留，而每次获授权的导入都会使用新的初始化向量并生成新的标签。

3.4 安全界限

我们分析普通 GCM 操作以及使用新鲜 IV 的重新标记，其中验证仅限于已为传输发放的 IV。对于使用 96 位 IV 和完整 128 位标签的密钥 K，其界限为

$$\varepsilon_K = \varepsilon_{\mathrm{AES},K} + \frac{Q(Q-1)}{2^{129}} + \frac{q_v(\ell+1)}{2^{128}}. \tag{7}$$

这三项分别反映 AES 安全性、密钥使用量累积以及 GHASH 伪造尝试，与标准 AES-GCM 界限 [20] 中的情况相同。这里 $\varepsilon_{\mathrm{AES},\ K}$ 界定了区分 AES 与随机置换的能力，而 Q 界定了在 K 下生成可观测加密输出和标签输出所需的不同 AES 输入数量。在伪造项中，q_v 界定验证尝试次数，ℓ 界定消息中 128 位密文块的数量，不包括长度块。普通加密每个 128 位数据块会增加一个载荷块输入，并向 Q增加一个标签掩码输入；每次重新绑定导入仅增加一个标签掩码输入，因为其中间明文仍受到保护。使用同一密钥的普通流量和重新标记共用这一预算。

对于源密钥和目标密钥，将这些界限记为 $\varepsilon_{\mathrm{src}}$ 和 $\varepsilon_{\mathrm{dst}}$。在固定允许观察内容的情况下，攻击者识别导出的两个等长候选明文中哪一个的成功概率至多为 $1/2 + \varepsilon_{\mathrm{src}}$。任何一次导入接受授权密文—标签对以外内容的概率至多为 $\varepsilon_{\mathrm{dst}}$。校正正确性和可信 IV 递进将这一身份验证界限转化为上述完整性和新鲜性保证。

机械化验证。我们使用 EasyCrypt [7]，在密文重新绑定的抽象模型中对校正恒等式以及机密性和完整性归约进行了机器检查。该模型考虑一个源密钥和一个目标密钥，并涵盖常规流量、重复导入和自适应验证尝试。模型使用抽象密码学原语，并假设通道状态和元数据可信、内部传输安全，且应用程序数据与密钥无关。CUDA 内核、驱动程序和硬件不在验证范围内。

4 案例研究

我们通过三个案例研究评估密文重新绑定原语的性能：模型切换、KV缓存卸载和跨实例 KV 共享。在每个案例研究中，服务运行时都会通过通用导出/导入接口保留并恢复应用程序状态。

4.1 模型切换

多模型服务系统会根据需求变化调整 GPU 驻留，以减少为不常用模型预留的资源 [12, 47]。服务运行时支持释放 GPU 权重内存，同时保留主机副本以便之后恢复 [10]。在机密计算（CC）下，通过传统传输路径进行的每次恢复仍然需要对完整的权重载荷进行 CPU 加密，即使权重未发生变化也是如此。

HiPerfCC-StateReuse 会保留一份已导出权重的密文副本，以便反复恢复。首次释放时，它会将已加载的权重导出到共享主机内存，并将相关元数据保存在 CVM私有内存中。只要权重和通道密钥保持不变，后续释放就无需再次导出。每次恢复时，系统都会使用新的 H2D IV 导入保留的密文，从而避免在 CPU 上进行大规模重新加密。导出成本会由多次恢复分摊。权重版本发生变化时，则需要重新导出。

我们将相同的导出/导入接口集成到 SGLang [57] 和 vLLM [10]的权重内存释放与恢复流程中。它们现有的内存管理流程会在释放和恢复期间保留权重的虚拟地址，从而维持张量和执行图所持有的引用。释放期间，运行时会等待任何必要的导出操作完成，然后再释放 GPU 物理内存。恢复时，运行时会还原内存映射并导入保留的密文，待权重可用于推理后才返回。

4.2 KV 缓存卸载

多轮对话以及共享长前缀的请求可以复用先前计算得到的 KV 缓存，但随着工作集不断增大，GPU 内存无法保留所有有用的前缀。CachedAttention 和 HiCache 等分层缓存系统可将 KV 状态保留在 GPU 内存之外，使后续请求能够恢复缓存的前缀，避免重复预填充 [13, 48]。在机密计算（CC）环境下，传统路径会在 CPU 上解密每份备份，并在每次恢复时重新加密。

我们通过将每个缓存页映射到共享主机内存中导出的密文段，把 *HiPerfCC-StateReuse* 集成到 SGLang HiCache 的主机内存层。相关的分段引用和密码学元数据保留在 CVM 私有内存中。

HiCache 备份选定页面时，会将所有层中的 K 和 V 区域提交导出。在导出完成之前，GPU源页保持有效且不变。只有导出完成后，主机副本才可用于恢复，GPU源页也才能回收。当请求的前缀不在 GPU 内存中、但存在于主机缓存中时，HiCache 会分配目标GPU页并导入相应的密文。GPU校正与散布会将恢复的 KV 数据放入目标布局中。只有导入完成后，注意力才能使用恢复的页面。

只要内容和通道密钥保持不变，保留的主机条目就可以继续复用。驱逐主机条目会移除其私有引用，只有在未完成的传输结束后，其存储才能重新使用。

4.3 跨实例 SharedKV池

为满足不断增长的请求需求，系统会在多个 GPU 上运行同一模型的多个实例。存储在主机内存或外部存储层中的共享 KV 缓存，可让请求复用其他实例计算出的前缀，从而避免重复预填充 [33]。Dynamo 等请求路由器在选择请求的执行位置时，会同时考虑缓存可用性和实例负载 [31]。在机密计算下，

传统路径会在接收实例每次恢复共享 KV 时，使用 CPU 对其重新加密。

我们扩展了 HiCache 集成，使同一 CVM 内不同 GPU 上服务相同模型的 SGLang 实例能够共享导出的 KV 状态。导出的副本可为对等实例提供缓存命中，直到其所有者撤回该副本以回收主机容量。

实例 *A* 通过第 2.5 节所述的过程导出选定的键值页，从而填充池。只有在导出完成后，它才会使用这些页面对应的前缀缓存键发布这些页面。Dynamo 使用来自池的发布和移除事件，在实例之间路由请求时记录这些共享前缀 [33]。

实例 *B* 发生共享缓存命中时，会创建一个引用 *A*已导出页面的主机缓存条目，而不会将有效载荷复制到另一个主机缓冲区中。在导入数据之前， *B* 会检查共享副本是否仍然可用，并将该传输注册为活动状态。随后，它使用自己的 H2D 通道，将保留密文导入已分配的 GPU 页面。传输会一直保持活动状态，直到确认导入完成，之后注意力便可使用恢复的 KV 数据。

在主机容量压力下， *A* 会撤回新导入中的一页。它向对等实例发送失效通知，并将移除操作报告给路由器。撤回操作与可用性检查和传输注册串行执行：先完成注册的导入可能成功，而之后的尝试则被视为缓存未命中。确认活动导入完成后，即使 *B* 仍持有对已撤回页面的主机缓存引用， *A* 也可以回收主机存储空间。已恢复到 *B*的 GPU 中的 KV 仍可继续使用。

5 评估

我们的评估表明，通过避免批量 CPU 解密和重新加密，*HiPerfCC-StateReuse* 仅需 Multi-context 机密计算每吉比字节 CPU 时间的一小部分，同时在 H200 上实现接近关闭 CC 时的传输带宽。传输微基准测试还分别揭示了批处理与流水线处理的影响。这些传输收益使 SGLang 和 vLLM 中从保留的主机副本加载模型的延迟都接近关闭 CC 时的水平。随后，我们通过单实例 KV缓存卸载和跨实例 KV 共享，研究其对服务吞吐量和延迟的影响。我们还量化了内存开销。此外，在 RTX PRO 6000 上进行的传输和 KV缓存卸载实验评估了这些收益在不同 GPU平台上的变化情况。

5.1 实验设置

实现与测试平台。 *HiPerfCC-StateReuse* 修改了 NVIDIA 的开源图形处理器内核模块 [28] ，并提供了包含服务适配器的 C++/CUDA 库。我们的主要测试平台是一台 AMD SEV-SNP [1] 机密虚拟机，配备 16 个虚拟中央处理器、480 吉比字节内存和两块 H200 NVL GPU，运行于 AMD Zen 4 主机上。每块 GPU 都通过 PCIe 5.0 ×16 连接到主机。我们还在 AMD Zen 5 主机上评估了一块 96 GB RTX PRO 6000 Blackwell Server Edition GPU，并为机密虚拟机分配了 16 个虚拟中央处理器。我们使用 NVIDIA 驱动程序 610.57.04、CUDA 13.0 和 PyTorch 2.13.0 [35]。我们的服务集成基于 SGLang 0.5.21 [57] 和 vLLM 0.30.0，并使用 Dynamo 1.6.1 [31] 路由跨实例请求。

对比系统。 *NativeCC* 在 GPU 机密计算下，使用单个上下文中的传统 CUDA 传输，并通过 CPU 进行有效载荷密码运算。多上下文 *CC* 遵循 Serialized Bridge [52], , 并行处理这些传输：每个 GPU 使用多个 CPU 工作线程和八个 CUDA上下文。*HiPerfCC-StateReuse* 使用密文导出与导入，默认启用批处理和流水线处理。关闭 *CC* 则提供了一个参考方案，在禁用 GPU 机密计算的情况下使用原生传输。

KV 服务配置。 在每组对比中，所有路径均使用相同的模型、精度和主机内存预算。我们将 SGLang 的静态 GPU 内存占比设为 0.9，并在各路径中固定每个模型的 GPU 缓存容量。H200 键值实验使用总计 460 GB 的主机缓存预算，并在两个实例之间平均分配共享 KV。所有路径均使用直写备份，并在所需的 KV 恢复完成之前暂缓批次准入。阻塞式恢复在调度器线程之外运行，遵循与 HiCache 异步加载提交相同的原则 [38]。

KV 服务工作负载。 对于多轮复用，我们使用基于 SGLang 的 HiCache 多轮基准测试 [41], 的合成工作负载，记为 多轮。每个会话包含 10 轮，初始有 8,192 个输入令牌，之后每轮新增 256 个输入令牌。每轮生成 64 个令牌，后续轮次包含之前的输入和生成的响应。在 H200 上，Llama 和 Qwen 使用 240 个会话，GLM 使用 100 个会话。在 RTX PRO 6000 上，Llama 使用 150 个会话。

对于共享文档复用，遵循先前关于分层 KV 缓存的研究 [49], , 我们使用 LooGLE 短依赖问答子集中的 105 篇文档 [24] 以及 NarrativeQA 测试集中的 40 篇文档 [22]。生成长度依据这些数据集提供的响应文本确定。

我们对三种工作负载均扫描请求到达率；单实例服务最多允许 128 个请求同时在途，共享 KV 最多允许 256 个请求同时在途。LooGLE 和 NarrativeQA 使用泊松到达。多轮工作负载和 LooGLE 从空缓存开始。遵循 Strata [49],, 单实例 NarrativeQA 从预填充主机 KV 和空 GPU 缓存开始。

测量方法。 传输带宽按每秒完成的有效载荷字节数计算。CPU 开销为基准测试线程的用户态与内核态时间总和，按每吉比字节的有效载荷计算。我们报告三种工作负载完整重放期间的服务输出令牌吞吐量和平均首令牌时间（TTFT）；首令牌时间从客户端派发到收到首个令牌进行测量。测量前先进行主机池分配和编译预热。

5.2 传输性能

我们在 H200 和 RTX PRO 6000 上使用 1 吉比字节的连续缓冲区，测量持续导出（设备到主机）和导入（主机到设备）的带宽。每种配置先预热 2 秒，随后测量 10 秒。

传输带宽。图4 （上）显示，*HiPerfCC-StateReuse* 在两个平台和两个方向上，带宽均优于原生 CC。在 H200 上，其带宽达到 54–57 GB/s，接近关闭 CC 时的带宽，并比多上下文 CC 高出 1.9–2.3×。在 RTX PRO 6000 上，其导出性能与多上下文 CC 持平，导入带宽则提升了 1.6×，但与关闭 CC 时的带宽仍有较大差距。

在 RTX PRO 6000 上，更快的 CPU 端载荷密码学提升了原生 CC 和多上下文 CC 的带宽，从而缩小了密文重新绑定带来的相对收益。Zen 5 主机的全宽

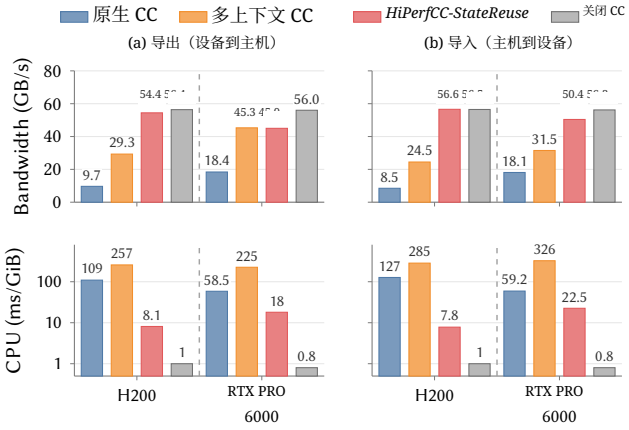

图 4：1 GiB 负载的传输性能。上：带宽。下：每 GiB 的总 CPU 时间（对数刻度）。

512 位矢量执行可能有助于提升密码运算速度。H200 主机的 Zen 4 核心则将 512 位操作分两步执行，每步 256 位 [2, 4]。

移除 CPU 载荷加密后，CE 并行度成为影响 *HiPerfCC-StateReuse* 带宽的重要因素。CC 传输需要支持硬件加密和解密的复制引擎 [28]。在 RTX PRO 6000 上，驱动程序提供两个这样的复制引擎，我们分别将其分配用于导出和导入。H200 提供了足够多的独立复制引擎，因此每个方向可以使用两个。RTX PRO 6000 较低的 CE 并行度导致其与关闭 CC 之间仍存在带宽差距。

收集和散布非连续 KV 数据几乎不会损失带宽：在两个平台上，分页缓冲区传输的性能均保持在连续传输的 3% 以内。

CPU 开销。 *HiPerfCC-StateReuse* 在两个平台和两个方向上，相较于多上下文 CC，均可将每 GiB 的 CPU 时间降低 92–97%（图4，下）。多上下文 CC 将负载加密和解密分布到 CPU 工作线程上，而密文重新绑定避免了这种按字节进行的 CPU 工作。因此，即使 RTX PRO 6000 的导出带宽相当，其 CPU 需求也显著更低。

批处理与流水线处理。在两个平台上进行 1 吉比字节传输时，第2.6 节中的两项优化结合使用，相较于两者均未启用的配置，可将带宽提高 23–60%。表1报告了在暂存分配和复制引擎数量保持不变时的带宽和 CPU 开销。

批处理通过在多个分段间分摊驱动程序入口调用和受保护命令提交的开销，提高带宽并降低 CPU 开销。在禁用流水线处理时，批处理可使两个平台的导出带宽提高 26–33%，而对导入带宽的提升较小。在 H200 上，它还将每 GiB 的 CPU 时间降低 70–71%。

启用批处理时，流水线处理主要通过提升带宽发挥作用，在两个平台和方向上的带宽均提高了 11–30%。未启用批处理时，唯一显著的提升是 RTX PRO 6000 的导入带宽提高了 25%。对于导出，单独提交的分段可能会在收集的初始化向量到达 GPU 进行摘要计算之前就完成传输，导致几乎没有可用于

表 1：1 吉比字节传输场景下批处理与流水线处理的消融实验。带宽：带宽（GB/s）；CPU：CPU 时间（毫秒/吉字节）。

GPU	方向	均非 带宽	均非 CPU	批处理 带宽	批处理 CPU	流水线处理 BW	流水线处理 CPU	Both 带宽	Both CPU
H200	导出	34.0	19.1	42.8	5.7	34.9	18.9	54.4	8.1
	导入	46.1	27.5	49.2	8.0	46.1	27.3	56.6	7.8
RTX PRO 6000	导出	30.5	29.0	40.5	18.2	30.8	28.9	45.0	18.0
	导入	37.7	23.2	38.8	22.3	47.2	23.2	50.4	22.5

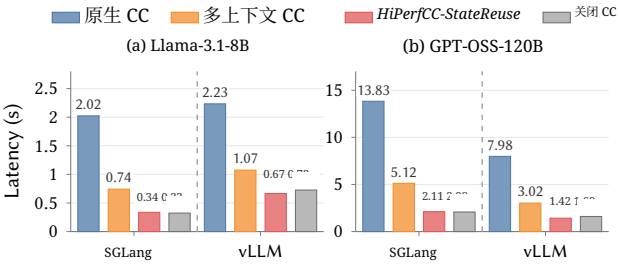

图 5：H200 上完整模型恢复的延迟。

重叠的工作。批处理会在摘要计算开始时保留更多待处理的 CE 传输，从而实现更大程度的重叠。对于导入，H200 已能在其两个复制引擎之间实现 GPU处理与传输的重叠，但仍需等待暂存区复用。禁用流水线处理时，RTX PRO 6000 的单个导入复制引擎则必须等待校正和散布操作释放其暂存缓冲区。双缓冲使下一批数据能够在 GPU处理继续进行时到达。

5.3 模型切换

设置。我们使用 SGLang 和 vLLM，在一台 H200 上评估 Llama-3.1-8B-Instruct（BF16），并按照 Serialized Bridge [52], 评估 GPT-OSS-120B（MXFP4）。在 SGLang 完全释放内存或 vLLM 进入一级休眠后，我们测量完整恢复或唤醒调用所需的时间，包括 GPU 内存分配、映射和权重恢复。

恢复性能。图5 显示，*HiPerfCC-StateReuse* 在两种模型和引擎中，相较于多上下文 CC，将完整恢复延迟降低了 38–59%，使其接近关闭 CC 的基准水平。例如，在 SGLang 中恢复 GPT-OSS 仅需 2.11 秒，而多上下文 CC 下需要 5.12 秒。

在 vLLM 中，GPU 分配和映射限制了较小 Llama 模型的延迟降幅，因为传输其权重所需的时间较短。重新创建其分配和映射（包括 KV 缓存区域）约需 0.37 秒，而权重导入需 0.29 秒。对于较大的 GPT-OSS 模型，权重传输占恢复时间的主要部分，因此更快的导入能够更大幅度地缩短完整唤醒延迟。

5.4 KV 缓存卸载

设置。 我们在一块 H200 上使用 Llama-3.1- 80 亿参数-Instruct [25], Qwen3.8-27B [40], 和 GLM-4.5-Air [55] 的 4 位 AWQ 检查点 [8]，评估 SGLang HiCache。这些模型涵盖了一个稠密 Transformer

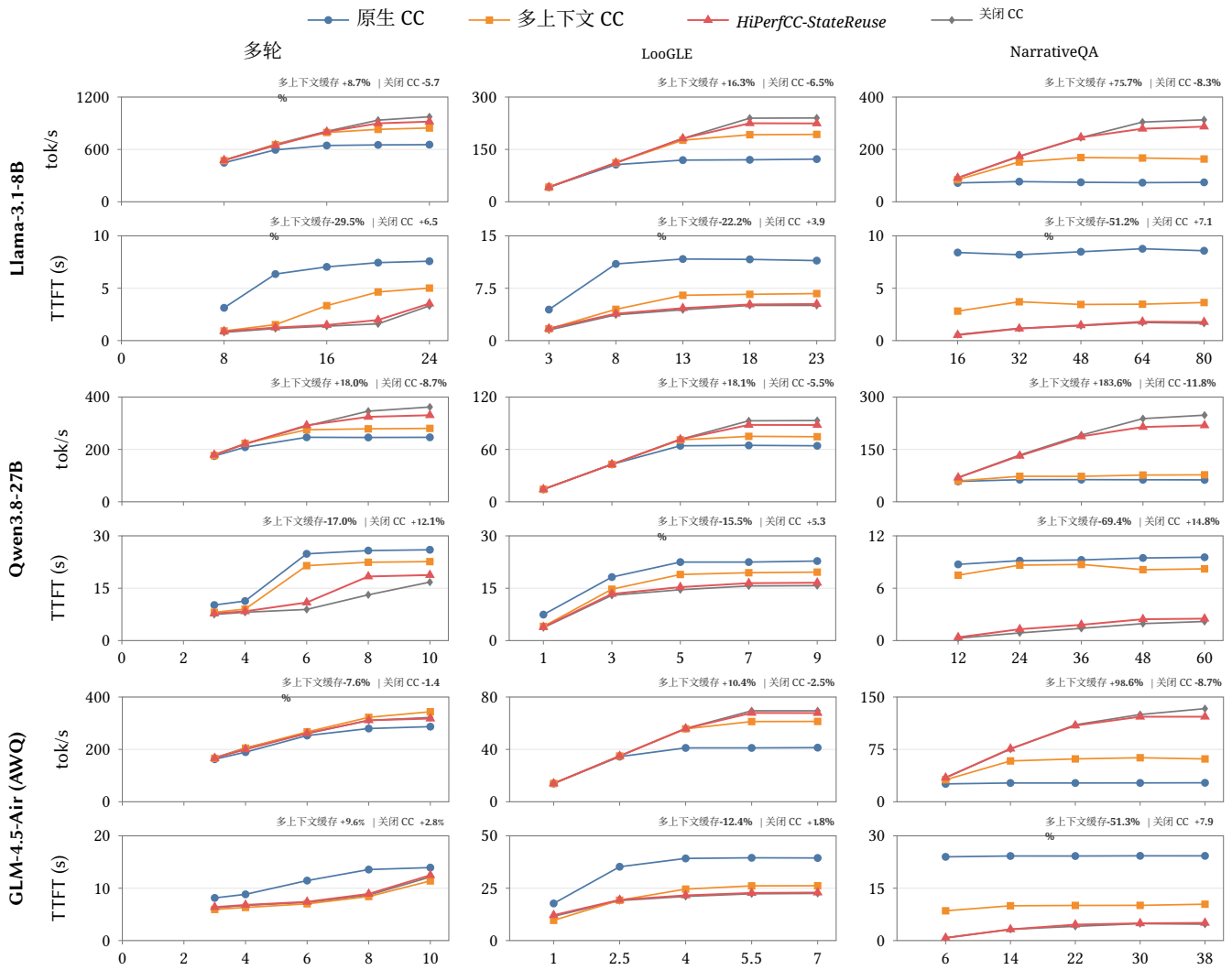

图6：H200 键值卸载结果。横轴显示请求到达率（请求/秒）。标签显示*HiPerfCC-StateReuse*在最大负载下相对于多上下文 CC (多上下文缓存) 和关闭 CC 的百分比变化。

（Llama）采用全注意力、（Qwen）采用线性注意力与全注意力的混合结构，以及（GLM）采用混合专家（MoE）Transformer。我们使用第 5.1 节中介绍的多轮、LooGLE 和 NarrativeQA 工作负载。

服务性能。对于每个模型–工作负载组合中评估的四种路径，在最高负载下，*HiPerfCC-StateReuse* 在38×个模型–工作负载组合上的几何平均吞吐量方面达到多上下文 CC 吞吐量的 1 倍，并将平均首令牌时间缩短了 28.8%（图6）。更快的键值恢复缩短了请求在批次准入前的等待时间，从而改善首令牌时间，并在更高负载下实现更高吞吐量。

在各类工作负载中，NarrativeQA 为这三个模型带来的吞吐量和首令牌时间提升幅度最大。由于其主机 KV 已预先填充，且回答较短，因此测得的重放过程中需要进行的新计算较少。多轮和 LooGLE 从空缓存开始进行初始预填充，而多轮还会在每轮处理新输入，并生成较长的响应。这些额外计算限制了更快恢复所带来的端到端收益，同时也为将恢复操作与其他请求的 GPU 工作重叠执行创造了机会。

多轮场景下的 GLM 展示了有限的提升空间：原生 CC 和 *HiPerfCC-StateReuse* 的吞吐量都接近关闭 CC 时的吞吐量，这表明恢复并非主要瓶颈。在此工作负载中，只有前序轮次完成后，后续轮次才符合执行条件，因此执行时序也会影响请求交错和缓存复用。在评估的最高负载下，多上下文 CC 从主机内存恢复的键值比关闭 CC 时少约 27%，这有助于解释其为何优于关闭 CC 和*HiPerfCC-StateReuse*。

在各模型中，Qwen 在三个工作负载上的吞吐量提升均高于多上下文 CC。例如，在 NarrativeQA 上，与 Llama 或 GLM 相比，多上下文 CC 为 Qwen 带来的性能提升低于原生 CC。SGLang 的混合缓存按页组织主机数据，而设备 KV 则按层组织。对于

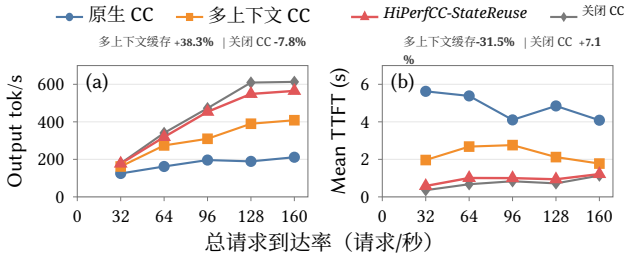

图 7：在两块 H200 GPU 上运行的两个 Llama-3.1-8B 实例，已启用 KV 共享。吞吐量为两个实例的总和。

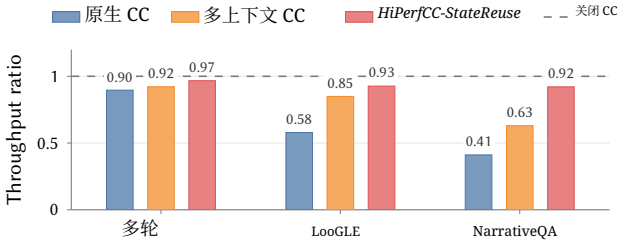

图 8：在 RTX PRO 6000 上使用 Llama-3.1-8B 进行 KV 卸载。吞吐量按关闭 CC 时的吞吐量归一化。

原生 CC 和多上下文 CC，这种布局不匹配会使 KV 恢复被拆分为许多跨越多个页面和层的小型拷贝。Llama 和 GLM 使用的布局则允许将相邻页面合并为较大的拷贝。多个上下文可以并行处理这些小型拷贝，但仍会产生提交和同步开销。*HiPerfCC-StateReuse* 会批量处理密文导入，并在 GPU 上恢复设备布局，从而同时降低这类开销和 CPU 载荷加密的开销。

在较高提供负载下，几条首令牌时间曲线趋于平稳。达到 128 个请求的在途请求上限后，每次新调度都必须等待较早的请求完成。这限制了服务器端并发，并使额外等待发生在客户端，且此时首令牌时间的测量尚未开始。

5.5 跨实例共享键值池

设置。 我们评估两个 SGLang 实例，它们在同一个 CVM 中各使用一块 H200，以 BF16 提供 Llama-3.1-8B-Instruct 服务。四种路径均启用跨实例 KV 共享，并通过 Dynamo 重放 Narra-tiveQA [22] through Dynamo [31]。重放之前，每份文档都会在两个实例之一上预填充，并备份到主机内存；由此产生的 GPU 缓存状态会予以保留。工作集超出 GPU 缓存总容量，但可容纳在共享主机池中。

服务性能。当请求到达率总计为每秒 160 个请求时，*HiPerfCC-StateReuse* 实现了 1。38× 多上下文 CC 的吞吐量，并将平均首令牌时间降低了 31.5%，达到关闭 CC 时吞吐量的 92.2%（图7）。

所有路径都支持跨实例键值复用，因此这些提升证明了：当请求使用由另一块 GPU 生成的键值时，加速恢复过程能够带来收益。*HiPerfCC-StateReuse* 会跨 GPU 通道复用保留密文，从而避免 CPU 重新加密；而多上下文 CC 在每次恢复时仍会执行该操作。

HiPerfCC-StateReuse 在整个负载扫描过程中，首令牌时间始终低于多上下文 CC，尽管绝对首令牌时间会随负载呈非单调变化。我们的执行日志显示，对于相同的请求轨迹，负载较高的运行从主机内存恢复的键值更少。在同一实例上共享文档前缀的并发请求会复用已恢复的键值，因此一次恢复即可服务多个请求，从而减少每个请求所需的恢复工作。

5.6 内存开销

HiPerfCC-StateReuse 使用可复用的 GPU 缓冲区和 CVM 私有主机元数据。我们的配置为每个传输方向分配 1 GiB 的 GPU 暂存区，每块 GPU 共计 2 GiB。

对于每个 2 MiB 密文段，可信元数据会保留其 12 字节的 D2H 初始化向量，以及针对每个不同目标 H2D 密钥的 16 字节认证摘要 g，涵盖同一块 GPU 和其他 GPU 上的候选导入通道。在我们的单GPU H200配置中，每条库分段记录占用 112 字节，相当于每 GiB 密文容量约占 0.055 MiB。

5.7 GPU平台间的通用性

设置。我们在一块 RTX PRO 6000 上使用 BF16 格式的 Llama-3.1-8B-Instruct，借助 SGLang Hi-Cache 和第 5.1 节所述的三种工作负载，重复进行了 KV缓存卸载实验。

服务性能。在每种工作负载下接近关闭 CC 饱和点的相同负载处，*HiPerfCC-StateReuse* 的吞吐量达到多上下文 CC 的 1.19× 倍；在三种工作负载上的几何平均值中，其吞吐量达到关闭 CC 时吞吐量的 93.8%。相较于多上下文 CC，其平均首令牌时间平均减少了 28.2%。图 8 报告了按关闭 CC 时的吞吐量归一化后的吞吐量。与 H200 上的趋势一致，NarrativeQA 相较于多上下文 CC 展现出最大的吞吐量增幅。

如第5.2节所示，RTX PRO 6000 主机上更快的 CPU侧密码学增强了传统传输基线的表现，而单个复制引擎限制了*HiPerfCC-StateReuse*的导入带宽。这些差异缩小了其相较于 H200 的传输优势。

6 讨论与局限性

主机内存共享与可用性。拒绝服务（DoS）攻击不属于本文采用的传统 CPU–GPU机密计算威胁模型 [1, 15]。准备密文池需要将主机页面从 CVM私有内存转换为共享内存，以供 GPU直接内存访问。在 SEV-SNP 下，客户机与虚拟机监控程序协调执行这些更改 [1]。共享页面存放 GPU 导出的密文，而密钥、摘要以及将每个分段绑定到其导出的元数据，仍受私有内存保护。正如第 3.2 节所分析的，CPU 根据私有摘要派生标签，因此篡改共享密文无法授权被篡改的内容，只会导致拒绝服务攻击。即使进行普通的 CC 传输，恶意主机也可以扣留资源或干扰传输，从而造成可用性问题。

访问模式泄漏与安全范围。侧信道攻击和访问模式泄漏不属于本文采用的传统 CPU–GPU机密计算威胁模型 [1, 15]。现有机密计算系统即使保留了有效载荷机密性和传输认证，仍可能暴露应用程序的访问模式。例如，

共享 KV 可通过响应时间和服务顺序泄露用户缓存的提示词前缀 [42, 46],，而在 SEV-SNP 和 TDX 上，不受信任的主机可以通过页级和微架构侧信道观测应用程序的访问轨迹 [17, 45]。

在我们的设计中，主机也可以识别同一导出的KV段的重复导入，因为保留密文存储在共享内存中，即使使用新的 IV 和认证标签，它也保持不变。如果主机已将某个导出的分段与已知提示前缀关联起来，那么识别后续导入可能会暴露对该前缀的访问。尽管共享内存中不变的密文可能使访问模式泄漏更容易发生，但它不会损害有效载荷加密，也不会泄露独立加密的分段之间的明文相等性。要防止此类访问模式泄漏，需要采用额外机制来隐藏缓存复用和可观察到的应用程序行为；对于基于TEE的系统而言，这是一个更广泛的挑战，超出了本工作的范围。

密钥轮换与保留状态生命周期。每字节重新标记所消耗的密钥预算远低于普通传输。长期运行的服务会轮换通道密钥，因为随着使用同一密钥加密的数据增多，安全界限也会增大。公式（7）中的三项里，AES 项假定可忽略不计，而全长标签以及身份验证失败时终止机制会使伪造项保持较小。剩余的二次密钥使用项 $Q(Q-1)/2^{129}$随累计加密工作量增长，因此决定基于流量的使用上限。NVIDIA 的驱动程序根据该项 [28], 设置轮换阈值，默认预算为 2^{-60}。

使用 2 MiB 分段时，该预算相当于每个密钥可进行约 390 GB 的普通传输。重新绑定每次 2 MiB 导入需要一次 AES 计算，以生成一个 16 字节的标签掩码。因此，同一预算允许生成约 390 GB 的标签掩码；若密钥仅用于重新标记，则可支持约 51 PB 的累计导入量。同一密钥下的普通传输和受保护元数据也会消耗部分预算，从而减少轮换前可支持的重新绑定数据量。

要在轮换期间保留密文，就必须保存恢复密文所需的状态，并更新用于验证其导入的摘要。D2H 轮换会更改新导出的密钥，而现有密文仍可通过其原始密钥和初始化向量恢复。库会保留活动分段或进行中的操作所需的源密钥版本，并回收已过时的版本。H2D 轮换不会更改密文，但需要使用目标密钥生成新的认证摘要 g。当获知 H2D 密钥即将达到上限时，库会通过经过身份验证的路径导入保留的数据，并在后台于 GPU 上计算新摘要。摘要就绪后，库会请求切换密钥，并在后续导入时使用相应的摘要。如果驱动程序在准备完成前轮换密钥，则没有新摘要的条目将无法导入。

在对 D2H密钥轮换和 H2D 摘要刷新进行的独立测量中，所有保留的分段均仍可导入。在 H200 上，我们估计，在扣除稳态完成间隔后，D2H密钥轮换会使已完成的 1 吉比字节导出之间的间隔增加约 130 毫秒。在一项独立测量中，为约 430 吉比字节的保留数据刷新 g，每个 H2D 密钥约耗时 15 秒。

在以恢复为主的工作负载中，密文重新绑定处理了大部分 H2D 负载流量，减缓了 H2D 密钥预算的消耗，并减少了摘要所需的刷新次数。在 H200 KV 缓存实验中，每次重放的累计密文导入量达到约 9.3 TB，不到单个密钥 51 PB 重新标记容量的 0.02%。我们观察到了 D2H密钥轮换，但在这些运行中没有观察到 H2D 密钥轮换或摘要刷新。

7 相关工作

机密 GPU 数据移动。 先前研究对机密 GPU 工作负载中的传输和运行时开销进行了分析 [34, 51]。一些设计降低了受保护传输开销，但需要修改硬件或 GPU CC 协议，而现有机密 GPU 不支持这些修改。LÆGIS 提出扩展 GPU，并配备 HBM 驻留的初始化向量存储区，从而使 CPU 预加密与 UVM 页迁移顺序解耦 [50]。TensorTEE 在模拟环境中评估，通过在张量级统一 CPU 和 NPU 可信执行环境（TEE）的保护粒度，使两者之间无需重新加密即可传输数据 [16]。Fastrack 提出修改 GPU CC 协议，以建立直接的客户端–GPU 安全通道，避免对客户端输入进行 CPU 解密和重新加密 [44]。AMD SEV-TIO 和 Intel TDX Connect 提供可信 I/O，使可信设备能够直接访问 CVM私有内存，无需使用共享主机缓冲区，但前提是主机、设备、固件和软件兼容 [3, 18]。

相比之下，针对现有机密 GPU 的优化主要通过重叠或并行执行 CPU 密码运算，仍需承担 GPU 状态备份与恢复中按字节计算的开销。PipeLLM 预测传输缓冲区和初始化向量，以便让 CPU 加密与 GPU 计算重叠 [43]。Serialized Bridge 使用持久化 CUDA 上下文池，解决机密 CPU–GPU 模型加载中的串行化问题 [52]。

保留密文提供了另一种减少重复加密工作的途径。PipeLLM 将未更改的交换数据复用密文视为未来硬件和库支持的一个机会 [43]。FLASH 会在检查点中保留 GPU 生成的密文，从而避免在创建检查点时进行 CPU 解密和重新加密 [23]。其恢复路径会在 CVM 内解密保存的 GPU 状态，并通过传统的受保护 H2D 传输将其恢复。相比之下，密文重新绑定通过现有 CE 接口导入保留的密文，并根据目标通道的密钥和新 IV 调整身份验证和明文恢复。*HiPerfCC-StateReuse* 利用这一原语，在 CVM 信任域内实现重复恢复和跨 GPU 共享，无需更改 GPU 硬件或固件，即可避免大量 CPU 解密和重新加密。

模型与 KV 状态管理。 Aegaeon 为并发模型服务提供 GPU 池化，而 ServerlessLLM 优化模型加载、部署和迁移 [12, 47]。CachedAttention 和 HiCache 在内存与存储层级之间组织 KV 复用 [15, 48]。Strata 通过 GPU 辅助 I/O 和缓存感知调度来解决碎片化 KV 传输问题 [49]。Mooncake 和 Dynamo 将分布式 KV 复用与请求调度协同起来 [31, 39]。*HiPerfCC-StateReuse* 通过降低 CC 环境下传输保留的 GPU 状态所需的 CPU 密码运算成本，补充了这些部署、缓存和调度技术。

8 结论

HiPerfCC-StateReuse 实现了机密虚拟机信任域内重复复用密文和跨 GPU 密文复用的密文重新绑定。保留的认证摘要支持 CPU 更新标签而无需处理有效载荷，同时 GPU 校正可在使用新鲜通道状态进行 CE 解密后还原原始明文。我们分析了其在所述威胁模型下的机密性、完整性和信道新鲜性。我们的批处理流水线驱动程序和库无需改动 GPU 硬件或固件。与多上下文 CC 相比，*HiPerfCC-StateReuse* 在 H200 和 RTX PRO 6000 上将每 GiB 的 CPU 时间降低了 92–97%，并在 H200 上实现了 1.9–2.3× 的传输带宽。在 H200 上，它将模型恢复延迟降低了 38–59%；在 KV卸载场景下，其几何平均输出词元吞吐量达到基线的 1.38× 倍；在跨实例 KV 共享场景下，其吞吐量达到基线的 1.38× 倍。

参考文献

[1] 超威半导体。2020年。AMD SEV-SNP：通过完整性保护等措施强化虚拟机隔离。白皮书 70366。超威半导体。 https://docs.amd.com/v/u/en-US/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more [2] AMD。2023年。第 4 代 AMD EPYC 处理器架构。白皮书，第三版。https://www.amd.com/content/dam/amd/en/documents/products/epyc / 4th-gen-epyc-processor-architecture-white-paper.pdf [3] AMD。2023年。AMD SEV-TIO：面向安全加密虚拟化的可信 I/O。白皮书。 https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf [4] AMD。2025年。第 5 代 AMD EPYC 处理器架构。白皮书，第一版。https://docs.amd.com/api/khub/documents/UIqhAbjRhgnzgzzdVU4pUw/content [5] EmilyApsey、Phil Rogers、Michael O'Connor 和 Rob Nertney。2023年。用于安全可信人工智能的 NVIDIA H100 GPU 机密计算。NVIDIA 技术博客。 https://developer.nvidia.com/blog/confidential-computing-on-h100-gpus-for-secure-and-trustworthy-ai/ [6] Julien Aubry。2026年。GPU 上的机密计算如何工作（H100、H200、Blackwell）。VoltageGPU 技术博客。 https://voltagegpu.com/blog/how- confidential-computing-works-on-a-gpu [7] Gilles Barthe、Benjamin Grégoire、Sylvain Heraud 和 Santiago Zanella Béguelin。2011年。面向密码学工作者的计算机辅助安全证明。载于《密码学进展——CRYPTO 2011》（《计算机科学讲义》，第 6841 卷）。Springer，第 71–90 页。[8] cyankiwi。2025年。GLM-4.5-Air-AWQ-4bit。Hugging Face模型仓库。https: // huggingface.co/cyankiwi/GLM-4.5-Air-AWQ-4bit [9] Morris Dworkin。2007年。分组密码工作模式建议：伽罗瓦/计数器模式（GCM）和 GMAC。NIST 特别出版物 800-38D。美国国家标准与技术研究院。doi:10.6028/NIST.SP.800-38D [10] Embedded LLM。2025年。使用 vLLM 休眠模式实现零重载模型切换。vLLM 博客。 https://vllm.ai/blog/2025-10-26-sleep-mode [11] Cédric Fournet。2024年。Microsoft Azure 机密计算。瑞士联合研究中心春季研讨会，苏黎世。 https://www.microsoft.com/en-us/research/wp-content/uploads/2024/05/Confidential-Computing-Azure-Research-Swiss-Joint-Research-Centre-Spring-Workshop-Zurich-2024.pdf 幻灯片：机密模式下的内存管理方式。[12] Yao Fu、Leyang Xue、Yeqi Huang、Andrei-Octavian Brabete、Dmitrii Ustiugov、Yuvraj Patel 和 Luo Mai。2024年。ServerlessLLM：面向大语言模型的低延迟无服务器推理。载于第 18 届 USENIX 操作系统设计与实现研讨会（OSDI 24）。USENIX 协会，加利福尼亚州圣克拉拉，第 135–153 页。 https://www.usenix.org/conference/osdi24/presentation/fu [13] Bin Gao、Zhuomin He、Puru Sharma、Qingxuan Kang、Djordje Jevdjic、Junbo Deng、Xingkun Yang、Zhou Yu 和 Pengfei Zuo。2024年。利用 CachedAttention 为多轮对话提供高性价比大语言模型服务。载于 2024 年 USENIX 年度技术会议（USENIX ATC 24）。USENIX 协会，第 111–126 页。https://www.usenix.org/conference/atc24/presentation/gao-bin-cost [14] Cheng Gu、Reese Levine、Zhenkai Zhang、Tyler Sorensen 和 Yanan Guo。2026年。牢笼之后：利用内存屏障对 NVIDIA MIG 缓存分区实施的侧信道攻击。载于第 35 届 USENIX 安全研讨会（USENIX Security 26）。USENIX 协会，第 5373–5392 页。 https://www.usenix.org/conference/usenixse curity26/presentation/gu-cheng [15] Zhongshu Gu、Enriquillo Valdez、Salman Ahmed、Julian James Stephen、Michael Le、Hani Jamjoom、Shixuan Zhao 和 Zhiqiang Lin。2026年。蓝图、引导与桥梁：NVIDIA GPU 机密计算的安全审视。 载于《机器学习与系统》第8卷。MLSys，1546–1568。 https://proceedings.mlsys.org/paper _ 2026 906419 502 files/paper/ / 575 617 489 1 696 67 hash/ cd b cc a a a -Abstract-Conference.html [16] Husheng Han、Xinyao Zheng、Yuanbo Wen、Yifan Hao、Erhu Feng、Ling Liang、Jianan Mu、Xiaqing Li、Tianyun Ma、Pengwei Jin、Xinkai Song、Zidong Du、Qi Guo 和 Xing Hu。2024。TensorTEE：统一异构 TEE 粒度，实现高效安全的协同张量计算。载于第29届 ACM 编程语言与操作系统体系结构支持国际会议论文集，第4卷（ASPLOS）。ACM，282–297。doi:10.1145/3622781. 3674168 [17] Tristan Hornetz、Hosein Yavarzadeh、Albert Cheu、Adria Gascon、Lukas Gerlach、Daniel Moghimi、Phillipp Schoppmann、Michael Schwarz 和 Ruiyi Zhang。2026。TDXRay：针对真实工作负载的 Intel TDX 微架构侧信道分析。载于 IEEE 安全与隐私研讨会（S&P）。https: // tdxray.cpusec.org/assets/tdxray_sp26.pdf [18] 英特尔公司。2025。Intel TDX Connect 架构规范。https: //cdrdv2-public.intel.com/862706/Intel%20TDX%20Connect%20EAS%20354629 %20004.pdf [19] 英特尔公司。2026。多密钥全内存加密（MKTME）侧信道对 Intel Trust Domain Extensions（Intel TDX）的影响。软件安全指南 768686，版本 1.1。英特尔公司。https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/mktme-side-channel-impact-on-intel-tdx.html [20] Tetsu Iwata、Keisuke Ohashi 和 Kazuhiko Minematsu。2012。破解并修复 GCM 安全性证明。载于《密码学进展——CRYPTO 2012》（《计算机科学讲义》，第7417卷）。Springer，31–49。https://iacr.org/a rchive/crypto2012/74170032/74170032.pdf [21] Hodong Kim、Gyeongsup Lim、Seunghee Shin、Youngjoo Shin 和 Junbeom Hur。2026。TIMESLICE-SANDWICH：利用分时调度的 GPU 侧信道攻击。载于第35届 USENIX 安全研讨会（USENIX Security 26）。USENIX 协会，5295–5314。 https://www.usenix.org/conference/usen ixsecurity26/presentation/kim-hodong [22] Tomáš Kočiský、Jonathan Schwarz、Phil Blunsom、Chris Dyer、Karl Moritz Hermann、Gábor Melis 和 Edward Grefenstette。2018。NarrativeQA 阅读理解挑战。计算语言学协会汇刊 6（2018），317–328。doi:10.1162/tacl_a_00023 [23] Jonghyun Lee、Taewoon Kang、Pratheek B、Yongqin Wang、Rachit Rajat、Mengyuan Li、Gunjae Koo、Daniel Wong 和 Murali Annavaram。2026。FL ASH：故障引导、经认证的安全混合密钥检查点，面向机密 GPU。载于2026年 ACM SIGOPS 年度技术会议论文集。计算机协会，纽约，NY，美国，876–891。 doi:10.1145/3828523. 3857076 [24] Jiaqi Li、Mengmeng Wang、Zilong Zheng 和 Muhan Zhang。2024。LooGLE：长上下文语言模型能否理解长上下文？ 载于第62届计算语言学协会年会论文集（第1卷：长篇论文）。计算语言学协会，泰国曼谷， 16304 16333 10 18653 1 2024 859 – 。doi: . /v / .acl-long。Meta。 . [25] 2024 3 1 8 Llama . B Instruct 模型卡。Hugging Face模型仓库。 https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct [26] Ravan Nazaraliyev、Yicheng Zhang、Sankha Baran Dutta、Andres Marquez、Kevin Barker 和 Nael Abu-Ghazaleh。2025。不那么清爽：利用 RFM Rowhammer 缓解机制攻击 GPU。载于第34届 USENIX 安全研讨会（USENIX Security 25）。USENIX 协会，西雅图，WA，5641–5660。https://www.usenix . org/conference/usenixsecurity25/presentation/nazaraliyev [27] NVIDIA。2025。英伟达 Blackwell 和 Hopper GPU 安全 AI。白皮书 WP-12554-001，v1.3。NVIDIA。 https://docs.nvidia.com/nvidia-secure-ai-with-blackwell-and-hopper-gpus-whitepaper.pdf [28] NVIDIA。2026。英伟达 Linux 开放 GPU 内核模块源代码。源代码，发行版 610 6 10 57 04 57 04 . . . https://github.com/NVIDIA/open-gpu-kernel-modules/tree/ . [29] NVIDIA。2026。英伟达开放 GPU 内核模块：UVM 通道提交。源代码，发行版 610.57.04。 https://github.com/NVIDIA/open-gpu-kernel-modules/blob/610.57.04/kernel-open/nvidia-uvm/uvm_channel.c [30] NVIDIA。2026。英伟达可信计算解决方案 R595 GA 发行说明。RN-12817-001，版本 02。 https://docs.nvidia.com/595trd1-trusted-computing- solutions-release-notes.pdf [31] NVIDIA。无日期。KV 感知路由。英伟达 Dynamo 文档。https: // docs.nvidia.com/dynamo/knowledge-base/concepts/system-architecture/kv-aware-routing 版本 1.6.1。访问日期：2026年11月7日。[32] NVIDIA。无日期。信任与威胁模型。英伟达机密计算文档，版本 1.0.0。https://docs.nvidia.com/conf idential- computing/virtual-machines/v1.0.0/threat-model 访问日期：2026年11月16日。[33] NVIDIA。无日期。使用 HiCache。英伟达 Dynamo 文档。https://docs.nvidia.com/dynamo/latest/knowledge-base/modular-components/backends/sg-lang/hi-cache 版本 1.6.1。访问日期：2026年11月7日。[34] Eunseong Park、Timo Thans、Vishnu Kumar Kalidasan、Qinghao Hu 和 Wenjie Xiong。2026。机密 GPU 上大语言模型服务的实证研究。载于2026年

IEEE 系统与软件性能分析国际研讨会（ISPASS）。IEEE, 311 322 10 1109 69572 2026 00038 – 。doi: . /ISPASS . . Adam [35] Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga、Alban Desmaison、Andreas Kopf、Edward Yang、Zachary DeVito、Martin Raison、Alykhan Tejani、Sasank Chilamkurthy、Benoit Steiner、Lu Fang、Junjie Bai 和 Soumith Chintala。2019。PyTorch：一种命令式风格的高性能深度学习库。载于《神经信息处理系统进展》，H. Wallach、H. Larochelle、A. Beygelzimer、F. d'Alché Buc、E. Fox 和 R. Garnett（编），第32卷。Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/pap er/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf [36] PCI-SIG。2022。完整性与数据加密（IDE）修订版 A。PCI Express 工程变更通知。https://pcisig.com/PCI% 20Express/ECN/Base/Inte grityandDataEncryption_A [37] PCI-SIG。2022。TEE 设备接口安全协议（TDISP）。PCI Express 工程变更通知。https://pcisig.com/PCI% 20Express/ECN/Base/TE EDeviceInterfaceSecurityProtocol [38] Jianliang Qi。2026。 [Perf] HiCache：在加载器线程上执行主机到设备回载入队（选择启用）。SGLang GitHub 拉取请求 #38472。https: //github.com/sgl-project/sglang/pull/38472[39] Ruoyu Qin、Zheming Li、Weiran He、Jialei Cui、Feng Ren、Mingxing Zhang、Yongwei Wu、Weimin Zheng 和 Xinran Xu。2025。Mooncake：以更多存储换取更少计算——面向大语言模型聊天机器人的 KVCache 中心架构。载于第23届 USENIX 文件与存储技术会议（ FAST 25）。USENIX 协会，155–170。 https://www.usenix.org/conference/fast25/prese ntation/qin [40] Qwen 团队。2026。 Qwen3.8-27B 模型卡。Hugging Face模型仓库。 https://huggingface.co/Qwen/Qwen3.8-27B [41] SGLang 团队。2026。HiCache 多轮基准测试。SGLang GitHub 仓库。 https://github.com/sgl-project/sglang/blob/main/benchmark/hicache/bench _ multiturn.py 访问日期：2026年11月。[42] Linke Song、Zixuan Pang、Wenhao Wang、Zihao Wang、XiaoFeng Wang、Hongbo Chen、Wei Song、Yier Jin、Dan Meng 和 Rui Hou。2024。早起的鸟儿捉到泄露：揭示大语言模型服务系统中的定时侧信道。arXiv 预印本 arXiv: 2409 20002 10 48550 2409 20002 . . doi: . /arXiv. . [43] 2025 Yifan Tan、Cheng Tan、Zeyu Mi 和 Haibo Chen。 . PipeLLM：通过推测式流水线加密提供快速且机密的大语言模型服务。载于第30届 ACM 编程语言与操作系统体系结构支持国际会议论文集，第1卷。计算机协会， 843 857 10 1145 3669940 3707224 – 。doi: . / . Yongqin [44] Wang、Rachit Rajat、Jonghyun Lee、Tingting Tang 和 Murali Annavaram。2024。 Fastrack：利用 GPU 可信执行环境实现安全机器学习的快速 I/O。arXiv 预印本 arXiv:2410.15240。doi:10.48550/arXiv.2410.15240[45] Luca Wilke、Jan Wichelmann、Anja Rabich 和 Thomas Eisenbarth。2023。SEV-Step：面向 AMD-SEV 的单步执行框架。 IACR密码硬件与嵌入式系统汇刊 2024，1（2023年12月），180–206。 doi:10.46586/tches.v2024.i1.180-206[46] Guanlong Wu、Zheng Zhang、Yao Zhang、 Weili Wang、Jianyu Niu、Ye Wu 和 Yinqian Zhang。2025。我知道你问了什么：多租户大语言模型服务中通过 KV 缓存共享导致的提示词泄露。载于网络与分布式系统安全（ NDSS）研讨会。互联网协会。doi: 10 14722 2025 241772 . /ndss. . Yuxing Xiang、 [47] Xue Li、Kun Qian、Yufan Yang、Diwen Zhu、Wenyuan Yu、Ennan Zhai、Xuanzhe Liu、Xin Jin 和 Jingren Zhou。2025。Aegaeon：面向市场上并发大语言模型服务的高效 GPU池化。载于 ACM SIGOPS 第31届操作系统原理研讨会论文集。计算机协会， 1030 1045 10 1145 3731569 3764815 – 。doi: . / . Zhiqiang Xie。 [48] 2025 . SGLang HiCache：利用您喜爱的存储后端实现快速分层 KV 缓存。LMSYS 组织博客。 https://www.lmsys.org/blog/2025- 09-10-sglang-hicache/[49] Zhiqiang Xie、Ziyi Xu、 Mark Zhao、Yuwei An、Vikram Sharma Mailthody、Scott Mahlke、Michael Garland 和 Christos Kozyrakis。2026。Strata：面向长上下文语言模型服务的分层上下文缓存。载于第20届 USENIX 操作系统设计与实现研讨会（OSDI 26）。USENIX 协会，西雅图，WA，1–16。https://www.usenix.org/conference/osdi26/pre sentation/xie-zhiqiang [50] Yang Yang 和 Adwait Jog。2026。LÆGIS：定位并解决基于 GPU 的机密计算的性能开销。载于2026年 ACM/IEEE 第53届年度国际计算机体系结构研讨会（ISCA）。IEEE，465–481。doi:10.1109/ISCA66397.2026.00046[51] Yang Yang、 Mohammad Sonji 和 Adwait Jog。2025。剖析基于 GPU 的系统中机密计算的性能开销。载于2025年 IEEE 系统与软件性能分析国际研讨会（ISPASS）。IEEE， 1 16 10 1109 64960 2025 00020 – 。doi: . /ISPASS . . Hang Yin 和 Kevin [52] Wang。 . 序列化桥接：理解并恢复 Blackwell GPU 机密计算下的大语言模型服务性能。 arXiv 预印本 arXiv: 2606 23969 10 48550 2606 23969 . . doi: . /arXiv. . Joanna Young 和 Sam Lugani。 . 机密加速器如何提升人工智能工作负载安 [53] 2026 全。Google Cloud博客。https://cloud.google.com/blog/prod ucts/identity-security/how-confidential-accelerators-can-boost-ai-workload-security [54] Peterson Yuhala、Christian Göttel、Jämes Ménétrey、Valerio Schiavoni、 David Kozhaya 和 Pascal Felber。2025年。《Intel SGX 和 TDX 中的实时保证》。载于 第*37*届*Euromicro*实时系统会议（*ECRTS 2025*）（莱布尼茨国际信息学会议录（ *LIPIcs*），第*335*卷）。Dagstuhl城堡—莱布尼茨信息学中心，德国 Dagstuhl，8:1– 8:25。 doi:10.4230/LIPI cs.ECRTS.2025.8[55] Z.ai。2025年。GLM-4.5-Air模型卡。 Hugging Face模型仓库。 https: //huggingface.co/zai-org/GLM-4.5-Air[56] Shixuan Zhao、Zhongshu Gu、Salman Ahmed、Enriquillo Valdez、Hani Jamjoom 和 Zhiqiang Lin。2025年。《GPU Travelling：利用TEE支持的GPU进行高效机密协作训练》。载于 *2025*年*ACM SIGSAC*计算机与通信安全会议论文集。计算机协会，纽约，美国纽约州，2653–2667。 doi:10.1145/3719027.3765029[57] Lianmin Zheng、 Liangsheng Yin、Zhiqiang Xie、Chuyue Sun、Jeff Huang、Cody Hao Yu、Shiyi Cao、Christos Kozyrakis、Ion Stoica、Joseph E. Gonzalez、Clark Barrett 和 Ying Sheng。2024年。《SGLang：结构化语言模型程序的高效执行》。载于神经信息处理系统进展，第37卷。Curran Associates, Inc.，62557–62583。 doi:10.52202/079017-2000