ccAI: A Compatible and Confidential System for AI Computing

Chenxu Wang*†‡
Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology
Shenzhen, Guangdong, China
Department of Computer Science and Engineering, Southern University of Science and Technology
Shenzhen, Guangdong, China
Department of Computing, The Hong Kong Polytechnic University
Kowloon, Hong Kong, China
12150073@mail.sustech.edu.cn

Danqing Tang*
Ant Group
Hangzhou, Zhejiang, China
tangdanqing.tdq@antgroup.com

Changxu Ci*
Ant Group
Hangzhou, Zhejiang, China
cichangxu.ccx@antgroup.com

Junjie Huang
Department of Computer Science and Engineering, Southern University of Science and Technology
Shenzhen, Guangdong, China
12431254@mail.sustech.edu.cn

Yankai Xu
Department of Computer Science and Engineering, Southern University of Science and Technology
Shenzhen, Guangdong, China
12432712@mail.sustech.edu.cn

Fengwei Zhang§¶
Department of Computer Science and Engineering, Southern University of Science and Technology
Shenzhen, Guangdong, China
zhangfw@sustech.edu.cn

Jiannong Cao
Department of Computing, The Hong Kong Polytechnic University
Kowloon, Hong Kong, China
csjcao@comp.polyu.edu.hk

Jie Song
Ant Group
Hangzhou, Zhejiang, China
charlie.sj@antgroup.com

Shoumeng Yan§
Ant Group
Hangzhou, Zhejiang, China
shoumeng.ysm@antgroup.com

Tao Wei
Ant Group
Hangzhou, Zhejiang, China
lenx.wei@antgroup.com

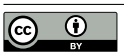
Zhengyu He
Ant Group
Hangzhou, Zhejiang, China
zhengyu.he@antgroup.com

Abstract

Confidential xPU computing has emerged as a prominent technique for effectively securing users' AI computing workloads on heterogeneous systems equipped with xPUs. Although the industry adopts this technology in cutting-edge hardware (e.g. NVIDIA H100 GPU) to safeguard high-performance AI computing, most clouds still rely on legacy xPUs and suffer from data leakage problems. Moreover, although the academy proposes several confidential xPU designs, these solutions have yet to be widely deployed in heterogeneous clouds. A key limitation is the compatibility challenge, which requires non-trivial engineering effort to address. Therefore, there is an urgent need to design a compatible and confidential xPU protection system that is tailored for today's AI computing platforms.

To address these issues, we present ccAI, a heterogeneous system that ensures both high compatibility and strong security for AI computing based on xPU. By leveraging the PCIe interconnect as a common abstraction layer, ccAI provides a unified solution for securing diverse xPU types while preserving user transparency, effectively overcoming key compatibility limitations of existing approaches. Specifically, ccAI achieves this through two core components: (1) a dedicated hardware module, the PCIe Security Controller (PCIe-SC), which enforces computing security of diverse types of xPU and (2) a collaborative software component, the Adaptor, deployed within the Trusted VM (TVM), which orchestrates secure workload processing without modifying applications or

*Chenxu Wang, Danqing Tang and Changxu Ci are co-first authors.
†Also with Department of Computer Science and Engineering, Southern University of Science and Technology, China.
‡Also with Department of Computing, The Hong Kong Polytechnic University, China.
§Fengwei Zhang and Shoumeng Yan are corresponding authors.
¶Also with Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, China.



MICRO '25, Seoul, Republic of Korea

ACM ISBN 979-8-4007-1573-0/25/10
https://doi.org/10.1145/3725843.3756104

drivers. To handle the complexity of PCIe packet transmissions, ccAI incorporates a flexible and fine-grained processing framework, within which ccAI additionally optimizes frequent I/O interactions and security-critical operations to minimize performance overhead. We implement a prototype of ccAI and evaluate it across multiple real-world xPU platforms using a range of Large Language Models (LLMs). Results show that ccAI effectively protects xPU computing with low (0.05% – 5.67%) performance overhead

Keywords

AI computing, confidential xPU computing, PCIe

ACM Reference Format:

Chenxu Wang, Danqing Tang, Changxu Ci, Junjie Huang, Yankai Xu, Fengwei Zhang, Jiannong Cao, Jie Song, Shoumeng Yan, Tao Wei, and Zhengyu He. 2025. ccAI: A Compatible and Confidential System for AI Computing. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25), October 18–22, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3725843.3756104

1 Introduction

Artificial Intelligence (AI) computing has become increasingly prevalent, profoundly transforming a wide range of data-driven applications. These include personalized video processing [62], healthcare diagnostics [74], and Large Language Model (LLM) inference [21, 60, 61]. As AI workloads grow in scale and complexity, developers are increasingly turning to heterogeneous cloud platforms such as Google Cloud [27], Microsoft Azure [46], and Alibaba Cloud (Aliyun) [4]) to meet performance demands. Unlike traditional cloud infrastructures, heterogeneous clouds leverage high-performance accelerators, known as xPUs, to accelerate AI tasks. These accelerators include Graphics Processing Units (GPUs) [5, 8, 52], Neural Processing Units (NPUs) [10], and Field Programmable Gate Arrays (FPGAs) [7] accelerators. With their computational power, xPUs have become indispensable for modern AI systems.

Despite their performance advantages, xPUs introduce significant security problems. Vulnerabilities in xPU software stacks [16–20] and insecure hardware designs [48] allow adversaries to extract sensitive data, such as input datasets and intermediate results, or even leak proprietary AI models during execution [73]. To address this threat, confidential xPU computing has emerged as a promising defense mechanism, supported by extensive research over the past decade [39, 40, 44, 75, 84, 85]. Recently, NVIDIA commercialized this concept with the release of the H100 GPU [50], the first GPU to offer built-in confidentiality support. The H100 delivers strong data protection for high-performance AI workloads while maintaining performance comparable to legacy xPUs. This has prompted adoption by major cloud providers such as Google [28] and Microsoft [47]. As a result, confidential xPU computing will become a standard feature in future heterogeneous clouds.

However, the H100 GPU alone cannot fully address the data leakage problem in today's heterogeneous clouds. Due to cost and deployment constraints, most cloud providers continue to rely on general xPUs, which lack native confidentiality support. Although studies has proposed confidential computing solutions for these devices (summarized in Figure 1), they have not achieved widespread deployment in real-world AI platforms due to two compatibility limitations: First, many studies lack support for multiple xPU types. Hardware-based approaches [50, 83] (Figure 1c) and privileged software or Trusted Execution Environment (TEE)-based methods [39, 44, 85] (Figures 1a–b) are often tightly coupled to specific xPU architectures, workflows, or hardware features. TDISP-based solutions [6, 9, 37] (Figure 1e) require advanced PCIe capabilities, such as IDE support [66], and compliance from both the platform and the xPU. As a result, they are incompatible with older or non-compliant hardware. Second, many designs fail to ensure user transparency. Isolated platform architectures [93] (Figure 1d), for example, require non-trivial modifications to user applications, such as invoking custom APIs for secure data transfer or task submission. In addition, most studies (Figures 1a/b/c/e) require changes to the xPU software stack, including drivers and runtime libraries, within the Trusted VM (TVM). These modifications increase migration effort and reduce usability.

These limitations highlight the urgent need for a solution that provides strong security while maintaining compatibility and transparency in heterogeneous clouds. In this paper, we present ccAI (Figure 1f), a novel system designed to deliver robust security without compromising compatibility or ease of use. ccAI aims to achieve three goals.

- (**G1**) High compatibility: ccAI must support a wide range of xPU types and integrates with existing xPU software.
- (**G2**) Strong data security: ccAI must end-to-end confidentiality, integrity, and isolation throughout the computation process.
- (**G3**) Low performance overhead: ccAI must introduce low latency to AI workload execution.

The primary design challenge (**C1**) is overcoming the compatibility problems of existing studies. To address this, ccAI adopts a PCIe-based architecture that operates at the PCIe packet level of DMA and MMIO transactions, a common interface across all xPU types. Instead of relying on device-specific protection, ccAI offloads security enforcement to a dedicated hardware module called PCIe Security Controller (PCIe-SC), which sits between the xPU and the PCIe bus. The PCIe-SC monitors and secures all PCIe packet exchanges between the TVM and the xPU, providing consistent protection independent of the xPU type. To preserve user transparency, ccAI avoids any modifications to xPU applications or xPU software stacks (e.g., drivers or user-layer libraries). Instead, it introduces a lightweight software component, the Adaptor, deployed within the TVM. The Adaptor collaborates with the PCIe-SC to transparently manage security-critical operations, including data encryption and metadata handling, for xPU workloads. We detail ccAI's system design in §3.

Implementing ccAI introduces additional challenges. One challenge (**C2**) is secure and flexible management of PCIe packets. Unlike prior works that rely on coarse-grained access control, such as securing MMIO or DMA channels, ccAI operates at the PCIe packet level, where each packet has complex and heterogeneous attributes including format, type, and address space. Moreover, many studies [39, 40, 85] either ignore PCIe bus attacks or employ inflexible defenses such as full-link encryption. To address this, ccAI introduces a Packet Filter that systematically classifies PCIe packets based on their attributes and blocks unauthorized requests. For

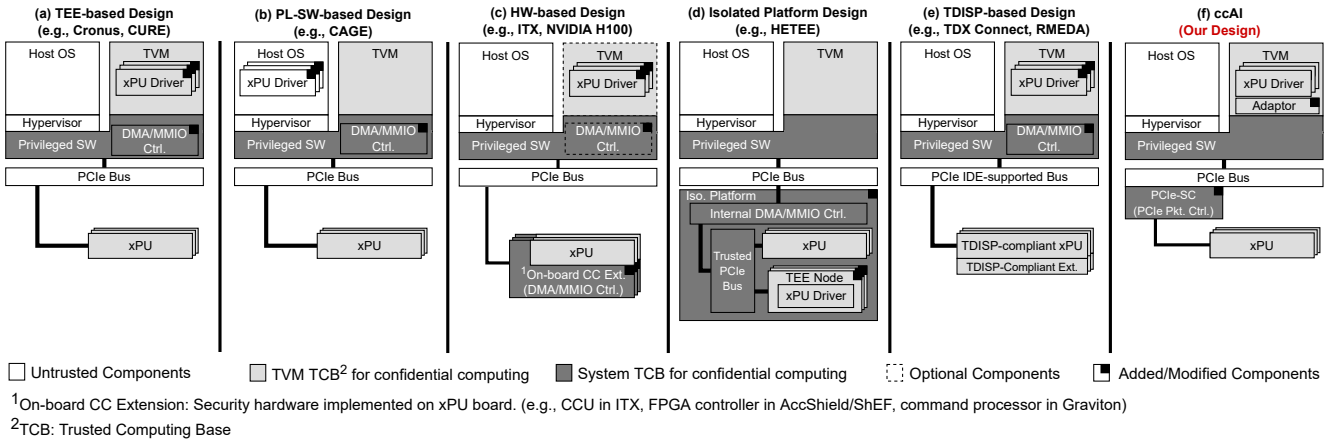


Figure 1: Architecture overview of state-of-the-art designs and ccAI.

authorized packets, the filter enforces fine-grained security policies, from strict confidentiality and integrity guarantees to transparent pass-through, based on packet attributes. ccAI also employs a set of Packet Handlers to perform cryptographic operations such as encryption and integrity verification. We elaborate on this packet-level protection in §4. Another challenge (**C3**) is the performance overhead introduced by security operations. Frequent I/O interactions between the TVM and PCIe-SC, especially during encryption and policy synchronization, can lead to significant latency. To mitigate this, we reduce redundant I/O read and write operations during computing and propose several optimizations to security operation. We describe these techniques in §5.

We implement a prototype of ccAI using an Intel server [34] to host TVM-side components such as the Adaptor, and an Intel Agilex 7 SoC FPGA [33] to realize the PCIe-SC as a proof of concept. The system runs on a general-purpose TVM and supports five distinct xPUs: NVIDIA A100 GPU [56], RTX4090Ti GPU [55], T4 GPU [59], Tenstorrent N150d NPU [78], and Enflame S60 GPU [24]. This demonstrates high compatibility across different vendors and architectures. We compare ccAI's compatibility guarantees with those of state-of-the-art approaches and conduct a comprehensive security analysis, including an evaluation of the Trusted Computing Base (TCB). Furthermore, we evaluate the prototype using a suite of LLMs, including OPT [90], Llama-2 [79], Babel [92], and Deepseek [87], across multiple xPUs. Experimental results show that ccAI achieves strong security and high compatibility with low (0.05% – 5.67%) performance overhead.

We highlight the following key contributions:

- We propose ccAI, a compatible and confidential system for xPU-based AI computing in heterogeneous clouds. By securing the PCIe communication channels between TVMs and multi-type xPUs, ccAI ensures confidentiality, integrity, and isolation throughout xPU computation.
- We implement a proof-of-concept prototype of ccAI and integrate it with real-world xPU devices. The prototype seamlessly supports diverse xPUs types and transparently runs existing software stacks.
- We evaluate ccAI using a range of LLM models across multiple xPUs. Results show that ccAI introduces low performance overhead and is practical for real-world deployment.

2 Background and Motivation

2.1 xPU, PCIe and Packets

xPUs, including Graphic Processing Units (GPUs) [52], Neural Processing Units (NPUs) [10], and Field Programmable Gate Array (FPGA) accelerators [7], are critical to today's AI computing. In heterogeneous clouds, xPUs interact with host CPUs through two top-layer interfaces: (1) Direct Memory Access (DMA) for data/code exchange and (2) Memory-Mapped I/O (MMIO) for transferring commands or accessing register status. However, the implementation of DMA and MMIO varies across xPU types due to the hardware heterogeneity. For instance, commercial GPUs [5, 52] usually equip an on-board Memory Management Unit (MMU), while TPUs [26] lack this component. Besides hardware variance, the software stacks for different xPUs (e.g., NVIDIA GPU driver [58] and Xilinx FPGA driver [88]) are largely varied. This makes the control of DMA/MMIO vary in different xPU-equipped system. Together, these hardware and software differences make it difficult to design a one-size-fits-all protection mechanism that works with multiple xPU types and their software stacks.

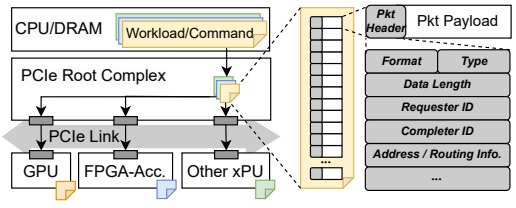


Figure 2: Overview of PCIe Fabric and PCIe packet.

Nevertheless, xPUs in heterogeneous clouds rely on a common and low-level channel for DMA and MMIO operations: The PCIe. As illustrated in Figure 2, the PCIe root complex routes workloads and commands between the host side (e.g., CPU and main memory) and PCIe-attached devices such as xPUs. Since the PCIe interface is universally adopted in xPU computing, it provides a consistent

foundation for ccAI to design a compatible protection mechanism that supports diverse xPU types (**G1**).

To implement DMA and MMIO operations, the PCIe interface transmits data, code, and commands using PCIe packets, the fundamental units of PCIe communication. As shown in Figure 2, each DMA or MMIO transaction consists of multiple PCIe packets, all routed through the same PCIe Root Complex to reach the target xPU. To control data flow, each PCIe packet includes a header containing critical attributes such as packet format, packet type, requester and completer IDs, accessed address space, and routing information. By inspecting these header attributes, ccAI can differentiate between authorized and malicious requests (**G2**). However, the complexity of PCIe header attributes, including numerous valid field combinations, and the diversity of packet sources, such as the TVM and untrusted guest software, pose a significant challenge in designing an efficient and secure packet filtering mechanism.

2.2 Threat Model

We assume a powerful adversary who aims to leak or tamper with sensitive data (i.e., inputs, intermediate data, and execution results) and code of confidential xPU tasks. On the CPU side, the adversary controls the privileged software stack, including the privileged OS, hypervisor, and peripheral drivers. In this case, the adversary attempts to access or tamper with the xPU applications, software stacks, and the Adaptor in ccAI. On the xPU side, we follow the state-of-the-art [50, 68, 93] and assume that the adversary can attack the PCIe bus (e.g., via the snooping attack [72]) to access or tamper with PCIe transmission. In addition, the adversary may attempt to tamper with xPUs, ccAI's PCIe-SC, and the connection between these components.

We trust the TVM (e.g., Intel TDX [36]) to protect the xPU applications and software stacks from the privileged software. The CPU-side TVM is protected by privileged software (e.g., TDX module) and CPU-side security primitives. Same as most state-of-the-arts, we do not consider the side-channel and denial-of-service attacks, which orthogonal works [2, 41, 86] can address. Finally, we assume that the firmware of the xPU is free of malicious code and its integrity is protected. This indicates that ccAI trusts the authenticity of the hardware vendor.

3 ccAI Design

The principle of ccAI is to achieve a compatible and secure system for xPU-based AI computing in heterogeneous clouds. Guided by this principle, we aim to achieve three critical goals:

- **Compatibility:** ccAI must operate in heterogeneous clouds that support general-purpose TVMs and general xPU devices. This requires ccAI to be independent of specific CPU-side security features, particular xPU architecture, and hardware modifications to either the CPU or xPU devices.
- **Security:** ccAI must ensure confidentiality, integrity, and isolation in the xPU computing environment. Moreover, ccAI must protect sensitive xPU workloads, including data and model, throughout their entire lifecycle: When stored in the TVM, transmitted over the PCIe bus, and processed on the xPU.
- **Performance:** ccAI must introduce low performance overhead during computing. To meet the stringent efficiency requirements of heterogeneous clouds, ccAI's execution workflow must be carefully optimized.

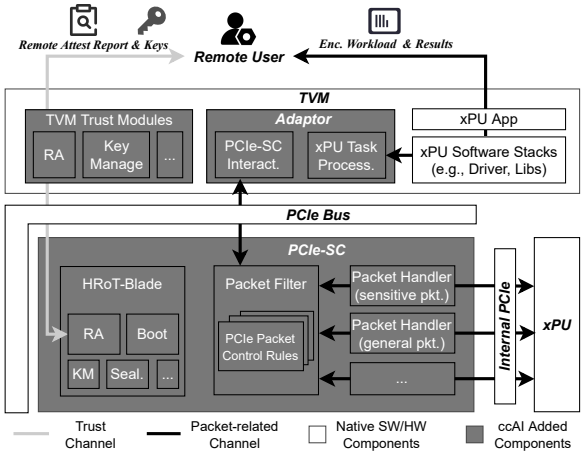


Figure 3: Architecture overview of ccAI.

Figure 3 illustrates the architecture of ccAI. Designed for clouds equipped with general-purpose TVM, ccAI requires no modifications to xPU software or hardware, nor does it depend on specialized CPU security primitives, thereby achieving high compatibility (**G1**). To ensure strong data security (**G2**), ccAI integrates a standard TVM with a dedicated PCIe-SC, which collectively protect xPU workloads against powerful adversaries. The TVM is responsible for safeguarding xPU applications and xPU software stacks, including xPU drivers and user-layer libraries, with isolation mechanisms. Meanwhile, the PCIe-SC mediates all access to the xPU and performs critical security operations, such as data de/encryption and integrity verification. Additionally, ccAI includes trust establishment components — such as an HRoT-Blade (a hardware root-of-trust module) and TVM-side trust modules — to securely initialize the system and support remote attestation. These components are detailed in §6. By optimizing the workflow and security operations, ccAI enables LLMs to run on high-performance xPUs with minimal performance overhead, meeting the efficiency demands of heterogeneous clouds (**G3**).

C1. However, the primary challenge of ccAI is to address the compatibility problems in previous studies, especially in supporting multi-type xPUs and ensuring user transparency. On the one hand, different xPUs are usually implemented with different hardware architectures and support their own software stacks. On the other hand, most xPU software stacks do not provide security support to workloads.

Solution to C1. We propose two major solutions to address the aforementioned problems. First, to support multi-type xPUs, we anchor our protection mechanism on the PCIe channel, which is the common connection between TVMs and multi-type xPUs. Specifically, our protection focuses on managing the basic PCIe transmission unit — PCIe packet. This is because the PCIe packet is commonly used in various types of xPUs, carrying the data/code and command payloads for DMA/MMIO interaction with the TVM

(mentioned in §2.1). Second, to ensure user transparency, we do not directly modify the xPU applications or drivers. Instead, we design a TVM-side Adaptor to process the xPU workloads (e.g., encrypting the data) and submit them to PCIe-SC. Note that the Adaptor supports software-based updates (e.g., kernel patch) to mitigate the effort to support new xPUs. Overall, ccAI effectively addresses compatibility issues while ensuring data security.

PCIe-SC. By managing packets in PCIe transmission, the PCIe-SC controls and processes the DMA/MMIO interaction with xPU. As shown in Figure 3, the PCIe-SC consists of three major components: First, the Packet Filter. To support multi-type xPUs, the Packet Filter intercepts all PCIe packets coming from or sent to the xPU to analyze their metadata (e.g., the xPU's Bus/Device/Function, packet type and address space) for filtering and processing. Specifically, the Packet Filter stores a set of access control rules designed for the DMA/MMIO security requirements (e.g., the authorized sender and sensitive address space) of the target xPU. Second, a set of Packet Handlers. These handlers are designed to securely process (e.g., perform de/encryption and check PCIe packet integrity) authorized packets. They include sensitive PCIe packets (e.g., data and code for AI model) and general packets like interrupts. Third, ccAI also designs an HRoT-Blade for trust establishment processes, such as remote attestation and exchange keys. Overall, the PCIe-SC supports and protects multi-type xPUs by managing PCIe packet interactions. We further detail this process in §4.

TVM-side Adaptor. As shown in Figure 3, the TVM-side Adaptor is delegated to achieve two major functions: (1) Interacting with PCIe-SC and (2) processing xPU tasks. For the interaction with PCIe-SC, the Adaptor sends request packet via MMIO-based operations. For processing the xPU tasks, this process must ensure user privacy — introducing no changes to xPU application and driver. To achieve this, the Adaptor first sends packets to query the essential metadata (e.g., address position and size) of the processed task. Next, based on the metadata from PCIe-SC, the Adaptor locates the sensitive data/code, performs encryption, and transmits the encrypted task via a bounce buffer. Moreover, the TVM-side Adaptor supports compatible system updates for supporting new xPU devices. Specifically, ccAI updates the Adaptor with software-based kernel patches. With secure boot guarantees, the updated patch is directly activated on the TVM, providing interaction support (e.g., memory allocation for confidential workloads) of the new xPU software.

ccAI deployment. Lastly, we detail the deployment process of ccAI. We deploy ccAI on a TVM-supported cloud where the user requests a TVM, provides an xPU computing workload, and receives the execution results. To deploy ccAI, each TVM installs the Adaptor, trust modules, and the native xPU software stacks (e.g., xPU driver and user-layer libraries). Meanwhile, the PCIe-SC is equipped on the server's PCIe port and connects xPU devices with an internal PCIe bus. In the secure boot process, ccAI verifies the TVM and its PCIe-SC to ensure the integrity of the confidential xPU computing environment. Next, ccAI provides a set of trust modules to attest the local TVM, xPU devices, and PCIe-SC and generates a remote attestation report for the user's attestation. To ensure workload confidentiality, the user follows ccAI's trust establishment process to exchange keys with PCIe-SC and its TVM. This trust establishment process helps to encrypt the xPU workloads with signatures and securely transmit the xPU workloads in the network. Once the TVM receives xPU workloads, it decrypts the workload and leverages ccAI's workflow to perform confidential xPU computing. After computing, the TVM receives the encrypted execution results from PCIe-SC, decrypts them and wait the remote user to access.

Table 1: Categorization of PCIe packet access control. The *A1 – A4* denote four security actions for processing a packet.

Packet Access Permission	Actions
Prohibited	(*A1*) Disallow
Write-Read Protected	(*A2*) Integrity Check (Crypt.) + En/Decryption
Write Protected	(*A3*) Integrity Check (Plain) + Security Verify
Full Accessible	(*A4*) Transparent Transmission

4 Security Design

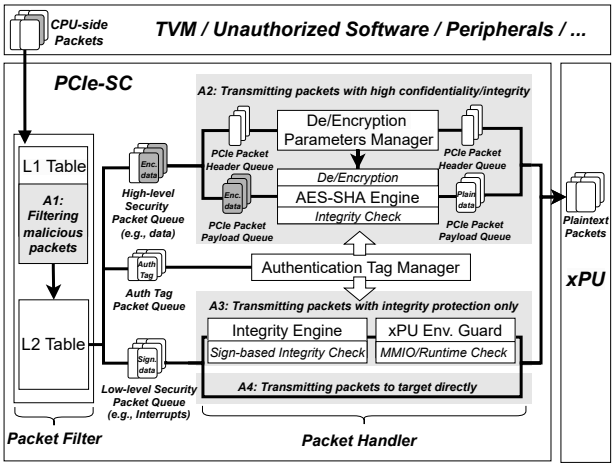


Figure 4: Overview of ccAI security design.

As aforementioned in §3, ccAI must ensure the security of xPU computing (**G2**). Since TVM builds isolated security regions for xPU workloads on the CPU side, we focus on protecting the PCIe channel between xPU and TVM. Figure 4 shows the workflow of our PCIe protection. During xPU computing, ccAI receives PCIe packets from varied software/hardware components. To filter and process these packets, we design two major components: (1) A Packet Filter, which blocks malicious packets and classifies authorized packets for further security actions, and (2) a set of Packet Handlers, which execute security operations (e.g., de/encryption, security checks, and integrity verification) on packets. After processing, the PCIe-SC transmits packets with plaintext data to the xPU and finally handles the execution results using the same process.

C2. However, filtering and handling PCIe packets in confidential xPU computing can be challenging: Packets carry diverse attributes and expected values (e.g., the address and ID of the requester and destination). Even packets of the same type may require different security actions due to their attribute values. Thus, a one-size-fits-all mechanism is insufficient to filter and manage PCIe packets for xPUs.

Solution to C2. To address this challenge, we systematically analyze PCIe packets in confidential xPU computing and design a

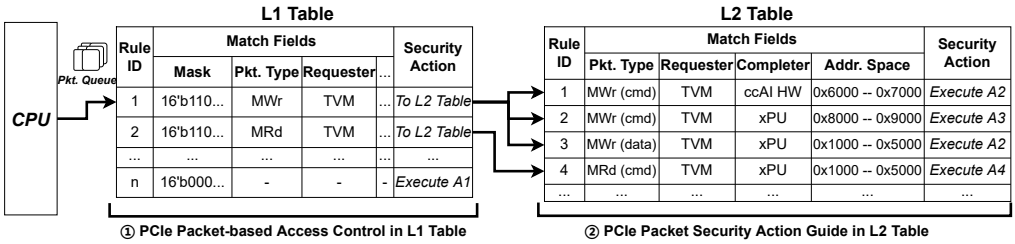


Figure 5: Workflow of Packet Filter.

new packet security categorization — we classify four categories of packet access permissions and corresponding security actions (see Table 1). The Packet Filter is required to recognize *prohibited* packets from unauthorized software/hardware and filter malicious packets (*A1*). For the authorized packets, we further categorize them into three additional packet access permissions: (1) *Write-Read Protection*, (2) *Write Protection*, and (3) *Full Accessible*, as a guide to the corresponding security actions. Specifically, the *Write-Read Protection* access control serves the packets with sensitive data (e.g., user data, model parameters, and execution results). These packets require careful confidentiality and integrity protection over the PCIe bus (*A2*). For packets related to xPU computing but with non-sensitive payloads (e.g., generic model code and MMIO-based control/register values), we set *Write Protection* control for these packets. Specifically, we provide integrity protection and additional security verification (e.g., checking the correctness of the xPU page table register) on the computing environment (*A3*). Lastly, for packets with general functions (e.g., interrupt requests), we set them as *Full Accessible* and directly transmit these packets to their destination (*A4*). Our design securely and efficiently guides PCIe packet filtering and management in confidential xPU computing.

4.1 Packet Filter

L1/L2 Table. The L1 and L2 tables work in sequence to filter *Prohibited* packets and classify authorized packets, with a fine-grained detection. To avoid over-engineering (e.g., preparing all rules for each xPU/TVM) and defend against malicious changes to every packet attribute, we add the `Mask` attribute. This allows users to flexibly control the attribute values for packet checking. As shown in Figure 5①, we only permit the memory read/write requests from authorized TVMs to proceed to the xPU. Moreover, the L2 table determines the permissions of the authorized packets to guide security actions. The key reason for distinguishing different permissions is the combination of three packet attribute values: Packet type, interacting parties, and address space sensitivity. For example, in Figure 5②, when handling the memory read/write request packet, the Packet Filter considers whether the PCIe packet hits the address space of the workload (e.g., data and code bounce buffer). If it hits, we classify the packet as a *Write-Read Protection* packet. However, if packets only perform write operations to non-sensitive address space, we consider these packets as *Write Protection*. Lastly, in the L2 table, we treat read requests for non-sensitive information (e.g., interrupt status), and memory read requests without sensitive payload, as *Full Accessible* packets. For these packets, we directly transmit them without additional security checks.

Dynamic and secure configuration. ccAI supports dynamic policy updates to Packet Filter via a dedicated configuration space. Authorized users modify policies through the Adaptor. However, the adversary may also attack the configuration space (e.g., injecting a malicious configuration and leaking sensitive data). To mitigate this attack, ccAI encrypts the security policies before storing them in the configuration space. When applying these policies, ccAI extracts the policies and decrypts them with corresponding keys, ensuring secure configuration of Packet Filter.

4.2 Packet Handler

Once the Packet Filter blocks the unauthorized PCIe packets, the Packet Handler processes packets using the provided security actions. As shown in Figure 4, the Packet Handler processes the high-level security packets (e.g., packets with sensitive data and model parameters) with complex de/encryption and integrity check (*A2*), while it processes the low-level security packets (e.g., packets with generic AI models or insensitive xPU MMIO values) with integrity protection only (*A3*) or no additional processing (*A4*).

Key observation of xPU workloads. To support complex xPU confidentiality tasks, we first analyze workloads from varied xPU devices and applications. A critical observation emerged: Although the memory access patterns for xPUs workloads are largely varied, the workflows of PCIe packet processing are standardized. This allows us to design a general security workflow for all xPU packets, consisting of three major steps: (1) Analyzing confidential packet headers and their authentication tag packets, (2) extracting the packet payloads and performing security operations, and (3) merging the header and processed payloads for transmission. We detail key components to achieve the workflow as follows.

Control panels. Based on the aforementioned workflow, we decouple the control functions from the hardware engine and delegate them to two control panels: First, a `De/Encryption Parameters Manager` for the de/encryption confidentiality guarantee. This panel aims to manage cryptographic requirements for different tasks. To achieve this, it analyzes the packet headers and records the essential de/encryption parameters, helping to process packet payloads. Second, an `Authentication Tag Manager` for integrity checks. It handles a unique authentication tag packet queue, matching authentication tag packets and the corresponding xPU task's packets based on the tag attribute. Additionally, it extracts the authentication codes and verifies the integrity of the sensitive payload. Overall, these control panels flexibly schedule the security operations on packets and satisfy different packet processing requirements.

xPU environment guard. Besides protecting the xPU workloads, the Packet Handler additionally includes an xPU environment

guard, which supports cleaning the xPU computing environment. The xPU environment guard checks and cleans the xPU computing environment when terminating an xPU task, preventing the adversary from accessing unused data after computing. To achieve this, the xPU environment guard triggers a cold boot reset on the xPU, cleaning its memory, caches, registers, and TLB status. For xPUs that support software-based reset, the xPU environment guard can notify the Adaptor to send an environment reset packet, such as a packets with cache and TLB reset MMIO commands.

5 ccAI Optimization

As discussed in §3 and §4, ccAI ensures compatibility and security by designing a novel interaction workflow between TVM and multi-type xPUs. In this design, the PCIe-SC addresses the security gap between TVMs and xPUs by filtering and securely processing PCIe packets. Meanwhile, the TVM-side Adaptor enables native xPU software to manage xPU workloads without sacrificing user transparency.

C3. However, ccAI's new workflow risks degrading the performance of heterogeneous clouds, such as reducing I/O throughput and increasing AI computing latency. This is because ccAI introduces additional interaction between PCIe-SC and TVM/xPU. Moreover, the essential security operations (e.g., de/encryption) also introduce performance overhead in xPU computing.

Solution to C3. To address this challenge, we focus on optimizing the frequent interactions by reducing the redundant (1) I/O read and write operations in DMA. Meanwhile, we reduce overhead from security-critical processes. The effectiveness of our optimizations is validated in §8.5 (**G3**). We detail our optimization solutions as follows.

Optimization on I/O read. In ccAI's xPU computing, the xPU often initiates DMA request to access sensitive data/code in TVM memory. We observe that such process can introduce redundant I/O read operations — The Adaptor can repeatedly query PCIe-SC for the metadata of the DMA process (e.g., transmission payload size and address). To reduce the I/O read operations, we do not store the DMA metadata in ccAI and wait for the Adaptor to fetch. Instead, the PCIe-SC collects DMA metadata in batches and provides them to TVM. In this step, we allocate a temporary buffer in TVM memory to store the metadata batches. This allows the Adaptor to directly read the metadata and encrypt data for DMA, without frequent I/O read interaction with PCIe-SC.

Optimization on I/O write. After the Adaptor completes data encryption and prepares for data transfer, it typically sends a PCIe packet request (via an I/O write operation) to notify the PCIe-SC to initiate further transmission. In a non-optimized design, this process can generate redundant requests. For instance, the Adaptor may split a large-scale data decryption task into smaller subtasks, each of which can generate a PCIe packet request when the encryption is finished. To reduce this redundancy, we require the Adaptor to process data (e.g., perform encryption) in batches. Once the entire data region is processed, we use only one I/O write operation to notify PCIe-SC for data transfer. Based on this, we reduce the frequency of write operations.

Optimization on security operations. Besides optimizing the frequent interactions, we also consider two solutions to optimize the security operations in ccAI. First, on the TVM-side Adaptor, we leverage hardware-based instructions (e.g., Intel AES-NI [32]) to offload the de/encryption and memory copy process. Such instructions leverage well-designed hardware acceleration mechanism and perform faster than software-based instructions. Second, permitted by privileged software (e.g., TDX module), we can allocate additional CPU threads and cores to process the security operations in parallel. This design prevents security operations from becoming a bottleneck in high-throughput AI workloads.

6 Trust Establishment

ccAI enables users to securely provide workloads to the ccAI-equipped platform and verify their computing environments. To achieve this, ccAI deploys trust modules on both CPU-side TVMs and the PCIe-SC, alongside a suite of trust establishment processes. We detail ccAI's trust establishment design as follows.

Secure boot. Leveraging the Hardware Root of Trust (HRoT) on the CPU-side platform, ccAI adapts the native secure boot and software (e.g., Adaptor) measurement for TVMs. For the PCIe-SC' secure boot, we design a TPM-compatible [82] trust module, called HRoT-Blade, to ensure component integrity during boot. In this process, the HRoT-Blade decrypts the PCIe-SC's bitstream file (e.g., Packet Filter) and firmware stored in an external flash memory, then measures the integrity of each component via a pre-defined chain of trust. ccAI updates the measurement results in a dedicated register — the Platform Configuration Register (`PCR`) — which is used for generating attestation reports. Once the results are successfully attested, ccAI loads the binary file into the boot loader and boots up the PCIe-SC.

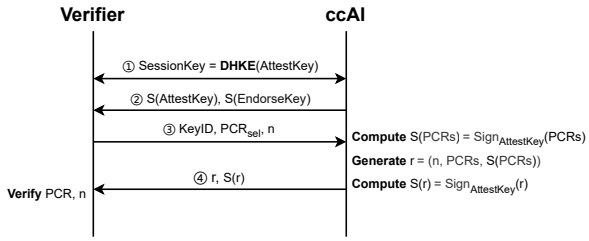


Figure 6: Remote Attestation Protocol of ccAI.

Remote attestation. ccAI designs a standard remote attestation protocol based on trusted attestation frameworks [80, 81], similar to those used in building TEEs [14, 15, 75, 84, 91, 93]. This protocol establishes a secure channel with a user verifier, enabling verification of the user's own ccAI-xPU set components. Figure 6 shows its four major steps: First, the verifier and ccAI perform a key exchange using the Diffie-Hellman protocol [22], generating a shared `SessionKey` to de/encrypt subsequent messages. Second, the verifier requests the Attestation Key (AK) and Endorsement Key (EK) certificates from the ccAI-equipped platform. It further validates them with the corporate Root Certificate Authority (CA). Both EK and AK are stored in the HRoT-Blade: The EK is pre-installed by the vendor during manufacturing, while the AK is randomly generated at system boot. Third, the verifier sends a challenge (e.g., `KeyID` for xPU selection, `PCR` selection, with a random nonce) to its TVM. The `PCR` selection is tied with two components: (1) The CPU-side

HRoT (recording CPU firmware) and (2) the HRoT-blade (recording PCIe-SC firmware). The TVM forwards the challenge to both HRoT components for attestation. Once HRoT receives the challenge, it signs the required PCR with AK to compute certificates (S(PCRs) in Figure 6), and combines with nonce and PCR to generate the report. Lastly, the TVM returns the report r and its certificate to the verifier. Using the CA, the verifier validates the nonce and signature, confirms the authenticity of the PCR, and uses the PCR to verify the overall integrity of ccAI system.

For measuring xPU devices, if xPU devices equip HRoT, the PCIe-SC can collaborate with this HRoT to verify the device authenticity and firmware integrity (shown in Figure 6). If xPU devices delegate attestation to ccAI, it can achieve this by employing software-based attestation [38]. Moreover, xPU vendors can provide external interfaces (e.g., serial peripheral interfaces) to connect directly to the HRoT-Blade, helping ccAI to attest the xPU computing environment.

Workload key management. To support workload en/decryption, ccAI manages shared symmetric keys for data transmission over the untrusted PCIe fabric. The TVM and PCIe-SC negotiate the symmetric keys and store keys in their own trust modules. ccAI also dynamically updates the Initialization Vector (IV) for the symmetric keys to ensure encryption randomness. As mentioned in [51], IV exhaustion can lead to several attacks due to IV reuse [23, 29, 42]. Thus, ccAI follows the solution used in NVIDIA H100 [51] (e.g., generating and exchanging a new key) to mitigate this risk. Lastly, when xPU computing terminates, both the TVM and the PCIe-SC securely destroy shared symmetric keys to prevent data leakage.

Sealing. The sealing mechanism defends against physical attacks on the PCIe-SC, the xPU device, and their internal PCIe connection during computing. To achieve this, ccAI seals the aforementioned hardware in a chassis, in which it runs physical sensors (e.g., pressure and temperature sensors) to monitor the physical integrity status. The HRoT-Blade periodically retrieves the physical status via an integrated circuit (I^2C) bus and updates in PCR registers, enabling the remote user to attest the physical integrity of the chassis.

7 Implementation

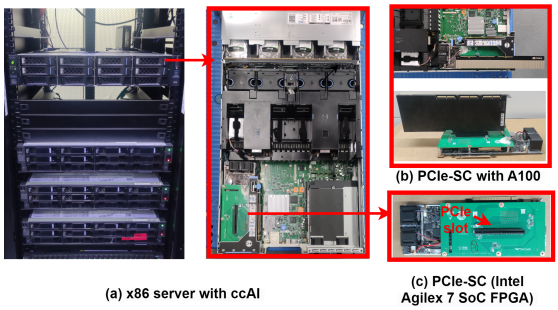


Figure 7: ccAI prototype system on real-world clouds.

We implemented a prototype of ccAI on an Intel server equipped with 256 GB of memory and a 96-core CPU (see Figure 7a). On CPU side, we run a Ubuntu 22.04 OS as its kernel, within which we integrate our Adaptor, trust modules, and a set of xPU software stacks. To validate ccAI's confidential xPU computing supports, we select five distinct xPUs: An NVIDIA A100 GPU [56], an NVIDIA RTX4090Ti GPU [55], an NVIDIA T4 GPU [59], a Tenstorrent N150d NPU [78], and an Enflame S60 GPU [24]. For the software stacks, we support CUDA 12.1 and NVIDIA 550.90.07 GPU driver for NVIDIA GPUs, tt-buda and ttkmd-1.29 software stacks for Tenstorrent NPU, and EFSMI library v1.4.0.606 and Enflame driver v1.4.0.3 for Enflame S60 GPU.

To verify the security supports of PCIe-SC, we prototype it on an Intel Agilex 7 SoC FPGA (see Figure 7c). This FPGA implementation includes a set of configurable IP cores for cryptographic operations, integrity verification, PCIe packet-based filter (for DMA/MMIO), and other essential functionalities. The PCIe-SC connects to xPU (e.g., NVIDIA A100 GPU) via a standard PCIe slot (see Figure 7b). We elaborate on our prototype in the following sections.

7.1 Adaptor

As aforementioned in §3, the Adaptor fulfills two major functions: (1) Providing additional confidential xPU computing support for the generic xPU software stacks and (2) interacting with the PCIe-SC. We detail how ccAI prototypes achieve these functions as follows.

Confidential xPU support. Rather than modifying the native xPU drivers, we create a new kernel module (called `ccAI_adaptor`) to provide confidential xPU support. Specifically, we design a pair of de/`encrypt_data` functions to locate the address of packet payload contents (e.g., data and code) and perform de/encryption with the specified algorithm (e.g., AES-128 in our prototype). In this step, we leverage a set of hardware-assisted de/encryption instructions, called Intel AES-NI [32], to optimize AES de/encryption in TVM.

PCIe-SC interaction. We allocate a 64KB MMIO region and implement additional kernel functions to achieve PCIe-SC interaction with the TVM. This implementation comprises two components:

First, we implement the control of PCIe-SC security components. For Packet Filter, we provide a `pkt_filter_Manage` function to handle L1/L2 tables, including configuring the MMIO space for security rules transmission, loading and transmitting rules, and activating them. For Packet Handlers, we implement a set of control bits and kernel functions for the security operations (i.e., de/encryption, integrity check, and xPU environment protection). These functions enable or disable the hardware engines of each operation, initialize the `De/Encryption Parameters Manager` and `Authentication Tag Manager` control panels, and configure the panels. Moreover, we implement a `hw_init` function to initialize the PCIe-SC.

Second, we implement a set of H2D and D2H functions to achieve the data transmission between the TVM Host and the PCIe-SC Device. Our implementation includes three tasks: (1) Configuring the address, size, and other attributes of the H2D/D2H buffers, (2) starting/terminating the H2D/D2H execution, and (3) monitoring the running status of each position on H2D/D2H queues.

7.2 PCIe-SC

As described in §4, the PCIe-SC consists of two major parts: The Packet Filter and the Packet Handler. We detail our implementation of these two components as follows.

Packet Filter. For initializing the Packet Filter, we allocate a 4KB Upstream Bar space on the PCIe-SC to build L1/L2 tables. Authorized users can add specific security policies (32 bytes per policy) by invoking the `pkt_filter_manage` function in the Adaptor. The Packet Filter focuses on three key information in the PCIe packet Header [63–65, 67]: (1) The packet type, which is a combination of format and memory access attributes (e.g., memory read/write configurations), (2) the route IDs, which contain the PCIe information of the requested and completed devices, and (3) the payload metadata, which describes the (`start_address`) and (`end_address`) for triggering packet-specific security operations. To extract the necessary information, the PCIe-SC uses an integrated PCIe switch to receive packets, then parses them according to the standard PCIe packet format. Since all PCIe-based xPUs rely on this format for DMA/MMIO interactions with the CPU host, the Packet Filter supports different xPU devices.

Packet Handler. To implement the Packet Handler, we delegate the control panels (i.e., `De/Encryption Parameters Manager` and `Authentication Tag Manager`) to manage parameters for de/encryption and integrity check, such as the key length, key, initial vector (with 12-byte nonce and 4-byte counter), and authentication tag (16-byte length). Additionally, we implement an AES-GCM-SHA hardware engine for de/encryption and integrity checks. We also implement an environment check module to validate the MMIO values and clean the xPU environment.

8 Evaluation

In this section, we evaluate the compatibility, security, performance overhead, and optimization of the ccAI prototype with respective to six research questions:

RQ1: How does ccAI compare with the state-of-the-art in compatibility support?

RQ2: Can ccAI defend against the privileged adversary?

RQ3: How much performance overhead does ccAI introduce to different evaluation metrics when running LLMs?

RQ4: How much performance overhead does ccAI introduce on multi-type LLMs and xPUs?

RQ5: How effective is our optimization on ccAI?

RQ6: How much performance overhead does ccAI introduce on stress-test scenarios?

8.1 RQ1: Comparison to State-of-the-art

We discuss the compatibility issues on ccAI and the state-of-the-art studies, with detailed report in Table 2. Our compatibility analysis focus on three major aspects: (1) User transparency, (2) support of multi-type xPUs, and (3) support of heterogeneous clouds. ccAI's system design is friendly to xPU application developers. Unlike several state-of-the-art that introduce customized user-layer APIs [39, 83, 91, 93], ccAI directly adapts to native xPU application without additional changes. Additionally, ccAI achieves high compatibility with commercial xPUs: ccAI neither alters the xPU software stacks (modified in most studies) nor changes xPU hardware logic (as seen in typical hardware-based designs [50, 83, 84, 91]). Notably, rather than supporting specific xPUs or TDISP-compliant xPUs, ccAI aims to support legacy xPUs for AI computing. For heterogeneous cloud support, ccAI leverages general TVMs to protect xPU applications, software stacks, and the TVM-side Adaptor — yet it does not rely on TVM-specific security primitives. Note that ccAI follows existing IOMMU settings in TVM or privileged software, without additional changes. Our PCIe-SC also functions as a standard PCIe switch.

Comparison to H100. We compare ccAI with NVIDIA H100 GPU [50] in security and performance. For security, both H100 and ccAI ensure a confidential computing environment for xPUs, including three core capabilities: (1) Isolating xPU from software and physical adversary targeting the TVM and PCIe, (2) providing de/encryption support for xPU workloads, (3) enabling secure boot, attestation and key management for trust establishment. In performance, studies [77, 94] show that H100's confidential computing introduces a moderate (more than 20%) performance overhead on execution metrics like end-to-end (E2E) Latency. In ccAI, it achieves a low (0.05% – 5.67%) latency overhead due to the fine-grained de/encryption mechanisms. Note that H100 and ccAI exhibit comparable overhead on throughput.

Comparison to HETEE. Next, we discuss an AI computing protection design for a heterogeneous system, HETEE [93]. ccAI is better suitable for general TVM-equipped servers for three reasons. First, HETEE relies on the rack-scale PCIe resource-sharing features (e.g., PCIe Express fabric chips) to deploy its security controller and isolated computing nodes — features not supported by all servers. Instead, ccAI is designed for general servers. Second, HETEE requires specialized APIs to receive user data/models, along with a corresponding manifest from remote users. This adds engineering effort for adapting general xPU applications. However, ccAI does not require such API changes. Third, HETEE adapts microserver (i.e., proxy nodes) as a TEE for xPU software, which requires hardware changes on the microserver's PCB board. ccAI avoids such changes and protects xPU software with existing TVMs.

Comparison to TDISP. TDISP is an emerging standard for confidential xPU design, with functionality that partially overlaps with ccAI. Nevertheless, the full hardware implementation of TDISP (e.g., cloud with PCIe IDE [66], TDISP-based CPU architecture [6, 9, 37], and multiple TDISP-compliant xPU) will take years to mature. Currently, AI-supported clouds [4, 25] still lack TDISP support, and adopting TDISP can incur non-negligible hardware costs. Compared with TDISP, ccAI adapts general TVMs and xPUs for confidential AI computing.

Comparison to secure PCIe. Furthermore, we explain why ccAI aims at protecting PCIe packet instead of another choice — secure PCIe channel that encrypting all transmission content. We summarize two key reasons: First, most xPU devices (e.g., general NVIDIA A100 GPUs [56]) lack built-in support for cryptographic operations. Without such support, these xPUs cannot directly decrypt sensitive payloads transmitted over a secure PCIe channel. Second, mainstream xPU software stacks prefer a close-source implementation for PCIe transmission interfaces (e.g., `CudaMemCopy` in CUDA [49]). Modifying these stacks to support secure PCIe channel can be challenging and often impractical.

Comparison to Cronus and HyperTEE. Lastly, we discuss two xPU protection designs, Cronus [40] and HyperTEE [13], which introduce a new TEE system design for xPU computing. ccAI outperforms both in software/hardware compatibility, for two reasons:

Table 2: Comparison between ccAI and the state-of-the-art in compatibility issues. The green and red entry: High compatibility and low compatibility design.

Design Type	System Design	User Transparency		Multi-type xPU Support		Supported Heterogeneous Clouds	
		App Changes	xPU SW Changes	xPU HW Changes	Supported xPU	Supported TEE/TVM	Host PL-SW Changes
CPU TEE-based Designs	ACAI [75]	No	Yes	No	TDISP-compliant xPU	Arm CCA	RMM, Monitor
	Cronus [40]	No	Yes	No	General xPU	Arm SEL2	S-Hyp, Monitor
	CURE [12]	No	Yes	No	GPU	Customized RISC-V TEE	Monitor, CPU Firmware
	HIX [39]	Customized API	Yes	No	GPU	Intel SGX	CPU Firmware
	Portal [70]	No	Yes	No	GPU	Arm CCA	RMM, Monitor
	HyperTEE [13]	Customized API	Yes	No	DNN Accelerator	Customized RISC-V TEE	Monitor
PL-SW-assisted Designs	CAGE [85]	No	Yes	No	GPU	Arm CCA	Monitor
	Honeycomb [44]	No	Yes	No	GPU	AMD SEV	SVSM, Monitor
	MyTEE [30]	No	Yes	No	GPU	Customized Arm TEE	Monitor
Hardware Designs	ITX [83]	Customized API	Yes	Yes	IPU	General TVM	No
	NVIDIA H100 [50]	No	Yes	Yes	GPU	Intel TDX, AMD SEV	No
	Graviton [84]	No	Yes	Yes	GPU	Intel SGX	No
	ShEF [91]	Customized API	Yes	Yes	FPGA-Acc.	General TVM	No
Isolated Platform	HETEE [93]	Customized API	No	No	General xPU	Customized proxy TEE	No
TDISP-based Designs	Intel TDX Connect [37]	No	Optional	Optional	TDISP-compliant xPU	Intel TDX	TDX Connect
	ARM RMEDA [11]	No	Optional	Optional	TDISP-compliant xPU	Arm CCA	RMM
	AMD SEV-TIO [6]	No	Optional	Optional	TDISP-compliant xPU	AMD SEV	SEV Firmware
ccAI (Ours)		**No**	**No**	**No**	**General xPU**	**General TVM**	**No**

First, ccAI does not modify the existing xPU software stacks or APIs invoked by the xPU application. By contrast, Cronus and HyperTEE mandate these changes to collaborate xPU software with their security mechanisms. Second, ccAI is not tied to a specific TVM architecture. Instead, Cronus is designed on the Arm SEL2 platform, and HyperTEE designs a hardware security IP on the CPU chip. Nevertheless, several security supports (e.g., xPU page table management and access control) in Cronus and HyperTEE can further enhance the security design of our PCIe-SC.

8.2 RQ2: Security Analysis of ccAI

We follow the threat model in §2.2 and conduct a detailed security analysis on ccAI. Same as most hardware-based designs, ccAI defends against adversaries from the untrusted host, unauthorized TVMs, malicious devices, and the untrusted PCIe fabric. We elaborate our security analysis as follows.

Attacks from host/TVM. The adversary may attempt to access or tamper with the sensitive data/models in the TVM. To achieve this, the adversary can access the TVM from an untrusted host OS, hypervisor, or a registered TVM. Nevertheless, ccAI defends against these attacks by hardware-based TVM security primitives (e.g., Intel TDX). Specifically, privileged software (e.g., TDX module) configures these primitives to securely protect the TVM's address space. Besides accessing TVM, the adversary may attempt to access the protected xPU from the host or unauthorized TVM — typically by sending packet read/write requests to the protected xPU. However, our PCIe-SC verifies packet metadata in the L1/L2 table and blocks unauthorized packet requests.

Attacks from malicious devices. Besides the access attempts from the CPU side, the adversary may control a malicious device to undermine ccAI's security guarantees. One direct attack is to access TVM via the malicious device. However, privileged software restricts such access attempts (e.g., by configuring the IOMMU to isolate TVM from malicious devices). Furthermore, the adversary may use the malicious device to attack xPUs, but the Packet Filter effectively blocks such access attempts. Another potential attack entails faking a PCIe packet request and sending it to the xPU. However, the packet contents still fail the integrity check, as the adversary lacks de/encryption keys. Note that ccAI also addresses packet replay attacks by leveraging initial vectors.

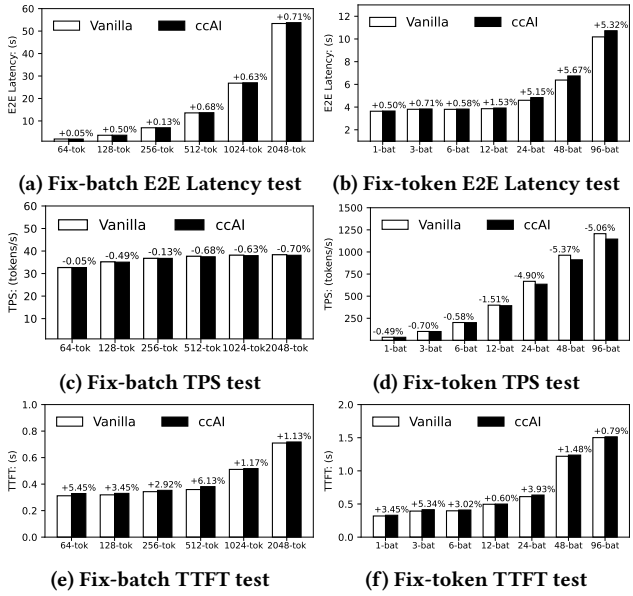
Attacks from PCIe. The adversary may compromise PCIe packet transmission over the untrusted PCIe fabric [31, 43, 72, 76]. For example, a sophisticated attacker can snoop on the PCIe bus to directly access transmitted packets, extracting or tampering with sensitive data, code, and MMIO-based commands. ccAI mitigates this risk by encrypting packet payloads containing sensitive data and implementing integrity checks for packets involved in confidential xPU computing. Additionally, the adversary may compromise the integrity of PCIe packet transmission, such as by altering packet order, replaying packets, or deleting them. To defend against this attack, ccAI incorporates an additional check to verify the order of packet transmission. The adversary may also route packets carrying sensitive data or models to unexpected TVMs or other peripherals. Even in such cases, the packet payload cannot be leaked as they lacks the necessary decryption keys.

Attacks on xPU. Last, the adversary may attempt to compromise the PCIe-SC, the xPU device, and the internal PCIe connection between these components — most commonly by tampering with their firmware. ccAI defends against this attack through the implementation of a secure boot process, where integrity is carefully measured and reported via remote attestation. Notably, today's xPUs support firmware signature checking [45], a feature ccAI leverages to enable this secure boot workflow. Moreover, the adversary may attempt to physically compromise the hardware during xPU computing. However, the sealing mechanism in §6 can effectively detect such physical tampering.

TCB addition size. We use cloc [3] and Quartus [35] to respectively measure the software and hardware TCB addition in ccAI prototype, whose breakdown is reported in Table 3. ccAI introduces 3.1K Lines of Code (LoC) on each CPU-side TVM without additional changes on privileged software. For hardware changes, the PCIe-SC consumes 218.6K Adaptive Look-Up Tables (ALUTs), 195.7K logic registers (Regs), and 630 Block RAMs (BRAMs). Overall, our prototype does not expose a large attack surface to the adversary.

Table 3: Breakdown of TCB addition in ccAI. The HRoT-Blade part is implemented on an embedded Cortex-A53 hard processor system (HPS) and does not introduce additional hardware cost. The Others part includes PCIe switch, clocks and connections.

Components	LoC	ALUTs	Regs	BRAMs
TVM				
Adaptor	2.1K	–	–	–
Trust Modules	1.0K	–	–	–
PCIe-SC				
Packet Filter	–	11.3K	32.4K	310
Packet Handlers	–	175.5K	56.8K	72
HRoT-Blade	–	0	0	0
Others	–	31.5K	106.5K	248
Total	3.1K	218.6K	195.7K	630



(a) Fix-batch E2E Latency test (b) Fix-token E2E Latency test

(c) Fix-batch TPS test (d) Fix-token TPS test

(e) Fix-batch TTFT test (f) Fix-token TTFT test

Figure 8: Performance overhead in Llama-2-7B-Chat model. Note that the left-side figures (a/c/e) set the batch size as 1 and the right-side figures (b/d/f) set the size of token as 128.

8.3 RQ3: LLM benchmarks Evaluation with Different Metrics

Evaluation on Llama-2. We evaluate ccAI's performance using Llama-2 chat model [79] — a widely adopted LLM. For the experimental setup, we fix the model parameters size at 7 billion and vary two input variables: (1) Tokens, which indicate the number of words in a chat question, and (2) batch, which represents the number of asked questions at once. Prompts used in the experiments are adapted from the ShareGPT [69] and Hellaswag [89] datasets with changes. We compile the benchmarks with CUDA [49] libraries and run them on NVIDIA A100 GPU. Following NVIDIA's standard evaluation guidelines [54], we selected three key metrics for analysis: First, the end-to-end latency (E2E Latency), which indicates the total time to generate a response to chat questions. Second, the tokens per second (TPS), which is defined as the number of output tokens generated per second. Third, the time to first token (TTFT), which shows the time elapsed until the first token is generated.

Figure 8 shows the execution results. ccAI introduces low (0.05% – 5.67%) performance overhead across all metrics and benchmarks. The evaluation results in E2E and TPS benchmarks are similar in two aspects: First, significant increases in tokens/batches does not lead to a sharp rise in performance overhead. For example, expanding the input token size from `1024-tok` to `2048-tok` adds merely 0.08% overhead to the E2E Latency. Similarly, the `96-bat` benchmark does not incur higher performance overhead than `48-bat` benchmark (e.g., 5.37% TPS overhead in `96-bat` but 5.06% in `48-bat`). Second, when we increase the input batch size from 12 to 24, the overhead shows a relatively large increase in both the E2E Latency and TPS. For instance, TPS overhead increases by 3.39% between `12-bat` and `24-bat`, but only 0.47% between `24-bat` and `48-bat` benchmarks. For the TTFT evaluations, we observe that ccAI performs better on benchmarks with larger-size tokens (e.g., 5.45% in `64-tok` and 1.13% in `2048-tok`). However, TTFT overhead fluctuates as batch size increases. This fluctuation can be attributed to the relatively short input sequences in the evaluation. Note that the TTFT of each batch is relatively small and easily affected by the changed PCIe transmission speed.

8.4 RQ4: Evaluation on Different LLMs/xPUs

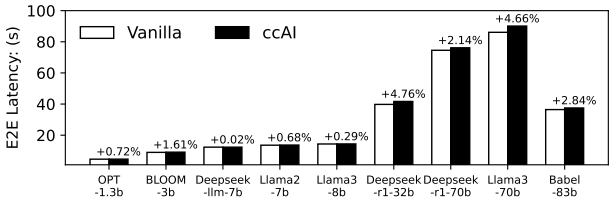


Figure 9: Performance overhead on different LLMs. The title of each benchmark indicates the LLM type and parameter size (counted as billions parameters). The Babel-83b is quantized to `INT2` (`INT8` for Deepseek-r1-32b, `INT4` for Deepseek-r1-70b and Llama3-70b) and it has a relative small E2E Latency.

Evaluation on other LLMs. We measure ccAI across a diverse set of LLMs with varying parameter scales. Specifically, our experiments includes two light-weight LLMs (i.e., OPT-1.3b [90] and BLOOM-3b [71]), three medium-weight LLMs (i.e., Deepseek-llm-7b [87], Llama-2-7b [79], and Llama-3-8b [1]), and four heavy-weight benchmarks (i.e., Deepseek-r1-32b [21], Deepseek-r1-70b [21], Llama-3-70b [1] and Babel-83b [92]). For consistency across all benchmarks, we fix the batch size as 1 and token size as 512. All benchmarks are compiled and run in the same environment as the Llama-2 evaluation, with the E2E Latency as the primary measurement metric. Figure 9 shows the evaluation results. Overall, ccAI introduces a low (0.72% – 4.76%) performance overhead on the selected benchmarks. The heavy-weight LLMs introduce a relatively high performance overhead than light-weight LLMs (e.g., 1.61% on BLOOM-3b but 4.76% on Deepseek-r1-32b). This trend can be partially attributed to limitation in our prototype's PCIe transmission bandwidth. However, crucially, this bandwidth-related overhead does not scale linearly with model parameter size (e.g., 2.14% on Deepseek-r1-70b and 2.84% on Babel-83b).

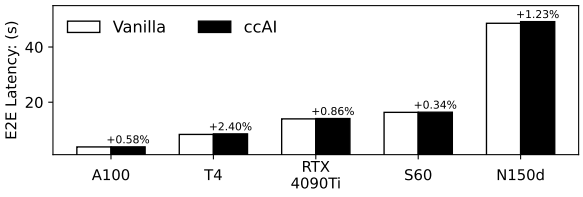


Figure 10: Performance overhead of the selected xPUs.

Evaluation on multi-type xPU. Next, to assess ccAI's compatibility and performance across diverse xPU architectures, we evaluate on five distinct xPU devices: An NVIDIA A100 GPU [56], an NVIDIA RTX4090Ti GPU [55], an NVIDIA T4 GPU [59], a Tenstorrent N150d NPU [78], and an Enflame S60 GPU [24]. We measure NVIDIA A100 GPU, NVIDIA RTX4090Ti GPU, and Enflame S60 GPU by running the Llama2-7b model. However, due to memory limitations, we measured the NVIDIA T4 GPU and Tenstorrent N150d NPU using a lightweight OPT-1.3b benchmark. Across all tests, we fix the token size as 512 and batch size as 1. Figure 10 shows that ccAI introduces 0.58% – 2.40% performance overhead across all five xPU devices. This result confirms two key attributes of ccAI: First, ccAI achieves compatibility with multi-type xPUs, including GPUs from NVIDIA/Enflame and an NPU from Tenstorrent. Second, ccAI ensures low overhead with different xPUs.

8.5 RQ5: Evaluation of Optimization

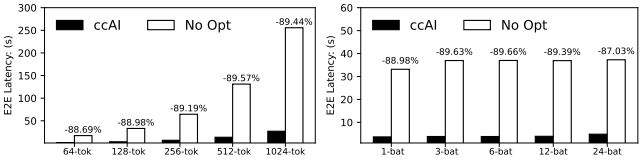
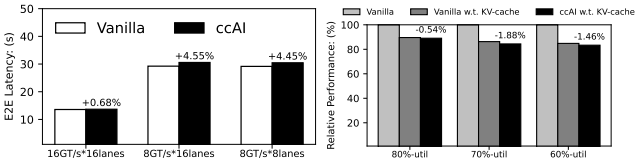


Figure 11: Performance comparison between the non-optimized mechanism and ccAI in Llama-2-7B-Chat. Note that the left benchmark configures the batch size as 1 and the right benchmark sets the size of token as 128.

Lastly, we measure the effectiveness of our performance optimization (described in §5). Specifically, we compare the ccAI prototype with the non-optimized version on the Llama-2-7B-chat benchmark, measuring them with different tokens and batch sizes on NVIDIA A100 GPU. Figure 11 shows the comparison results. ccAI reduces 88.7% – 89.8% E2E Latency overhead on the selected benchmarks. Figure 11 further reveals that changes in token/batch size have minimal impact on our optimization's effectiveness. For instance, when increasing the input token size from 64 to 1024, the optimized prototype introduces more latency but still reduces 89.44% performance overhead. Moreover, increasing the input batch size from 1 to 24 only leads to a marginal 1.95% decrease (from 88.98% to 87.03%) in optimization efficacy. Collectively, our evaluation confirms both the effectiveness (via large overhead reduction) and robustness (via steady optimization efficacy on varied workloads) of our optimization mechanism.

8.6 RQ6: Evaluation on Stress Test Scenarios

Limited PCIe bandwidth. To understand how ccAI performs when PCIe bandwidth is shared/limited, we perform a stress test



(a) Limited PCIe bandwidth **(b) Limited xPU memory**

Figure 12: Performance results in stress test scenarios.

by adjusting both the speed and lane count of PCIe. Experiments are run on NVIDIA A100 GPU with Llama-2-7b, using a fixed configuration (token size as 512 and batch size as 1). Figure 12a shows the results. The decreasing PCIe bandwidth increases latency for both the native system and ccAI. Critically, however, ccAI does not introduce higher performance overhead when PCIe speed/lanes are limited (e.g., 4.45% on 8GT/s speed with eight lanes).

KV-cache. Next, we test ccAI in a scenario where xPU memory is limited, forcing frequent swapping of the KV-cache to CPU memory. We set a 3 GB KV-cache and limit memory utilization percentage (from 80% to 60%) on NVIDIA A100 GPU, triggering KV-cache swapping. We run Llama-2-7b and select inputs from ShareGPT, with input tokens ranging from 4 to 924 (batch size as 1). Considering the input tokens of each test are largely varied, we report relative performance slowdown (instead of E2E Latency) in Figure 12b. In the KV-cache swapping scenario, both ccAI and the native system reduce performance to ~83%. Importantly, ccAI only introduces a low addition (less than 2%) on overhead, confirming ccAI's ability to maintain low overhead even when xPU memory is limited.

9 Discussion

Customized packets. The xPU vendors may design customized PCIe packets to support proprietary management functionalities (e.g., customized `message` packets for power management). Nevertheless, such packets do not violate the standard PCIe format (e.g., PCIe header and payload format), so that they can be recognized and transferred in general PCIe Root Complex. This ensures ccAI's PCIe-SC can still analyze the packet Header of the customized packets and perform basic security operations. Moreover, if xPU vendors require to handle these packets with specific rules (e.g., encrypting sensitive `message` PCIe packets), they can add such rules into Packet filter via PCIe-SC's MMIO registers.

PCIe-SC for multiple xPUs and users. In our implementation, each PCIe-SC serves a single xPU that is owned by a TVM. In the future, ccAI can upgrade the PCIe-SC to support multiple xPUs or xPU with multi-user support (e.g., NVIDIA MIG-enabled xPUs [53]). To achieve this, the PCIe-SC can establish an isolated secure channel with every connected xPU, or user content on a multi-user xPU. Next, the PCIe-SC distinguishes each xPU, or virtual functions on a xPU, by unique PCIe identifiers (e.g., Bus/Device/Function ID on PCIe). Based on this, ccAI can handle packets with unique security policies and route packets to correct xPU. Moreover, if the xPU vendor allows, ccAI's PCIe-SC design can integrate into the xPU board to further reduce the hardware cost and complexity.

Supporting non-PCIe xPUs. While ccAI currently targets PCIe-attached xPUs, its design can be adapted to several xPUs with non-PCIe connectors (e.g., NVIDIA SXM on Hopper GPUs [57]). We consider two requirements for such support: First, the connector

must transmit DMA/MMIO requests with a basic unit (similar to PCIe packets). Second, this unit must contain accessible metadata (similar to PCIe packet Header) to guide ccAI for security operations, and the format must be open source. If these requirements are satisfied, ccAI can mirror existing security design to filter these units, perform de/encryption, and securely route to target xPUs.

10 Conclusion

In this paper, we present ccAI, a compatible and confidential system designed for xPU-accelerated AI computing in heterogeneous cloud environments. To address key limitations of existing studies, such as lack of support for multiple xPU types and insufficient user transparency, ccAI introduces two core components: A TVM-side Adaptor and a PCIe Security Controller (PCIe-SC). These components work collaboratively to protect xPU software, hardware devices, and the PCIe communication between the TVM and xPU, while ensuring high compatibility and preserving application transparency. ccAI anchors its security enforcement at the PCIe packet level, enabling uniform protection across diverse xPUs with varying hardware architectures and software stacks. To handle the complexity of PCIe traffic, ccAI employs a fine-grained and flexible packet processing mechanism that classifies, filters, and securely processes packets based on their attributes. Furthermore, ccAI optimize performance overhead caused by frequent I/O interactions and cryptographic operations. We implement a prototype of ccAI and evaluate it using multiple real-world xPUs and a suite of LLMs. Results show that ccAI effectively secures xPU computing with low (0.05% – 5.67%) performance overhead.

Acknowledgments

We would like to thank the anonymous reviewers and COMPASS members for their insightful comments. This work is partly supported by the National Natural Science Foundation of China under Grant No.62372218, No.U24A6009. This work is also supported in part by HK RGC Collaborative Research Fund (No. C5032-23GF), and Research Institute for Artificial Intelligence of Things, The Hong Kong Polytechnic University. This work is also in part supported by Ant Group.

References

[1] Abhinav Jauhri Aaron Grattafiori, Abhimanyu Dubey et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783* (2024).
[2] Shaizeen Aga and Satish Narayanasamy. 2017. InvisiMem: Smart Memory Defenses for Memory Bus Side Channel. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*.
[3] AlDanial. 2021. cloc. https://github.com/AlDanial/cloc.
[4] Alibaba. 2025. Alibaba Cloud AI and Data Intelligence. https://alibabacloud.com/en/solutions/ai/data-intelligence?_p_lc=1.
[5] AMD. 2023. AMD Radeon RX Graphics Cards. https://www.amd.com/en/graphics/radeon-rx-graphics.
[6] AMD. 2023. AMD SEV-TIO: Trusted I/O for SecureEncrypted Virtualization. https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf.
[7] AMD. 2024. FPGA Leadership Across Multiple Process Nodes. https://www.xilinx.com/products/silicon-devices/fpga.html.
[8] ARM. 2023. Arm Mali Graphics Processing Units (GPUs). https://developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus.
[9] ARM. 2023. Arm System Memory Management Unit Architecture Specification. https://developer.arm.com/documentation/ihi0070/latest/.
[10] ARM. 2023. Ethos-N78. https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-n78.
[11] ARM. 2023. Introducing Arm Confidential Compute Architecture Guide. https://developer.arm.com/documentation/den0125/latest/.
[12] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In *Proceedings of the 30th USENIX Security Symposium*.
[13] Yunkai Bai, Peinan Li, Yubiao Huang, Michael C Huang, Shijun Zhao, Lutan Zhao, Fengwei Zhang, Dan Meng, and Rui Hou. 2024. HyperTEE: A Decoupled TEE Architecture with Secure Enclave Management. In *2024 57th IEEE/ACM International Symposium on Microarchitecture*. IEEE.
[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* (2016).
[15] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*.
[16] CVE. 2017. CVE-2017-17176. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-17176.
[17] CVE. 2018. CVE-2018-12010. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12010.
[18] CVE. 2019. CVE-2019-2318. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-2318.
[19] CVE. 2020. CVE-2020-5991. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-5991.
[20] CVE. 2022. CVE-2022-21821. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-21821.
[21] Haowei Zhang Daya Guo, Dejian Yang et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).
[22] Whitfield Diffie and Martin Hellman. 1976. New Directions in Cryptography. *IEEE Transactions on Information Theory* (1976).
[23] Morris J Dworkin. 2007. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Technical Report NIST SP 800-38D. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf
[24] Enflame Tech. 2025. Enflame S60. https://www.enflame-tech.com/s60.
[25] Google. 2022. GPUs on Compute Engine. https://cloud.google.com/compute/docs/gpus/.
[26] Google. 2024. Tensor Processing Units (TPUs) - Google Cloud. https://cloud.google.com/tpu.
[27] Google. 2025. AI Infrastructure ML and DL Model Training. https://cloud.google.com/ai-infrastructure/.
[28] Google. 2025. Announcing A3 supercomputers with NVIDIA H100 GPUs, purpose-built for AI. https://cloud.google.com/blog/products/compute/introducing-a3-supercomputers-with-nvidia-h100-gpu.
[29] Shay Gueron and Vladislav Krasnov. 2014. The Fragility of AES-GCM Authentication Algorithm. In *Proceedings of the 11th International Conference on Information Technology: New Generations*.
[30] Seung-Kyun Han and Jinsoo Jang. 2023. MyTEE: Own the Trusted Execution Environment on Embedded Devices. In *Proceedings of the 30th Annual Network and Distributed System Security Symposium*.
[31] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. 2020. DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*.
[32] Intel. 2025. Intel Advanced Encryption Standard Instructions (AES-NI). https://www.intel.com/content/www/us/en/developer/articles/technical/advanced-encryption-standard-instructions-aes-ni.html.
[33] Intel. 2025. Intel Agilex 7 FPGA F-Series 027. https://www.intel.com/content/www/us/en/products/sku/208599/intel-agilex-7-fpga-fseries-027-r25a/ordering.html.
[34] Intel. 2025. Intel® 64 and IA-32 Architectures Software Developer Manuals. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.
[35] Intel. 2025. Quartus Prime Design Software. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html.
[36] Intel Corporation. 2022. Intel Trust Domain Extensions. https://cdrdv2.intel.com/v1/dl/getContent/690419.
[37] Intel Corporation. 2023. Intel TDX Connect TEE-IO Device Guide. https://cdrdv2-public.intel.com/772642/whitepaper-tee-io-device-guide-v0-6-5.pdf.
[38] Andrei Ivanov, Benjamin Rothenberger, Arnaud Dethise, Marco Canini, Torsten Hoefler, and Adrian Perrig. 2023. SAGE: Software-based Attestation for GPU Execution. In *Proceedings of the 2023 USENIX Annual Technical Conference*.
[39] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. 2019. Heterogeneous Isolated Execution for Commodity GPUs. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*.
[40] Jianyu Jiang, Ji Qi, Tianxiang Shen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Gong Zhang, Xiapu Luo, and Heming Cui. 2022. CRONUS: Fault-isolated, Secure and High-performance Heterogeneous Computing for Trusted Execution

Environment. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture*.
[41] Zhipeng Jiao, Hua Chen, Jingyi Feng, Xiaoyun Kuang, Yiwei Yang, Haoyuan Li, and Limin Fan. 2023. A Combined Countermeasure Against Side-Channel and Fault Attack with Threshold Implementation Technique. In *Chinese Journal of Electronics*.
[42] Antoine Joux. 2006. *Authentication Failures in NIST version of GCM.* Technical Report. National Institute of Standards and Technology. https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/joux_comments.pdf
[43] Mohamed Amine Khelif, Jordane Lorandel, Olivier Romain, Matthieu Regnery, Denis Baheux, and Guillaume Barbu. 2019. Toward a Hardware Man-in-the-Middle Attack on PCIe Bus for Smart Data Replay. In *Proceedings of the 22nd Euromicro Conference on Digital System Design*.
[44] Haohui Mai, Jiacheng Zhao, Christos Kozyrakis, Mingyu Gao, Hongren Zheng, Quanxi Li, Zibin Liu, Cong Wang, Huimin Cui, and Xiaobing Feng. 2023. Honeycomb: An Secure, Efficient GPU Execution Environment with Minimal TCB. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation*.
[45] Michael Larabel. 2017. NVIDIA Sends Out Signed Firmware Images For GP108 Pascal GPUs. https://www.phoronix.com/news/NVIDIA-GP108-Firmware.
[46] Microsoft. 2025. Azure OpenAI Service. https://cazure.microsoft.com/en-us/products/ai-services/openai-service.
[47] Microsoft. 2025. ND-H100-v5 sizes series. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/gpu-accelerated/ndh100v5-series?tabs=sizebasic.
[48] Rijoy Mukherjee, Sneha Swaroopa, and Rajat Subhra Chakraborty. 2024. Security Vulnerabilities in AI Hardware: Threats and Countermeasures. In *2024 IEEE 33rd Asian Test Symposium (ATS)*.
[49] NVIDIA. 2022. CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit.
[50] NVIDIA. 2022. NVIDIA CONFIDENTIAL COMPUTING. https://www.nvidia.com/en-us/data-center/solutions/confidential-computing/.
[51] NVIDIA. 2023. *Confidential Compute on NVIDIA Hopper H100.* Technical Report. NVIDIA Corporation. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf
[52] NVIDIA. 2023. Graphics Cards. https://www.nvidia.com/en-us/geforce/graphics-cards/.
[53] NVIDIA. 2024. NVIDIA Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.
[54] NVIDIA. 2024. NVIDIA NIM LLMs Benchmarking . https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html.
[55] NVIDIA. 2025. GeForce RTX 4090 Graphics Cards for Gaming. https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/.
[56] NVIDIA. 2025. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/a100/.
[57] NVIDIA. 2025. NVIDIA H100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/h100/.
[58] NVIDIA. 2025. NVIDIA Linux Open GPU Kernel Module Source. https://github.com/NVIDIA/open-gpu-kernel-modules.
[59] NVIDIA. 2025. NVIDIA T4 Tensor Core GPU for AI Inference. https://www.nvidia.com/en-us/data-center/tesla-t4/.
[60] OpenAI. 2025. ChatGPT-OpenAI. https://openai.com/chatgpt.
[61] OpenAI. 2025. DALL-E-2-OpenAI. https://openai.com/index/dall-e-2/.
[62] OpenAI. 2025. Sora - OpenAI. https://openai.com/index/sora.
[63] PCI-SIG. 2010. PCI Express Base Specification Revision 3.0. https://members.pcisig.com/wg/PCI-SIG/document/download/8265.
[64] PCI-SIG. 2017. PCI Express Base Specification Revision 4.0, Version 1.0. https://members.pcisig.com/wg/PCI-SIG/document/10912?downloadRevision=active5.
[65] PCI-SIG. 2019. PCI Express Base Specification Revision 5.0, Version 1.0. https://members.pcisig.com/wg/PCI-SIG/document/13005.
[66] PCI-SIG. 2020. IDE Security IP for PCIe 5.0. https://pcisig.com/sites/default/files/files/PCIe%20Security%20Webinar_Aug%202020_PDF.pdf.
[67] PCI-SIG. 2022. PCI Express Base Specification Revision 6.0, Version 1.0. https://members.pcisig.com/wg/PCI-SIG/document/16609.
[68] Wei Ren, William Kozlowski, Sandhya Koteshwara, Mengmei Ye, Hubertus Franke, and Deming Chen. 2023. AccShield: a New Trusted Execution Environment with Machine-Learning Accelerators. In *Proceedings of the 60th ACM/IEEE Design Automation Conference*.
[69] RyokoAI. 2025. ShareGPT52K. https://huggingface.co/datasets/RyokoAI/ShareGPT52K.
[70] Fan Sang, Jaehyuk Lee, Xiaokuan Zhang, and Taesoo Kim. 2025. PORTAL: Fast and Secure Device Access with Arm CCA for Modern Arm Mobile System-on-Chips (SoCs). In *Proceedings of the 46th IEEE Symposium on Security and Privacy*.
[71] Teven Le Scao, Angela Fan, Christopher Akiki, et al. 2022. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *arXiv preprint arXiv:2211.05100* (2022).
[72] Mert Side, Fan Yao, and Zhenkai Zhang. 2022. LockedDown: Exploiting Contention on Host-GPU PCIe Bus for Fun and Profit. In *Proceedings of the 7th IEEE European Symposium on Security and Privacy*.
[73] Siladitya Ray. 2023. Samsung Bans ChatGPT Among Employees After Sensitive Code Leak. https://www.forbes.com/sites/siladityaray/2023/05/02/samsung-bans-chatgpt-and-other-chatbots-for-employees-after-sensitive-code-leak/.
[74] K. Singhal, Shekoofeh Azizi, Tao Tu, Said Mahdavi, and Jason Wei. 2022. Large Language Models Encode Clinical Knowledge. *Nature* (2022).
[75] Supraja Sridhara, Andrin Bertschi, Benedict Schlüter, Mark Kuhne, Fabio Aliberti, and Shweta Shinde. 2023. ACAI: Protecting Accelerator Execution with Arm Confidential Computing Architecture. In *Proceedings of the 33rd USENIX Security Symposium*.
[76] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. 2021. Invisible Probe: Timing Attacks with PCIe Congestion Side-channel. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy*.
[77] Yifan Tan, Cheng Tan, Zeyu Mi, and Haibo Chen. 2025. Pipellm: Fast and Confidential Large Language Model Services with Speculative Pipelined Encryption. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*.
[78] Tenstorrent. 2025. Wormhole n150d. https://tenstorrent.com/en/hardware/wormhole.
[79] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open Foundation and Fine-tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023).
[80] tpm2-software community. 2019. *Remote Attestation with TPM2.* Trusted Computing Group. https://tpm2-software.github.io/tpm2-tss/getting-started/2019/12/18/Remote-Attestation.html#/
[81] Trusted Computing Group. 2019. *TCG Trusted Attestation Protocol (TAP) Information Model.* Technical Report. Trusted Computing Group. https://trustedcomputinggroup.org/wp-content/uploads/TNC_TAP_Information_Model_v1.00_r0.29A_publicreview.pdf
[82] Trusted Computing Group. 2025. *TPM 2.0 specification.* Trusted Computing Group. https://trustedcomputinggroup.org/resource/tpm-library-specification/
[83] Kapil Vaswani, Stavros Volos, Cedric Fournet, Antonio Nino Diaz, Ken Gordon, Balaji Vembu, Sam Webster, David Chisnall, Saurabh Kulkarni, Graham Cunningham, et al. 2023. Confidential Computing within an AI Accelerator. In *Proceedings of the 2023 USENIX Annual Technical Conference*.
[84] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*.
[85] Chenxu Wang, Fengwei Zhang, Yunjie Deng, Kevin Leach, Jiannong Cao, Zhenyu Ning, Shoumeng Yan, and Zhengyu He. 2024. CAGE: Complementing Arm CCA with GPU Extensions. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium*.
[86] Jinwen Wang, Ao Li, Haoran Li, Chenyang Lu, and Ning Zhang. 2022. RT-TEE: Real-time System Availability for Cyber-physical Systems using ARM TrustZone. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy*.
[87] Guanting Chen Xiao Bi, Deli Chen et al. 2024. DeepSeek LLM: Scaling Open-Source Language Models with Longtermism. *arXiv preprint arXiv:2401.02954* (2024).
[88] Xilinx. 2023. Xilinx DMA IP Reference drivers. https://github.com/Xilinx/dma_ip_drivers.
[89] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence?. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.
[90] Susan Zhang, Stephen Roller, Naman Goyal, et al. 2022. OPT: Open Pre-trained Transformer Language Models. *arXiv preprint arXiv:2205.01068* (2022).
[91] Mark Zhao, Mingyu Gao, and Christos Kozyrakis. 2021. ShEF: Shielded Enclaves for Cloud FPGAs. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.
[92] Yiran Zhao, Chaoqun Liu, Yue Deng, et al. 2025. Babel: Open Multilingual Large Language Models Serving Over 90% of Global Speakers. *arXiv preprint arXiv:2503.00865* (2025).
[93] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, et al. 2020. Enabling Rack-scale Confidential Computing using Heterogeneous Trusted Execution Environment. In *Proceedings of the 41st IEEE Symposium on Security and Privacy*.
[94] Jianwei Zhu, Hang Yin, et al. 2024. Confidential Computing on NVIDIA Hopper GPUs: A Performance Benchmark Study. *arXiv preprint arXiv:2409.03992* (2024).