sNPU: Trusted Execution Environments on Integrated NPUs

Erhu Feng[1†◇], Dahu Feng[1‡], Dong Du[†◇], Yubin Xia[†◇], Haibo Chen[†◇§]
[†]*Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University*
[‡]*Department of Precision Instrument, Tsinghua University*
[◇]*Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China*
[§]*Key Laboratory of System Software (Chinese Academy of Sciences)*

Abstract—**Trusted execution environment (TEE) promises strong security guarantee with hardware extensions for security-sensitive tasks. Due to its numerous benefits, TEE has gained widespread adoption, and extended from CPU-only TEEs to FPGA and GPU TEE systems. However, existing TEE systems exhibit inadequate and inefficient support for an emerging (and significant) processing unit, NPU. For instance, commercial TEE systems resort to coarse-grained and static protection approaches for NPUs, resulting in notable performance degradation (10%–20%), limited (or no) multitasking capabilities, and suboptimal resource utilization. In this paper, we present a secure NPU architecture, known as sNPU, which aims to mitigate vulnerabilities inherent to the design of NPU architectures. First, sNPU proposes *NPU Guarder* to enhance the NPU's access control. Second, sNPU defines new attack surfaces leveraging in-NPU structures like scratchpad and NoC, and designs *NPU Isolator* to guarantee the isolation of scratchpad and NoC routing. Third, our system introduces a trusted software module called *NPU Monitor* to minimize the software TCB. Our prototype, evaluated on FPGA, demonstrates that sNPU significantly mitigates the runtime costs associated with security checking (from upto 20% to 0%) while incurring less than 1% resource costs.**

I. INTRODUCTION

Trusted Execution Environment (TEE) is a trending topic in modern hardware-assisted security architectures. We have witnessed a plethora of TEE designs implemented across various Instruction Set Architectures (ISA), including Intel SGX [18] and TDX [42], AMD SEV [90], ARM TrustZone [4] and CCA [8], and RISC-V Penglai [26] and Keystone [54]. TEE provides an trusted execution environment with isolated hardware resources for secure tasks. Due to hardware security guarantees, attackers outside the TEE are incapable to compromise the secret in secure tasks.

Besides, with the evolution of AI applications such as LLM [11], [82], autonomous driving, and image recognition, there is a growing trend of offloading ML tasks to domain-specific accelerators known as Neural Processing Units (NPUs). To accelerate AI workloads, NPUs leverage specialized hardware structures such as matrix computation units (MCUs), scratchpads, and Networks-on-Chip (NoC). Due to the high demand for AI applications on mobile devices, current mobile chips [6], [15], [24], [28], [60], [86], [101], [107] also have integrated NPU cores within the SoC for improved energy efficiency and performance.

However, the integration of NPUs in SoCs introduces new attack surfaces that need to be considered in TEE systems. These attack surfaces can be categorized into three aspects: **(1)** ***Using a compromised NPU to attack CPU-side resources***: NPUs in mobile devices typically have access permissions to the CPU-side secure memory, which contains sensitive data such as personal facial features. If the NPU is compromised, a malicious NPU task can steal these secrets, posing a threat to CPU-side resources. **(2)** ***Internal attacks on the NPU***: Modern NPUs support concurrent execution for multiple tasks [17], [30], [49], [50], [53], [112]. However, the concurrent execution introduces a risk of stealing confidential models or data by leveraging internal NPU resources like scratchpad and NoC. Meanwhile, another team also disclosed a vulnerability leveraging the in-NPU structure called **LeftoverLocals** [98] at the same time. This vulnerability exploits the accelerator's local memory (scratchpad), to extract secret information from the model being processed. LeftoverLocals has been confirmed to affect platforms of Apple, AMD, and Qualcomm, indicating a widespread risk across various hardware implementations. **(3)** ***Using the CPU to attack the NPU***: CPU-side tasks can exploit NPU instructions or vulnerabilities [69]–[72] in the NPU driver to attack NPU-side tasks. To address these security concerns, a comprehensive TEE which combines both the CPU and NPU becomes crucial.

Some prior works have proposed coarse-grained and static TEE designs for CPU-NPU systems. One approach is temporarily designating the entire NPU as a secure device and migrating the whole NPU driver into the TEE. However, this solution results in severe underutilization of NPU resources, and a large TCB due to the complexity of the software stack. Other NPU TEEs [37], [38], [57], [58], [95] focus on the memory encryption and integrity protection. These approaches mainly consider physical attacks on DRAM (e.g., freezing memory), but lack protection for in-NPU structures, as the data inside NPU remains plaintext. Besides, TEE systems on other accelerators like GPU [21], [40], [45], [67], [80], [106] mainly focus on the protection for GPU global memory, CPU-GPU connection, etc. However, directly applying these mechanisms from GPU TEE to integrated NPUs is not enough.

[1]The two authors contributed equally to this work and should be considered co-first authors.

First, isolation mechanisms like IOMMU adopted in the GPU TEE are not efficient for integrated NPUs, as NPU requires larger memory bandwidth. Second, NPU has specialized hardware structures like scratchpads and NoCs, which bring new attack surfaces. Therefore, an NPU TEE should address potential attacks mentioned above and meet two fundamental requirements: *minimizing runtime performance overhead* and *achieving higher resource utilization*.

In this paper, we present sNPU, a TEE system for integrated NPUs, which addresses above NPU-related attacks through three novel designs. First, to defend against attacks leveraging the NPU's external behavior (e.g., memory access), we introduce a tile-based memory translation and checking unit, specifically designed to accommodate the characteristics of NPU memory access pattern. This design incurs (almost) zero runtime overhead while saving the checking energy. Second, to address the new attack surface utilizing in-NPU structures like scratchpad and NoC, sNPU leverages the observation that scratchpad has no association with the main memory, and can employ a more fine-grained and dynamic isolation mechanism for it. Meanwhile, sNPU also incorporates an NoC isolation mechanism with the offline route checking, ensuring the integrity of the NoC network. Third, to minimize potential attacks from the malicious software in the CPU side, sNPU reduces the software TCB of NPU stack. sNPU introduces an NPU Monitor within the secure world solely for essential security checks. Meanwhile, other software components like the AI framework and NPU driver can remain untrusted.

We have implemented a prototype of sNPU with Chipyard [5], which is a customized RISC-V SoC generator including the CPU, NPU (e.g., Gemmini [29] and NVDLA [79]) and other components in SoC. We have extended the security features of sNPU in the Gemmini with the NoC extension. To establish a comprehensive TEE system, we integrate the sNPU with the CPU side TEE, which divides hardware resources into the normal and secure worlds. Evaluation results show that the design of sNPU has no impact on NPU runtime performance (e.g., DMA, Scratchpad, and NoC) across various AI models. Additionally, security extensions in sNPU do not compromise NPU resource utilization for both secure and non-secure tasks. The extra hardware cost of these extensions is also minimal, estimated at less than 1%.

II. BACKGROUND AND RELATED WORK

A. Trusted Execution Environment (TEE)

TEE has been widely used in the contemporary computer architecture. CPU vendors have introduced their own TEE architectures, such as Intel SGX and TDX [18], [42], ARM TrustZone and CCA [4], [8], AMD SEV [90], RISC-V Penglai, CURE, and Keystone [9], [26], [54], and others [10], [19], [27], [89], [116]. In mobile systems, TrustZone serves as a mainstream implementation of TEE. It leverages the concept of *secure partition*, which effectively segregates hardware resources into different partitions, such as the normal world and the secure world. In this architectural design, no trust is placed in any software executing within the normal world, including untrusted operating systems and applications. A privileged software monitor (such as ARM EL3 or RISC-V M mode) oversees the secure partition, managing all hardware resources. Our paper builds upon this TEE design, extending its scope to include the NPU, thus, trusted AI workloads can harness secure hardware resources within the secure partition, including the CPU, NPU, and memory.

There are several other TEE designs like *Enclave* (e.g., SGX) and *CVM* (e.g., SEV, CCA, TDX). SGX provides a user-level TEE abstraction called Enclave. It can work together with untrusted user applications but possess strong isolation from them. Untrusted applications and kernels are unable to access the private memory of an Enclave. SEV, TDX, and CCA represent the CVM design, they leverage the secure processor or the lightweight CVM module to control all security-sensitive resources for VMs. Therefore, CVM can distrust the hypervisor (or VMM). Although Enclave and CVM provide more flexible TEE abstractions, the underlying mechanism still relies on the secure partition — Enclave and CVM also need to define which hardware resource is secure or not. Extending our NPU TEE design to incorporate Enclave or CVM remains an area for future work.

B. Neural Processing Unit (NPU)

NPUs [15], [24], [28], [29], [47], [60], [79], [101], [107] are specialized hardware accelerators that excel at performing neural network computations efficiently. In contrast to general-purpose processors, NPUs are specifically optimized to meet the unique requirements of neural networks. They offer capabilities such as massive parallelism, high-speed data processing and matrix computation. These features are essential for handling intensive computations involved in deep learning algorithms.

To meet these requirements, NPU adopts several special hardware structures. One key unit for the modern NPU is the matrix calculation unit (MCU) like tube [13], [14], [60], [91], systolic array [29], [35], [77], etc. With these matrix units, NPU can execute the matrix calculation like multiplication and convolution within one operation. Some NPUs also have other dedicated units for some specialized operations like sparse matrix calculation, activation function, etc.

Besides the matrix unit, NPUs often adopt a near data computing (NDC) architecture [29], [65], [73], [79] to minimize data retrieval overhead. For example, weights in neural networks are pre-stored in the SRAM/scratchpad near the matrix unit, allowing for quick access during computations. This reduces latency and energy consumption by eliminating the need to retrieve weights from main memory for each task. NDC optimizes data flow and improves computational efficiency by minimizing memory access bottlenecks.

Furthermore, NPUs also leverage multi-core architectures with a Network-on-Chip (NoC) network [35], [50], [65], [91], [117] to further parallelize data computation. The NoC network allows for direct data transfer among NPU cores without the need for additional memory load/store instructions.

Developers can map different layers of neural network into the different NPU cores and gain the performance benefits.

Integrated NPU v.s. Discrete NPU: An integrated NPU [15], [24], [28], [29], [47], [60], [76], [79], [101], [107] refers to a specialized hardware component that is built directly into an SoC (or processor). In this configuration, the NPU is tightly integrated with other components of the chip. On the other hand, a discrete NPU is a standalone hardware component that is separate from the main SoC. Compared with the discrete NPU, the integrated NPU has three main benefits: First, integrated NPUs can share the system cache with a unified address space, eliminating the need for additional memory transfers or encryption between the CPU and discrete NPU. Second, the integrated NPU can achieve better bandwidth and lower latency for accessing the system memory, enhancing overall performance. Third, the integrated design of NPUs significantly reduces energy consumption and chip size compared to discrete NPUs. Therefore, almost all mobile SoC adopts an integrated NPU design, facilitating the flourishing of AI workloads running on local devices. This paper focuses on integrated NPUs, and our techniques can also be applied to discrete NPUs.

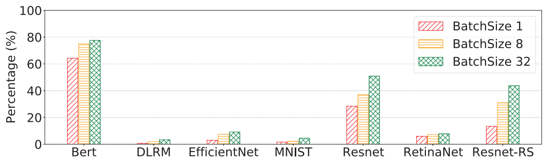


Fig. 1. **Overall FLOPS utilization of different inference workloads**.

C. Multi-tasking Requirements for NPUs

Low NPU utilization for a single ML workload: The latest NPUs [3], [32], [47] have adopted powerful computing capability (>100 TFLOPs) as well as large memory capacity (>100MB SRAM and >50GB HBM), which facilitate the support of multitasking requirements. We analyze the NPU utilization when running different ML workloads on the TPU [47], [112], as shown in Figure 1. Most ML workloads utilize less than 50% of the computational resource available in the TPU core. This underutilization is primarily attributed to temporal idleness of MCU and the inefficient use of memory bandwidth. To optimize the NPU's utilization, recent studies [49], [50], [112] have proposed the simultaneous execution of multiple ML tasks on a single NPU, ensuring that the service-level agreements (SLAs) of tasks are not compromised.

Simultaneous execution of both secure and non-secure tasks: The simultaneous execution of secure and non-secure tasks has become a topic of heightened interest, particularly within mobile devices [30] and autonomous vehicles [31]. For instance, developers may need to concurrently run a confidential model alongside other models that have been developed by potentially untrusted entities. Currently, smartphone vendors [94] tend to address this requirement by statically partitioning the computational resources of the NPU, like dedicating one NPU core to secure tasks while allocating another core for non-secure tasks. Furthermore, the simultaneous execution of secure and non-secure tasks is also evident in the field of autonomous driving. Here, secure tasks such as the occupancy network may be executed simultaneously with non-secure tasks, such as in-vehicle entertainment.

D. Limitation of Current NPU TEEs

The hardware-assisted security for NPU is still an emerging topic, and current solutions face limitations in achieving fine-grained isolation for in-NPU resources and multi-task supporting (both secure and non-secure). ITX [105] proposes confidential computing for AI accelerators. However, ITX specifically targets the Graphcore Intelligence Processing Unit (IPU), which is a separate AI accelerator located on the main board. The main focus of ITX is on establishing attestation between users and ITX, as well as secure transfer of secret data and models. TNPU [57] and other NPU TEEs [37], [38], [58], [95] focus on the memory encryption and integrity protection, and minimize overhead using the specific memory access pattern in NPU. In our work, we concentrate on the isolation of in-NPU structures like the scratchpad and NoC. The data in these structures remain plaintext even when using memory encryption, as introduced in prior works. In a multitasking scenario, a non-secure task may run concurrently with a secure task and potentially steal the secret model from the scratchpad [97], [98]. Therefore, our design is complementary to encryption-based NPU TEEs.

The industry currently adopts a coarse-grained NPU TEE design, we called **TrustZone NPU**. For example, a smartphone vendor extends the sMMU of the NPU with the TrustZone extension. Specifically, an additional secure bit is used in the sMMU to indicate whether the corresponding NPU is a secure device or not. However, this design only allows for switching the entire NPU into the secure world and does not support fine-grained and dynamic isolation for in-NPU resources. Furthermore, the TrustZone NPU design requires migrating the entire NPU driver into the TEEOS with a larger TCB and clearing all sensitive NPU context during mode switching. Due to the limitations mentioned above, the TrustZone NPU experiences poor resource utilization and performance, as demonstrated in our evaluation (see in §VI-B and §VI-C).

In this paper, we present a novel TEE system tailored for integrated NPUs. Our design achieves strong and (almost) zero-cost isolation for NPU-specific hardware structures such as scratchpad and NoC. By providing fine-grained isolation within the NPU, our design enables the concurrent execution of multiple neural network tasks, both secure and non-secure, within the same NPU. Furthermore, the isolation mechanism ensures that resource utilization is not compromised, allowing the NPU driver to adopt arbitrary allocation strategies based on task requirements. In summary, we introduce a fine-grained and dynamic isolation mechanism specifically designed for integrated NPUs, while supporting multi-tasking and maximizing resource utilization.

III. Design Overview

A. Goals

- **Utilization:** Modern NPUs support the concurrent execution of multi-networks to enhance resource utilization. Therefore, our design should also support the multi-tasking for both secure and non-secure tasks, and achieve fine-grained resource isolation.
- **Security:** Our design should consider the security of the NPU from three aspects. First, malicious users cannot leverage NPU to access the sensitive data in the system. Second, non-secure ML tasks cannot steal the secret model or data of secure ML tasks in the NPU. Third, attackers cannot leverage the NPU software running in the CPU side to attack ML tasks running in the NPU.
- **Performance:** Our design should not impact the performance of non-secure NPU tasks, and almost zero-cost for secure tasks.

B. Threat Model

The TCB of our system contains: the secure hardware (i.e., secure CPU core, sNPU hardware extension, memory protection engine, etc.), the most privileged mode monitor (e.g., EL3 in ARM, M mode in RISC-V) and software running in the secure world. We do not place trust in hardware and software components in the normal world like the NPU driver, scheduler and ML framework, etc. Moreover, we do not trust the NPU compiler as it is known to be fragile [97].

We also assume that both secure and non-secure tasks can run simultaneously to satisfy the demands of multi-tasking and SLA (service-level agreement) requirements. As NPU resources might be shared both **spatially** and **temporally** by these concurrent NPU tasks, a secure task must be wary of attacks from other tasks operating on the same NPU that exploit vulnerabilities in the NPU's internal structures. For instance, a confidential ML task must guard against model leakage through the NPU's SRAM [98].

sNPU does not consider the side channel attacks and physical attacks like freezing memory, bus snooping, etc. These attacks have been well studied in prior researches [37], [44], [56], [61], [81], [93], [95], [108], and our sNPU design complements these previous efforts.

IV. Detailed Design

We introduce sNPU, a comprehensive and secure NPU architecture designed to mitigate attacks and vulnerabilities targeting both NPU hardware and software. sNPU consists of three essential components: NPU Guarder, NPU Isolator, and NPU Monitor. NPU Guarder serves as a lightweight memory translation/access guard, effectively preventing unauthorized access attempts from the NPU. NPU Isolator focuses on lightweight inner resource isolation mechanisms tailored to NPU-specific architectures like scratchpad and NoC. NPU Monitor is a lightweight trusted software module dedicated to perform essential security checks.

A. NPU Guarder: Memory Access Guarder for NPU

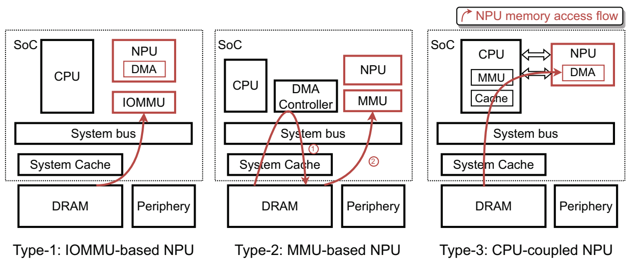


Fig. 2. **Different types of integrated NPUs:** Type-1 NPU leverages an integrated DMA engine to retrieve the data. Type-2 NPU relies on a system DMA engine for data copy, and then uses ld/st instructions. Type-3 NPU reuses the memory access capability in the CPU side.

Challenges of current NPU access control: Access control is not new but encounters several new challenges for integrated NPUs. First, integrated NPUs have different memory access paths, which may complicate the design of a unified access controller. Second, the NPU also requires a higher memory bandwidth, which needs a more efficient checking logic. Figure 2 illustrates different types of integrated NPUs, including IOMMU-based NPUs [51], [78], [79], [101], MMU-based NPUs [21], [86], and CPU-coupled NPUs [29]. The first two types of NPUs are MMIO devices, with one utilizing DMA for system memory access and the other employing ld/st instructions. The third type is coupled with the CPU core, allowing it to access the CPU cache. To prevent arbitrary memory access, some NPUs (Type-1 and Type-2) utilize a separate IOMMU/MMU to restrict NPU access, while others (Type-3) rely on the access checking mechanism on the CPU side. There is no unified memory access controller for integrated NPUs, which increases design complexity. Besides, IOMMU/MMU mechanisms introduce non-negligible overhead for NPUs due to page table walking, TLB ping-pong, etc. This overhead can result up to 15% to 20% performance loss for real-world applications [29].

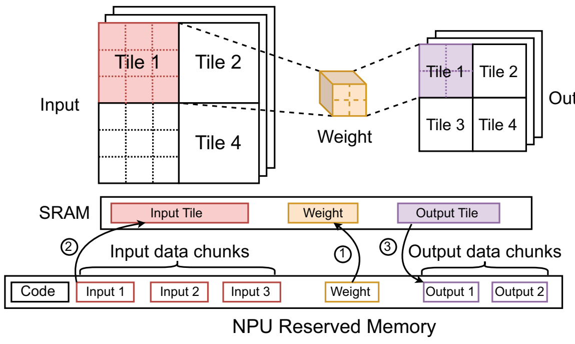


Fig. 3. **The memory access patterns of NPU:** During a single NPU calculation, only limited tiles of data are required or generated.

NPU memory access pattern: Unlike general-purpose units, the NPU is a domain-specific accelerator designed specifically for ML tasks. ML tasks typically involve multiple layers, each consisting of numerous matrix computations complemented by activation functions. For a single task, NPUs require limited types of data sets: the input data, weight (including bias) as well as the output result, as shown in Figure 3.

NPUs utilize Direct Memory Access (DMA) [109] to fetch data from the system DRAM/HBM into the in-NPU SRAM (i.e., scratchpad). The NPU driver employs a specific memory allocator to manage these DMA buffers (NPU reserved memory). For instance, Android/Linux introduces the ION heap [21], [63], NVIDIA Tegra utilizes NVMA [78] and Qualcomm MSM employs PMEM [62] as memory allocators designed for NPU's memory, separate from the system memory. To reserve the contiguous DMA buffer, NPU drivers either pin the memory during system boot or use Contiguous Memory Allocator (CMA) [102]. Before offloading tasks to the NPU, the NPU driver needs to allocate several chunks in the NPU-reserved memory, and the NPU will further partition each chunk into several tiles. During execution, the NPU initiates computational sequences by loading weights from the system memory into the SRAM. Subsequently, it retrieves input values either directly from the system memory or from the SRAM that retains the output result from the preceding layer's computation. Upon completing the calculation, the NPU stores the output result back to the system memory. In summary, the NPU only needs to load and store limited tiles of data (input, weight, and output) during one calculation, and the VA to PA mappings are consistent for each tile (belongs to a data chunk). This memory access and allocation pattern can be leveraged to overcome the challenge of NPU's memory access control.

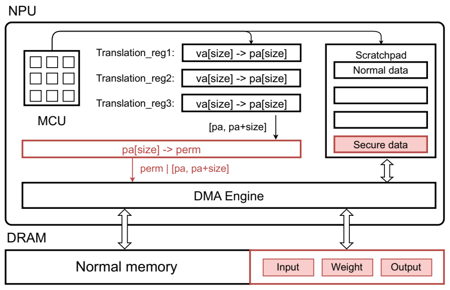


Fig. 4. **Lightweight address translation and checking in the NPU core**.

NPU Guarder: Figure 4 illustrates our NPU Guarder design, a lightweight memory access controller leveraging the specific memory access pattern in NPUs. It has two main benefits compared with MMU or IOMMU: (1) it has a lightweight design without checking overhead, and (2) it can be integrated inside the NPU.

First, to eliminate the runtime overhead of traditional paging-based memory access control, NPU Guarder employs coarse-grained memory checking and fine-grained translation mechanisms. For memory checking, it utilizes a checking register that records the access permission of a contiguous memory region, as sensitive data in the mobile system is typically stored in a pre-allocated secure memory region (e.g., TrustZone secure memory area). As for the address translation, NPU Guarder provides fine-grained translation registers in the tile level (e.g., input tile and output tile). Each translation register maps a specific region from virtual address to the corresponding physical address. Unlike the checking register, which is rarely modified, the translation registers may be updated before the NPU calculation (if needed).

Second, NPU Guarder integrates these checking and translation registers inside the NPU core, positioned before the DMA engine. Compared with a standalone module (e.g., IOMMU), the integration design reduces the complexity in the SoC. Furthermore, since the memory checking and translation are performed at the DMA-request level rather than the memory-packet level, NPU Guarder can save additional energy compared to IOMMU. When a DMA request is received, the DMA engine divides it into multiple fixed-size memory packets (e.g., 64 bytes). Therefore, NPU Guarder only checks for one time (and saves energy), while IOMMU needs to check O(N) times at the memory-packet level.

B. NPU Isolator: Inner Resource Isolation for NPU

Specialized hardware structures in NPU: To optimize AI workloads, NPUs incorporate specialized hardware structures such as scratchpads and NoCs (Network on Chip). The scratchpad is a high-speed, low-latency SRAM that requires explicit management by the programmer. As NPU needs a larger memory bandwidth, the scratchpad/SRAM has been widely used in modern NPU [32], [41], [47], [66] to accelerate AI workloads [20], [33]. Unlike caches, the scratchpad holds data that can only be accessed by its index (not the global address), without any association with the system memory. It does not include mechanisms like hits or misses, and it does not maintain the memory coherence. The NoC is the on-chip network that connects multiple NPU cores. NoC is indispensable for the multi-core NPUs [32], [47], [50], [66], [92], as it enables scalable computing resources while addressing the issue of unscalable memory bandwidth through interconnection. Most NoC networks utilize a package-based protocol [115]. A package typically consists of a head flit, several body flits, and a tail flit. The head flit contains route information, specifying the path between the source and target cores (e.g., x:+4, y:+2) [65], [85].

Scratchpad and NoC are specialized structures in the NPU to accelerate AI workloads, but they also introduce new vulnerabilities.

Lack of protecion for the scratchpad: As scratchpad is explicitly managed by the programmer, a compromised compiler or NPU driver can easily issue attacks on it. For example, the NPU driver can allocate a scratchpad entry which is already used by another task, and the NPU compiler can forcibly read the content in the scratchpad without write before. A recently disclosed vulnerability: **LeftoverLocal** [97] allows recovery of data from accelerator's local memory (scratchpad) created by the victim process. A real-world PoC has been developed, that attacker can listen into another user's interactive LLM session (e.g., llama.cpp). Due to the lack of underlying hardware

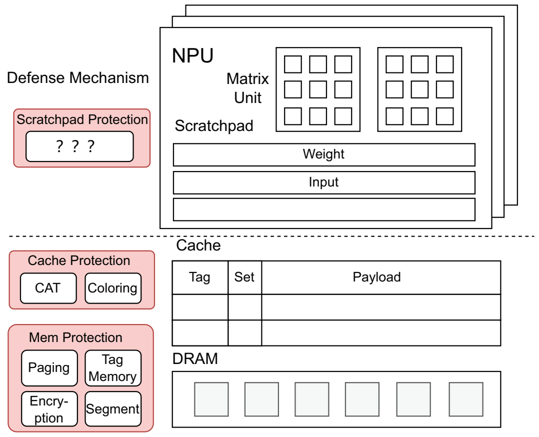


Fig. 5. **Existing protection mechanisms for hierarchical memory system:** Lack a specific protection mechanism for the scratchpad.

isolation support, LeftoverLocals has a widespread impact on various platforms, includes Apple, Qualcomm, AMD, and Imagination GPUs.

We have conducted a detailed research on existing memory protection mechanisms, but these mechanisms are not suitable for the scratchpad structure. Figure 5 illustrates the existing protection mechanisms for cache and memory. For instance, CAT [44] and page coloring [108] are employed to mitigate cache side channel attacks. Paging, tag memory [23], and segment mechanisms [111] are used for memory isolation. However, these mechanisms usually introduce non-trivial overhead, which are not suitable for the scratchpad due to its high-bandwidth requirement.

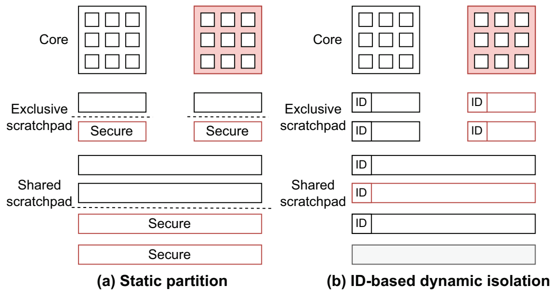


Fig. 6. **Different isolation mechanisms for scratchpads:** Figure (a) illustrates the static scratchpad partition. Figure (b) demonstrates the ID-based and fine-grained scratchpad isolation.

TABLE I
DIFFERENT ISOLATION MECHANISMS FOR SCRATCHPAD.

Isolation mechanism	Sharing Model		Metric		
	Temporal	Spatial	Utilization	Performance	SLA
Partition	Yes	Yes	Low	Low	Good
Flush (coarse-grained)	Yes	No	Low	Good	Poor
Flush (fined-grained)	Yes	No	Low	Low	Good
sNPU	Yes	Yes	High	Good	Good

Strawman solution: Table I presents a comparison of various isolation mechanisms for the scratchpad. Flushing all contents in the scratchpad before task scheduling is a straightforward approach but comes with several inherent limitations. For instance, flushing is not just zeroing out the contents in the scratchpad but needs to save and restore the task's context before scheduling. The current scheduling granularity for integrated NPUs is at the op-kernel level [12], [68], and the NPU must perform fine-grained heterogeneous computing with the CPU, due to the limited operators. However, this fine-grained flushing granularity can cause a significant performance overhead (larger than 25% in our evaluation and other works [112]). What's more, flushing only works when the scratchpad is temporally shared among different tasks, while current ML tasks tend to assume that scratchpads are spatially shared to achieve better performance. Even when considering only time-sharing, the granularity of flushing becomes a trade-off between performance and compliance with the service-level agreement (SLA) [25], [110] of ML tasks. Frequent flushing will cause a considerable performance overhead as mentioned before, whereas flushing at a coarser granularity might fail to meet SLA requirements with low resource utilization (NPU is waiting for the CPU computing). This dilemma has been corroborated by multiple studies [49], [50], [112].

Another straightforward solution is to partition scratchpads into several regions, as shown in Figure 6(a). The partition mechanism has already been employed in CPU and GPU TEEs, like ARM TrustZone [4], NVIDIA MIG [80]. However, such mechanism suffers from fragmentation issues, making it difficult to adjust the isolation boundary [59] at runtime. Moreover, partition also introduces a non-trivial overhead in real-world NPU tasks due to the low resource utilization (see in §VI-C).

ID-based scratchpad isolation: In NPU Isolator, we propose an ID-based isolation mechanism specifically for scratchpads, as shown in Figure 6(b). The key insight is that there is ***no address association between the scratchpad entry and system memory***, allowing us to store data in ***any*** scratchpad entries. Therefore, scratchpad can adopt a more fine-grained and dynamic isolation mechanism than cache and memory. We first add ID states for NPU cores and scratchpads, with the value 1 for secure and 0 for non-secure. Setting the ID state of the NPU core can only be done through a secure instruction. As for scratchpads, we define two access rules: (1) For exclusive scratchpad: we prohibit any read operations from a NPU core to a scratchpad entry that has a different ID state, while we do allow a NPU core forcibly writing data to a scratchpad entry, and update the scratchpad ID state to the NPU core's automatically. (2) For shared scratchpad: we prohibit non-secure NPU cores from accessing (both read and write) secure scratchpad entries, otherwise, a secure NPU core forcibly sets the accessed scratchpad's ID state to secure. A dedicated secure instruction is used to reset the scratchpad state from secure to non-secure. By enforcing these rules, we can achieve fine-grained and dynamic isolation for each scratchpad entry, and decouple the allocation strategy from its security check. It allows the NPU driver to reuse any allocation strategies to achieve better scratchpad utilization. Furthermore, since each scratchpad entry has a large payload (e.g., $\geq$128b), the resource overhead of one-bit ID state is negligible ($<$1%).

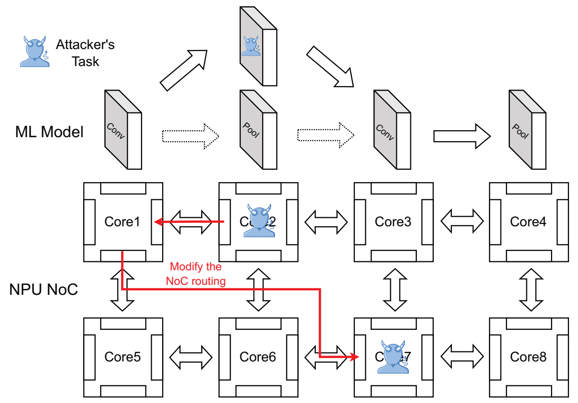


Fig. 7. **NoC attacks:** Tamper with the NoC route to hijack the data flow of the victim's NPU task.

Lack of the NoC isolation and integrity protection: In addition to protecting the scratchpad, NPU Isolator also ensures the isolation and integrity of the NoC network. NPUs can accelerate ML tasks by orchestrating different levels of the ML model across multiple NPU cores, allowing to transfer the intermediate result through the NoC network directly. It significantly reduces the overhead of storing and reloading data from memory. However, this usage of the NoC network introduces a new attack surface by breaking the route integrity, as depicted in Figure 7. Route integrity cannot be protected in the same way as code integrity adopted in the CPU TEE. For instance, if the NPU scheduler is compromised, it can schedule a malicious task to a wrong NPU core. Thus the attacker can either intercept secret intermediate results transmitted from the source core, or send malicious NoC packets to the victim core. By tampering with the route integrity of the NoC network, attackers can manipulate the entire ML tasks.

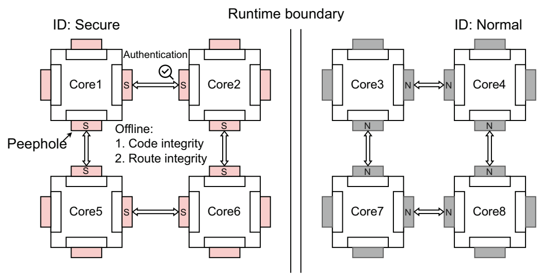


Fig. 8. **NoC protection:** Peephole mechanism along with the code and route integrity guarantees.

NoC isolation: To defend against above attacks on the NoC network, NPU Isolator introduces a lightweight NoC authentication mechanism called *peephole* with the offline route integrity check, as shown in Figure 8. The peephole mechanism generates an identity for the NoC packet (head flit) at the source core, which travels in the NoC network. When the target core receives this NoC packet, the peephole in target core authenticates this NoC request based on its identity. The NPU core's ID state can be used as an effective identity in the peephole mechanism. For example, the packet originating from a secure NPU core includes a secure bit as its identity. If the target core is a normal NPU core, the authentication process fails and this NoC request will be rejected. Otherwise, a secure NPU core will accept this request and receive the following packet (body flits).

Route integrity: Peephole mechanism provides isolation in the NoC network, however, to ensure a comprehensive NoC protection, we also need to consider the route integrity for ML tasks running on the multiple NPU cores. Besides the code integrity check which calculates the hash of ML task's code and compares it with the expected measurement, the route integrity check ensures the actual NoC routing aligns with the user's expectation. For instance, in the case of secure ML tasks requiring a 2×2 NoC network, a malicious NPU driver may allocate 1×4 NPU cores for these tasks, which may cause ML tasks interacting with two unexpected NPU cores during the execution. Therefore, before loading ML tasks to the multiple NPU cores, we need to verify whether the actual allocation of NPU cores matches the expected NoC configuration defined in the secure tasks. We will introduce more details in §IV-C.

C. NPU Monitor: Trusted Software Module for NPU

The NPU encompasses a large software stack consisting of various components such as the AI framework (e.g., TensorFlow [1], PyTorch [83]), compiler (e.g., TVM [12], CANN [88]), and NPU driver. Inclusion of the entire software stack within the TCB can introduce potential vulnerabilities and security risks. To mitigate this, we adhere to the design principle [99] of decoupling security from strategy, and only move a small monitor into the secure world. This monitor is responsible for performing security checks, managing critical resources, and acting as a bridge between the secure CPU and NPU.

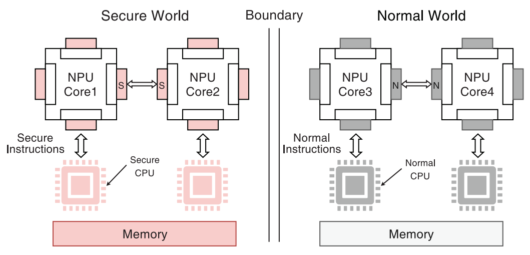


Fig. 9. **A comprehensive TEE system including CPU, NPU and memory.**

Interaction between NPU and CPU: Figure 9 illustrates the entire TEE system, comprising the CPU, NPU, and memory. There are two common methods for interaction between the NPU and CPU: the Memory-Mapped I/O (MMIO) interface and specific instructions (e.g., matrix extensions [7], [43], [74], [76] or RoCC [16]). To establish a complete trusted environment, sNPU restricts the CPU-NPU interactions to only allow the secure CPU to interact with the secure NPU. Specifically, the secure context (e.g., ID states and checking registers) can only be set by the secure CPU, utilizing new instructions dedicated to the sNPU design.

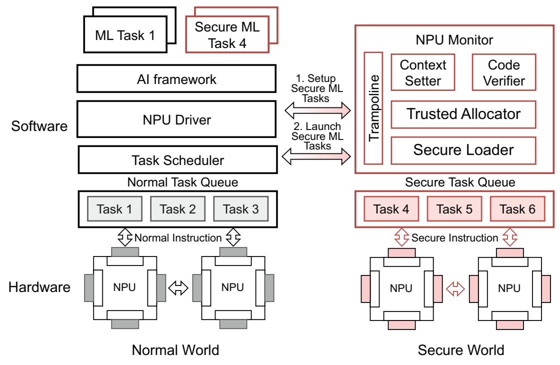


Fig. 10. **NPU Monitor:** A lightweight trusted software module responsible for critical security checks for secure NPU tasks.

NPU Monitor: The NPU Monitor plays a critical role in ensuring the confidentiality and integrity of multiple secure ML tasks. It consists of several shim modules: the context setter, trusted allocator, code verifier, and secure loader. Notably, the NPU Monitor only works for secure ML tasks. While for non-secure tasks, we do not apply any software checks and rely only on the hardware mechanisms to guarantee the isolation between secure and non-secure tasks.

Context setter is responsible for setting the NPU secure context, which includes NPU's ID state, checking and translation registers for secure tasks. The NPU context determines the hardware resources that the NPU can access, such as system memory and scratchpad.

Trusted allocator is responsible for allocating memory buffers in the reserved secure memory like input/output data and model of secure tasks. It also checks that there is no overlap for the scratchpad.

Code verifier first loads the code and sensitive model of the secure task into the secure task queue. It then calculates and verifies the measurement of the task code against the user's expectation.

Secure loader first guarantees the route integrity of the ML task. Unlike traditional CPU TEEs, a ML task may utilize multiple NPU cores connected with the NoC network. Secure loader verifies whether scheduled NPU cores match the topology of the expected NoC network. After verifying the route integrity, secure loader uploads the ML task into corresponding NPU cores.

In addition to the shim modules, there are two auxiliary components for NPU Monitor: the trampoline and the secure task queue. The trampoline serves as the intermediary (for data transferring) between the non-secure NPU driver and NPU Monitor, while secure task queue stores secure NPU tasks for scheduling.

Secure boot: The secure boot flow of sNPU is similar to prior works [4], [21], [57]. During the machine boot, the secure CPU verifies a minimal code of the trusted loader, which then loads and verifies the trusted firmware [84], [104]. The trusted firmware further loads and verifies software in the trusted world, such as TEEOS [103] and NPU Monitor, before loading the software running in the normal world. The Root-of-Trust for this secure boot chain remains in the SoC.

V. IMPLEMENTATION

NPU Guarder: Figure 11 depicts the microarchitecture of the NPU Guarder, emphasizing the key distinctions between the original design. In the NPU Guarder, a secure controller configures both checking and translation registers. The checking register comprises two primary fields: memory range and its authority (read, write, etc.). This tile-based mechanism is particularly well-suited for handling contiguous data, a common scenario in NPU. The translation register maintains a range mapping from physical addresses to virtual addresses. When a DMA request arrives, it translates the requested virtual address to the corresponding physical address according to the translation register. Subsequently, the physical address undergoes the permission check, and only authenticated requests are proceed to the DMA engine.

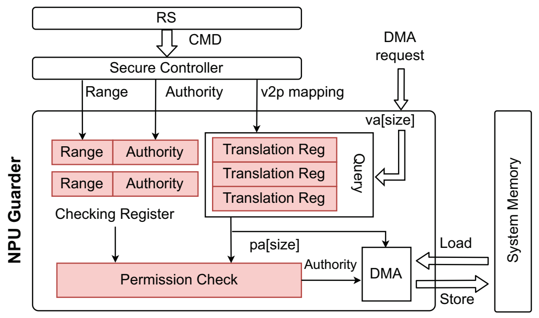


Fig. 11. **Microarchitecture of NPU Guarder:** It will translate and check the access permission for each DMA request.

NPU Isolator: The NPU Isolator contains two parts, the scratchpad isolation and NoC isolation. To achieve dynamic isolation for scratchpad, we only make slight extensions to the scratchpad interface and wordline. In addition to regular data, each scratchpad line now includes additional bits for the ID state. Considering that each wordline contains a large data block (128 bits for input/output scratchpad and 512 bits for accumulation scratchpad), the increase of one to two bits is negligible. In the local scratchpad, write operations are unrestricted and will overwrite the ID state. However, read operations are only allowed when the ID state of the wordline matches the state of the NPU. For the global scratchpad, both read and write operations are restricted, forbidding normal NPU cores from accessing secure wordline.

To support the peephole-based isolation for NoC, we redesign the router controller. Each NPU core possesses its own router controller, comprising send and receive engines, as shown in Figure 12. When the send engine receives a sending request, the router transitions from an idle state to a peephole state and generates the corresponding authentication ID. Subsequently, it triggers an authentication request to the destination core and awaits the return packet. If the authentication check passes, the router proceeds to transmit the entire data to the destination core. The receive engine has a mirrored working flow. Upon receiving the authentication request, the router verifies the authentication ID and waits for the subsequent data packet. Once all data is received and written to the scratchpad, the router transitions back to the idle state. Notably, authentication occurs only once. After verified, the router map locks, preventing other cores from using this channel.

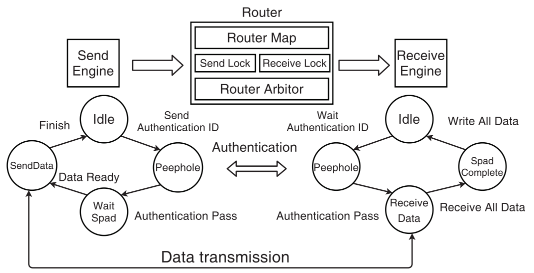


Fig. 12. **Secure protocol of sNPU router controller**.

NPU Monitor: We have implemented the NPU Monitor within a secure domain using PMP protection [96] in RISC-V CPUs. The primary code in the NPU Monitor is the cryptographic functions likes model decryption and code integrity measurement. Besides, we also develop a trusted allocator which can efficiently allocate memory slots of specific sizes (e.g., scratchpad size) in the secure memory. The context setter and secure loader guarantees the correctness of the NPU secure context and NoC route, respectively. To facilitate communication with software in the non-secure domain, we have designed a trampoline protocol that includes the function ID, arguments, and shared memory.

VI. EVALUATION

A. Experimental Setup

We implement the hardware prototype of sNPU on top of Chipyard [5], which is a customized RISC-V SoC generator designed for evaluating full-system hardware. The microarchitecture of the NPU design refers to Gemmini [29] and AuRORA [50], a systolic-array-based DNN accelerator. We have incorporated all security extensions of sNPU into the Gemmini, and further implement an in-body send/receive engine and router module to support NoC-based multi-core NPUs. For the CPU side TEE, our implementation is based on the Penglai [26], a RISC-V TEE system. We have extended the existing Penglai TEE by dividing all hardware resources into normal and secure worlds. This division ensures strong isolation for hardware resources between these two worlds, enhancing the security of the whole system. We evaluate the sNPU performance by running end-to-end DNN workloads using FireSim [48], a cycle-exact, FPGA-accelerated RTL simulator. The configuration is shown in Table II.

In our evaluations, we choose six different state-of-the-art DNN inference models including GoogleNet [100], AlexNet [52], YOLO-lite [39], MobileNet [36], ResNet [34], and Bert [22]. These DNNs include neural networks for mainstream computer vision (CV) and natural language processing (NLP), with different model sizes, DNN kernel types, computational and memory requirements.

Prerequisite: In our evaluation, we exclude the offline overhead, such as the task copy and model decryption. These operations can be performed in advance and do not impact the runtime performance. Furthermore, the modern mobile SoC [51], [94] supports to transfer the device's data (e.g., camera) directly to the secure memory. Thus, there is no additional copy for loading the sensitive input data. In this paper, we mainly focus on the runtime performance analysis between the sNPU with other Comparative systems.

Comparative Systems:

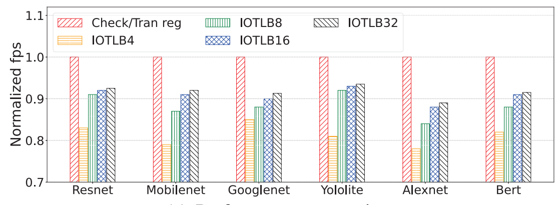
- **Normal NPU:** Normal NPU refers to the baseline in our evaluation without any secure protection mechanisms.
- **TrustZone NPU:** TrustZone is the most widely used TEE system in mobile devices. To support an isolated NPU core in TrustZone, current SoC vendors have enhanced the NPU's sMMU/IOMMU with the TrustZone extension (S/NS bit), and migrated the whole NPU driver inside the secure world. In the evaluation, we add extra protection mechanisms (e.g., flush-reload and partitioning) for internal scratchpads on the TrustZone NPU to meet the multi-tasking requirement.
- **sNPU:** sNPU incorporates the NPU Guarder and Isolator as additional security modules. We will compare the efficiency of sNPU's isolation mechanisms with previously mentioned methods.

TABLE II
SoC CONFIGURATIONS USED IN THE EVALUATION

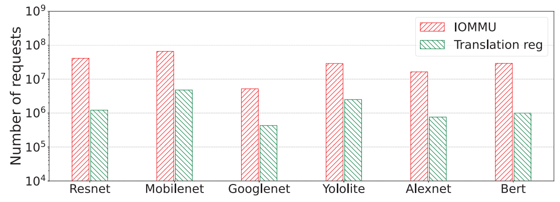
Parameter	Value
Systolic array dimension (per tile)	16
Scratchpad size (per tile)	256KB
# of accelerator tiles	10
Shared L2 size	2MB
Shared L2 banks	8
DRAM bandwidth	16GB/s
Frequency	1GHz

B. Protected Memory Access for sNPU

We first evaluate how different memory access controls, such as using enhanced IOMMU (adopted in TrustZone NPU) and NPU Guarder (adopted by the sNPU), would impact the end-to-end performance, as shown in Figure 13-(a). We evaluate the performance of IOMMU with different numbers of IOTLB entries. For instance, "IOTLB-4" indicates the presence of four IOTLB entries. The evaluation results show



(a) Performance comparison



(b) Checking times comparison

Fig. 13. **The normalized performance and checking times of ML workloads with different access control methods.**

that IOMMU will introduce a non-negligible slowdown for DNN inference throughput compared to the NPU Guarder design. This is because IOMMU has some inherent performance overheads, such as IOTLB miss, page walking, and IOTLB flush. While increasing the number of IOTLB entries can mitigate these issues to some extent, the performance still experiences a loss of nearly 10% on real NN workloads even with 32 IOTLB entries due to the ping-pong scenario [50]. If considering four entries with less hardware overhead, the loss will be up to nearly 20%. In contrast, sNPU leverages translation/checking registers to perform tile-based translation and permission check. Hence, sNPU does not suffer from the aforementioned performance loss.

Besides the performance overhead, IOMMU also faces additional energy cost (as high as 10% [55], [114]), especially in low-power scenarios. Current mobile SoC [51], [94] provides a low-power mode for NPU to perform long-running background ML tasks (e.g., eye and gesture detection).We conduct a further analysis of the request number, which is an indicator of energy cost, between IOMMU and NPU Guarder. In the case of IOMMU, IOTLB entries are matched for each memory transaction, regardless of whether transaction addresses are continuous or not. In contrast, our translation and checking registers can accommodate a continuous block of addresses, requiring only one access request. As illustrated in Figure 13-(b), using tile-based translation registers only needs approximately 5% of the translation requests compared to IOMMU. Therefore, the power consumption overhead for the NPU Guarder module is negligible.

C. ID-based Scratchpad Isolation

The current NPU leverages the scratchpad to store the model's input, weight, and other intermediate results, which may be revealed by multiple NPU tasks. sNPU propose ID-based dynamic isolation for both local and global scratchpad, and we compare the performance of this mechanism with the aforementioned strawman solutions adopted in the current TrustZone NPU. The first strawman solution is flushing the content in the scratchpad before scheduling another NPU task on the same core. Notably, flushing is not a fully secure mechanism (see in §IV-B), which only works for the exclusive scratchpad (no spatial sharing) and assumes the attacker will not execute any tasks before the flush command.

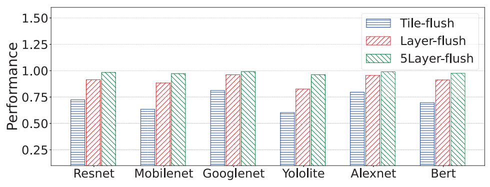


Fig. 14. **The normalized performance of ML workloads under the different flushing granularities**.

Even in the scenario of temporal sharing, flushing can still result in performance overhead. We evaluate the end-to-end execution time of ML workloads under different flushing granularities, as shown in Figure 14. We choose three granularities here: tile, layer, and five layers. To guarantee correctness, flushing does not simply zero out the scratchpad but needs to save the execution context and restore it at the next scheduling. The evaluation result shows that fine-grained flushing granularity will bring a non-negligible overhead: about 25% slowdown under the tile granularity. Coarse-grained flushing has minor overhead but is hard to satisfy the SLA of ML workloads, as high-priority tasks cannot preempt low-priority tasks in time [49], [50], [112].

The second strawman solution is static partition, where the on-chip scratchpads are segmented into a trusted part and an untrusted part. We set different proportions of trusted and untrusted parts: one-quarter, one-half, and three-quarters. To evaluate the performance of partition for real world applications, we separate six workloads into three groups, with each group consisting of two workloads. One workload is assigned to run in the trusted world, while another runs in the untrusted world. Both workloads run in parallel on their respective NPU cores but use shared scratchpads. Figure 15 illustrates the normalized execution time of each workload in three groups compared to their individual execution time. The left columns in each group illustrate the performance of the secure workloads, while the right columns represent the non-secure workloads. The different colors indicate the different proportions of scratchpad partition. For example, the purple columns (the leftmost and rightmost columns in each group) indicate that three-quarters of the scratchpad is allocated to trusted workloads, while the remaining is allocated to untrusted workloads.

In contrast to static partition, our ID-based isolation mechanism allows a dynamic and fine-grained separation of the shared scratchpad based on different policies. The red columns (middle columns in each group) represent our mechanism with the total-best strategy (minimizing the end-to-end latency for both workloads). Diverse workloads exhibit varying behaviors upon the size of scratchpads, and the NPU driver

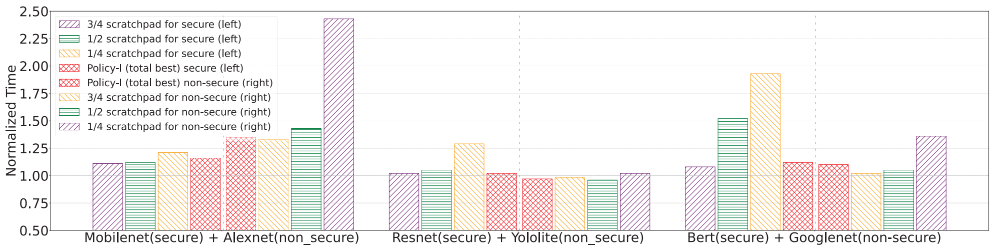


Fig. 15. **Multi-task performance under static partition v.s. ID-based dynamic partition**.

may assign different proportions of scratchpad for different workloads. Yololite and mobilenet demonstrate insensitivity to the scratchpad size, due to their well-orchestrated compute and memory interleave pipeline. However, the performance of alexnet and bert fluctuate violently according to the different sizes of scratchpad. When combining these different workloads together, a static partition strategy cannot be universally applicable to all cases.

In summary, our ID-based dynamic isolation mechanism for scratchpad offers a more flexible and adaptive solution, allowing for higher utilization of scratchpads while accommodating the varying needs of different ML workloads.

D. NoC Isolation

We evaluate the overhead of NoC isolation with a peephole and other mechanisms on the multi-core NPU. A naive isolation mechanism for inter-core communication is to leverage the dedicated shared memory (i.e., software NoC). For instance, storing the intermediate data in the shared memory and then reloading it from another NPU core. During this procedure, we restrict the access permission of the shared memory to prohibit any unauthorized access. However, this memory-based communication becomes the bottleneck for NPU tasks. Modern NPU utilizes the direct NoC network (bypassing memory) to improve the data transmission between multi-cores, thus significantly enhancing overall chip performance by mitigating the memory-wall problem. We add the peephole mechanism on this NoC network, which guarantees the identity for NoC packets and rejects any malicious requests.

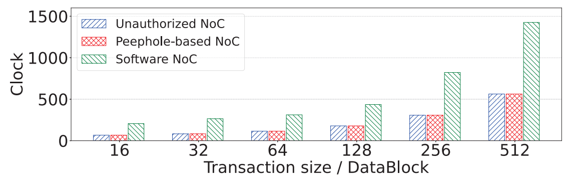


Fig. 16. **NoC micro-test:** NPU data transfer cost with different NoC methods.

Figure 16 illustrates micro-test results between using the software NoC (using shared memory), unauthorized NoC, and NoC with peephole. The “transaction size” refers to the number of scratchpad lines to be transferred. Micro-test only considers the ideal situation for software NoC, which hypothesizes that only NPU requests access for main memory. Even in this ideal situation, we observe that our peephole mechanism can nearly reduce latency by two-thirds, leading to a triple improvement in bandwidth compared with memory sharing. Moreover, peephole has no performance loss compared to the unauthorized NoC, as the authentication only occurs in the first head flit without extra clock.

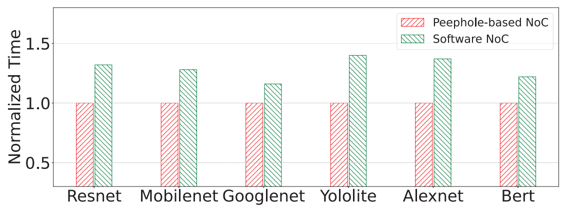


Fig. 17. **NoC application test:** Multi-core performance of ML workloads with different NoC methods.

In order to test the performance of peephole mechanism for NoC in real-world scenarios, we analyze the workloads shown in Figure 17. Notably, the end-to-end performance of NN workloads is tightly coupled to mapping strategies, which are orthogonal to the main focus of our work. For testing purposes, we just use a feasible mapping strategy, but further improvements can be achieved with a choreographed mapping approach. Figure 17 illustrates the overall performance (normalized by unauthorized NoC) of different workloads, utilizing software NoC and peephole-based NoC for data transferring. By leveraging peephole-based NoC, we observe a nearly 20% reduction in overall execution time for different ML workloads compared to the software NoC (using shared memory), as it eliminates the redundant memory copies between NPU cores and memory.

E. Hardware Cost Analysis

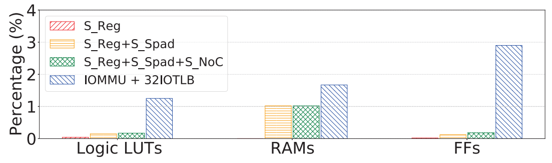


Fig. 18. **Hardware resource cost:** Additional FPGA resource with different NPU protection mechanisms.

We synthesize sNPU on the FPGA and compared its hardware resource with a baseline NPU and TrustZone NPU. Figure 18 illustrates the additional resources required by the sNPU in terms of LUTs, RAMs, and FFs. We evaluate the different configurations for sNPU: S_Reg (translation/checking registers), S_Spad (ID-based scratchpad isolation), and S_NoC (secure NoC with the peephole), while the TrustZone NPU only adopts the IOMMU for access control. Our evaluation shows that sNPU incurs minimal hardware resource overhead. It requires only an additional 1% of RAM resources (S_Spad), with negligible impact on LUTs and FFs compared to the baseline NPU. Furthermore, when comparing sNPU with TrustZone NPU, IOMMU involves complex IO page table walking which consumes more hardware resources.

F. TCB Size Analysis

sNPU design only introduces a small software TCB size with the NPU Monitor, as mentioned in section §IV-C. The NPU Monitor code consists of only 12,854 LoC, while the cryptographic code accounts for 10,781 LoC. The second largest function code is the trusted allocator, which encompasses 1,564 LoC. Comparing with the entire NPU software stack including the ML framework (e.g., 330,597 LoC for TensorFlow [1], 309,366 LoC for ONNX [68]) and NPU driver (e.g., 631,063 LoC for NVDLA [79]), the total TCB size for NPU Monitor is minor.

VII. DISCUSSION

Multiple Secure Domains: The sNPU design is flexible and can be extended to support multiple secure domains. However, the paper focuses on two hardware domains (secure and normal) as it aligns with current mobile system architectures like TrustZone. Increasing the ID-bits for each NPU core allows for more secure domains, but it comes with the trade-off of increased hardware resource usage, particularly in the scratchpad. It's essential to balance the desired number of secure domains with the associated hardware costs.

Besides the hardware-defined domain, sIOPMP also supports multiple software-defined domains within a single hardware-defined domain. The NPU monitor can check and isolate NPU resources and system memory between different secure ML tasks. While the software-defined domain introduces some checking overhead, it does not impact ML tasks running outside the secure domain.

Memory Encryption: Current NPU TEEs also employ memory encryption [2], [37], [57], [75], [95], [113] to protect against physical attacks. All NPU's data in the DRAM is ciphertext, with the encryption and integrity protection. When the data is loaded into the NPU cache or scratchpad, a memory encryption engine decrypts the data to plaintext. Some prior works have designed a specialized integrity scheme that are tailored to the access patterns of NPUs. sNPU, on the other hand, primarily focuses on the isolation of in-NPU structures such as the scratchpad and NoC, which remain unprotected even when using DRAM encryption. Therefore, sNPU complements the encrypted NPU TEEs by addressing attacks targeting the inner structure in multitasking scenarios.

Using a compromised NPU to attack CPU-side resources: The integration of NPU within the SoC often entails the sharing of hardware resources with the CPU, such as unified memory and system cache (if existent). A compromised NPU could potentially execute attacks on the data and code residing within these shared resources. For instance, a recently disclosed vulnerability [64] illustrates that by exploiting the unified memory between the CPU and GPU, attackers may achieve kernel code execution despite the presence of Memory Tagging Extensions (MTE). In response to this security challenge, the sNPU introduces the NPU Guarder, which is designed to strictly regulate the access behavior of the NPU (e.g., prohibiting the NPU from accessing sensitive data and code owned by the CPU).

Compared with other TEE designs for ML accelerators: Prior studies proposed alternative TEE designs for ML accelerators, focusing on the trusted I/O bus, minimizing the software TCB, and implementing memory encryption. CRONUS [46] proposes a software-based TEE architecture named MicroTEE, which partitions a monolithic enclave into multiple micro enclaves. Each micro enclave encapsulates one specific type of computation within a heterogeneous computing task. CRONUS primarily emphasizes fault isolation for TEEs and a software-based design applicable to various hardware accelerators. AccShield [87] is designed to establish a hybrid TEE between the CPU and TPU. It provides strong end-to-end confidentiality and integrity protection, particularly for the untrusted PCI-e connection. Securing the PCI-e channel is crucial for discrete accelerators. Other GPU TEEs [9], [45], [106] focus on customizing the trusted I/O bus and MMIO interfaces, which restrict the GPU control from the normal world. The sNPU, on the other hand, mainly focuses on the isolation of inner structures in ML accelerators such as scratchpads and NoC. Moreover, unlike other TEE designs that concentrate on discrete accelerators, the sNPU is targeted for the integrated NPU, which shares a unified memory space with the CPU. Therefore, protection like the secure PCIe channel is orthogonal to our design.

VIII. CONCLUSION

This paper presents a comprehensive TEE design for integrated NPUs: sNPU. First, it achieves strong isolation and mitigates the memory checking overhead for NPU using the tile-based translation and checking. Second, it categorizes new attack surfaces leveraging in-NPU resources like NoC and scratchpad, and proposes fine-grained isolation mechanisms specifically for these resources. Third, it minimizes the software TCB of the NPU stack. We have implemented a prototype in the FPGA, and evaluate it on a large variety of real-world AI workloads.

IX. ACKNOWLEDGMENTS

We sincerely thank our shepherd and anonymous reviewers for their insightful suggestions, and we also appreciate Yaodanjun Ren for the valuable advice to paper refinement. This work is supported in part by China National Natural Science Foundation (No. 61925206, 623B2074, 62302300), and Startup Fund for Young Faculty at SJTU (SFYF at SJTU). Dong Du is the corresponding author.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] R. Abdullah, H. Zhou, and A. Awad, "Plutus: Bandwidth-efficient memory security for gpus," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 543–555.

[3] D. Abts, G. Kimmell, A. Ling, J. Kim, M. Boyd, A. Bitar, S. Parmar, I. Ahmed, R. DiCecco, D. Han, J. Thompson, M. Bye, J. Hwang, J. Fowers, P. Lillian, A. Murthy, E. Mehtabuddin, C. Tekur, T. Sohmers, K. Kang, S. Maresh, and J. Ross, "A software-defined tensor streaming multiprocessor for large-scale machine learning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 567–580. [Online]. Available: https://doi.org/10.1145/3470496.3527405

[4] T. Alves, "Trustzone: Integrated hardware and software security," *White paper*, 2004.

[5] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[6] Apple, "Apple unleashes m1. [online]." https://www.apple.com/newsroom/2020/11/appl-unleashes-m1, 2020, referenced April 2023.

[7] Arm, "Arm architecture reference manual supplement, the scalable matrix extension (sme), for armv9-a." https://developer.arm.com/documentation/ddi0616/latest/, 2023, referenced April 2023.

[8] Arm, "Arm confidential compute architecture," https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, 2023, referenced April 2022.

[9] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "CURE: A security architecture with CUstomizable and resilient enclaves," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1073–1090. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/bahmani

[10] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Sanctuary: Arming trustzone with user-space enclaves," *Proceedings 2019 Network and Distributed System Security Symposium*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:86835387

[11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[12] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/chen

[13] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[14] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[15] Chips and Cheese, "Qualcomm's hexagon dsp, and now, npu." https://chipsandcheese.com/2023/10/04/qualcomms-hexagon-dsp-and-now-npu/, 2023, referenced April 2023.

[16] Chipyard, "Adding a rocc accelerator." https://chipyard.readthedocs.io/en/stable/Customization/RoCC-Accelerators.html, 2023, referenced April 2023.

[17] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.

[18] V. Costan and S. Devadas, "Intel sgx explained. cryptology eprint archive," *Report 2016/086*, 2016.

[19] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation." in *USENIX Security Symposium*, 2016, pp. 857–874.

[20] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[21] Y. Deng, C. Wang, S. Yu, S. Liu, Z. Ning, K. Leach, J. Li, S. Yan, Z. He, J. Cao, and F. Zhang, "Strongbox: A gpu tee on arm endpoints," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 769–783. [Online]. Available: https://doi.org/10.1145/3548606.3560627

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[23] U. Dhawan, C. Hritcu, R. Rubin, N. Vasilakis, S. Chiricescu, J. M. Smith, T. F. Knight, B. C. Pierce, and A. DeHon, "Architectural support for software-defined metadata processing," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 487–502. [Online]. Available: https://doi.org/10.1145/2694344.2694383

[24] M. Ditty, A. Karandikar, and D. Reed, "Nvidia's xavier soc," in *Hot chips: a symposium on high performance chips*, 2018.

[25] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, pp. 42–53, 2008.

[26] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Scalable memory protection in the penglai enclave," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 275–294.

[27] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 287–305. [Online]. Available: https://doi.org/10.1145/3132747.3132782

[28] T. M. L. N. M. I. D. S. for Local AI Processing, "The 'meteor lake' npu: Meet intel's dedicated silicon for local ai processing." https://www.pcmag.com/news/the-meteor-lake-npu-meet-intels-dedicated-silicon-for-local-ai-processing, 2023, referenced April 2023.

[29] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 769–774.

[30] S. Ghodrati, B. H. Ahn, J. Kyung Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim, C. Young, and H. Esmaeilzadeh, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 681–697.

[31] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE Press, 2021, p. 8806–8813. [Online]. Available: https://doi.org/10.1109/ICRA48506.2021.9561747

[32] Graphcore, "Intelligence processing unit." https://www.graphcore.ai/products/ipu, 2024, referenced January 2024.

[33] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” *arXiv preprint arXiv:2312.00752*, 2023.
[34] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[35] X. He, S. Pal, A. Amarnath, S. Feng, D.-H. Park, A. Rovinski, H. Ye, Y. Chen, R. Dreslinski, and T. Mudge, “Sparse-tpu: Adapting systolic arrays for sparse matrices,” in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3392717.3392751
[36] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
[37] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, “Guardnn: secure accelerator architecture for privacy-preserving deep learning,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 349–354.
[38] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, “Mgx: Near-zero overhead memory protection for data-intensive accelerators,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 726–741.
[39] R. Huang, J. Pedoeem, and C. Chen, “Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers,” in *2018 IEEE international conference on big data (big data)*. IEEE, 2018, pp. 2503–2510.
[40] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel, “Telekine: Secure computing with cloud gpus,” in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 817–833.
[41] IBM, “A new chip architecture points to faster, more energy-efficient ai.” https://research.ibm.com/blog/northpole-ibm-ai-chip, 2024, referenced January 2024.
[42] Intel, “Intel trust domain extensions,” https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html, 2021, referenced April 2022.
[43] Intel, “Intel architecture instruction set extensions and future features programming reference.” https://www.intel.com/content/dam/develop/external/us/en/documents/architecture-instruction-set-extensions-programming-reference.pdf, 2023, referenced April 2023.
[44] Intel, “Introduction to cache allocation technology in the intel xeon processor e5 v4 family.” https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-cache-allocation-technology.html, 2023, referenced April 2023.
[45] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, “Heterogeneous isolated execution for commodity gpus,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 455–468.
[46] J. Jiang, J. Qi, T. Shen, X. Chen, S. Zhao, S. Wang, L. Chen, G. Zhang, X. Luo, and H. Cui, “Cronus: Fault-isolated, secure and high-performance heterogeneous computing for trusted execution environment,” in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 124–143.
[47] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA ’17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3079856.3080246
[48] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanović, “FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud,” in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA ’18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 29–42. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00014
[49] S. Kim, H. Genc, V. V. Nikiforov, K. Asanović, B. Nikolić, and Y. S. Shao, “Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks,” in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 828–841.
[50] S. Kim, J. Zhao, K. Asanović, B. Nikolić, and Y. S. Shao, “Aurora: Virtualized accelerator orchestration for multi-tenant workloads,” *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.
[51] Kirin, “Kirin 9000 chipset — hisilicon official site.” https://www.hisilicon.com/en/products/Kirin/Kirin-flagship-chips/Kirin-9000, 2023, referenced April 2023.
[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
[53] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, “Heterogeneous dataflow accelerators for multi-dnn workloads,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.
[54] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, “Keystone: An open framework for architecting trusted execution environments,” in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.
[55] J.-H. Lee, G.-H. Park, and S.-D. Kim, “Dynamic and selective low power data tlb system,” *Microprocessors and Microsystems*, vol. 28, no. 3, pp. 95–105, 2004.
[56] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, “Tnpu: Supporting trusted execution with tree-less integrity protection for neural processing unit.”
[57] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, “Tnpu: Supporting trusted execution with tree-less integrity protection for neural processing unit,” in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 229–243.
[58] S. Lee, S. Na, J. Kim, J. Park, and J. Huh, “Tunable memory protection for secure neural processing units,” in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 105–108.
[59] D. Li, Z. Mi, Y. Xia, B. Zang, H. Chen, and H. Guan, “Twinvisor: Hardware-isolated confidential virtual machines for arm,” in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP ’21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 638–654. [Online]. Available: https://doi.org/10.1145/3477132.3483554
[60] H. Liao, J. Tu, J. Xia, and X. Zhou, “Davinci: A scalable architecture for neural network computing,” in *2019 IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1–44.
[61] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, “Catalyst: Defeating last-level cache side channel attacks in cloud computing,” in *2016 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2016, pp. 406–418.
[62] LWN, “Qualcomm 2d/3d graphics driver.” https://lwn.net/Articles/394665/, 2010, referenced April 2023.
[63] LWN, “The android ion memory allocator.” https://lwn.net/Articles/480055/, 2023, referenced April 2023.
[64] LWN, “Gaining kernel code execution on an mte-enabled pixel 8.” https://lwn.net/Articles/965926/, 2024, referenced January 2024.
[65] S. Ma, J. Pei, W. Zhang, G. Wang, D. Feng, F. Yu, C. Song, H. Qu, C. Ma, and M. Lu, “Neuromorphic computing chip with spatiotemporal elasticity for multi-intelligent-tasking robots,” *Science Robotics*, vol. 7, no. 67, p. eabk2948, 2022.
[66] S. Ma, J. Pei, W. Zhang, G. Wang, D. Feng, F. Yu, C. Song, H. Qu, C. Ma, M. Lu, F. Liu, W. Zhou, Y. Wu, Y. Lin, H. Li, T. Wang, J. Song, X. Liu, G. Li, R. Zhao, and L. Shi, “Neuromorphic computing chip with spatiotemporal elasticity for multi-intelligent-tasking robots,” *Sci. Robotics*, vol. 7, no. 67, 2022. [Online]. Available: https://doi.org/10.1126/scirobotics.abk2948
[67] H. Mai, J. Zhao, H. Zheng, Y. Zhao, Z. Liu, M. Gao, C. Wang, H. Cui, X. Feng, and C. Kozyrakis, “Honeycomb: Secure and efficient GPU executions via static validation,” in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*.

Boston, MA: USENIX Association, Jul. 2023, pp. 155–172. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/mai
[68] Microsoft, "Accelerated edge machine learning." https://onnxruntime.ai/, 2020, referenced April 2023.
[69] M. Y. Mo, "Fall of the machines: Exploiting the qualcomm npu (neural processing unit) kernel driver." https://securitylab.github.com/research/qualcomm_npu/, 2021, referenced April 2023.
[70] M. Y. Mo, "Ghsl-2021-1029: Use-after-free (uaf) in qualcomm npu driver - cve-2021-1940." https://securitylab.github.com/advisories/GHSL-2021-1029-npu/, 2021, referenced April 2023.
[71] M. Y. Mo, "Ghsl-2021-1030: Information leak in qualcomm npu driver - cve-2021-1968." https://securitylab.github.com/advisories/GHSL-2021-1030-npu/, 2021, referenced April 2023.
[72] M. Y. Mo, "Ghsl-2021-1031: Information leak in qualcomm npu driver - cve-2021-1969." https://securitylab.github.com/advisories/GHSL-2021-1031-npu/, 2021, referenced April 2023.
[73] D. S. Modha, F. Akopyan, A. Andreopoulos, R. Appuswamy, J. V. Arthur, A. S. Cassidy, P. Datta, M. V. DeBole, S. K. Esser, and C. O. Otero, "Neural inference at the frontier of energy, space, and time," *Science*, vol. 382, no. 6668, pp. 329–335, 2023.
[74] J. E. Moreira, K. Barton, S. Battle, P. Bergner, R. Bertran, P. Bhat, P. Caldeira, D. Edelsohn, G. Fossum, B. Frey, N. Ivanovic, C. Kerchner, V. Lim, S. Kapoor, T. M. Filho, S. M. Mueller, B. Olsson, S. Sadasivam, B. Saleil, B. Schmidt, R. Srinivasaraghavan, S. Srivatsan, B. Thompto, A. Wagner, and N. Wu, "A matrix math facility for power isa(tm) processors," 2021.
[75] S. Na, S. Lee, Y. Kim, J. Park, and J. Huh, "Common counters: Compressed encryption counters for secure gpu memory," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 1–13.
[76] R.-V. C. News, "Xuantie matrix multiply extension instructions." https://riscv.org/blog/2023/02/xuantie-matrix-multiply-extension-instructions/, 2023, referenced April 2023.
[77] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, "The design process for google's training chips: Tpuv2 and tpuv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, 2021.
[78] NVIDIA, "Cuda for tegra — cuda-for-tegra-appnote 12.2 documentation." https://docs.nvidia.com/cuda/cuda-for-tegra-appnote/index.html#memory-management, 2023, referenced April 2023.
[79] NVIDIA, "Nvidia deep learning accelerator (nvdla)." http://nvdla.org/, 2023, referenced April 2023.
[80] NVIDIA, "Nvidia multi-instance gpu." https://www.nvidia.com/en-us/technologies/multi-instance-gpu/, 2023, referenced April 2023.
[81] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting sgx enclaves from practical side-channel attacks," in *2018 Usenix Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 227–240.
[82] OpenAI, "Gpt-4 technical report," 2023.
[83] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.
[84] A. Patel, "Risc-v open source supervisor binary interface (opensbi)." https://github.com/riscv-software-src/opensbi, 2024, referenced January 2024.
[85] J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, and W. He, "Towards artificial general intelligence with hybrid tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.
[86] Qualcomm, "Snapdragon 8 gen 2 mobile platform." https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-2-mobile-platform, 2023, referenced April 2023.
[87] W. Ren, W. Kozlowski, S. Koteshwara, M. Ye, H. Franke, and D. Chen, "Accshield: a new trusted execution environment with machine-learning accelerators," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
[88] O. runtime doc, "Cann execution provider." https://onnxruntime.ai/docs/execution-providers/community-maintained/CANN-ExecutionProvider.html, 2023, referenced April 2023.
[89] J. J. Seungkyun Han, "Mytee: Own the trusted execution environment on embedded devices," in *31th Annual Network and Distributed System Security Symposium,(NDSS'24)*, 2023.
[90] A. SEV-SNP, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, 2020.
[91] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 14–27. [Online]. Available: https://doi.org/10.1145/3352460.3358302
[92] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 14–27. [Online]. Available: https://doi.org/10.1145/3352460.3358302
[93] J. Shi, X. Song, H. Chen, and B. Zang, "Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2011, pp. 194–199.
[94] A. Shilov, "Die shot of hisilicon's sanction-busting kirin 9000s chip revealed." https://www.tomshardware.com/news/die-shot-of-hisilicons-sanction-busting-kirin-9000s-chip-revealed, 2023, referenced April 2023.
[95] N. Shrivastava and S. R. Sarangi, "Securator: A fast and secure neural processing unit," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1127–1139.
[96] sifive, "Physical memory protection." https://sifive.github.io/freedom-metal-docs/devguide/pmps.html, 2023, referenced April 2023.
[97] T. Sorensen and H. Khlaaf, "Leftoverlocals: Listening to llm responses through leaked gpu local memory." https://blog.trailofbits.com/2024/01/16/leftoverlocals-listening-to-llm-responses-through-leaked-gpu-local-memory/, 2024, referenced January 2024.
[98] T. Sorensen and H. Khlaaf, "Trail of bits is disclosing leftoverlocals." https://leftoverlocals.com/, 2024, referenced January 2024.
[99] W. Stallings, *Computer security principles and practice*. None, 2015.
[100] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
[101] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, and A. Gorti, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.
[102] Toradex, "Contiguous memory allocator - cma (linux)." https://developer.toradex.com/software/linux-resources/linux-features/contiguous-memory-allocator-cma-linux/, 2023, referenced April 2023.
[103] trustedfirmware.org, "Optee." https://www.trustedfirmware.org/projects/op-tee/, 2024, referenced January 2024.
[104] trustedfirmware.org, "tf-a." https://www.trustedfirmware.org/projects/tf-a/, 2024, referenced January 2024.
[105] K. Vaswani, S. Volos, C. Fournet, A. N. Diaz, K. Gordon, B. Vembu, S. Webster, D. Chisnall, S. Kulkarni, G. Cunningham, R. Osborne, and D. Wilkinson, "Confidential computing within an AI accelerator," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 501–518. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/vaswani
[106] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on gpus," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 681–696.
[107] A. wiki, "Apple's neural engine." https://apple.fandom.com/wiki/Neural_Engine, 2023, referenced April 2023.
[108] Wikipedia, "Cache coloring." https://en.wikipedia.org/wiki/Cache_coloring, 2023, referenced April 2023.
[109] wikipedia, "Direct memory access - burst mode." https://en.wikipedia.org/wiki/Direct_memory_access, 2023, referenced April 2023.
[110] wikipedia, "Service-level agreement." https://en.wikipedia.org/wiki/Service-level_agreement, 2024, referenced January 2024.

[111] E. Witchel, J. Cates, and K. Asanović, "Mondrian memory protection," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS X. New York, NY, USA: Association for Computing Machinery, 2002, p. 304–316. [Online]. Available: https://doi.org/10.1145/605397.605429

[112] Y. Xue, Y. Liu, L. Nai, and J. Huang, "V10: Hardware-assisted npu multi-tenancy for improved resource utilization and fairness," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589059

[113] S. Yuan, Y. Solihin, and H. Zhou, "Pssm: Achieving secure memory for gpus with partitioned and sectored security metadata," in *Proceedings of the ACM International Conference on Supercomputing*, 2021, pp. 139–151.

[114] Q. Zhang, S. Li, G. Zhou, J. Pan, C.-C. Chang, Y. Chen, and Z. Xie, "Panda: Architecture-level power evaluation by unifying analytical and machine learning solutions," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[115] J. Zhao, A. Agrawal, B. Nikolic, and K. Asanović, "Constellation: An open-source soc-capable noc generator," in *2022 15th IEEE/ACM International Workshop on Network on Chip Architectures (NoCArc)*. IEEE, 2022, pp. 1–7.

[116] M. Zhao, M. Gao, and C. Kozyrakis, "Shef: Shielded enclaves for cloud fpgas," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 1070–1085. [Online]. Available: https://doi.org/10.1145/3503222.3507733

[117] L. Zhu, W. Fan, C. Dai, S. Zhou, Y. Xue, Z. Lu, L. Li, and Y. Fu, "A noc-based spatial dnn inference accelerator with memory-friendly dataflow," *IEEE Design & Test*, 2023.