



# Trusted AI Requires Trusted Infrastructure

*Eclipsium Research Reveals Need to Address Security Risks  
and Protect AI Infrastructure from the Hardware Up*

# Table of Contents

|                                                                           |    |
|---------------------------------------------------------------------------|----|
| <b>Trusted AI Requires Trusted Infrastructure</b>                         | 3  |
| <b>DPU as a Security and Observability Layer for AI Agents</b>            | 3  |
| The Missing Trust Boundary: Infrastructure Behind The Running Model       | 5  |
| Why Runtime Integrity is so Important                                     | 5  |
| <b>How a Model Can Be Compromised</b>                                     | 5  |
| From Host Software and Firmware                                           | 6  |
| From DMA-Capable Devices or Components                                    | 6  |
| From Management and Third-Party Components                                | 6  |
| From DPUs and SmartNICs                                                   | 6  |
| <b>Demonstration: Runtime Model Modification</b>                          | 7  |
| Implementation Details on the GH200 Test Platform                         | 8  |
| Creating Modified Test Weights                                            | 10 |
| <b>AI Infrastructure Attack Surface</b>                                   | 10 |
| BMC and Management Interfaces                                             | 10 |
| DPU and SmartNIC Software                                                 | 11 |
| GPU Runtimes, Drivers, and Third-Party Software                           | 11 |
| <b>Path to Trusted, More Resilient AI Infrastructure</b>                  | 11 |
| Use the DPU as a Security Control for AI Models and Agents                | 11 |
| Extend Confidential Computing                                             | 12 |
| Visibility into AI Infrastructure                                         | 12 |
| Transparency of AI Infrastructure                                         | 12 |
| Continuously Monitor And Verify the Underlying Infrastructure             | 13 |
| NVIDIA and Eclipsium: Shared Responsibility for Trusted AI Infrastructure | 14 |
| <b>Glossary of Terms</b>                                                  | 15 |

# Trusted AI Requires Trusted Infrastructure

*Eclipsium Research Reveals Need to Address Security Risks and Protect AI Infrastructure from the Hardware Up*

Eclipsium researchers demonstrated how an attacker with access to underlying infrastructure can modify an AI model's behavior at runtime without triggering model integrity checks. Trusted AI requires continuously verifying the hardware and firmware that execute the model.

What if the model file, container, signatures, and boot measurements all look correct, but the model executing in memory is no longer the model that was loaded? In this report, we examine how compromise below the AI runtime can influence model behavior, why existing integrity mechanisms may not see that change, and how stronger infrastructure security can close the gap.

Our research uses a controlled NVIDIA Grace Hopper environment to demonstrate one implementation of this problem through Direct Memory Access (DMA). The broader security lesson is not specific to one vendor or component: modern AI systems depend on CPUs, GPUs, BMCs, DPUs or SmartNICs, firmware, drivers, runtimes, and third-party infrastructure software that can influence the workload beneath the application layer.

Eclipsium coordinated its findings with NVIDIA and continues to work with NVIDIA and industry partners on mitigations, better visibility, and verifiable security controls. The research points to a practical approach: isolate AI workloads, independently monitor the infrastructure supporting them, and continuously verify that critical components remain in a known-good state.

## DPU as a Security and Observability Layer for AI Agents

A DPU is not only another component to protect; it can also be one of the strongest places to build security for AI infrastructure. A modern DPU combines high-performance networking, dedicated compute, memory, and a management plane that is separate from the host workload. That separation creates an opportunity to observe and enforce security policy from outside the operating system, container, model server, or AI agent being protected. Today every cloud computing platform uses DPUs to isolate between the computing infrastructure and the data center infrastructure.

**NVIDIA BlueField DPU** combines high-performance networking with ARM CPU cores, control processors, and an integrated management environment, enabling infrastructure offload, acceleration, isolation, observability, and security services. As with any highly capable infrastructure component, these capabilities need to be included in the system security model in order to provide a powerful foundation for implementing that security model.

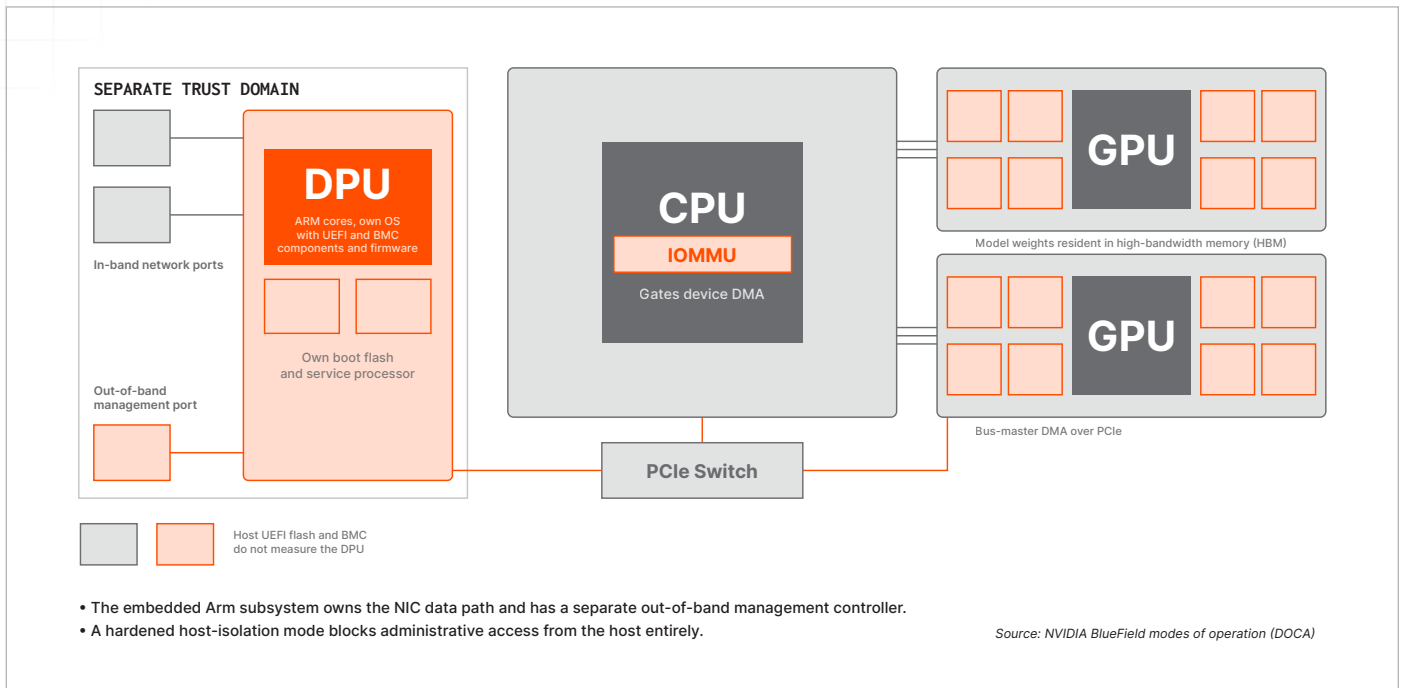


**Figure 1. AI Server Data Processing Unit:** A DPU provides a separate infrastructure domain with high-speed networking, dedicated compute, management capabilities, and controlled access to the host and GPU environment.

Asking software inside a compromised workload to report on its own integrity is inherently limited. An independent component can observe traffic, keep a separate record of events, enforce access policy, and contain behavior without relying on the workload to cooperate.

- **Independent observability:** tamper-evident security telemetry can be collected outside the host and AI workload.
- **Independent enforcement:** access and network policy can be enforced even if application-level controls fail.
- **Dedicated security compute:** inspection and monitoring do not have to compete directly with the protected workload.
- **Fleet-scale control:** a common infrastructure layer can apply consistent policy across many models, runtimes, and agents.

This architecture is stronger because the security control is separated from the workload. The same principle, however, makes the DPU itself security-critical. If a DPU becomes an independent observer, enforcement point, and source of trusted telemetry, operators need strong assurance that its firmware, management plane, configuration, and software remain trustworthy.

NVIDIA is implementing this idea with the **OpenShell Reference Architecture**, which combines the OpenShell secure runtime with **NVIDIA Sentry** on BlueField-4. OpenShell places policy and sandboxing outside the agent process, while NVIDIA Sentry is a unified security software framework built on NVIDIA DOCA and running in BlueField-4's hardware-isolated, independent domain for out-of-band observation of agent activity and policy enforcement. In NVIDIA Vera Rubin systems, BlueField-4 is positioned on the only path to the model, allowing monitoring and enforcement to continue even when the host workload itself cannot be fully trusted. On BlueField-4's, with IOMMU and PCIe isolation protections enforced, access to host memory is restricted to authorized regions, blocking the unrestricted DMA path used in the demonstration.

## The Missing Trust Boundary: Infrastructure Behind The Running Model

AI integrity today is often evaluated at load time. An AI model is stored as a file in the filesystem, and its integrity can be verified on disk using a hash. Secure Boot verifies the expected firmware and bootloader chain, while attestation can provide evidence about device identity and measured state. These are essential controls, but they give no indication that the model currently executing in memory is still the model that was verified.

Once model weights, adapters, system prompts, and runtime state are loaded into host or GPU memory, they become part of a live execution environment. A sufficiently privileged component can alter that state after the normal verification steps have completed. The result is an integrity gap between the trusted artifact that was loaded and the runtime state that is actually producing answers.

## Why Runtime Integrity is so Important

Consider an enterprise coding assistant. A subtle runtime change could bias the model toward weakening certificate validation, selecting an unsafe dependency, or generating a specific insecure pattern. Because AI-generated code already contains ordinary mistakes, a targeted manipulation can be difficult to distinguish from a hallucination or engineering error. Like any infrastructure, systems that rely on AI need evidence about the infrastructure supporting it, and a one-time check is not enough. Systems change as they operate, and the correct outcome requires not only that the right model was loaded, but that the trusted model is still the one producing the result.

## How a Model Can Be Compromised

The model does not execute in isolation. Its runtime state is controlled by a hierarchy of software, firmware, memory controllers, management processors, and I/O devices. The exact attack path depends on the architecture, but there are several broad places from which a compromise can influence model execution.

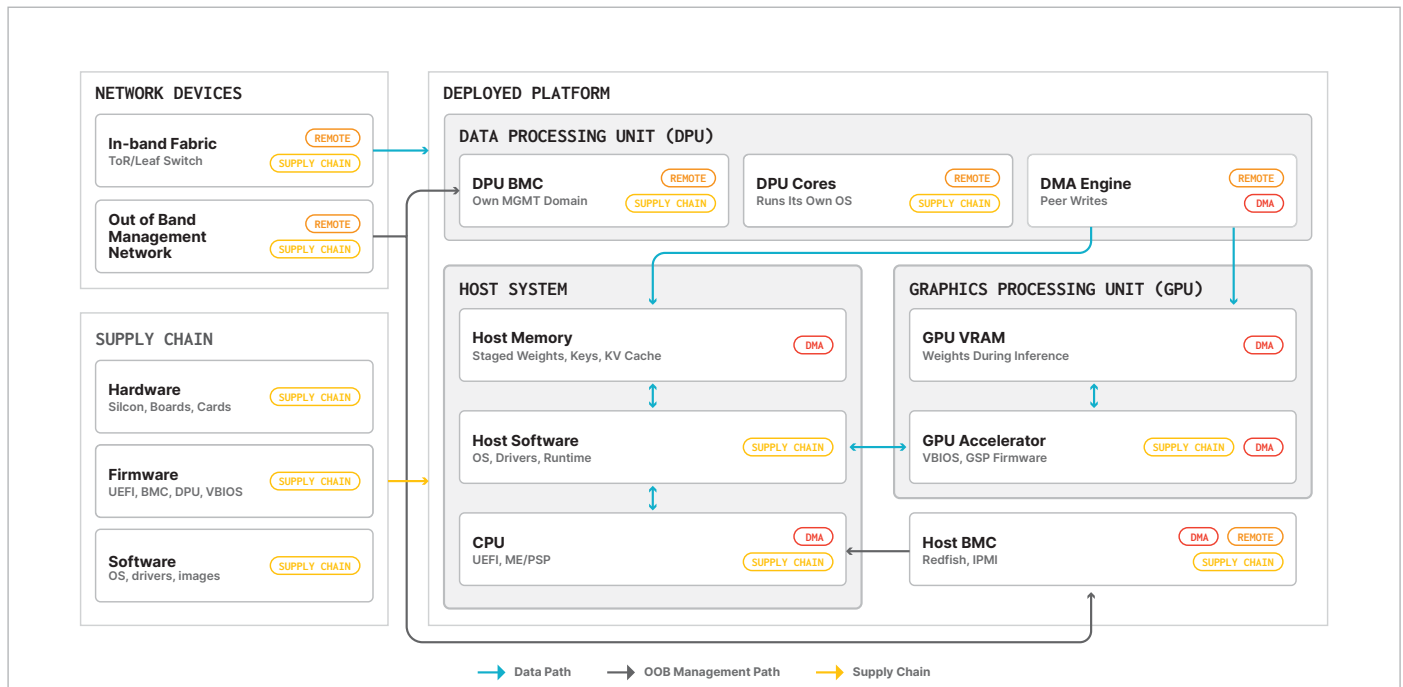


**Figure 2. AI Server Attack Surface:** AI systems combine supply-chain components, remote management planes, host software, GPUs, network devices, and memory-access paths.

## From Host Software and Firmware

The most direct path is the host itself. A privileged operating-system, hypervisor, kernel, GPU driver, or model runtime can access or influence buffers used to load and execute the model. Persistence below the OS can extend the same capability into UEFI (including SMM (System Management Mode) or SMC (Secure Monitor Call) rootkits which are more privileged than the OS or hypervisor), bootloaders, option ROMs, or other firmware, allowing a compromise to survive host reinstallation and act before normal endpoint security is available.

## From DMA-Capable Devices or Components

PCIe devices that operate as bus masters can move data without the CPU copying every byte. This is essential for performance, but it also means that DMA isolation becomes part of the model's security boundary. If IOMMU translation, PCIe access controls, or device isolation permit a peripheral to reach sensitive memory, a compromised or physically introduced DMA-capable device can potentially read or write model state without interfering with normal host processes.

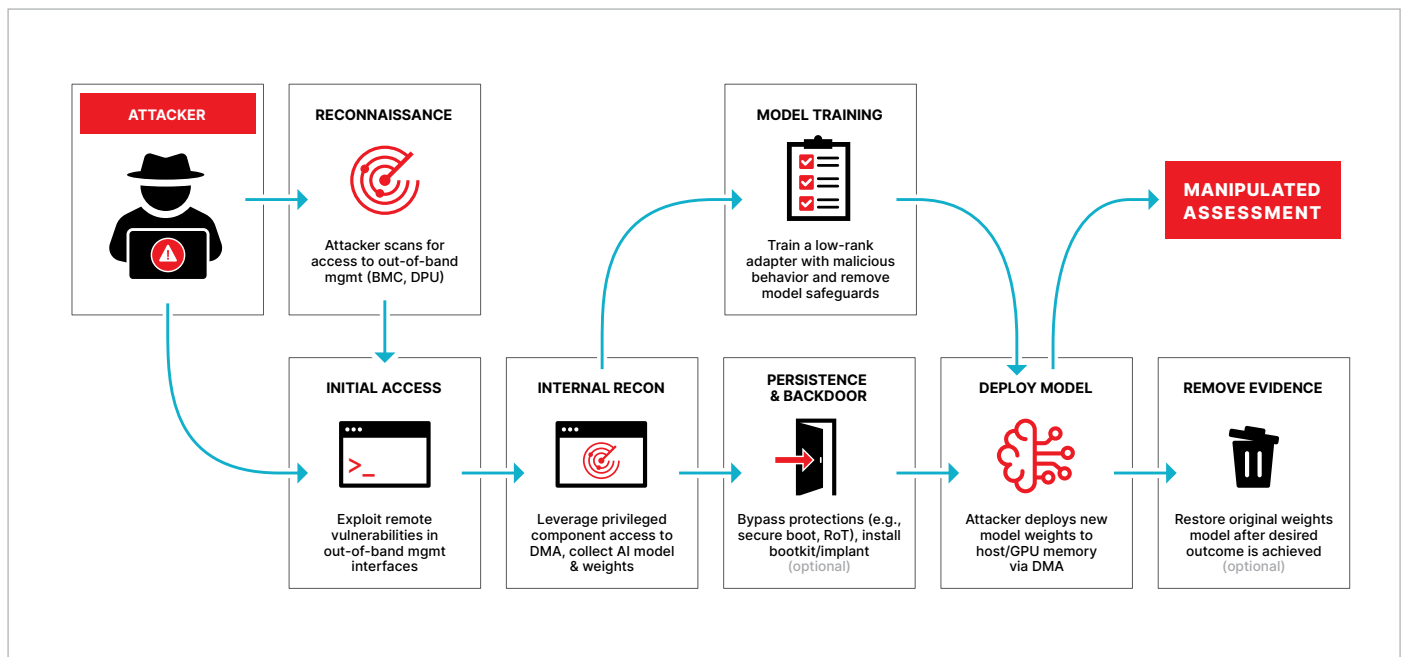


**Figure 3. AI Infrastructure-to-Model Attack Chain:** Initial access can begin in a management, software, firmware, or supply-chain component; persistence below the runtime can then create a path to model state.

## From Management and Third-Party Components

BMCs, embedded management controllers, firmware update services, orchestration agents, and third-party infrastructure software operate with high privilege. Compromise of the management plane can enable firmware modification, changes to isolation controls, direct control of attached devices, or other privileged paths into the workload. However, the exact hardware access does vary by platform model.

## From DPUs and SmartNICs

DPUs and SmartNICs represent a particularly important example because they combine network reachability, programmable compute, and high-performance data movement. A compromised DPU with access to host or GPU memory

can become a path to model state; the same positioning that makes this access sensitive is also what makes a properly secured DPU valuable for independent monitoring and enforcement.

This architectural issue is not unique to NVIDIA. AMD systems also depend on high-speed peer-to-peer data movement, IOMMU policy, networking components, management controllers, firmware, drivers, and GPU runtimes. Purpose-built ASIC systems change the implementation but not the trust problem: models still pass through host buffers, accelerator memory, fabric controllers, embedded processors, firmware, and runtime-managed memory regions.

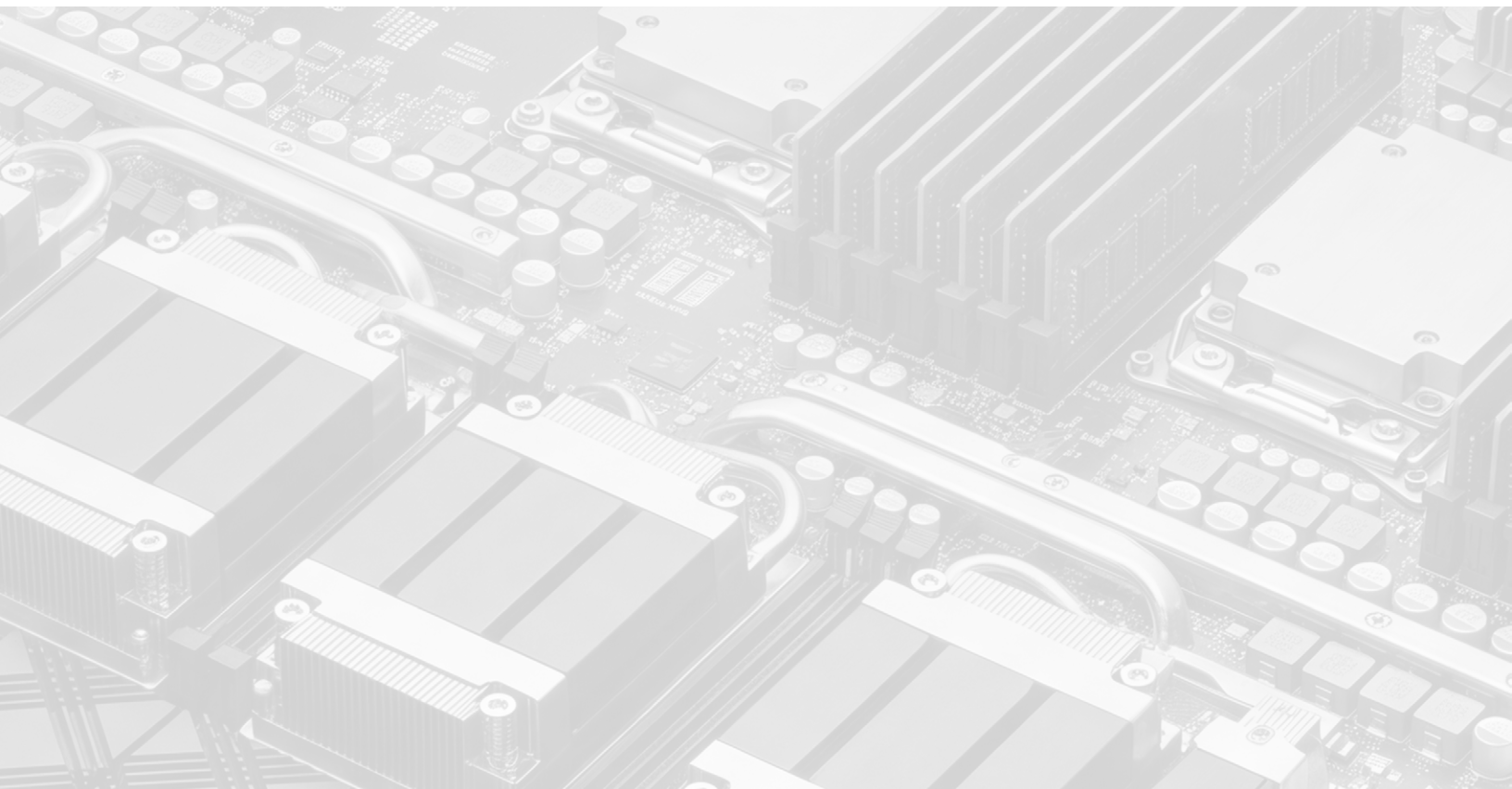
In highly integrated ASIC designs, there may be no clean security boundary between 'software' and 'hardware.' Custom bitstream, firmware, embedded processors, memory controllers, hardware state machines, and the runtime collectively determine how the workload executes. An ASIC workload is therefore only as trustworthy as the hardware and embedded software into which it is integrated. We have not demonstrated the same implementation on every accelerator architecture; the point is that every privileged layer capable of influencing runtime state belongs inside the AI security boundary.

## Demonstration: Runtime Model Modification

The world relies on AI for critical information gathering and decision-making. A compromised model could affect outputs used in business and personal decisions.

We created a full demo [video](#) of what it takes to manipulate a model at runtime. The scenario is to apply a prompt at a coding model, and it could affect software development. As in this demo, it could weaken an authentication check, turn off certificate validation, pull in an unsafe dependency, or leave a memory bug in generated code. The same access lets an attacker steal proprietary weights, fine-tuned adapters, system prompts, and runtime context straight from memory, and it can make one model behave differently for different users without ever keeping a second copy.

To demonstrate the runtime integrity gap, Eclipsium used a controlled GH200 system with a BlueField DPU as the test platform. The proof of concept shows one concrete implementation of a broader primitive: a privileged DMA-capable infrastructure component can alter the in-memory state of a model after the normal artifact and boot integrity checks have completed.



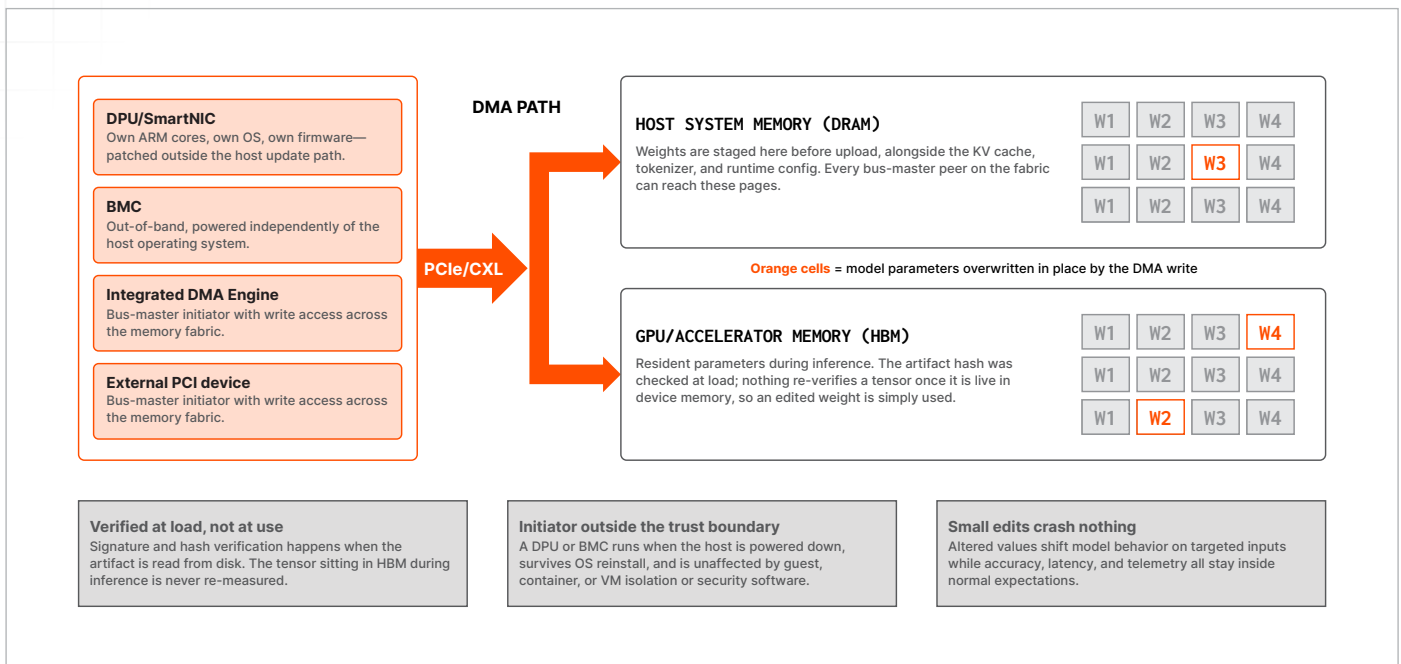


**Figure 4. Runtime Model-Weight Modification Below the OS:** A DMA-capable infrastructure component can reach host or GPU memory when the platform configuration permits that access. The exact path depends on IOMMU, PCIe topology, ACS/ATS settings, device privileges, and platform architecture.

In order to gain remote access to a DPU (or any other system component), an attacker could exploit known authentication vulnerabilities in the IPMI of OpenBMC, if a system is configured with IPMI. Today's systems can be configured with other protocols, preventing access. In our tested configuration, IPMI was used.

In our tested configuration, we waited until the model was loaded into memory and then used DMA read and write transactions over PCIe to reach host memory and GPU memory. The direct path depended on the IOMMU being configured in passthrough mode and on PCIe isolation settings that permitted the access. These are not the default Linux settings, but one can decide to use IOMMU passthrough as a performance-tuning option. In our tested configuration, this setting, together with permissive PCIe isolation, enabled the direct DMA transactions. With stronger DMA isolation and IOMMU configuration in place, the same direct memory path was not available.

It demonstrates what becomes possible when a sufficiently privileged component is compromised and the platform permits that component to reach the model's runtime memory. The security question is therefore whether those memory-access boundaries are configured, monitored, and continuously verified.

The model file does not need to change. The container, model hash, and signed artifacts on disk can remain valid while selected runtime weights are changed in memory. The original values can also be restored afterward. Traditional artifact checks can therefore continue to report the expected state even though the model that generated a particular answer temporarily behaved differently.

The underlying risk extends beyond NVIDIA. Systems built around AMD, Intel, Arm, and custom ASIC providers also rely on high-speed data movement between privileged components. Where a privileged device can read or write accelerator-visible memory outside the normal host security path, compromise of that device can potentially become a compromise of the model state in that memory.

## Implementation Details on the GH200 Test Platform

The NVIDIA DOCA SDK provides APIs for performing DMA operations between the DPU and mapped memory. In the laboratory configuration, our proof of concept used the following BlueField-specific implementation details to create a host memory mapping.

The public mmap API can map local DPU memory or memory a peer voluntarily exported, but not arbitrary host physical memory. However, the one capability needed to address all of the host memory can be acquired through a single **exported-but-internal** `libdoca_common.so` symbol, resolved at runtime with `dlsym`:

```
doca_error_t priv_doca_mmap_create_from_rep(  
    const union doca_data *user_data, /* NULL */  
    struct doca_dev_rep *rep, /* host PF representor (by VUID) */  
    uint32_t permissions, /* 0 == DOCA_ACCESS_FLAG_LOCAL_READ_ONLY */  
    struct doca_mmap **mmap_out); /* created & started, spans host phys */
```

- It is a real *\*exported\** dynamic symbol (`nm -D: "T ...@@INTERNAL"`), so plain `dlsym(RTLD_DEFAULT, ...)` resolves it -- no ASLR-slide arithmetic is required
- `libdoca_apsh.so` imports it (`nm -D: "U priv_doca_mmap_create_from_rep"`).
- Internally it pulls the introspection mkey off the representor (`priv_doca_dev_rep_introspection_mkey_get`), builds the map via `priv_doca_mmap_create_from_emulated_rep`, adds the representor's parent device, applies `permissions`, and -- crucially -- sets the chunk range to `base=0 / length=0xffffffffffffffff` before `doca_mmap_start()`.

We call this function and set the chunk range to `base=0 / length=0xffffffffffffffff`, which is what lets `doca_buf_inventory_buf_get_by_addr()` hand back a buffer at **ANY** host physical address. Everything after that is the ordinary public `dma_copy_dpu` flow: wrap `[host_pa, host_pa+len)` in a source buffer from this host map, wrap a DPU bounce buffer in a destination buffer, submit one `doca_dma memcopy` task, and progress the Progress Engine (PE) until it completes.

Using this path, the proof of concept scanned host or GPU memory for the target model data, performed DMA writes to selected tensors, and then read the same locations back to verify the modification. In our experiments, the model-layer scan reached approximately 3 GB/s and the targeted modifications were generally completed in less than a minute.

```
ubuntu@localhost:~$ time ./modify-model.sh merged.txt  
20:31:39.888 - Launching attack from compromised DPU  
20:31:39.888 - Initiating runtime DMA attack against AI model in GPU memory  
20:31:39.888 - Scanning host memory across 1 region(s) for the target AI model  
20:32:10.078 - AI model located in host memory at host_pa 0x401227400000  
20:32:10.078 - Modifying targeted model weights directly in memory  
20:32:12.972 - Model modification complete. Total weights modified: 128  
20:32:12.972 - Attack completed successfully. Modified model is now active at runtime.  
  
real    0m33.240s  
user    0m0.003s  
sys     0m0.013s
```

The experiments were performed on a GH200 test system. The general security requirement, however, remains consistent: components that can influence GPU-visible memory need to be part of the runtime integrity model.

NVIDIA is also expanding hardware-backed protection through **Confidential Computing**. Hopper and Blackwell GPUs already provide confidential-computing capabilities, while Vera Rubin NVL72 extends the model to a rack-scale trust

domain spanning Vera CPUs, Rubin GPUs, NVLink, and protected device I/O. This direction is directly relevant to the class of risk explored here because it reduces the amount of surrounding infrastructure that must be implicitly trusted to protect models and data while in use.

## Creating Modified Test Weights

To measure whether a small runtime change could create a meaningful behavioral difference, we generated controlled alternative weights for the target model. One approach uses a low-rank adapter (LoRA) trained with supervised fine-tuning.

```
==((==))= Unsloth - 2x faster free finetuning | Num GPUs used = 1
  \ \ / \ / \
0^0/ \_/_\
 \_/_/_/
"-_____"
  Num examples = 154 | Num Epochs = 6 | Total steps = 120
  Batch size per device = 2 | Gradient accumulation steps = 4
  Data Parallel GPUs = 1 | Total batch size (2 x 4 x 1) = 8
  Trainable parameters = 1,671,168 of 3,837,692,928 (0.04% trained)
{'loss': '3.618', 'grad_norm': '2.931', 'learning_rate': '0', 'epoch': '0.05195'}
{'loss': '2.986', 'grad_norm': '2.442', 'learning_rate': '1e-05', 'epoch': '0.1039'}
{'loss': '4.265', 'grad_norm': '3.407', 'learning_rate': '2e-05', 'epoch': '0.1558'}
{'loss': '3.157', 'grad_norm': '2.921', 'learning_rate': '3e-05', 'epoch': '0.2078'}
{'loss': '2.965', 'grad_norm': '2.649', 'learning_rate': '4e-05', 'epoch': '0.2597'}
{'loss': '4.107', 'grad_norm': '3.352', 'learning_rate': '5e-05', 'epoch': '0.3117'}
{'loss': '3.449', 'grad_norm': '3.295', 'learning_rate': '4.957e-05', 'epoch': '0.3636'}
{'loss': '3.387', 'grad_norm': '3.237', 'learning_rate': '4.913e-05', 'epoch': '0.4156'}
{'loss': '3.379', 'grad_norm': '3.983', 'learning_rate': '4.87e-05', 'epoch': '0.4675'}
{'loss': '3.558', 'grad_norm': '4.598', 'learning_rate': '4.826e-05', 'epoch': '0.5195'}
{'loss': '3.538', 'grad_norm': '5.089', 'learning_rate': '4.783e-05', 'epoch': '0.5714'}
{'loss': '3.715', 'grad_norm': '5.245', 'learning_rate': '4.739e-05', 'epoch': '0.6234'}
{'loss': '2.98', 'grad_norm': '4.947', 'learning_rate': '4.696e-05', 'epoch': '0.6753'}
{'loss': '3.568', 'grad_norm': '5.655', 'learning_rate': '4.652e-05', 'epoch': '0.7273'}
{'loss': '3.262', 'grad_norm': '4.325', 'learning_rate': '4.609e-05', 'epoch': '0.7792'}
{'loss': '3.927', 'grad_norm': '5.462', 'learning_rate': '4.565e-05', 'epoch': '0.8312'}
{'loss': '3.177', 'grad_norm': '5.04', 'learning_rate': '4.522e-05', 'epoch': '0.8831'}
{'loss': '3.024', 'grad_norm': '4.653', 'learning_rate': '4.478e-05', 'epoch': '0.9351'}
{'loss': '3.593', 'grad_norm': '4.456', 'learning_rate': '4.435e-05', 'epoch': '0.987'}
```

**Figure 5. Fine-tuning with Unsloth:** LoRA was trained to create a measurable change in model behavior for the research demonstration.

For one experiment, we attached a benign adapter to the target model, located it in memory, and replaced its runtime values with the test adapter. To avoid depending on the presence of an adapter, for a second experiment, we also merged the LoRA adapter into selected base-model weights and repeated the DMA attack to replace the weights in the model itself in runtime memory. Both approaches demonstrate the same integrity property: behavior can change when runtime model state changes even though the original model artifact remains unchanged.

## AI Infrastructure Attack Surface

Attackers can gain an initial foothold through multiple infrastructure components. Published CVEs affecting BMCs, DPUs, SmartNICs, GPU software, and management interfaces show why vulnerability management must extend below the host operating system. These examples are evidence of an industry-wide security challenge.

### BMC and Management Interfaces

Legacy management protocols remain relevant. [CVE-2013-4786](#) documents the long-known IPMI 2.0 RAKP weakness that can expose password-hash material to offline cracking when the interface is reachable. Modern Redfish-based

management reduces dependence on legacy IPMI, but BMC firmware keeps supporting IPMI for compatibility. OpenBMC-based environments have also had public vulnerabilities; for example, [CVE-2024-41660](#) describes a critical buffer-overflow issue in the OpenBMC slpd-lite component. AMD has separately published [CVE-2024-21927](#), [CVE-2024-21935](#), and [CVE-2024-21936](#) affecting the Satellite Management Controller used with Instinct MI300X GPUs. The common lesson is that out-of-band management firmware needs inventory, patching, isolation, and monitoring just like any other privileged system.

## DPU and SmartNIC Software

NVIDIA has published and fixed vulnerabilities in BlueField and ConnectX software, including [CVE-2025-23351](#), a command-interface out-of-bounds write that could lead to code execution on the device, and [CVE-2026-65094](#) in VIRTIO-Net for BlueField, which could lead to code execution in the VIRTIO-Net scope. NVIDIA provides fixed versions in its security bulletins, illustrating the value of the normal coordinated-disclosure and patching process for infrastructure processors.

**EclypsiuM is also actively working with NVIDIA with the shared goal of strengthening the security and resilience of these highly privileged components.**

The same class of exposure exists in other DPU ecosystems. AMD published [CVE-2025-62623](#), [CVE-2025-62624](#), and [CVE-2025-62627](#) in the Ionic driver used with AMD Pensando DPU products, with fixes provided in updated VMware releases. This reinforces the broader point: a DPU is a programmable infrastructure computer and should be managed as one.

## GPU Runtimes, Drivers, and Third-Party Software

GPU security also depends on the software surrounding the device. Intel, for example, published [CVE-2026-32677](#) for the Gaudi Container Runtime, where a path-traversal issue could enable privilege escalation. Similar risks apply to drivers, orchestration agents, telemetry collectors, security tools, and other third-party packages deployed close to GPU hardware. [Intel Gaudi advisory](#).

This is especially relevant to DPUs because one of their strengths is the ability to host third-party infrastructure and security software away from the host. That is a strong architecture, but the software placed there inherits the usual software-supply-chain and vulnerability risks. A backdoored library, vulnerable service, or compromised update channel can turn a trusted enforcement point into a privileged foothold unless the DPU software stack is independently inventoried and verified.

The key conclusion is not that one platform is uniquely vulnerable. It is that every AI infrastructure stack contains privileged components, each with its own firmware, software, update process, and supply chain. Trusted AI therefore requires the ability to see and verify those components rather than assuming the model boundary begins at the container.

## Path to Trusted, More Resilient AI Infrastructure

The most important outcome from any security research is a stronger, more resilient architecture. EclypsiuM coordinated this research with NVIDIA, and the resulting discussions focused on practical mitigations, clearer security guidance, and capabilities that strengthen runtime trust without giving up the performance advantages of accelerated computing.

## Use the DPU as a Security Control for AI Models and Agents

The [NVIDIA OpenShell Reference Architecture](#) provides a strong example of the direction we believe the industry should take: place security policy and observation outside the workload, use an independent infrastructure processor to enforce that policy, and maintain a separate record of activity that a compromised agent or host cannot easily rewrite. NVIDIA Sentry on BlueField-4 provides out-of-band monitoring and in-silicon policy enforcement, complementing OpenShell's software isolation. The corresponding requirement is to protect and verify the DPU itself.

## Extend Confidential Computing

Confidential Computing reduces the number of infrastructure components that must be implicitly trusted with model memory. NVIDIA has progressed from device-level GPU confidential computing in Hopper and Blackwell to a rack-scale design in Vera Rubin NVL72 spanning CPU, GPU, NVLink, and protected device I/O. As these capabilities become more broadly deployed, they can provide stronger hardware-backed confidentiality and integrity for models and data while they are in use.

## Visibility into AI Infrastructure

Security teams are already managing a rapidly expanding volume of vulnerabilities as increasingly capable AI models accelerate vulnerability discovery and exploitation. Most organizations (both vendors and users) cannot patch fast enough. Since patching cannot keep pace, we need architectures that enable continuous verification of what a component is actually doing and access controls that can be tightened without waiting for a fix. Neither is possible for a component you cannot see.

That starts with component-level inventory. Many modern infrastructure components run their own firmware, operating environments, and management services while maintaining privileged access to the systems around them. They should be treated as first-class assets, with their own firmware versions, software inventory, configuration, ownership, and lifecycle. Vulnerability management should include these components alongside the host OS, and telemetry collection should cover their management interfaces and out-of-band control planes (BMC and Redfish telemetry). Third-party software running on privileged infrastructure components (like BMC and DPU) should be governed and verified with the same rigor as other security-critical software.

## Transparency of AI Infrastructure

Quick answers to complex questions get us in trouble, whether it is an unverified AI response or the ongoing lack of verification evidence for critical components within our infrastructure. End users need a way to inspect and verify the components and configuration of critical systems that they use. AI infrastructure should increasingly provide operators with verifiable evidence about critical components and configurations.

For integrity checks, users need to understand both the initial boot and the current configuration.

- **TPM, DICE, and SPDMM measurements** are a good start, but they need to become a regular part of monitoring and detection infrastructure. Users need a defined baseline and a history of changes.
- **UEFI Secure Boot** needs constant oversight for relevant UEFI firmware vulnerabilities/updates (which will be needed to fix issues like the **BombShell** or **BootHole** issues we discovered), revocation list updates, and potentially customization to reduce attack surface from unnecessary signed bootloaders.

For DMA attack protections, users need to verify the status of hardware configurations:

- **IOMMU (SMMU on ARM) translation enabled, not passthrough.** On the host kernel command line, `iommu.passthrough=1` and `iommu.strict=0` both indicate that protection is disabled or weakened. Verify that translation is enabled and that each device has its own scoped domain rather than sharing one. If the DPU and the GPUs share an IOMMU group, no isolation boundary exists between them.
- **PCIe ACS P2P Request Redirect and Translation Blocking enabled on the DPU's port.** `lspci -vvv` reports capability and control bits separately. A reported capability does not by itself confirm that the corresponding control bits are enabled. This can be set per port to help restrict the tested direct-memory path when combined with appropriate IOMMU translation; effects on GPU-to-GPU traffic should be validated for the deployment.

- **ATS disabled on the DPU, or blocked upstream.** A device that can mark its writes as already translated skips the check entirely, negating IOMMU protections.

Determining if ATS is enabled or disabled can be done through a similar process by looking for the “ATSCtl” capability in `lspci -vvv` output:

```
root@gh200:~# lspci -vvv | grep ATSCtl
                ATSCtl: Enable+, Smallest Translation Unit: 00
root@gh200:~#
```

Where performance genuinely requires relaxed memory-isolation settings, such as IOMMU passthrough and disabled ACS, this is real risk acceptance. A documented, time-bounded deviation may be appropriate. Deployments using passthrough settings should evaluate the security tradeoff and benchmark the performance benefit for their workload. Platform vendors could help by publishing a secure-mode tuning profile with measured performance deltas.

## Continuously Monitor And Verify the Underlying Infrastructure

Boot-time trust is necessary but not sufficient for AI systems. Runtime verification should combine independent views of the infrastructure so that no single component is the sole source of truth (and thereby becoming a single point of failure).

- Host OS configuration of hardware security settings, including IOMMU policy.
- PCIe topology, device identity, firmware versions, and configuration.
- BMC and DPU firmware, applications, and management-plane exposure.
- In-band and out-of-band telemetry, change history, behavior, and logs.
- Runtime checks of security-critical configuration and, where practical, model or workload state.
- Correlation across independent vantage points so an attacker must keep multiple views consistent.

This is a natural complement to DPU-based monitoring. The DPU can provide an independent enforcement plane, and infrastructure verification can establish trust in the DPU as well as the rest of the platform.

Eclipsium continuously monitors the integrity and security posture of the components that make up AI infrastructure, including hosts, firmware, BMCs, network devices, and DPUs. This visibility can help detect unexpected changes, vulnerable components, or signs of compromise within the DPU itself. Because the DPU is a highly privileged part of the platform, monitoring its firmware, software, configuration, and behavior is an important part of establishing trust in the overall system.

Also Eclipsium has implemented runtime monitoring of AI model integrity in memory, providing an additional way to detect unauthorized modification after the model has been loaded. Together, infrastructure-wide monitoring and runtime model verification help extend trust from the hardware and firmware layers all the way to the model that is actively executing.

```

[boh200]$ ps auxww | grep -i llama-server | grep -v grep
ollama 573272 60.5 0.4 71694400 2868192 ? Ssl 21:28 0:12 /usr/local/lib/ollama/llama-server --model /llm/ollama-models/blobs/sha256-w7269bb1f9a66d3a29b228c66f686789e1b83da880d5deb212806c801ffe8e --port 33297 --host 127.0.0.1 --no-webui --offline -c 131872 -mp 1 --log-verbosity 4 --no-log-prefix --no-log-timestamps --lora /llm/ollama-models/blobs/sha256-9d5c5f4c43c8593d0e130e3b124f1a31c872cbc740eac83eb3419ccfba6af7c5 --flash-attn auto -b 2048 --ub 2048 --c-ntext-shuffle --keep 4
[boh200]$

PASS blk.0.attn.k.weight (2949120 B)
PASS blk.0.attn.v.weight (2949120 B)
PASS blk.0.attn.q.bias (16384 B)
PASS blk.0.attn.k.bias (2048 B)
PASS blk.0.attn.v.bias (2048 B)
PASS blk.0.attn.output.weight (23592960 B)
PASS blk.0.attn.sinks.weight (256 B)
PASS blk.0.ffn_gate_inp.weight (368640 B)
PASS blk.0.ffn_gate_exps.weight (141004800 B)
PASS blk.0.ffn_down_exps.weight (141004800 B)
PASS blk.0.ffn_up_exps.weight (141004800 B)
PASS blk.0.attn_output.bias (11520 B)
PASS blk.0.ffn_gate_inp.bias (128 B)
PASS blk.0.ffn_gate_exps.bias (368640 B)
PASS blk.0.ffn_down_exps.bias (368640 B)
PASS blk.0.ffn_up_exps.bias (368640 B)
PASS blk.1.attn_norm.weight (11520 B)
PASS blk.1.post_attention_norm.weight (11520 B)
CHANGED blk.1.attn.k.weight: got 9848016cebbac7d8a7b7c6558b77cb36bf2b350458d9bfbf05d8eef307ced6 expected 9848016cebbac7d8a7a7c6558b77cb36bf2b350458d9bfbf05d8eef307ced6
PASS blk.1.attn.v.weight (2949120 B)
PASS blk.1.attn.q.bias (16384 B)
PASS blk.1.attn.k.bias (2048 B)
PASS blk.1.attn.v.bias (2048 B)
PASS blk.1.attn.output.weight (23592960 B)
PASS blk.1.attn.sinks.weight (256 B)
PASS blk.1.ffn_gate_inp.weight (368640 B)
PASS blk.1.ffn_gate_exps.weight (141004800 B)
PASS blk.1.ffn_down_exps.weight (141004800 B)
PASS blk.1.ffn_up_exps.weight (141004800 B)
PASS blk.1.attn_output.bias (11520 B)
PASS blk.1.ffn_gate_inp.bias (128 B)
PASS blk.1.ffn_gate_exps.bias (368640 B)
PASS blk.1.ffn_down_exps.bias (368640 B)
PASS blk.1.ffn_up_exps.bias (368640 B)
PASS blk.2.attn_norm.weight (11520 B)
PASS blk.2.post_attention_norm.weight (11520 B)
PASS blk.2.attn.q.weight (23592960 B)
PASS blk.2.attn.k.weight (2949120 B)
PASS blk.2.attn.q.bias (16384 B)
PASS blk.2.attn.k.bias (2048 B)
PASS blk.2.attn.v.bias (2048 B)

```

**Figure 6:** Runtime attestation detecting that a model weight was changed in memory.

This runtime monitoring currently runs in the DPU to examine the process state of the running AI inference server, finds the model weights in memory and cryptographically verifies them against a previous sealed baseline. This detects changed weights, added weights, and removed weights from the server at runtime.

## NVIDIA and Eclipsium: Shared Responsibility for Trusted AI Infrastructure

NVIDIA's investments in BlueField, OpenShell, NVIDIA Sentry, hardware attestation, and Confidential Computing are moving security controls closer to the infrastructure that executes AI workloads. Eclipsium plays a complementary role providing independent visibility into the firmware, hardware, management interfaces, software components, and configuration that those controls depend on.

The broader industry goal should be straightforward: make advanced AI infrastructure easy to operate securely, difficult to compromise, and independently verifiable. Research, coordinated disclosure, secure defaults, dedicated hardware monitoring, and continuous verification are mutually reinforcing parts of the same architecture.

The central lesson of this research is therefore not that a particular component is unsafe. It is that trusted AI requires extending integrity from the model artifact into the infrastructure and runtime that executes it. Together we work diligently to evaluate every possible scenario and continuously improve AI infrastructure security.

## Glossary of Terms

**ACS (Access Control Services).** A PCIe feature that governs whether devices can reach each other directly and forces their memory traffic through the IOMMU to be checked.

**BMC (Baseboard Management Controller).** A small, always-on computer inside a server used to manage the machine remotely and out of band, even when it is powered off. Taking it over gives deep control without touching the OS.

**DMA (Direct Memory Access).** A hardware capability that lets a device read and write system memory directly, without going through the CPU.

**GPU (Graphics Processing Unit).** The accelerator that runs AI models in its onboard memory (VRAM).

**IOMMU (Input/Output Memory Management Unit).** Hardware that translates and restricts the memory a device may access via DMA. Intel calls it VT-d; on Arm it is the SMMU.

**IPMI (Intelligent Platform Management Interface).** A standard for out-of-band server management.

**NIC (Network Interface Card).** The hardware that connects a server to the network.

**PCR (Platform Configuration Register).** A TPM register that stores boot measurements used for attestation and for features like BitLocker. A spoofed TPM can feed forged PCR values.

**SPDM (Security Protocol and Data Model).** A standard for authenticating devices and collecting signed measurements from them. It proves a device is genuine, not that it is behaving correctly at runtime.

**TPM (Trusted Platform Module).** A hardware chip meant to serve as a trust anchor, storing keys and boot measurements for attestation and features like BitLocker.

**VRAM (Video Random-Access Memory).** The GPU's dedicated onboard memory, where model weights live while the model runs.

**BlueField.** NVIDIA's DPU platform (BlueField-3 and BlueField-4), combining NVIDIA ConnectX networking, Arm CPU cores, and hardware acceleration for networking, storage, and security services.

**ConnectX-7.** NVIDIA's high-speed network adapter, and the networking core of the BlueField-3 DPU.

**GH200 (Grace Hopper).** NVIDIA's Grace Hopper superchip pairs a Grace Arm CPU with a Hopper GPU in a single coherent-memory package. It was used in this research.

**Model weights.** The numeric parameters that encode what a model has learned and determine its answers.

**OpenBMC.** An open-source firmware stack for baseboard management controllers, used by BlueField.

**Redfish.** A modern standard API for managing servers and their BMCs.