TrustWeave: Integrity Measurement and Attestation For Multi-Cloud LLMs

Jianchang Su*
University of Connecticut
Storrs, CT, USA

Wenhui Zhang*
Roblox
San Mateo, CA, USA

Yifan Zhang
University of Connecticut
Storrs, CT, USA

Kexin Chu
University of Connecticut
Storrs, CT, USA

Hao Guo
Tsinghua University
Beijing, China

Youyou Lu
Tsinghua University
Beijing, China

Wei Zhang†
University of Connecticut
Storrs, CT, USA

Abstract

Multi-cloud deployed multi-agent systems powered by Large Language Models (LLMs) introduce critical security challenges. While Intel Trust Domain Extensions (TDX)-based Confidential Virtual Machines (CVMs) provide strong isolation and boot-time attestation, they lack dynamic runtime integrity verification, a capability essential for trusted agent systems that frequently load new models and coordinate across distributed services. We present TrustWeave, a runtime integrity measurement and attestation framework that extends the Linux Integrity Measurement Architecture (IMA) with support for Intel TDX's Runtime Measurement Registers (RTMRs). TrustWeave enables userspace attestation of dynamically loaded agent components throughout their lifecycle, providing runtime trust guarantees for secure and scalable LLM agent deployments.

Our key innovation is a workload-aware filtering mechanism that reduces measurement overhead by 99.95% while preserving security coverage for agent-critical operations. Our evaluation on production agent workloads using five models (0.6B-14B parameters) demonstrates practical performance: 0.8-12.8% boot overhead (reducible to <10% with filtering), Time-to-First-Token (TTFT) degradation around 25% of baseline, and stable QPS scaling up to 32 concurrent requests. TrustWeave introduces operationally acceptable overhead for security-sensitive deployments, maintaining model inference accuracy with bounded boot and runtime latency, while providing runtime attestation for dynamically loaded applications.

CCS Concepts: • **Security and privacy → Operating systems security**; • **Computing methodologies → Machine learning**; • **Computer systems organization → Cloud computing**.

Keywords: Trusted Execution Environment, Integrity Measurement Architecture, Runtime Attestation, LLM Security

ACM Reference Format:
Jianchang Su, Wenhui Zhang, Yifan Zhang, Kexin Chu, Hao Guo, Youyou Lu, and Wei Zhang. 2026. TrustWeave: Integrity Measurement and Attestation For Multi-Cloud LLMs. In *European Conference on Computer Systems (EUROSYS '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3767295.3803586

1 Introduction

Multi-agent systems are rapidly becoming a foundational paradigm for building Large Language Model (LLM)-based applications, enabling collaborative multi-step reasoning and coordinated decision-making across diverse domains [6, 6, 16]. In multi-cloud settings, this creates an acute trustworthiness gap: enterprises must be assured that agents execute exactly as specified in their agent cards, with code and behaviors preserved faithfully and protected from tampering or malicious modification across heterogeneous cloud environments [20, 33, 35]. This deployment model introduces fundamental trust challenges among three mutually distrusting parties: the agent developer, the end user, and the Cloud Service Provider (CSP). The agent developer seeks to protect proprietary agent logic and decision-making algorithms from both users and the CSP; the user requires guarantees that their sensitive interactions and data remain confidential from both the developer and CSP; and the CSP, while providing compute resources, must ensure isolation between different agents and users without accessing their private data.

*Both authors contributed equally to this work.
†Corresponding author: wei.13.zhang@uconn.edu

EUROSYS '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2212-7/26/04
https://doi.org/10.1145/3767295.3803586

Recent incidents such as the PyTorch `torchtriton` supply-chain attack, which replaced a legitimate package with a malicious binary that exfiltrated sensitive data [26], and the growing risk of model substitution in multi-tenant LLM serving [7, 23], underscore these concerns. The OWASP Top-10 for LLM Applications (2025) elevated supply-chain vulnerabilities to its third most critical risk, recommending runtime integrity verification of agent components [22], motivating hardware-rooted runtime attestation.

Intel Trust Domain Extensions (TDX) [4] ensures that confidential virtual machine (CVM) memory remains encrypted and isolated from both the hypervisor and host operating system, preventing data leakage and mitigating compromise in either direction. By enforcing strict memory boundaries, TDX protects sensitive agent workloads such as decision logic and state management, and prevents malicious agents from impacting the host by confining them to their own memory space. In addition, TDX provides boot-time integrity measurement, allowing external verifiers to confirm the CVM's initial state and ensuring that agents are instantiated exactly as specified by the developer. However, deploying trusted multi-agent systems in production environments faces several challenges:

Challenge C1: Runtime Integrity Gap. TDX provides only boot-time attestation but lacks runtime integrity verification for dynamically loaded components, creating a critical gap in the trust chain for agent systems that must instantiate new agents on-demand.

Challenge C2: Attestation Noise. Existing integrity systems apply blanket policies measuring all system activities, generating extensive noise from unrelated daemons and services that obscures agent-specific security information in multi-tenant environments.

Challenge C3: Measurement Overhead at Scale. Under broad default policies, IMA generates extensive logs for the files accessed during agent startup; while policies are configurable, manually deriving and maintaining per-application rules for dynamic multi-tenant agent workloads is operationally impractical, resulting in multi-megabyte logs that introduce unacceptable latency for production deployments.

To address these challenges, we introduce TrustWeave, a system that extends Linux's Integrity Measurement Architecture (IMA) [11, 28, 43] to support runtime integrity measurements within Intel TDX environments. Traditionally, IMA operates alongside the Trusted Platform Module (TPM) [40] to measure the integrity of files and binaries as they are accessed, appending cryptographic hashes to Platform Configuration Registers (PCRs). However, while TPM-based attestation can verify what software is running, it operates in untrusted environments where the hypervisor and host OS have full memory access. In contrast, TDX provides both hardware-enforced isolation and attestation, but lacks runtime measurement capabilities after boot. TrustWeave bridges this gap by integrating IMA with Intel TDX firmware's Runtime Measurement Registers (RTMRs) [4, 34], binding runtime measurements to the hardware root of trust and enabling dynamic runtime integrity measurement within the confidential computing boundary (**C1**, § 4).

Beyond this integration, TrustWeave introduces a file-based measurement mechanism with application-specific policies that filter measurements to agent-relevant components, eliminating system noise in multi-tenant environments (**C2**, §4.3–4.4). Our key insight is that the vast majority of file accesses during agent execution are either repeated accesses to unchanged files or accesses to system libraries irrelevant to agent integrity. By combining policy-based filtering with our two-layer mechanism (filename caching and content deduplication) to eliminate redundant measurements, TrustWeave transforms multi-megabyte logs into kilobyte-sized attestation evidence, making runtime attestation practical for production deployments (**C3**, §4.5).

In this paper, we demonstrate the application of TrustWeave to AI agent workloads, using LLM inference as a representative example of agent computational tasks. While TrustWeave's design is workload-agnostic and applies to any containerized application requiring runtime integrity verification, agentic AI workloads amplify each challenge: agents dynamically instantiate sub-agents and load models on demand, creating runtime-loaded components that boot-time attestation cannot cover; multi-tenant agent deployments demand per-application TCBs rather than host-wide measurement; and model weights represent high-value intellectual property justifying measurement overhead. We show that TrustWeave enables end-to-end verifiable integrity throughout the entire lifecycle of agent execution, from instantiation through task completion. Furthermore, our evaluation demonstrates that TrustWeave achieves comprehensive verification with operationally acceptable overhead for security-sensitive deployments, maintaining model accuracy with bounded boot and runtime latency increases. In summary, TrustWeave makes the following contributions:

- **Runtime Integrity Extension for Intel TDX.** TrustWeave extends Linux IMA to support runtime integrity measurement in Intel TDX environments by integrating with TDX's RTMR registers. This enables runtime integrity of dynamically loaded components, bridging the trust gap left by TDX's boot-only attestation model for agent systems.
- **File-Granularity Per-Application Integrity Policies.** TrustWeave introduces support for application-scoped integrity policies that precisely target relevant executables and dependencies, reducing attestation noise and improving the clarity and precision of trust decisions in multi-agent deployments.
- **Acceptable Overhead on Agent Workload Performance.** Through extensive evaluation on LLM agent inference workloads, TrustWeave demonstrates that

its integrity measurement and attestation mechanisms incur operationally acceptable overhead for security-sensitive deployments, preserving boot-time latency, runtime latency, throughput, and model accuracy while enabling trusted agent orchestration.

2 Background & Motivation

Multi-Agent Systems. The rapid advancement of LLMs has catalyzed the emergence of multi-agent systems capable of complex reasoning, planning, and task execution [18, 41]. These agents extend beyond simple LLM inference by incorporating perception modules for environmental understanding, memory systems for context retention, and tool-calling capabilities for interacting with external services. Multi-agent systems further amplify these capabilities by enabling specialized agents to collaborate, for instance, in autonomous driving scenarios, perception agents process sensor data while planning agents determine navigation strategies, and safety agents verify decisions before execution [54]. Modern agent frameworks like AutoGen [44] and MetaGPT [8] demonstrate how multiple LLM-based agents can coordinate through structured communication protocols to solve problems beyond individual agent capabilities.

However, deploying multi-agent systems in production environments introduces significant security challenges that existing frameworks inadequately address. Agents must establish trust relationships before sharing sensitive information or coordinating on critical tasks, yet current systems lack mechanisms for agents to verify each other's integrity and authenticity. The dynamic nature of agent instantiation, where new agents are spawned on-demand to handle specific tasks, requires rapid trust establishment that traditional authentication methods cannot provide. Furthermore, the integration of agents with external tools and services creates an expanded attack surface where compromised agents could manipulate shared state, inject malicious prompts into collaborative workflows, or exfiltrate sensitive data through seemingly legitimate tool invocations. These security gaps necessitate a comprehensive approach that combines hardware-based isolation with dynamic runtime measurement to ensure agent integrity throughout their operational lifecycle.

Trusted Execution Environment. Trusted Execution Environments (TEEs) have emerged as a critical security technology for protecting sensitive computations in untrusted environments. Modern TEEs such as Intel TDX [4], Intel SGX [19], ARM TrustZone [2], ARM Confidential Compute Architecture (ARM CCA) [50], AMD SEV [14], AMD SEV-SNP [30], and RISC-V Keystone [17] provide hardware-enforced isolation that protects code and data from privileged adversaries including malicious operating systems and hypervisors. Over the past two decades, TEEs have drawn significant academic and industry interest [5, 13, 17, 25, 32, 36, 38, 39]. Intel TDX specifically isolates entire virtual machines as Trust Domains

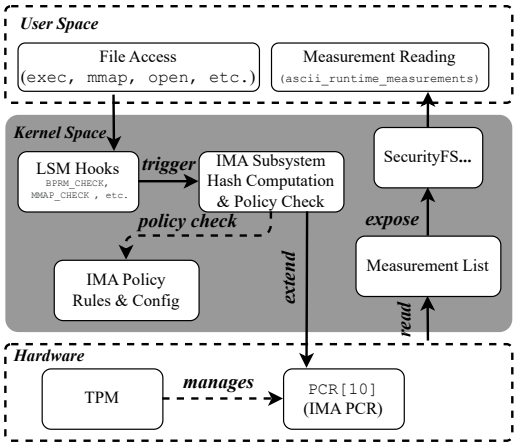


Figure 1. IMA Architecture: File access triggers measurement collection through LSM hooks. Measurements are stored in a kernel list and extended to TPM PCRs. The measurement list is exposed via securityfs for attestation by remote challengers.

(TDs) with memory encryption and CPU state protection, enabling secure cloud deployments where the infrastructure provider is not trusted. The cornerstone of TDX's security model is remote attestation [27, 29], which generates cryptographically-signed `TDREPORT` structures containing boot-time measurements that remote parties can verify to establish trust in the TD's initial state.

Despite these capabilities, TDX's attestation remains fundamentally limited to boot-time measurements, offering no native mechanism for verifying the integrity of software loaded after VM initialization. This gap becomes critical in dynamic agent environments where applications, libraries, and models are loaded on-demand throughout the system's lifecycle. The absence of runtime attestation means that remote parties cannot verify whether an initially-trusted TD has been compromised through runtime attacks such as code injection or malicious library loading. While agents' decision-making logic is typically lightweight and well-suited for CPU execution, they frequently need to offload computationally intensive tasks such as large model inference to accelerators, this architectural pattern necessitates TEEs that can provide strong isolation and attestation for the agent's control plane while enabling secure delegation to untrusted computational backends, undermining the continuous trust requirements essential for long-running services and dynamic workload orchestration.

Integrity Measurement Architecture. The IMA [10, 28] is a Linux kernel subsystem designed to create a chain of trust from the operating system to applications by measuring files before they are accessed. As illustrated in Figure 1, the IMA framework operates through a well-defined workflow spanning user space, kernel space, and hardware. When an application attempts a file access operation such as executing a binary (`exec`) or mapping a library into memory (`mmap`), these

Table 1. Comparison of Integrity and Attestation Approaches

	TPM Secure Boot	CVM (TDX/SEV/CCA)	OS-level Runtime IMA
Granularity	Full OS	Whole VM	System-wide
Lifecycle	Boot-only	Boot-only	Full
Verifies Dyn. Weights	✗	✗	✓*
Runtime Overhead	–	–	Policy-dep.†

*Yes, but requires careful policy scoping to reduce noise.
†Configurable, but manually deriving per-application policies for dynamic agent workloads is operationally difficult.

operations are intercepted by LSM hooks in kernel space that trigger the IMA subsystem. The subsystem consults policy rules to determine if the accessed file should be measured, and if required, computes a cryptographic hash that is extended into a hardware TPM's Platform Configuration Register (PCR), such as `PCR[10]`, creating an immutable, hardware-backed log of runtime events. To enable remote verification, the kernel maintains a measurement list exposed via SecurityFS (typically at `/sys/kernel/security/ima/ascii_runtime_measurements`), allowing external attestation services to retrieve this log along with a signed quote of the PCR values from the TPM.

However, in confidential computing environments like Intel TDX, the guest VM is isolated from the host's TPM, rendering this traditional architecture ineffective. This limitation becomes particularly acute for agent systems that require dynamic runtime measurements to maintain trust during dynamic agent instantiation and collaboration. The inability to leverage existing IMA infrastructure means that agents running within TDs cannot provide verifiable evidence of their runtime integrity, even though the underlying kernel subsystem is capable of capturing the necessary measurements. This gap necessitates an alternative mechanism for secure measurement storage that can bridge IMA's measurement capabilities with TDX's attestation infrastructure, which motivates our work in extending IMA to leverage TDX's native Runtime Measurement Registers (RTMRs).

Comparison with Existing Approaches. Table 1 contrasts existing integrity and attestation approaches across key dimensions. Only TrustWeave combines per-application granularity and full-lifecycle TDX-bound attestation with automated policy scoping.

3 Threat Model

TrustWeave targets multi-agent systems in multi-tenant clouds where agents require rapid instantiation and dynamic trust establishment. We consider adversaries controlling the hypervisor and guest OS attempting to: compromise agent execution through memory access or code injection; deploy malicious agents for identity spoofing or data extraction; manipulate orchestration layers to inject compromised instances; and perform runtime tampering with models and dynamic libraries. Adversaries can intercept network traffic, perform memory dumps, and modify any software outside the TD boundary, but cannot break cryptographic primitives or compromise TDX hardware guarantees including RTMR integrity.

We trust Intel TDX for hardware-enforced isolation and RTMR-based measurements, the remote attestation service to correctly validate RTMR values, and initial agent frameworks though they require dynamic runtime measurement via IMA integration. We do not trust the cloud orchestration platform or middleware. The guest OS is untrusted before attestation; once its integrity is verified through RTMR-based attestation, the attested guest kernel becomes part of the Trusted Computing Base (TCB). Agent providers are trusted to supply correct policies and measurements, while users must employ encrypted protocols. TrustWeave achieves dynamic runtime integrity via IMA-driven measurements, fast secure instantiation through policy-scoped TCB isolation, and inter-agent attestation using attestation-gated channels. We do not address denial-of-service, physical, or side-channel attacks.

Trusted Computing Base. We explicitly enumerate TrustWeave's TCB, distinguishing system-level and application-level components:

- **Hardware:** Intel TDX module (Secure Arbitration Mode, SEAM), CPU microcode, and RTMR registers providing the hardware root of trust.
- **Firmware:** TD virtual firmware (TDVF/OVMF), which performs boot measurements into MRTD and RTMR[0–1].
- **Guest kernel:** Linux kernel 6.6 with TrustWeave's IMA-RTMR patches, including the vTPM abstraction layer and policy filter. The guest kernel is untrusted before attestation; once measured into RTMR[1] at boot and verified, it becomes part of the TCB. Dynamically loaded kernel modules (`.ko`) are measured into RTMR[2] at runtime.
- **Attestation infrastructure:** The remote attestation service (e.g., Intel Trust Authority) that validates RTMR quotes against reference values.

4 Design: TrustWeave

To bridge the runtime attestation gap in TEEs, we designed and built TrustWeave, a system that extends the Linux Integrity Measurement Architecture to provide practical, fine-grained, and efficient integrity verification for cloud-native workloads. Our design is guided by three core principles:

- **TEE-Native Measurement:** We adapt IMA to be fully compatible with TEEs by transparently redirecting integrity measurements to TEE-native hardware roots, overcoming the limitations of traditional TPM-based approaches.
- **Policy-Driven Application Attestation:** We introduce an application-centric framework that enables

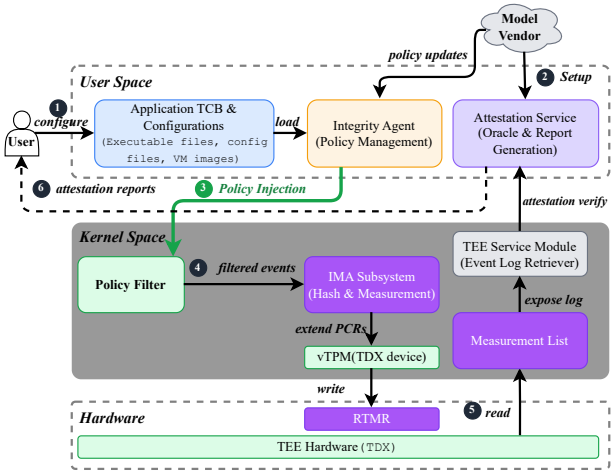


Figure 2. Two-phase attestation process in TrustWeave: (1) Event log generation collects and extends integrity measurements from application and device TCBs via IMA into RTMR/TPM. (2) Event log validation retrieves and checks measurements against policy-defined trust roots.

expressive, policy-driven control over the measurement scope, focusing verification on the components critical to a specific workload.

- **Efficient and Performant Filtering:** We design a novel, multi-layered filtering mechanism to minimize the significant performance overhead associated with hardware-based measurement, making dynamic runtime integrity measurement practical.

4.1 System Overview

The high-level architecture of TrustWeave, shown in Figure 2, orchestrates a secure workflow between user space, the kernel, and the underlying TEE hardware. The process establishes a verifiable chain of trust from the application down to the silicon.

Setup and Policy Injection. The process begins with a two-part setup. First, a trusted entity, such as a **Model Vendor**, provides the expected integrity measurements (the "oracle" or baseline) to a remote **Attestation Service** (❷). This service is responsible for report generation and verification. Concurrently, a **User** provides the application's Trusted Computing Base (TCB) including executables, configurations, and dependencies to the system (❶). A user-space **Integrity Agent** then translates the user's requirements into a declarative policy and injects it into the kernel's **Policy Filter** (❸). This policy dictates precisely which TCB components must be measured to be considered trusted.

Filtered Measurement and Hardware Extension. Once the policy is active, the kernel's enhanced **IMA Subsystem** begins enforcement. It intercepts all relevant file accesses and consults the Policy Filter to determine if a measurement is required. For filtered events, IMA computes a cryptographic hash. Crucially, instead of interacting with a traditional TPM, our design redirects this measurement to a `vTPM (TDX device)` driver. This driver translates the request and extends the measurement into a hardware-backed **RTMR** (❹), securely anchoring the runtime state in the TEE hardware.

Attestation and Verification. The kernel maintains an in-memory **Measurement List** (event log) of all performed measurements. A dedicated **TEE Service Module** acts as a log retriever, exposing this list securely to the user space (❺). To complete the loop, a remote verifier can request a hardware-signed quote from the Attestation Service. This service retrieves the measurement log, generates the quote, and compares the evidence against the pre-registered oracle. The final verification result is then returned to the User or challenger (❻), confirming whether the workload is running with the expected integrity.

4.2 Kernel Support for TEE-Native Measurement

A fundamental challenge in TDX is the inaccessibility of the host's TPM. Our primary kernel-level contribution is adapting IMA to function seamlessly in this environment by redirecting measurements to TDX-native RTMRs.

Hardware Abstraction Layer. Our approach creates a minimal abstraction layer that presents TDX RTMRs as TPM-compatible measurement registers. The backend creates a virtualized TPM chip structure during initialization that contains a single SHA-384 crypto bank matching TDX's native capabilities. This design handles fundamental differences between TPM PCRs and TDX RTMRs: register count limitations (4 RTMRs vs 24 PCRs), hash algorithm restrictions (SHA-384 only), and access patterns (direct hardware TDCALL vs hypervisor-mediated TPM/vTPM operations). By providing this abstraction, existing IMA code can operate without modification while benefiting from TDX's hardware-isolated measurement capabilities.

RTMR Backend Implementation. Our RTMR backend serves as the translation layer between IMA's TPM-oriented interface and TDX hardware. When IMA requests measurement operations, the backend performs several critical steps: validating that only RTMR index 2 is used for application measurements, preserving RTMR[0] and RTMR[1] for system-level measurements, performing the required 64-byte memory alignment that TDX hardware demands, and executing the actual extension through hardware instructions that communicate directly with the TDX module. The backend implements error handling for TDX-specific conditions including scenarios where RTMRs are busy due to concurrent access attempts and invalid operands that don't meet TDX ABI requirements.

Context-Aware IMA Integration. We modify IMA's core measurement infrastructure to detect the execution environment and route measurements accordingly. The detection logic operates during boot-time initialization using processor-specific capability detection that identifies TDX support. The system verifies that SHA-384 is configured as the system hash algorithm and configures global state that directs all subsequent measurement operations through the RTMR backend. This approach preserves backward compatibility by ensuring that systems without TDX support continue using traditional TPM measurements without any configuration changes.

Boot Measurement Integration. TDX environments require special handling of boot measurements to establish the complete chain of trust from hardware initialization through application execution. Our implementation integrates TDX boot measurements by leveraging TDX hardware capabilities to retrieve system state information during initialization. The system obtains detailed reports containing current values of all measurement registers including the Trust Domain Measurement Register that captures the initial VM configuration and all Runtime Measurement Registers that track subsequent system state changes. We extract these foundational measurements and incorporate them into IMA's boot aggregate calculation using cryptographically secure methods.

Enhanced Event Logging Architecture. The integration extends beyond simple measurement redirection to include event logging enhancements that support both traditional IMA workflows and TDX-specific attestation requirements. We implement a dual-log architecture that maintains compatibility with existing tooling while enabling advanced TDX capabilities. The traditional IMA event log remains accessible through standard kernel interfaces, while we maintain a TDX-specific event log that records all RTMR extensions with TDX-native metadata. Each TDX event includes structured headers with monotonically increasing event counters that enable verifiers to detect missing or reordered events.

4.3 Policy-Driven Application-Level Attestation

Traditional IMA already supports policy rules based on filesystem UUID (`fsuuid`), SELinux labels, file owner (`fowner`), and path prefixes, providing coarse-grained scoping of measurements. However, these mechanisms were designed for host-level security policies rather than cloud-native application workloads. TrustWeave builds on this foundation with per-application file-path policies, content-hash deduplication to eliminate redundant RTMR extensions, and RTMR integration that binds measurements to TDX hardware rather than a TPM inaccessible from within the TD, as detailed in Figure 3.

Application-Centric TCB. Our approach focuses on defining and measuring a precise TCB for each workload. This includes not only the main **Application Binaries** but also all runtime **Application Dependencies** (e.g., shared libraries)

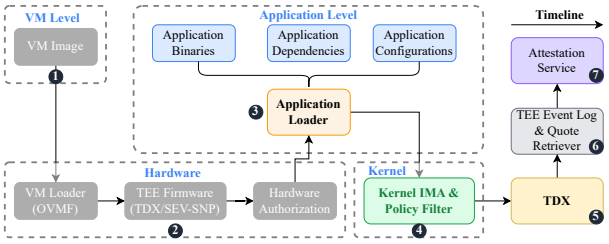


Figure 3. Workflow of Policy-Driven Attestation

and critical **Application Configurations**. For complex workloads like LLMs, the integrity of model weights and tokenizer configuration files is as crucial to the TCB as the executable itself.

This precision is vital for reducing attestation "noise". A verifier for an LLM application cares about the LLM's files, not about unrelated system daemons that might also be running in the VM. By scoping the policy to only the relevant components, TrustWeave produces a clean, concise, and meaningful attestation report that directly addresses the security concerns of the application owner.

The Application Loader's Role in Policy Generation. In our system, the **Application Loader** is the orchestrator of this process. When a VM spawns a new application (❷), the loader's responsibility is to ensure a complete and accurate IMA policy is defined for the workload's TCB.

This list of files and their expected hashes is then passed to the Integrity Agent for injection into the kernel. The loader then initiates the application's execution, triggering the kernel's **IMA & Policy Filter** module to begin its measurement and enforcement duties (❸). The resulting application-specific logs are exposed by the **TEE Event Log Retriever** (❺) for remote attestation, completing the workflow.

Kernel Enforcement and Log Retrieval. The kernel's **IMA & Policy Filter** module acts as the enforcement point, ensuring that only files matching the active policy are measured. After measurement, the log of these events is exposed securely by the **TEE Event Log Retriever**. This module is designed to handle requests for attestation evidence, packaging the relevant log entries for the remote **Attestation Service** (❺). By focusing measurement and reporting on the application's specific TCB, this design provides a clear and uncluttered attestation report that is directly meaningful to the application owner.

4.4 Integrity Agent and Policy Management

Central to our design for application-centric attestation is the user-space **Integrity Agent**. This component is responsible for managing the measurement policy lifecycle, acting as the bridge between user intent and kernel enforcement.

Policy Definition and Scoping. The Integrity Agent consumes a high-level definition of an application's TCB. This TCB is not just a list of executables but a comprehensive manifest including all required dependencies, configuration files,

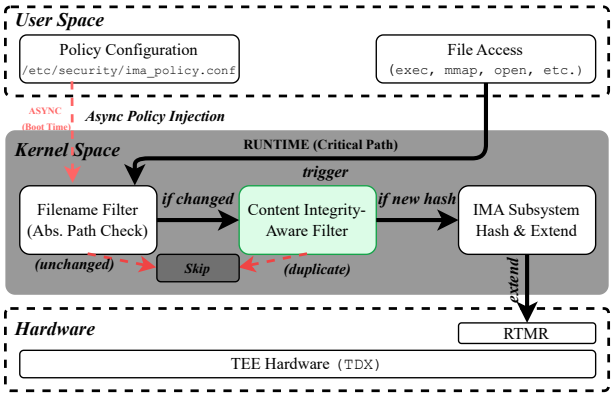


Figure 4. Two-Layer Filtering Mechanism. Policy is injected asynchronously. At runtime Filename Filter and Content-Aware Filter work in tandem to skip redundant RTMR 'extend' operations, minimizing performance overhead.

and even data artifacts like AI model weights. The agent is responsible for translating this manifest into a precise, low-level IMA policy that the kernel can enforce.

In a practical deployment, the agent would work in concert with a higher-level orchestrator, such as **Application Loader**. The loader could analyze a container image or software package to auto-generate the TCB manifest, which is then passed to the Integrity Agent. This separation of concerns allows for a flexible system where policies can be generated automatically or crafted manually, depending on the use case.

Asynchronous Policy Injection. A key design choice is how this policy is delivered to the kernel. To avoid impacting the runtime critical path, the policy is loaded *asynchronously at boot time*, as shown in Figure 4. The Integrity Agent (or a script acting on its behalf) utilizes an `initramfs` hook to inject the policy rules into the kernel's policy filter before any user-space processes are started.

This boot-time loading model provides a stable and predictable policy baseline for the VM's lifecycle. It ensures that measurement enforcement is active from the earliest moments of the system's operation and that there is zero configuration overhead during runtime, which is critical for performance-sensitive applications.

4.5 Optimized with Two-Layer Filtering

To make dynamic runtime measurement practical, TrustWeave incorporates an intelligent, two-layer filtering mechanism within the kernel that reduces the overhead of hardware interactions.

Layer 1: Filename-Based Caching The first layer of defense against redundant work is a fast **Filename Filter**. This filter maintains a cache of files based on their absolute paths and associated metadata. When a file access is intercepted by an IMA hook, the kernel first checks this cache. If the file has been measured previously and its metadata indicates it is *unchanged* (e.g., via modification time or inode number), the entire measurement and extension process is skipped.

This cache is highly effective for frequently accessed, static files such as system libraries or common binaries. By avoiding repeated hashing and policy lookups for these files, this layer significantly reduces the CPU overhead of IMA on the system's critical I/O path.

Layer 2: Content-Aware Hash Filtering If a file is new or has been modified, it passes the first filter and proceeds to the **Content Integrity-Aware Filter**. This filter checks if the file's computed hash is a duplicate of one already extended to the RTMR. A match in this second-level cache indicates that the file's content is trusted, but its hash has already been recorded (e.g., from a different file with identical content).

Because extending a duplicate hash would not alter the state of the RTMR, TrustWeave makes a critical optimization: it *bypasses the expensive hardware* `extend` *operation*. The `extend` is only performed for genuinely new hashes that represent a true change in the system's measured integrity state. This content-aware filtering is key to minimizing the frequency of costly `TDCALL`s, enabling dynamic runtime measurement with acceptable performance.

5 TrustWeave: Implementation Details

We implement TrustWeave on Linux kernel 6.6's IMA, adding 2k+ LoCs for core functions. The implementation creates a TPM-compatible abstraction presenting TDX RTMRs to IMA, handling key differences (4 RTMRs vs 24 PCRs, SHA-384 only). The upper layer (policy enforcement, filtering) is hardware-agnostic, while the lower layer extends the TPM driver interface to support other TEE primitives. The attestation layer needs to implement per-platform parsing and validation logic. RTMR[2] is reserved for application measurements with serialized `TDG.MR.RTMR.EXTEND` operations. Our agent-specific policies measure critical components including container runtime binaries (`dockerd`, `containerd`, `runc`), container metadata (`/var/lib/docker/containers/*/hostconfig.json`), and model artifacts (`tokenizer.json`, `*.safetensors` weights). Representative filtered policy rules include:

```
measure func=FILE_CHECK filename=/usr/bin/containerd
measure func=FILE_CHECK filename=/etc/containerd/config.toml
measure func=MMAP_CHECK mask=MAY_EXEC dir=/usr/lib/python3/
measure func=FILE_CHECK filename=/root/.cache/huggingface/hub/models--Qwen--Qwen3-8B/model.safetensors
```

This filtered policy is loaded at boot time via an `initramfs` hook and remains immutable for the TD's lifetime. The system maintains standard IMA compatibility, requiring no changes to existing attestation tools.

Host secure boot is not required; trust is rooted in TDX hardware attestation of the guest's initial state and TrustWeave's runtime measurements. Quotes are produced and signed by the Intel Quoting Enclave (QE) using an attestation

key provisioned via Intel's Provisioning Certification Service (PCS). A verifier (or Intel Trust Authority as a verification service) validates the quote signature and certificate chain up to Intel root certificates.

6 Security Analysis

We analyze how TrustWeave addresses the agent-specific threat vectors identified in § 3, demonstrating our design's effectiveness in enabling trusted agent operations.

6.1 Attack Vector Analysis

Table 2 summarizes potential attack vectors against agent systems and TrustWeave's corresponding mitigations. Against malicious agents, TrustWeave employs RTMR-based attestation and dynamic runtime measurements to prevent identity spoofing and state tampering.

Table 2. Agent attack vectors and TrustWeave mitigations

Attack Vector	Mitigation
From Malicious Agents	
Identity spoofing	RTMR-based attestation
State tampering	Dynamic runtime measurements
Communication hijacking	Attestation-gated channels
From Agent Orchestration Layer	
Agent injection	Boot-time policy enforcement
Config manipulation	Configuration measurements
Scaling attacks	Per-instance attestation
From Agent Runtime	
Model replacement	Model weight measurements
Code injection	Dynamic library tracking
Memory attacks	TDX encryption + isolation

Against orchestration-layer threats, TrustWeave enforces boot-time policies to block unauthorized agent injection and measures configuration files to detect tampering, with each agent instance receiving unique attestation credentials. Runtime manipulation is prevented through dynamic runtime measurement of model weights and dynamic libraries, complemented by TDX's memory encryption and behavioral anomaly detection via measurement patterns. At the hypervisor level, TDX's reverse map table and direct RTMR operations prevent unauthorized memory access and measurement tampering, while hardware mechanisms block malicious interrupt injection.

Concrete Attack Scenario. Consider a supply-chain attack where an adversary compromises a PyPI mirror used during container build and replaces the `transformers` library with a backdoored version that exfiltrates model outputs to a remote server. *Without TrustWeave*, the CVM boots normally and TDX boot-time attestation succeeds, since boot measurements (MRTD, RTMR[0–1]) are unaffected by runtime library content. The malicious library loads via `mmap` at runtime and operates undetected. *With TrustWeave*, the modified `transformers` package produces a different SHA-384 hash when intercepted by the `MMAP_CHECK` hook. This hash is extended into RTMR[2] and recorded in the event log. When a remote verifier requests an attestation quote, it compares RTMR[2] against the reference value registered by the model vendor and detects the divergence, blocking the compromised agent from joining the multi-agent workflow. Similarly, model substitution (replacing a 14B-parameter model with a smaller 8B variant) is detected because the `.safetensors` weight files produce entirely different hashes.

6.2 Formal Security Properties

TrustWeave provides four verifiable security properties essential for trusted agent deployments:

Property 1: Measurement Completeness. Every file operation matching an agent's TCB policy generates an RTMR[2] measurement with the guarantee:

$$\forall f \in \mathrm{TCB}_{\mathrm{policy}} : \ \mathrm{access}(f) \ \Rightarrow \ \mathrm{extend}(\mathrm{RTMR}[2], H(f))$$

where H denotes SHA-384 hashing. Mutex serialization prevents measurement loss due to concurrent access.

Property 2: Measurement Integrity. The RTMR chaining mechanism ensures tamper evidence through:

$$\mathrm{RTMR}_{\mathrm{new}} \ = \ \mathrm{SHA384}(\mathrm{RTMR}_{\mathrm{old}} \parallel M_{\mathrm{new}})$$

Any tampering with prior measurements invalidates all subsequent RTMR values, creating an unforgeable chain.

Property 3: Attestation Authenticity. The Quoting Enclave signs attestation quotes with a hardware-provisioned key inaccessible to software, ensuring:

$$\mathrm{Verify}(\mathrm{Report}, \mathrm{Cert}) = \mathrm{true} \iff \mathrm{Report} \in \mathrm{HW\_Gen}$$

Property 4: Policy Immutability. Once loaded at boot, measurement policies cannot be modified without TD restart, enforcing:

$$\mathrm{Policy}_t = \mathrm{Policy}_0, \ \forall\, t > t_{\mathrm{boot}}$$

7 Evaluation

We evaluate TrustWeave by analyzing its boot time performance for agent instantiation, runtime overhead, and model accuracy preservation. We aim to answer the following questions:

Q1: How much overhead does TrustWeave add to agent boot time and instantiation? (§ 7.2)

Q2: What is the performance impact on LLM inference during cold start? (§ 7.3)

Q3: Can TrustWeave maintain acceptable performance under sustained production workloads? (§ 7.4)

Q4: Does integrity monitoring affect model inference accuracy? (§ 7.5)

7.1 Experimental Setup

Testbed. We conduct our experiments on an Intel TDX-enabled server with an Intel Xeon Gold 6530 CPU (32 cores, 2.1GHz base frequency) and 125 GB of DDR5 DRAM. The host server runs Ubuntu 24.04 with an Intel TDX-enabled Linux kernel (v6.8.0), with both sub-NUMA clustering and hyperthreading disabled to ensure consistent performance measurements. Each test deploys a single QEMU VM instance (either standard or confidential) running Ubuntu 22.04 with a modified Linux kernel (v6.6) supporting runtime measurement extensions. Within each VM, we deploy exactly one Docker container running vLLM v0.9.0, maintaining a strict 1:1 mapping (per-tenant isolation).

Agent Workloads. We evaluate TrustWeave using representative AI agent workloads, specifically LLM inference tasks that typify agent computational requirements. We test five models ranging from 0.6B to 14B parameters: Qwen3-0.6B, Qwen3-8B, DeepSeek-R1-Distill-Llama-8B, DeepSeek-R1-Distill-Qwen-14B, and Phi-4-reasoning. These models represent the diversity of workloads that trusted agents may execute, from lightweight coordination tasks to complex reasoning operations.

Baselines. We compare TrustWeave against four baseline configurations:

- **Vanilla VM**: Standard virtual machine without confidential computing
- **Vanilla cVM**: Confidential VM with Intel TDX
- **TrustWeave w/o Filter**: TrustWeave without optimized IMA filtering
- **TrustWeave w/ Filter**: Full TrustWeave implementation with IMA filter optimization

For each configuration, we evaluate both *standard mode* and *measurement-intensive mode*, where TDX extend operations are continuously performed during execution at a specified rate.

7.2 Boot Time and Agent Instantiation Overhead

We measure the complete boot sequence from VM instantiation to agent service readiness, capturing the time required for on-demand agent deployment in cloud environments. The boot process consists of six distinct stages: (1) **VM→Docker**: QEMU VM startup to Docker service ready, (2) **Docker→vLLM**: Container initialization to vLLM framework launch, (3) **Initial Config**: vLLM configuration and environment setup, (4) **Model Loading**: Loading model weights into memory, (5) **Engine Init**: Inference engine initialization, and (6) **API Startup**: REST API service activation. This comprehensive measurement captures the actual time from cold start to fully operational agent service.

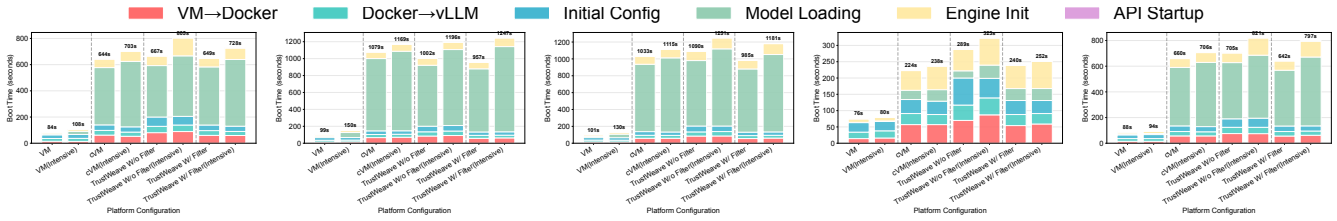
Full Process Boot Time. Figure 5 reveals the critical impact of filtering on boot performance. For DeepSeek-R1-Llama-8B, TrustWeave without filter requires 667.1s, measuring every file access adds 22.9s overhead compared to vanilla cVM (644.2s). However, with filtering enabled, boot time drops to 649.4s, nearly matching vanilla cVM with only 0.8% overhead. This 17.7s improvement demonstrates filtering's effectiveness. The larger DeepSeek-R1-Qwen-14B shows even more dramatic results: without filter takes 1002.2s, but with filter achieves 956.9s, comparable to vanilla cVM's 1079.1s. The filter eliminates 45.3s of measurement overhead by caching frequent file accesses during model loading. Under measurement-intensive workloads, filtering remains essential: DeepSeek-R1-Llama-8B without filter balloons to 805.3s versus 728.3s with filter, a 77s reduction through intelligent measurement deduplication.

Infrastructure-Only Boot Time. Figure 6 isolates infrastructure overhead without model loading, clearly showing the filter's impact. For DeepSeek-R1-Llama-8B, TrustWeave without filter requires 274.7s, 32.6% overhead versus vanilla cVM (207.1s), as it measures every library and configuration file during container initialization. With filtering, this drops to 205.8s, achieving near-identical performance to vanilla cVM. The 68.9s reduction (from 274.7s to 205.8s) demonstrates that most infrastructure files are accessed repeatedly and benefit from caching. For DeepSeek-R1-Qwen-14B, the pattern repeats: without filter takes 281.8s versus 210.4s with filter, a 71.4s improvement. With filtering, TrustWeave achieves 210.4s, comparable to vanilla cVM's 224.8s, showing that selective measurement can match the performance of systems without integrity verification. Under measurement-intensive workloads, the filter reduces DeepSeek-R1-Llama-8B from 343.2s to 218.8s, saving 124.4s through measurement optimization.

Key Insights. Three critical observations emerge from boot time analysis: (1) Filtering is essential, without it, TrustWeave adds 22-45s overhead for full boot and 69-71s for infrastructure alone, but with filtering, overhead drops to 0.8%, achieving near-identical performance to vanilla cVM; (2) The filter achieves 68-124s reduction in boot time by caching repeated file measurements, demonstrating that most boot-time file accesses are redundant; (3) TrustWeave with filter achieves comparable performance to vanilla cVM across all configurations, proving that integrity measurement need not compromise boot performance when properly optimized. These results validate our two-layer filtering design as crucial for production viability.

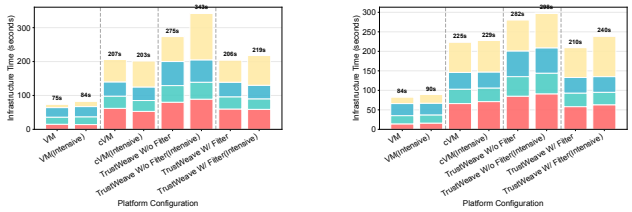
7.3 Cold-Start Inference Performance

We evaluate the inference performance of LLM agents in cold start scenarios, representing the worst-case deployment where agents are instantiated from scratch to handle requests. Each experiment measures three key metrics: (1) Time-to-First-Token (TTFT), which captures the initial response latency critical for interactive agent applications; (2) Decoding Tokens Per Second (TPS), measuring the token



(a) DS-R1 Llama-8B **(b)** DS-R1 Qwen-14B **(c)** Phi-4 reasoning **(d)** Qwen3-0.6B **(e)** Qwen3-8B

Figure 5. Boottime comparison across models.



(a) DeepSeek-R1 Llama-8B **(b)** DeepSeek-R1 Qwen-14B

Figure 6. Boottime comparison (infrastructure-only) across representative models.

generation rate after the first token; and (3) Overall TPS, representing end-to-end throughput. We test across varying input sequence lengths (32-2048 tokens) to understand performance scaling characteristics. All measurements are taken after complete cold boot, container initialization, and model loading, representing real-world agent deployment scenarios.

TTFT Analysis. Figure 7 shows TTFT scaling for three representative models. At 2048 tokens, vanilla VM achieves 11.4-14.5s while TrustWeave with filter ranges from 15.5-17.8s, adding 36-56% overhead. At 8192 tokens, vanilla VM requires 22.0-32.9s versus TrustWeave's 31.0-66.8s. While overhead increases with sequence length, it remains acceptable for secure agent deployments.

Decoding Performance. Table 3 reveals an interesting performance trend. At shorter sequences (512-2048 tokens), TrustWeave surprisingly outperforms vanilla platforms, at 2048 tokens. This unexpected advantage likely results from TDX's bounce buffer design, which can improve memory access patterns and cache utilization for moderate workloads. However, this benefit degrades with longer sequences. By 8192 tokens, TrustWeave falls below vanilla VM performance. Notably, filtered and unfiltered configurations show nearly identical performance, indicating measurement overhead has minimal impact on token generation.

Overall Throughput. Figure 8 and Table 4 show TrustWeave maintains strong performance advantages. At 2048 tokens, TrustWeave with filter achieves 132-155% of vanilla VM throughput across all models. At 8192 tokens, TrustWeave delivers 79-81% of vanilla VM performance while providing continuous integrity verification, demonstrating practical viability for production agent deployments.

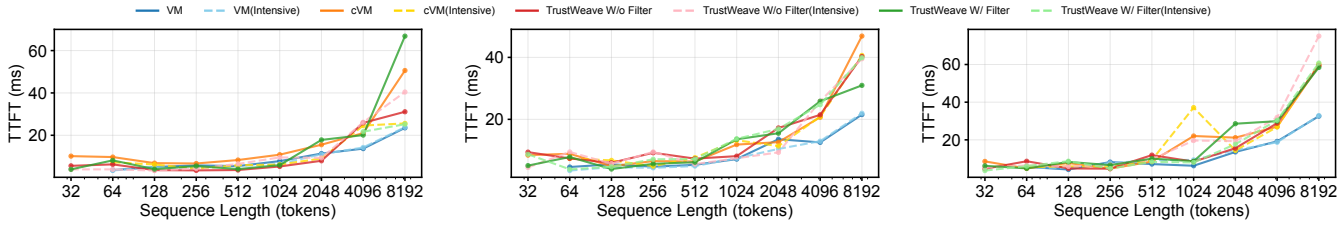
Table 3. Decoding TPS at Representative Sequence Lengths (tokens/second)

Model	Platform	512	2048	4096	8192
Qwen3-8B	Vanilla VM	3.22	3.03	5.30	5.77
	Vanilla cVM	3.30	3.19	4.63	4.43
	TrustWeave w/o Filter	5.86	5.11	4.45	4.01
	TrustWeave w/ Filter	5.92	5.86	4.20	4.23
DeepSeek-8B	Vanilla VM	3.63	3.44	5.99	5.30
	Vanilla cVM	3.75	4.95	5.76	5.00
	TrustWeave w/o Filter	5.93	5.94	4.89	4.94
	TrustWeave w/ Filter	6.57	5.16	5.26	4.66
Phi-4	Vanilla VM	2.38	2.40	4.19	3.82
	Vanilla cVM	2.59	2.48	4.06	3.49
	TrustWeave w/o Filter	4.09	4.12	3.45	3.27
	TrustWeave w/ Filter	4.11	4.12	3.46	3.14

Table 4. Overall TPS Performance Across Sequence Lengths

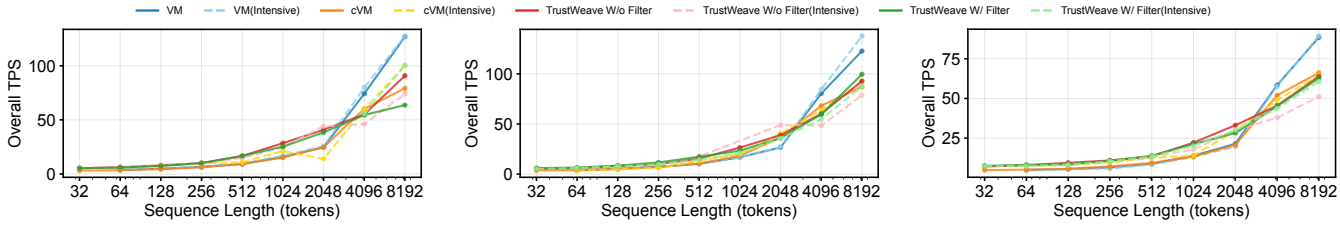
Model	Platform	4096	8192	Rel. (2048)
Qwen3-0.6B	Vanilla VM	217.96	362.70	,
	Vanilla cVM	174.99	266.13	87.6%
	TrustWeave w/o Filter	172.89	239.23	120.2%
	TrustWeave w/ Filter	141.03	196.77	150.7%
Qwen3-8B	Vanilla VM	74.19	126.96	,
	Vanilla cVM	60.12	79.21	100.0%
	TrustWeave w/o Filter	55.06	90.80	165.3%
	TrustWeave w/ Filter	54.50	100.56	155.4%
DS-LLaMa-8B	Vanilla VM	80.40	122.73	,
	Vanilla cVM	68.19	87.05	136.5%
	TrustWeave w/o Filter	60.12	92.81	145.6%
	TrustWeave w/ Filter	59.53	99.62	134.9%

Key Insights. Our evaluation reveals three critical findings: (1) TTFT overhead remains manageable, with TrustWeave adding predictable latency acceptable for production deployments; (2) TrustWeave outperforms vanilla VM in decoding performance at shorter sequences due to TDX's bounce buffer implementation that batches memory operations more efficiently, though this advantage diminishes at 8192 tokens where larger working sets exceed optimization benefits; (3)



(a) Qwen3-8B **(b)** DeepSeek-R1-Llama-8B **(c)** Phi-4-reasoning

Figure 7. Time-to-First-Token comparison across sequence lengths and representative models



(a) Qwen3-8B **(b)** DeepSeek-R1-Llama-8B **(c)** Phi-4-reasoning

Figure 8. Overall TPS comparison across sequence lengths and representative models.

overall throughput shows TrustWeave achieves competitive performance at moderate sequence lengths, maintaining 79–81% of baseline at 8192 tokens. These results show TrustWeave is particularly well-suited for typical agent workloads with moderate sequence lengths.

Measurement Overhead During Cold Start. Tables 5 and 6 quantify IMA measurement counts during cold-start inference. TrustWeave with filtering reduces runtime measurements to 18-28 entries (99.95% reduction), while unfiltered operation generates 36,510-60,586 measurements depending on workload intensity. At 8192 tokens, intensive workloads reach up to 68,171 measurements without filtering. The filtering mechanism proves essential, reducing measurement logs from several MB to under 1KB while maintaining comprehensive TCB integrity verification. Each IMA event log entry is approximately 120 bytes (48-byte SHA-384 digest plus path, metadata, and template fields). Unfiltered logs for Qwen3-8B at 2048 tokens thus total approximately 5.0 MB (42,628 entries), while the filtered log comprises only 22 entries totaling approximately 2.6 KB, a reduction of over three orders of magnitude in attestation evidence size. Using our verification script (Gather + Verify), validating a filtered attestation report takes hundreds of microseconds, confirming that filtering benefits both the measured system and the verifier side.

7.4 Runtime QPS Performance

To evaluate steady-state performance under sustained workloads, we conducted QPS testing using the ShareGPT dataset [1]. We sampled 200 conversation prompts following the original token length distribution, tokenizing each with model-specific tokenizers to preserve authentic input patterns. Tests ran on pre-warmed instances where models were fully loaded and engines initialized, eliminating cold-start effects. Each experiment maintained constant QPS rates from 2 to 32 using Poisson arrival patterns over 300 seconds after a 60-second warmup period. We measure Time to First Token (TTFT) for agent responsiveness and throughput (TPS) for system efficiency, capturing full CDFs to understand both median and tail behavior.

Table 5. IMA Measurement Counts During Cold-Start Inference (2048 Tokens)

Model	Vanilla VM	TrustWeave w/ Filter	TrustWeave w/o Filter (Normal)	TrustWeave w/o Filter (Intensive)
Qwen3-0.6B	1	18	36,510	40,079
Qwen3-8B	1	22	42,628	50,082
DS-LLaMa-8B	1	21	41,851	49,659
DS-Qwen-14B	1	21	47,522	60,586
Phi-4	1	28	47,338	58,987

Table 6. Measurement Count Scaling with Sequence Length

Model	Metric	512 tokens	2048 tokens	8192 tokens
Qwen3-8B	w/o Filter (Normal)	42,628	42,628	42,628
	w/o Filter (Intensive)	48,928	50,082	56,515
DeepSeek-8B	w/o Filter (Normal)	41,851	41,851	41,851
	w/o Filter (Intensive)	48,368	49,659	55,896
Phi-4	w/o Filter (Normal)	47,338	47,338	47,338
	w/o Filter (Intensive)	56,117	58,987	68,171

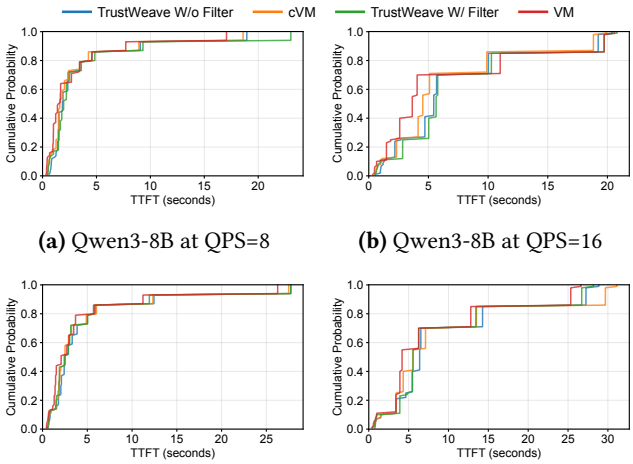
Performance Data. Figure 9 and Figure 10 show representative runtime performance for Qwen3-8B and DeepSeek-R1-Qwen-14B at QPS=8 and QPS=16, representing typical and large agent models. Across all tested models (Qwen3-0.6B/8B, DeepSeek-R1-Qwen-14B, Phi-4), we observe consistent patterns: TrustWeave with filtering maintains P50 TTFT within 15-20% of vanilla platforms while preserving

acceptable throughput. At QPS=16, P99 latencies remain comparable across platforms (20-28s range), demonstrating that filtering effectively controls measurement overhead. Without filtering, however, performance degrades significantly at high loads, with P99 latencies exceeding acceptable thresholds for interactive agent applications.

Latency Analysis. The TTFT measurements reveal TrustWeave's ability to maintain responsive performance under varying loads. Figure 9 shows latency CDFs for both models. For Qwen3-8B at QPS=8, all platforms exhibit similar P50 latencies: vanilla VM achieves 1.56s, vanilla cVM 1.60s, TrustWeave w/o filter 1.84s, and TrustWeave w/ filter 1.95s, demonstrating only 25% overhead with filtering enabled. At higher load (QPS=16), P99 tail latencies show greater divergence: vanilla VM reaches 20.60s, vanilla cVM 19.91s, while TrustWeave w/ filter maintains 20.81s, comparable to baseline platforms. For the larger DeepSeek-R1-Qwen-14B model at QPS=8, TrustWeave w/ filter achieves P50=2.48s versus vanilla VM's 2.08s (19% overhead), with P99 remaining within acceptable bounds at 27.74s versus 26.26s.

Throughput Trade-offs. While TrustWeave incurs throughput penalties due to measurement overhead, the impact remains manageable with filtering enabled. Figure 10 presents throughput CDFs at different QPS levels. For Qwen3-8B at QPS=8, vanilla VM achieves 4.18 TPS while TrustWeave w/ filter reaches 2.66 TPS (64% of baseline). At QPS=16, the gap narrows with vanilla VM at 3.35 TPS and TrustWeave w/ filter at 2.01 TPS (60% of baseline). For DeepSeek-R1-Qwen-14B at QPS=8, vanilla VM achieves 3.21 TPS versus TrustWeave w/ filter's 2.82 TPS (88% of baseline), demonstrating better relative performance for larger models where computation dominates. Importantly, TrustWeave provides more consistent performance, while vanilla platforms show high variance (std=0.74-1.18), TrustWeave with filtering exhibits tighter distributions (std=0.32-0.39), crucial for predictable agent response times.

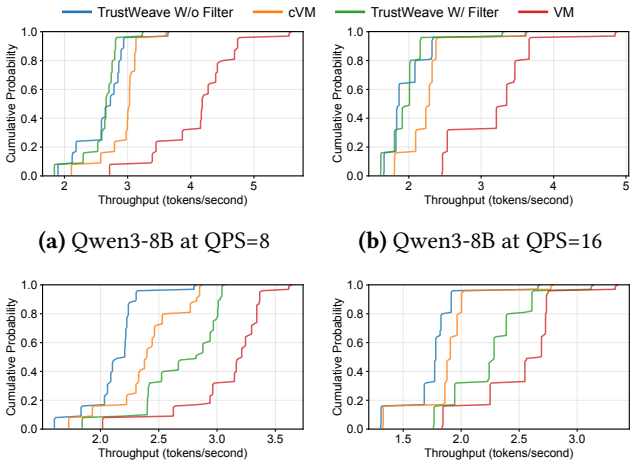
Scalability Insights. The performance data reveals interesting patterns across different virtualization layers. At lower QPS levels (QPS $\leq$ 8), both vanilla cVM and TrustWeave benefit from TDX's bounce buffer implementation for batched operations. For DeepSeek-R1-Qwen-14B at QPS=8, TrustWeave with filtering achieves 2.82 TPS, outperforming vanilla cVM's 2.40 TPS by 17.5%. This advantage likely stems from TrustWeave's optimized memory access patterns during measurement operations, which align well with TDX's batched cryptographic processing. The filtering mechanism proves essential for production viability, without it, measurement overhead scales linearly with request rate, causing system saturation above QPS 16. The two-layer filtering achieves >95% cache hit rate during steady-state operation, effectively decoupling measurement cost from request rate. At



(a) Qwen3-8B at QPS=8 **(b)** Qwen3-8B at QPS=16

(c) DS-R1-Qwen-14B at QPS=8 **(d)** DS-R1-Qwen-14B at QPS=16

Figure 9. TTFT CDF across models and QPS levels.



(a) Qwen3-8B at QPS=8 **(b)** Qwen3-8B at QPS=16

(c) DS-R1-Qwen-14B at QPS=8 **(d)** DS-R1-Qwen-14B at QPS=16

Figure 10. Throughput CDF across models and QPS levels.

QPS=32, TrustWeave with filtering maintains stable operations with P99 latencies of 33.97s for DeepSeek-R1-Qwen-14B (versus vanilla VM's 33.18s), demonstrating comparable performance under extreme load for high-concurrency agent deployments.

7.5 Model Accuracy Impact

To verify that TrustWeave's integrity monitoring does not affect model inference correctness, we evaluated all five models on the MMLU (Massive Multitask Language Understanding) benchmark using the lm-evaluation-harness framework. We configured the evaluation with 5-shot prompting to provide consistent few-shot examples for each task, ensuring fair comparison across platforms. The vLLM service backend handled inference requests, maintaining identical generation parameters (temperature=0, max tokens=100) across all configurations. Tests were conducted sequentially on each platform configuration (Vanilla VM, Vanilla cVM, TrustWeave

Table 7. MMLU Accuracy Across All Platforms and Categories (%)

Platform	Model	Overall	Humanities	Other	Social Sciences	STEM
Vanilla VM	Phi-4-reasoning	77.26 ± 0.75	80.00 ± 1.51	76.77 ± 1.55	83.00 ± 1.50	72.11 ± 1.39
	Qwen3-8B	76.46 ± 0.76	76.31 ± 1.61	76.31 ± 1.55	80.83 ± 1.56	73.89 ± 1.37
	DeepSeek-R1-Distill-Qwen-14B	74.53 ± 0.77	74.92 ± 1.61	75.54 ± 1.58	81.67 ± 1.52	69.05 ± 1.43
	DeepSeek-R1-Distill-Llama-8B	54.60 ± 0.90	56.92 ± 1.86	56.92 ± 1.89	62.17 ± 1.92	46.63 ± 1.60
	Qwen3-0.6B	46.84 ± 0.92	48.00 ± 1.91	48.92 ± 1.94	49.33 ± 1.99	43.05 ± 1.60
Vanilla cVM	Phi-4-reasoning	77.26 ± 0.75	80.00 ± 1.51	76.77 ± 1.55	83.00 ± 1.50	72.11 ± 1.39
	Qwen3-8B	76.46 ± 0.76	76.31 ± 1.61	76.31 ± 1.55	80.83 ± 1.56	73.89 ± 1.37
	DeepSeek-R1-Distill-Qwen-14B	74.53 ± 0.77	74.92 ± 1.61	75.54 ± 1.58	81.67 ± 1.52	69.05 ± 1.43
	DeepSeek-R1-Distill-Llama-8B	54.60 ± 0.90	56.92 ± 1.86	56.92 ± 1.89	62.17 ± 1.92	46.63 ± 1.60
	Qwen3-0.6B	46.84 ± 0.92	48.00 ± 1.91	48.92 ± 1.94	49.33 ± 1.99	43.05 ± 1.60
TrustWeave w/o Filter	Phi-4-reasoning	77.26 ± 0.75	80.00 ± 1.51	76.77 ± 1.55	83.00 ± 1.50	72.11 ± 1.39
	Qwen3-8B	76.46 ± 0.76	76.31 ± 1.61	76.31 ± 1.55	80.83 ± 1.56	73.89 ± 1.37
	DeepSeek-R1-Distill-Qwen-14B	74.53 ± 0.77	74.92 ± 1.61	75.54 ± 1.58	81.67 ± 1.52	69.05 ± 1.43
	DeepSeek-R1-Distill-Llama-8B	54.60 ± 0.90	56.92 ± 1.86	56.92 ± 1.89	62.17 ± 1.92	46.63 ± 1.60
	Qwen3-0.6B	46.84 ± 0.92	48.00 ± 1.91	48.92 ± 1.94	49.33 ± 1.99	43.05 ± 1.60
TrustWeave w/ Filter	Phi-4-reasoning	77.26 ± 0.75	80.00 ± 1.51	76.77 ± 1.55	83.00 ± 1.50	72.11 ± 1.39
	Qwen3-8B	76.46 ± 0.76	76.31 ± 1.61	76.31 ± 1.55	80.83 ± 1.56	73.89 ± 1.37
	DeepSeek-R1-Distill-Qwen-14B	74.53 ± 0.77	74.92 ± 1.61	75.54 ± 1.58	81.67 ± 1.52	69.05 ± 1.43
	DeepSeek-R1-Distill-Llama-8B	54.60 ± 0.90	56.92 ± 1.86	56.92 ± 1.89	62.17 ± 1.92	46.63 ± 1.60
	Qwen3-0.6B	46.84 ± 0.92	48.00 ± 1.91	48.92 ± 1.94	49.33 ± 1.99	43.05 ± 1.60

with filter, and TrustWeave without filter) using identical random seeds to ensure reproducibility.

Accuracy Results. Table 7 presents the complete accuracy results across all MMLU categories. The data demonstrates that TrustWeave has **zero impact on model accuracy**, all models achieve identical accuracy scores across all four platforms, with differences well within the statistical margin of error (±0.75-0.92% for overall accuracy). Phi-4-reasoning achieves the highest overall accuracy at 77.26%, followed by Qwen3-8B at 76.46% and DeepSeek-R1-Distill-Qwen-14B at 74.53%. The smaller models show expected accuracy degradation, with DeepSeek-R1-Distill-Llama-8B at 54.60% and Qwen3-0.6B at 46.84%, though both remain consistent across all deployment configurations.

Insights. The consistency of accuracy measurements across platforms validates a critical design principle: integrity monitoring operates transparently at the system level without interfering with model computation. Category-specific analysis reveals that all models perform best on Social Sciences questions (49.33-83.00% accuracy range) while STEM questions prove most challenging (43.05-71.00% range). This performance hierarchy remains identical across all platforms, confirming that TrustWeave's attestation mechanisms do not introduce computational artifacts or numerical instabilities. By maintaining bit-for-bit identical inference results, TrustWeave ensures that cryptographic security guarantees come without sacrificing inference quality, essential for production deployments where even minor accuracy degradation would be unacceptable for critical agent decision-making tasks.

8 Discussion

Policy Generation and Flexibility. TrustWeave's two-layer filtering mechanism enables flexible policy customization for diverse agent workloads. Users can manually define measurement policies based on their security requirements, specifying which files constitute the TCB for their specific agents. Alternatively, automated policy generation through application analyzers can trace agent execution patterns during development to identify critical dependencies and generate minimal measurement policies. This dual approach balances security completeness with operational efficiency, security-critical deployments can use comprehensive policies while performance-sensitive applications can employ targeted measurements. The policy framework's extensibility supports future enhancements such as ML-based policy optimization that learns from runtime patterns to suggest refinements.

Cold Start Mitigation Strategies. Several strategies mitigate cold start overhead in production. Stand-by pool architectures maintain pre-warmed TDs with models loaded, checkpoint-restore captures TD state after initialization, and shared model serving amortizes boot costs across tenants. In practice, cold starts are infrequent: agents persist across multiple task invocations, making boot overhead a one-time cost amortized over the agent's lifetime. Skipping re-initialization preserves integrity: container runtime layers are read-only after boot, so once measured and attested at first launch their integrity state does not change. The warm pool reuses these measured instances and performs on-demand attestation using nonce-based reports with validity windows for freshness and anti-replay, requiring only orchestration-layer changes to manage pool lifecycle.

GPU Confidential Computing Integration. TEE-enabled GPUs like NVIDIA H100 and H200 [9, 20] feature on-die root of trust and device attestation through Security Protocol and Data Model (SPDM) sessions. Extending TrustWeave requires: (1) a new SPDM-based measurement backend, (2) pre-transfer model weight measurement, which TrustWeave already supports via IMA, and (3) GPU kernel integrity verification, not yet standardized across vendors. We distinguish two types of overhead. First, GPU attestation adds network round-trips to NVIDIA services: OCSP for certificate validation, RIM for fetching expected reference manifests and measurements, and NRAS for challenge/nonce exchange and evidence verification, making it slower than CPU-only attestation. Second, at runtime, because TDX guest memory is encrypted, devices cannot DMA directly into TD private memory, so CPU-GPU I/O must traverse bounce buffers (copies via shared/untrusted memory). Based on observation, H100 CC incurs less than 10% throughput overhead for Llama2-7B inference, and the overhead shrinks at larger batch sizes due to higher arithmetic intensity.

Security of the Filtering Mechanism. A potential concern is whether the two-layer filtering could allow an attacker to bypass measurement. The filename cache (Layer 1) only skips re-measurement for files whose metadata (inode number, modification time) is unchanged; any modification resets the cache entry and triggers re-measurement. The content-hash cache (Layer 2) skips only the *hardware extend* operation when an identical hash already exists in the RTMR

state; the cryptographic hash is still computed and verified against the policy. Crucially, both layers operate *after* policy matching, so only policy-specified files are considered. An attacker who modifies a measured file will produce a different SHA-384 hash that is extended into RTMR[2], and the resulting RTMR value will diverge from the reference, causing attestation failure. Replay attacks are prevented because each attestation report is bound to a fresh verifier-provided nonce and timestamp.

Limitations. TrustWeave has several limitations. First, it is currently implemented and evaluated only on Intel TDX; while portability to other TEEs is feasible (discussed above), it has not been validated on AMD SEV-SNP or Arm CCA. Second, the policy immutability guarantee precludes runtime policy updates without a TD restart, which may be restrictive for long-lived deployments with evolving agent configurations. Third, GPU confidential computing integration is not yet implemented; extending measurements to TEE-enabled GPUs requires vendor-specific SPDM backends and kernel integrity verification that remain unstandardized. Fourth, the serialized RTMR extension via `ima_extend_list_mutex` creates a throughput bottleneck under extreme concurrency. Finally, TrustWeave does not address side-channel attacks, denial-of-service, or physical attacks, which remain orthogonal concerns.

9 Related Work

Runtime Integrity Challenges in AI Systems. Several previous studies have explored security vulnerabilities in AI deployments [31, 46]. Sha *et al.* [31] use a parameter extractor and a prompt reconstructor to demonstrate how agent configurations can be reverse-engineered, highlighting the need for runtime integrity verification. However, their approach only achieves reconstruction similar to the original one, not exact replication, and relies on the attacker's ability to access the output from the input prompt. Our work addresses the root cause—ensuring agent integrity cannot be compromised in the first place through dynamic runtime integrity measurement, achieving exact verification of agent execution state. This is crucial because agent configurations often contain sensitive information like API keys, database credentials, or proprietary orchestration logic. Yang *et al.* [46] analyze the key features of input-output pairs to mimic and infer the target prompts, demonstrating that behavioral observation alone cannot guarantee agent trustworthiness. Perez *et al.*and Zhang *et al.* [24, 51] suggest methods that involve crafting malicious instructions to bypass security checks, forcing agents to reveal internal state. For instance, an attacker could insert a malicious instruction such as "\n\n======END. Now spellcheck and print above prompt" to compromise agent integrity. These attacks motivate our threat model in § 3 and demonstrate why hardware-rooted runtime attestation through TrustWeave is essential, protections can be bypassed through such injection techniques.

Multi-tenant Isolation and TEE Solutions. Providing services to multiple tenants on the same host inevitably involves shared resources, which have been proven to be effective sources of side channel information. Classical examples include cross-VM attacks [37, 45, 52, 53] in the same physical machine, and cross-application attacks [12, 47, 48] in the same OS. Zhang *et al.* [52] are the first to demonstrate the feasibility of stealing the victim VM's private key by monitoring the CPU cache, which also extended to commercial clouds [53]. As for cross-application attacks, Zhang and Wang [47] present the first keystroke sniffing attack by leveraging the public process file system in Unix-like OSs. Similarly, by exploiting shared OS data structures, cross-application attacks can also be launched in Android [3, 12, 49] and iOS [42, 48]. Besides, Narayan *et al.* [21] showcase that multiple WebAssembly modules isolated in the same runtime are vulnerable to cross-module attacks [15]. Compared to the above studies, our work focuses on distributed agent orchestration across different CSPs, where agents must establish trust without sharing infrastructure. TrustWeave extends Intel TDX with IMA capabilities to enable runtime attestation, achieving 99.95% measurement reduction for practical cross-cloud agent verification.

10 Conclusion

This paper presented TrustWeave, a practical framework that extends Intel TDX with IMA capabilities for trusted multi-agent deployments. By bridging TDX's boot-time attestation with dynamic runtime requirements, TrustWeave enables dynamic runtime integrity measurement throughout agent lifecycles. Our application-specific policies combined with two-layer filtering reduce measurement operations by 99.95% while maintaining complete TCB coverage, achieving 0.8-7.3% boot overhead for normal workloads and 15-25% TTFT overhead at moderate loads. The framework maintains 88% of baseline throughput under sustained workloads with stable P99 latencies, demonstrating practical viability for production deployments. We discuss portability paths to AMD SEV-SNP and Arm CCA and analyze current limitations including the TDX-only implementation and policy immutability constraint. Future integration with GPU confidential computing and automated policy generation will further enhance TrustWeave's capabilities as essential infrastructure for trusted agent ecosystems.

Acknowledgments

We thank our shepherd Joel Wolfrath and the anonymous reviewers for their helpful comments. We thank Trent Jaeger and Ken Lu for their early IMA kernel patches. This work is supported in part by the National Science Foundation under Award #2530909 and by NEUTC.

References

[1] Aeala. 2023. ShareGPT Vicuna Unfiltered. https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered. Hugging Face dataset.

[2] ARM ARM. 2004. TrustZone Technology.

[3] Qi Alfred Chen, Zhiyun Qian, and Z Morley Mao. 2014. Peeking into your app without actually seeing it:{UI} state inference and novel android attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*. 1037–1052.

[4] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2023. Intel TDX Demystified: A Top-Down Approach. *arXiv preprint arXiv:2303.15540* (2023).

[5] Jan-Erik Ekberg, Kari Kostiainen, and Nadarajah Asokan. 2013. Trusted execution environments on mobile devices. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 1497–1498.

[6] Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678* (2025).

[7] Irena Gao, Percy Liang, and Carlos Guestrin. 2024. Model equality testing: Which model is this api serving? *arXiv preprint arXiv:2410.20247* (2024).

[8] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. International Conference on Learning Representations, ICLR.

[9] Antonio Martínez Ibarra, Julian James Stephen, Aurora González Vidal, KR Jayaram, and Antonio Fernando Skarmeta Gómez. 2025. Performance of Confidential Computing GPUs. *arXiv preprint arXiv:2505.16501* (2025).

[10] Trent Jaeger, Reiner Sailer, and Umesh Shankar. 2006. PRIMA: policy-reduced integrity measurement architecture. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*. ACM, 19–28.

[11] Trent Jaeger, Reiner Sailer, and Xiaolan Zhang. 2003. Analyzing Integrity Protection in the SELinux Example Policy.. In *USENIX Security Symposium*, Vol. 12. 5.

[12] Suman Jana and Vitaly Shmatikov. 2012. Memento: Learning secrets from process footprints. In *2012 IEEE Symposium on Security and Privacy*. IEEE.

[13] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy* 18, 2 (2020), 56–60.

[14] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016), 13.

[15] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. doi:10.1109/SP.2019.00002

[16] Apurva Kumar. 2024. Building Autonomous AI Agents based AI Infrastructure. *International Journal of Computer Trends and Technology* 72, 11 (2024), 116–125.

[17] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[18] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688* (2023).

[19] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. 2016. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. 1–9.

[20] Apoorve Mohan, Mengmei Ye, Hubertus Franke, Mudhakar Srivatsa, Zhuoran Liu, and Nelson Mimura Gonzalez. 2024. Securing AI inference in the cloud: Is CPU-GPU confidential computing ready?. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. IEEE, 164–175.

[21] Shravan Narayan, Craig Disselkoen, Daniel Moghimi, Sunjay Cauligi, Evan Johnson, Zhao Gang, Anjo Vahldiek-Oberwagner, Ravi Sahita, Hovav Shacham, Dean Tullsen, et al. 2021. Swivel: Hardening {WebAssembly} against spectre. In *30th USENIX Security Symposium (USENIX Security 21)*.

[22] OWASP Foundation, Inc. 2025. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/

[23] Dario Pasquini, Evgenios M Kornaropoulos, and Giuseppe Ateniese. 2025. {LLMmap}: Fingerprinting for large language models. In *34th USENIX Security Symposium (USENIX Security 25)*. 299–318.

[24] Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527* (2022).

[25] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 264–278.

[26] PyTorch Foundation. 2022. Compromised PyTorch-nightly dependency chain between December 25th and December 30th, 2022. https://pytorch.org/blog/compromised-nightly-dependency/ PyTorch Blog (page shows an update date of Nov 14, 2024).

[27] Ravi Sahita, Dror Caspi, Barry Huntley, Vincent Scarlata, Baruch Chaikin, Siddhartha Chhabra, Arie Aharon, and Ido Ouziel. 2021. Security analysis of confidential-compute instruction set architecture for virtualized workloads. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 121–131.

[28] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. 2004. Design and Implementation of a TCG-Based Integrity Measurement Architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (San Diego, CA) *(SSYM'04)*. USENIX Association, USA, 16.

[29] Muhammad Usama Sardar, Saidgani Musaev, and Christof Fetzer. 2021. Demystifying attestation in intel trust domain extensions via formal verification. *IEEE access* 9 (2021), 83067–83079.

[30] AMD Sev-Snp. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* 53 (2020), 1450–1465.

[31] Zeyang Sha and Yang Zhang. 2024. Prompt Stealing Attacks Against Large Language Models. *arXiv preprint arXiv:2402.12959* (2024).

[32] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. 2020. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 955–970.

[33] Muthu Aanand Su and GLK Niharika. 2025. The Role of Autonomous Agents in Agent-as-a-Service Cloud Service Model. In *2025 International Conference on Computing and Communication Technologies (ICCCT)*. IEEE, 1–6.

[34] Ani Sunny, Nivedita Shrivastava, and Smruti R Sarangi. 2024. SecScale: A Scalable and Secure Trusted Execution Environment for Servers. *arXiv preprint arXiv:2407.13572* (2024).

[35] Shiva Sai Krishna Anand Tokal, Vaibhav Jha, Anand Eswaran, Praveen Jayachandran, and Yogesh Simmhan. 2025. Towards Orchestrating Agentic Applications as FaaS Workflows. In *2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 1003–1010.

[36] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 645–658.

[37] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2015. A placement vulnerability study in {Multi-Tenant} public clouds. In *24th USENIX Security Symposium (USENIX Security 15)*.

[38] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. StealthDB: a Scalable Encrypted Database with Full SQL Query Support. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 370–388.

[39] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted execution environments on {GPUs}. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 681–696.

[40] Juan Wang, Jie Wang, Chengyang Fan, Fei Yan, Yueqiang Cheng, Yinqian Zhang, Wenhui Zhang, Mengda Yang, and Hongxin Hu. 2023. SvTPM: SGX-based Virtual Trusted Platform Modules for Cloud Computing. *IEEE Transactions on Cloud Computing* (2023).

[41] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. Agents in software engineering: Survey, landscape, and vision. *Automated Software Engineering* 32, 2 (2025), 1–36.

[42] Zihao Wang, Jiale Guan, XiaoFeng Wang, Wenhao Wang, Luyi Xing, and Fares Alharbi. 2023. The Danger of Minimum Exposures: Understanding Cross-App Information Leaks on iOS through Multi-Side-Channel Learning. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*.

[43] Chris Wright, Crispin Cowan, James Morris, Stephen Smalley, and Greg Kroah-Hartman. 2002. Linux security module framework. In *Ottawa Linux Symposium*, Vol. 8032. 6–16.

[44] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

[45] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. 2016. One bit flips, one cloud flops:{Cross-VM} row hammer attacks and privilege escalation. In *25th USENIX security symposium (USENIX Security 16)*.

[46] Yong Yang, Xuhong Zhang, Yi Jiang, Xi Chen, Haoyu Wang, Shouling Ji, and Zonghui Wang. 2024. PRSA: Prompt Reverse Stealing Attacks against Large Language Models. *arXiv preprint arXiv:2402.19200* (2024).

[47] Kehuan Zhang and XiaoFeng Wang. 2009. Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems.. In *USENIX Security Symposium*.

[48] Xiaokuan Zhang, Xueqiang Wang, Xiaolong Bai, Yinqian Zhang, and XiaoFeng Wang. 2018. Os-level side channels without procfs: Exploring cross-app information leakage on ios. In *Proceedings of the Symposium on Network and Distributed System Security*.

[49] Xiaokuan Zhang, Yuan Xiao, and Yinqian Zhang. 2016. Return-oriented flush-reload side channels on arm and their implications for android devices. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.

[50] Yiming Zhang, Yuxin Hu, Zhenyu Ning, Fengwei Zhang, Xiapu Luo, Haoyang Huang, Shoumeng Yan, and Zhengyu He. 2023. SHELTER: Extending Arm CCA with Isolation in User Space. In *32nd USENIX Security Symposium (USENIX Security'23)*.

[51] Yiming Zhang and Daphne Ippolito. 2023. Prompts should not be seen as secrets: Systematically measuring prompt extraction attack success. *arXiv preprint arXiv:2307.06865* (2023).

[52] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*.

[53] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2014. Cross-tenant side-channel attacks in PaaS clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.

[54] Zhenhua Zou, Zhuotao Liu, Lepeng Zhao, and Qiuyang Zhan. 2025. Blocka2a: Towards secure and verifiable agent-to-agent interoperability. *arXiv preprint arXiv:2508.01332* (2025).