AgenTEE: Confidential LLM Agent Execution on Edge Devices

Sina Abdollahi
s.abdollahi22@imperial.ac.uk
Imperial College London
London, United Kingdom

Mohammad M Maheri
m.maheri23@imperial.ac.uk
Imperial College London
London, United Kingdom

Javad Forough
j.forough@imperial.ac.uk
Imperial College London
London, United Kingdom

Amir Al Sadi
a.al-sadi@imperial.ac.uk
Imperial College London
London, United Kingdom

Josh Millar
joshua.millar22@imperial.ac.uk
Imperial College London
London, United Kingdom

David Kotz*
David.F.Kotz@dartmouth.edu
Dartmouth College
Hanover, NH, USA

Marios Kogias
m.kogias@imperial.ac.uk
Imperial College London
London, United Kingdom

Hamed Haddadi
h.haddadi@imperial.ac.uk
Imperial College London
London, United Kingdom

Abstract

Large Language Model (LLM) agents provide powerful automation capabilities, but they also create a substantially broader attack surface than traditional applications due to their tight integration with non-deterministic models and third-party services. While current deployments primarily rely on cloud-hosted services, emerging designs increasingly execute agents directly on edge devices to reduce latency and enhance user privacy. However, securely hosting such complex agent pipelines on edge devices remains challenging. These deployments must protect proprietary assets (*e.g.*, system prompts and model weights) and sensitive runtime state on heterogeneous platforms that are vulnerable to software attacks and potentially controlled by malicious users.

To address these challenges, we present *AgenTEE*, a system for deploying confidential agent pipelines on edge devices. *AgenTEE* places the agent runtime, inference engine, and third-party applications into independently attested confidential virtual machines (cVMs) and mediates their interaction through explicit, verifiable communication channels. Built on Arm Confidential Compute Architecture (CCA), a recent extension to Arm platforms, *AgenTEE* enforces strong system-level isolation of sensitive assets and runtime state. Our evaluation shows that such multi-cVMs system is practical, achieving near-native performance with less than 5.15% runtime overhead compared to commodity OS multi-process deployments.

CCS Concepts: • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Security and privacy** → **Virtualization and security**; • **Computing methodologies** → **Machine learning**.

Keywords: Confidential Computing, Machine Learning, LLM, AI, Agents, Agentic Computing

ACM Reference Format:
Sina Abdollahi, Mohammad M Maheri, Javad Forough, Amir Al Sadi, Josh Millar, David Kotz, Marios Kogias, and Hamed Haddadi. 2026. *AgenTEE*: Confidential LLM Agent Execution on Edge Devices. In *Sixth European Workshop on Machine Learning and Systems (EuroMLSys '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3805621.3807660

1 Introduction

Recent advances in Large Language Models (LLMs) have enabled a new class of software, LLM agents[1], that can autonomously reason over user instructions, plan multi-step tasks, and interact with external services. An agent combines a language model with a runtime that maintains context, interprets user requests, selects relevant services, invokes them, and integrates their outputs into subsequent reasoning steps.

Traditionally, agents are deployed as cloud-hosted services, often granted access to users' data through APIs or synchronization mechanisms [3, 6]. In contrast, emerging edge-device designs execute both the agent runtime and the underlying model locally. Such deployments improve privacy and data locality by confining users' personal data to the device, giving users greater control over its exposure to external services [29, 30, 55].

*This work was performed while Professor Kotz was in residence at Imperial College London.

[1]In this paper we use *agents* and *LLM agents* interchangeably.

EuroMLSys '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2605-7/2026/04
https://doi.org/10.1145/3805621.3807660

Compared to traditional software, agents face a significantly broader attack surface. On one hand, they require extensive access to third-party services and user's data to perform complex tasks effectively. These accesses expose the agent to untrusted or potentially malicious inputs (*e.g.*, emails or documents) that enter its processing pipeline. On the other hand, their core reasoning engine—the LLM itself—cannot reliably distinguish trusted system instructions from untrusted inputs, making it inherently vulnerable to attacks such as data exfiltration, unintended action execution, or safeguard bypass. As a result, LLM agents demand stronger isolation guarantees than conventional applications.

Current agent deployments on edge devices primarily rely on OS mechanisms—such as multi-processing and system-call filtering (*e.g.*, seccomp [18])—to isolate agent components from the rest of the software stack and from co-resident applications [30, 55]. This is typically adequate for low-stakes, simple agents (*e.g.*, an agent retrieving public news from the internet). However, once an agent's workflow includes *proprietary assets* (*e.g.*, specialized models or confidential agent code), software-only isolation provides inadequate protection for sensitive runtime state and data-in-use, since these assets must remain confidential even from privileged platform software.

The problem becomes even more challenging where the agentic service components needs to be provided by *mutually distrustful stakeholders*. For example, one party may supply the runtime while another supplies the model, alongside third-party integrated applications.

What is fundamentally required is an execution environment that can simultaneously host components of agentic workloads (*i.e.*, agent, model, and third-party applications) and protect them from both the hosting platform and from each other. The execution substrate must prevent the commodity OS from inspecting sensitive runtime state and messages exchanged between these components, while enabling interaction without exposing proprietary assets.

Trusted Execution Environments (TEEs) provide a hardware-backed foundation for such a execution environment, offering protected execution and data-in-use confidentiality even against privileged software. In contrast to cryptographic-based approaches (*e.g.*, [32, 38, 39, 43, 51]), TEEs are often more practical for end-to-end systems, as they preserve near-native performance relative to conventional execution environments.

Despite these advantages, widely deployed TEE architectures such as Arm TrustZone [16] are a poor fit for isolating multi-component agent pipelines. They are primarily designed for small, vendor-specific services, such as authentication (*e.g.*, Face ID [19]) and digital rights management (*e.g.*, Widevine [7]), and face significant limitations when supporting complex, general-purpose workloads [47].

Recent extensions to Arm architecture help address this gap. Arm Confidential Compute Architecture (CCA) [9] enables general-purpose confidential virtual machines (cVMs), called *realms*, that execute in hardware-isolated memory, protected from the hosting operating system and hypervisor. Realms can run unmodified software and can do not face any inherit memory size limitation. Each realm is cryptographically measured at launch and supports remote attestation, allowing mutually distrustful components to verify identity and integrity before provisioning secrets or exchanging sensitive data.

In this paper, we argue that secure agentic systems executing on edge devices and handling proprietary assets fundamentally require hardware-backed confidential execution. We present *AgenTEE*, a framework that deploys the agent runtime, LLM inference engine, and third-party applications into independently attested realms, and mediates their interaction through explicit, verifiable communication channels. By confining proprietary assets and intermediate runtime state to hardware-protected memory, *AgenTEE* enables secure composition of mutually distrustful components on a single device with strong isolation and minimal performance overhead.

This paper makes the following **contributions**:

- We identify the security requirements of edge-device LLM agents and explain how Arm CCA satisfies these requirements.
- We design *AgenTEE*, a system for executing agent runtime and third-party applications within confidential execution environments, enabling mutual attestation and hardware-enforced isolation between components.
- We implement[2] and evaluate *AgenTEE* on Arm CCA, demonstrating that confidential edge-device LLM agents are practical and scalable on modern edge hardware, incurring less than 5.15% overhead compared to native execution.

2 Background & Motivation

In this section, we first provide background on LLM agents (Sec. 2.1) and discuss their system-level isolation requirements (Sec. 2.2). We then review existing TEEs for edge devices and position Arm CCA as a promising foundation for *AgenTEE* (Sec. 2.3).

2.1 LLM Agents

Figure 1 illustrates an LLM agent flow. An LLM agent integrates a language model with a runtime that maintains context, interprets user requests, selects relevant components, invokes them, and integrates their outputs into subsequent reasoning steps. The LLM serves as the agent's core reasoning component, interpreting natural language instructions,

[2]We will release our implementation open-source upon publication.

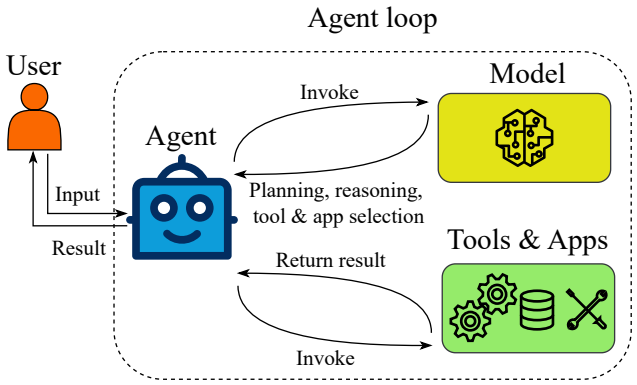

Figure 1. LLM agent architecture

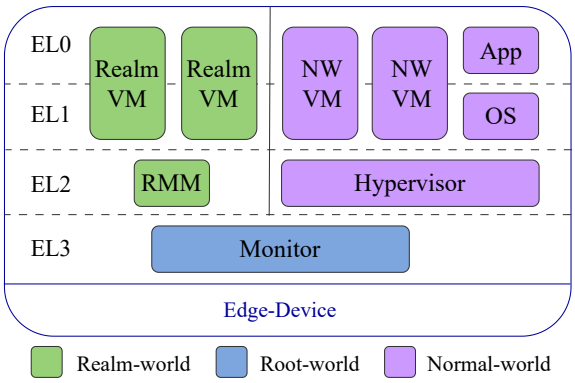

Figure 2. Arm Confidential Compute Architecture

generating plans, and selecting appropriate tools to invoke. Tools and third-party applications broaden the functional scope of the agent by enabling access to real-time information [56, 58], conducting complex calculations [45], and executing specialized tasks such as image recognition [42].

Agent-Model Interaction. Modern LLMs are trained to interpret role-labeled inputs, primarily the *system prompt* and the *user prompt*, and to prioritize the system prompt during reasoning and output generation [12]. A central responsibility of an agent is to construct and properly label the model input. The agent uses the user prompt to convey user requests, interaction history, and external input data, which may be untrusted or adversarial, while the system prompt encodes trusted instructions that govern the model's behavior, including its role, tone, safety constraints, and operational policies [12, 22]. The system prompt further defines accessible tools and external resources, as well as the conditions under which they may be invoked.

2.2 Isolation Requirements of Agent Workflow

In this section, we discuss several internal assets whose confidentiality and integrity are fundamental to the secure operation of an agent. These assets collectively form the agent's control plane and runtime state. We argue how leakage or unauthorized modification at any of these layers can undermine the agent's runtime security and reliability.

Agent. Agent code—including prompt templates (*e.g.*, system prompt), decision logic, orchestration rules, and memory management—is central to an agentic system's reliability and policy compliance, and may treated as proprietary [34]. Organizations invest significant effort in engineering these components to achieve efficiency, enforce policies, guide tool usage, and achieve safe behavior, making them a source of competitive advantage that requires protection from unauthorized access.

Prior works show that even partial leakage of these components significantly increases the feasibility of indirect and tailored prompt injection attacks [25, 33], data exfiltration [11, 41], and secret leakage [13]. This observation highlights that protecting internal agent logic is not merely an intellectual property concern, but a fundamental security requirement.

Inference Engine. The inference engine is the runtime component responsible for executing a trained model to produce outputs from input prompts. In doing so, it handles two confidentiality-critical assets: (1) the model weights, which encode the model's learned capabilities and behavior, and (2) transient runtime state, most notably the key–value (KV) cache, which stores intermediate attention states to accelerate LLM inference [36]. Protecting both assets is essential, since their exposure or manipulation can directly affect security and correctness.

Model weights carry substantial economic and operational value, reflecting significant investment in training and fine-tuning; proprietary weights therefore require strong confidentiality guarantees. Even when weights are public, integrity during inference remains crucial: LLMs can be highly sensitive to parameter-level perturbations, where a small fraction of parameter/feature outliers can disproportionately influence model quality [26, 31], and pruning even a single critical parameter can catastrophically degrade quality [57]. Beyond weights, the inference engine's runtime state also constitutes a security-sensitive asset. The primary example is the key–value (KV) cache [37, 54], which – while introduced for performance – encodes semantic representations of previously processed context. If an adversary can read or tamper with cached key–value states, system prompts or privileged instructions may be reconstructed or implicitly reactivated; moreover, cache manipulation can steer behavior or trigger policy bypass without changing the visible prompt. Accordingly, the KV cache requires both confidentiality and integrity protection.

Third-Party Applications. Third-party applications frequently require access to sensitive credentials, such as API keys, database secrets, or authentication tokens. They may also employ business logic to their local services. These secrets and business logic must be consistently protected

against adversaries, including malicious applications running on the same device.

2.3 TEEs on the Edge

Historically, edge-device TEEs such as Arm TrustZone [16] and Intel TDX [14] have been used to protect sensitive applications, including key management, digital rights management (DRM), federated learning [40], and machine learning inference [50]. However, edge-device TEEs have their own limitations, including restricted memory size [47] and limited accessibility for third-party developers [23, 35]. Such constraints complicate the deployment of complex, memory-intensive workloads, such as LLM agents.

To mitigate these limitations, Android introduced the Android Virtualization Framework (AVF) [10], which enables isolated VMs protected from the Android kernel. However, these VMs are not protected by hardware-enforced mechanisms and their security relies on the Android hypervisor, which remains part of the trusted computing base (TCB) of the VM.

Arm CCA. Arm CCA is an extension to the Armv9-A architecture targeting both edge devices and cloud servers. As illustrated in Figure 2, CCA introduces a new execution state called the *realm-world*, alongside the traditional normal-world (NW). CCA enables cVMs, referred to as realm VMs (or simply realms), which execute within the realm-world. Realm-world memory is protected through hardware-enforced isolation mechanisms, making it inaccessible to entities operating in the normal-world (including the OS and hypervisor). Consequently, even a compromised host operating system or hypervisor cannot directly access or modify realm memory or virtual CPU state.

In CCA, the hypervisor retains responsibility for resource management, including CPU scheduling and memory allocation. However, hypervisor operations on realm resources are mediated by a trusted lightweight firmware component called the Realm Management Monitor (RMM). By design, CCA significantly reduces the TCB: the hypervisor (approximately 300K lines of code) is excluded from the TCB, and security enforcement is delegated to the comparatively small RMM (approximately 27K lines of code).

CCA provides appropriate abstraction. CCA introduces several properties that make it a suitable foundation for *AgenTEE*. Firstly, CCA enables flexible, general-purpose confidential computation by moving beyond fixed-function TEEs (*e.g.*, TrustZone) to support general purpose realms, within a single device, making it a natural fit for *AgenTEE*. Unlike TrustZone, which relies on statically partitioned memory, CCA allows realms to be instantiated dynamically as long as sufficient RAM is available. These realms can execute existing software stacks with no modification, even if they were originally designed for a conventional execution environment. This supports more complex, multi-component agent pipelines—spanning the runtime, model, and third-party services—and makes it practical to deploy components from heterogeneous trust domains on commodity edge devices without specialized hardware configurations.

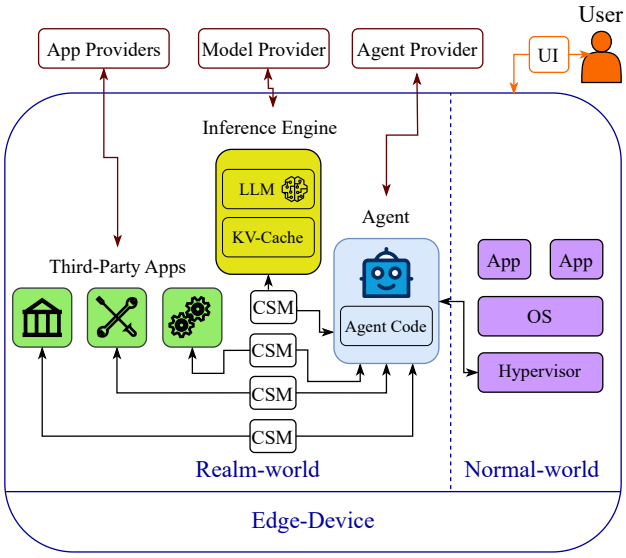

Figure 3. *AgenTEE*'s system architecture

Second, CCA's RISC-based architectural design enables extensibility and customization. Its minimal and modular design allows incremental extensions to support emerging workload requirements without redesigning the TCB. Prior work has demonstrated extensions for efficient inter-realm communication and protected data sharing [24], as well as confidential GPU acceleration [44, 48], enabling practical deployment of multi-realm and performance-sensitive workloads.

3 *AgenTEE* Overview

In this section, we provide an overview of *AgenTEE*, its threat model and pipeline.

3.1 System and Security Model

System Model. *AgenTEE* involves five major entities: *model providers*, *agent providers*, *application providers*, the *users*, and the *users' device*. Model providers train and distribute machine-learning models, agent providers implement the agent code, which are both proprietary. Application providers offer external services invoked by the agent (*e.g.*, banking, travel booking, or other cloud services). The users' devices are Armv9-A platform supporting the CCA architecture, where the normal-world runs a commodity operating system (*e.g.*, Android) that controls hardware resources, user interfaces (UI), and conventional applications, while also supporting the execution of cVMs in the realm-world.

Trust Model. We assume that users trust their own devices and the normal-world environment through which inputs are received and delivered to the agent[3]. In contrast, the agent provider, model provider, and third-party application providers do not trust one another, nor do they trust the NW running on users' devices. All stakeholders trust the underlying CCA hardware and firmware (*i.e.*, the RMM and Monitor in Fig. 2) and assume that its architectural security guarantees hold.

Threat Model. We assume that model provider, agent provider, and third-party application provider, and NW stack may attempt to obtain information about other parties' proprietary assets. Such assets include model weights, agent runtime code or APIs provided by third-party applications. Physical attacks and microarchitectural side-channel attacks against the TEE hardware are out of scope. We also assume no compromise of cryptographic primitives or violations of CCA's architectural guarantees.

3.2 *AgenTEE* Pipeline

Figure 3 shows the system architecture of *AgenTEE*. *AgenTEE* organizes the edge-device agent pipeline entirely within the realm world. The agent runtime, inference worker, and third-party applications are each deployed in separate cVMs. All interactions between these components occur through protected realm memory, preventing the host operating system or hypervisor from observing or manipulating sensitive runtime state.

The pipeline begins with each stakeholder deploying its component into a realm using the standard CCA initialization flow, which loads a publicly available image into realm memory. Upon launch, each realm establishes a secure network connection (TLS) with the respective owner, providing an RMM-signed attestation token. This token serves as cryptographic evidence that the realm is executing the software stack each owner expects. Once each owner verifies its realm, it uses the secure network connection to transmit proprietary assets (*e.g.*, model, prompt, or agent runtime code) to its realm.

To enable efficient and mutually authenticated inter-cVM communication, *AgenTEE* integrates CAEC [24] design, which provides Confidential Shared Memory (CSM) between realms. CSM enables hypervisor's inaccessible channels, allowing peer realms to exchange data through memory regions inaccessible to the normal world. *AgenTEE* adopts CAEC's inter-realm attestation protocol to ensure that communication occurs only between verified and authorized realms.

Once the agent realm establishes a trusted channel with the model realm, it signals readiness to the normal world. A user-interface application in the normal-world mediates user interactions by forwarding requests to the agent realm, while remaining unable to access or inspect the protected internal state of the agent or the model.

Security Analysis. *AgenTEE* mitigates the threats outlined in our model by deploying the agent runtime, inference engine, and third-party applications in separate realms. *AgenTEE* confines proprietary assets (and sensitive runtime state to hardware-isolated memory that is inaccessible to the normal world, including a compromised OS or hypervisor. Remote attestation ensures that each stakeholder provisions its assets only to a verified realm running the expected software stack, preventing unauthorized modification or impersonation.

4 Implementation

There is no native Arm CCA hardware yet commercially available. We therefore rely on OpenCCA [27], an open-source prototype implementation of Arm CCA. OpenCCA implements a prototype pf CCA of an embedded cost-effective hardware, Radxa Rock 5B [5]. Although this prototype cannot exactly forecast system performance on future CCA hardware, it provides a practical and best-effort approximation in the absence of commercial hardware support.

Software Stack. Our implementation builds on Trusted Firmware-A [20] (v2.11) as the Monitor and the reference RMM implementation [21] (v0.5.0). We use linux-cca [17] (v5+v7) as both host and guest operating system, and kvmtool-cca [15] (v3/cca) as the virtual machine manager (VMM). To enable CSM between realms, we apply CAEC's patches to the RMM, linux-cca, and kvmtool-cca used in the implementation.

User-Space CSM Interface. To facilitate application-level communication over CSM, we design a lightweight (184 line of code) Python module that abstracts CSM usage for user-space components. While CAEC [24] provides a Linux driver that exposes CSM regions to user space, it does not define how applications should structure communication over this shared memory. Our module builds on the exposed CSM regions and partitions each inter-realm CSM region into multiple logical half-duplex channels, enabling structured and efficient message passing between components executing in separate realms.

Implemented Agents. To evaluate *AgenTEE* under realistic edge-device agent workloads, we implement two representative LLM agents that share the same execution architecture but differ in prompt structure and computational intensity. We implemented both agents using LangChain framework [2].

Chatbot Agent: The chatbot agent represents a lightweight conversational assistant. It accepts free-form natural language queries, constructs a short prompt using a fixed system prompt, and forwards it to the LLM for response generation. This agent models common edge-device assistants used for question answering, summarization, and explanation, and

[3]More complex scenarios in which the NW lies outside the user's trust domain are out of scope for this work.

Table 1. Inference and end-to-end latency (second) for chatbot and itinerary agents under different mechanisms.

Agent			GPT2-Medium-q8_0 (437MB)		Llama-3.2-1B-Instruct-Q4_0 (773MB)	
			Inference	End-to-End	Inference	End-to-End
Chatbot	Isolation	NW Process	92.16	93.45	277.79	277.80
		NW VM	96.26	96.29	284.89	284.90
		AgenTEE	98.22	98.25	289.52	289.55
	Overhead	*AgenTEE* vs NW Process	6.57%	5.14%	4.22%	4.23%
		AgenTEE vs NW VM	2.04%	2.04%	1.63%	1.63%
Itinerary	Isolation	NW Process	163.50	163.82	462.30	462.31
		NW VM	168.98	168.99	467.92	473.26
		AgenTEE	170.76	170.77	481.25	485.18
	Overhead	*AgenTEE* vs NW Process	4.44%	4.24%	4.10%	4.08%
		AgenTEE vs NW VM	1.05%	1.05%	2.85%	2.52%

serves as a baseline workload with modest communication and inference demands.

Itinerary Planning Agent: The itinerary agent represents a task-oriented planning workload. Given a destination, trip duration, budget, interests, and additional constraints, the agent constructs a structured prompt that requests multi-day schedules, activity lists, and cost estimates. Compared to the chatbot agent, this workload produces longer prompts and responses and exercises both the communication channel and LLM inference more heavily.

5 Preliminary Evaluation

To evaluate the performance overhead of *AgenTEE*, we measure the dominant cost of an agent workflow: the agent–model interaction. Specifically, we measure the end-to-end latency of each agent–model query under different isolation scenarios to quantify the cost of hardware-backed isolation in *AgenTEE*.

We conduct experiments using two representative agents (*i.e.*, a chatbot and an itinerary planner), two models (GPT2-Medium [4] and Llama-3.2-1B [1]), and three isolation configurations. The isolation scenarios are as follows:(1) *AgenTEE*, where two cVMs communicate through CSM, assuming the entire NW is untrusted; (2) two NW VMs communicating via shared memory, where the NW hypervisor is trusted; and (3) two NW processes communicating through shared memory within the NW OS (baseline), where the NW OS and hypervisor are trusted. These configurations progressively relax the trust assumptions and represent decreasing levels of isolation. For each configuration, we measure both the inference latency within the inference engine and the end-to-end latency observed by the agent when issuing a model query.

Table 1 reports the median results over nine typical queries. The end-to-end overhead of *AgenTEE* compared to NW VMs remains below 2.53% across all settings, while the end-to-end overhead relative to NW processes is below 5.15%. These results demonstrate that *AgenTEE* introduces minimal performance overhead despite providing stronger isolation guarantees, remaining close even to the weakest isolation baseline, which is inter-process communication within the NW OS.

We note that our experiments are conducted on a resource-constrained prototype platform that is less powerful than modern edge devices (*e.g.*, smartphones). On contemporary mobile CPUs, we expect the absolute latency to be much less. In addition, upcoming CCA extension [8] will support secure assignment of specialized accelerators (*e.g.*, GPUs or NPUs) to realms, enabling the *AgenTEE*'s inference engine to leverage hardware acceleration for token generation, improving absolute latency.

6 Related Work

LLM Agents. Recent work explores the deployment of LLM agents on edge devices [30, 53, 55]. Prior studies recognize internal agent state—such as system prompts—as confidential assets whose exposure enable attacks including indirect prompt injection, data exfiltration, and tool manipulation [11, 13, 25, 33, 41]. Beyond prompt manipulation, model-centric threats target the integrity and confidentiality of model weights and runtime state: LLMs are highly sensitive to parameter perturbations [26, 31, 57].

Confidential Computing. Confidential computing research investigates hardware-backed isolation mechanisms to protect machine learning workloads, including TEEs such as Arm TrustZone [16, 28, 49]. More recently, Arm CCA has gained attention for extending TEE capabilities to support general-purpose confidential virtual machines, with several works exploring performance, accelerator integration, and system-level enhancements [44, 46, 48, 52]. CAEC [24] further introduces CSM between realms, enabling efficient data sharing across isolated cVMs and improving the practicality of multi-cVM deployments.

7 Conclusion

In this paper, we presented *AgenTEE*, a system for confidential edge device deployment of LLM agent pipelines. By leveraging Arm CCA, *AgenTEE* isolates the agent runtime, inference engine, and third-party components into independently attested confidential virtual machines and mediates their interaction through secure communication channels, protecting the confidentiality and integrity of all stakeholders. Our evaluation demonstrates that this approach is practical, achieving strong isolation guarantees with less than 5.15% runtime overhead compared to standard OS process-level deployments.

8 Acknowledgment

We thank the anonymous reviewers for their insightful comments and suggestions. The research was supported by the UKRI Open Plus Fellowship (EP/W005271/1, Securing the Next Billion Consumer Devices on the Edge), the Amazon Research Award "Auditable Model Privacy using TEEs", and the AI Security Institute (AISI) Systemic Safety Grants Programme (UKRI833).

Professor Kotz was supported by a Royal Society Wolfson Visiting Fellowship and by a collaborative award from the U.S. National Science Foundation (NSF) SaTC Frontiers program under award number 1955805.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of any sponsor. Any mention of specific companies or products does not imply any endorsement by the authors, by their employers, or by their sponsors.

References

[1] [n. d.]. bartowski/Llama-3.2-1B-Instruct-GGUF. https://huggingface.co/bartowski/Llama-3.2-1B-Instruct-GGUF Accessed Feb 2025.

[2] [n. d.]. LangChain. Langchain: Build context-aware, reasoning applications with langchain's flexible abstractions and ai-first toolkit. https://www.langchain.com/ Accessed Feb 2026.

[3] [n. d.]. Microsoft 365 Copilot hub. https://learn.microsoft.com/en-us/copilot/microsoft-365/ Accessed Feb 2026.

[4] [n. d.]. openai-community/gpt2-medium. https://huggingface.co/openai-community/gpt2-medium Accessed Feb 2025.

[5] [n. d.]. ROCK 5B. Retrieved April 14, 2025 from https://radxa.com/products/rock5/5b/

[6] [n. d.]. Using tools. https://developers.openai.com/api/docs/guides/tools Accessed Feb 2026.

[7] [n. d.]. Widevine. https://en.wikipedia.org/wiki/Widevine Accessed Feb 2026.

[8] 2024. MAD24-410 Arm Confidential Compute Architecture open-source enablement update. Retrieved March 9, 2025 from https://resources.linaro.org/en/resource/rEjhEezEvnNMC3LALzUTrr

[9] 2025. Arm Confidential Compute Architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture Accessed Feb 2025.

[10] 2025. AVF architecture. https://source.android.com/docs/core/virtualization/architecture#memory-ownership Accessed July 2025.

[11] 2025. Claude Code: Data Exfiltration with DNS (CVE-2025-55284). https://embracethered.com/blog/posts/2025/claude-code-exfiltration-via-dns-requests/ Accessed Feb 2026.

[12] 2025. Effective context engineering for AI agents. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents Accessed Feb 2026.

[13] 2025. How Devin AI Can Leak Your Secrets via Multiple Means. https://embracethered.com/blog/posts/2025/devin-can-leak-your-secrets/ Accessed Feb 2026.

[14] 2025. Intel Software Guard Extensions. Retrieved June 7, 2025 from https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html

[15] 2025. kvmtool-cca. https://gitlab.arm.com/linux-arm/kvmtool-cca/-/tree/cca/v3?ref_type=heads Accessed Feb 2025.

[16] 2025. Learn the architecture - TrustZone for AArch64. https://developer.arm.com/documentation/102418/latest/ Accessed Feb 2025.

[17] 2025. linux-cca. https://gitlab.arm.com/linux-arm/linux-cca/-/commit/fad35572db Accessed Feb 2025.

[18] 2025. seccomp(2) — Linux manual page. https://man7.org/linux/man-pages/man2/seccomp.2.html Accessed Feb 2026.

[19] 2025. Secure Enclave. https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web Accessed Feb 2025.

[20] 2025. TF-A. https://www.trustedfirmware.org/projects/tf-a Accessed Feb 2025.

[21] 2025. TF-RMM. https://www.trustedfirmware.org/projects/tf-rmm Accessed Feb 2025.

[22] 2026. Use system instructions (Generative AI on Vertex AI). https://docs.cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/system-instructions Accessed Feb 2026.

[23] Sina Abdollahi, Mohammad Maheri, Sandra Siby, Marios Kogias, and Hamed Haddadi. 2025. An Early Experience with Confidential Computing Architecture for On-Device Model Protection. *arXiv preprint arXiv:2504.08508* (2025).

[24] Sina Abdollahi, Amir Al Sadi, Marios Kogias, David Kotz, and Hamed Haddadi. 2025. Confidential, Attestable, and Efficient Inter-CVM Communication with Arm CCA. *arXiv preprint arXiv:2512.01594* (2025).

[25] Divyansh Agarwal, Alexander R Fabbri, Ben Risher, Philippe Laban, Shafiq Joty, and Chien-Sheng Wu. 2024. Prompt leakage effect and defense strategies for multi-turn llm interactions. *arXiv preprint arXiv:2404.16251* (2024).

[26] Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2025. Systematic outliers in large language models. *arXiv preprint arXiv:2502.06415* (2025).

[27] Andrin Bertschi and Shweta Shinde. 2025. OpenCCA: An Open Framework to Enable Arm CCA Research. *arXiv preprint arXiv:2506.05129* (2025).

[28] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. SANCTUARY: ARMing TrustZone with User-space Enclaves.. In *NDSS*.

[29] Wei Chen, Zhiyuan Li, Zhen Guo, and Yikang Shen. 2025. Octo-planner: On-device language model for planner-action agents. In *International Workshop on Engineering Multi-Agent Systems*. Springer, 141–156.

[30] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. 2025. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813* (2025).

[31] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems* 35 (2022), 30318–30332.

[32] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. PMLR, 201–210.

[33] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*. 79–90.

[34] Bo Hui, Haolin Yuan, Neil Gong, Philippe Burlina, and Yinzhi Cao. 2024. Pleak: Prompt leaking attacks against large language model applications. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 3600–3614.

[35] Mark Kuhne, Supraja Sridhara, Andrin Bertschi, Nicolas Dutly, Srdjan Capkun, and Shweta Shinde. 2024. Aster: Fixing the Android TEE ecosystem with Arm CCA. *arXiv preprint arXiv:2407.16694* (2024).

[36] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*. 611–626.

[37] Zhifan Luo, Shuo Shao, Su Zhang, Lijing Zhou, Yuke Hu, Chenxu Zhao, Zhihao Liu, and Zhan Qin. 2025. Shadow in the cache: Unveiling and mitigating privacy risks of kv-cache in llm inference. *arXiv preprint arXiv:2508.09442* (2025).

[38] Mohammad M Maheri, Sunil Cotterill, Alex Davidson, and Hamed Haddadi. 2025. ZK-APEX: Zero-Knowledge Approximate Personalized Unlearning with Executable Proofs. *arXiv preprint arXiv:2512.09953* (2025).

[39] Mohammad M Maheri, Hamed Haddadi, and Alex Davidson. 2025. Telesparse: Practical privacy-preserving verification of deep neural networks. *arXiv preprint arXiv:2504.19274* (2025).

[40] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*. 94–108.

[41] Zhenting Qi, Hanlin Zhang, Eric Xing, Sham Kakade, and Himabindu Lakkaraju. 2024. Follow my instruction and spill the beans: Scalable data extraction from retrieval-augmented generation systems. *arXiv preprint arXiv:2402.17840* (2024).

[42] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).

[43] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 707–721.

[44] Fan Sang, Jaehyuk Lee, Xiaokuan Zhang, and Taesoo Kim. 2025. PORTAL: Fast and Secure Device Access with Arm CCA for Modern Arm Mobile System-on-Chips (SoCs). In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4099–4116.

[45] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems* 36 (2023), 68539–68551.

[46] Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, et al. 2022. SOTER: Guarding Black-box Inference for General Neural Networks at the Edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 723–738.

[47] Sandra Siby, Sina Abdollahi, Mohammad Maheri, Marios Kogias, and Hamed Haddadi. 2024. GuaranTEE: Towards Attestable and Private ML with CCA. In *Proceedings of the 4th Workshop on Machine Learning and Systems*. 1–9.

[48] Supraja Sridhara, Andrin Bertschi, Benedict Schlüter, Mark Kuhne, Fabio Aliberti, and Shweta Shinde. 2024. ACAI: Extending Arm Confidential Computing Architecture Protection from CPUs to Accelerators. In *33rd USENIX Security Symposium (USENIX Security'24)*.

[49] Lizhi Sun, Shuocheng Wang, Hao Wu, Yuhang Gong, Fengyuan Xu, Yunxin Liu, Hao Han, and Sheng Zhong. 2022. LEAP: TrustZone Based Developer-Friendly TEE for Intelligent Mobile Apps. *IEEE Transactions on Mobile Computing* (2022).

[50] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. 2023. ShadowNet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1596–1612.

[51] Tim van Elsloo, Giorgio Patrini, and Hamish Ivey-Law. 2019. SEALion: A framework for neural network inference on encrypted data. *arXiv preprint arXiv:1904.12840* (2019).

[52] Chenxu Wang, Fengwei Zhang, Yunjie Deng, Kevin Leach, Jiannong Cao, Zhenyu Ning, Shoumeng Yan, and Zhengyu He. 2024. CAGE: Complementing Arm CCA with GPU Extensions. In *Network and Distributed System Security (NDSS) Symposium*.

[53] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158* (2024).

[54] Guanlong Wu, Zheng Zhang, Yao Zhang, Weili Wang, Jianyu Niu, Ye Wu, and Yinqian Zhang. 2025. I Know What You Asked: Prompt Leakage via KV-Cache Sharing in Multi-Tenant LLM Serving.. In *NDSS*.

[55] Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. 2024. Isolategpt: An execution isolation architecture for llm-based agentic systems. *arXiv preprint arXiv:2403.04960* (2024).

[56] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

[57] Mengxia Yu, De Wang, Qi Shan, Colorado J Reed, and Alvin Wan. 2024. The super weight in large language models. *arXiv preprint arXiv:2411.07191* (2024).

[58] Wanru Zhao, Vidit Khazanchi, Haodi Xing, Xuanli He, Qiongkai Xu, and Nicholas Donald Lane. 2024. Attacks on third-party apis of large language models. *arXiv preprint arXiv:2404.16891* (2024).