

MVVM: Deploy Your AI Agents—Securely, Efficiently, Everywhere

Yiwei Yang^{*}
 yyang363@ucsc.edu
 UC Santa Cruz
 Santa Cruz, California, USA

Yusheng Zheng
 yzhen165@ucsc.edu
 UC Santa Cruz
 Santa Cruz, California, USA

Kexin Chu
 kexin.chu@uconn.edu
 University of Connecticut
 Storrs, Connecticut, USA

Aibo Hu[†]
 huab@shanghaitech.edu.cn
 ShanghaiTech University
 Shanghai, China

Xinqi Zhang
 xzhang22@scu.edu
 Santa Clara University
 Santa Clara, California, USA

Wei Zhang
 wei.13.zhang@uconn.edu
 University of Connecticut
 Storrs, Connecticut, USA

Brian Zhao
 bwzhao@ucsc.edu
 UC Santa Cruz
 Santa Cruz, California, USA

Dawei Xiang
 ieb24002@uconn.edu
 University of Connecticut
 Storrs, Connecticut, USA

Andi Quinn
 aquinn1@ucsc.edu
 UC Santa Cruz
 Santa Cruz, California, USA

Abstract

The proliferation of LLM-driven agents demands secure execution, rapid fail-back across heterogeneous environments, stringent data confidentiality, continuous availability, low-latency response, and formally safe outputs. We introduce *MVVM*, a WebAssembly-based secure container that transparently live-migrates agent workspaces between edge and cloud, provides end-to-end privacy, employs fault-tolerant multi-tier replication with latency-aware speculative execution, and incorporates runtime output verification.

MVVM introduces two key innovations: (1) a two-way sandboxing framework leveraging hardware enclaves and accelerator extensions that protects both the agent from malicious hosts and the host from compromised agents; (2) an efficient cross-platform migration mechanism using WebAssembly and WASI’s platform-agnostic design, enabling seamless movement across ARM phones, RISC-V MCUs, x86 servers, and heterogeneous accelerators; and three astonishing use cases: (1) privacy-aware daemon that automatically determines whether to execute locally or remotely based on data sensitivity and resource availability; (2) multi-tier replication with intelligent quality degradation that maintains service availability despite network failures or resource constraints; (3) a comprehensive execution framework combining speculative execution for 10× latency reduction with parallel validation that ensures output safety without compromising responsiveness.

Our evaluation demonstrates that *MVVM* is validated on three separate devices across 18 workloads. The system achieves 200ms failover times, 8.9× speedup through speculation, and 94.2% accuracy in detecting harmful outputs with only 3-5% overhead. Security analysis confirms protection

against malicious hosts and cross-tenant data leakage. *MVVM* makes secure, practical deployment of privacy-sensitive LLM applications possible across the computing continuum.

1 Introduction

Imagine using your phone’s AI assistant to compose a message incorporating private photos, personal notes, and live video. In traditional cloud deployments, all this sensitive data traverses public networks to shared servers—an unacceptable privacy risk. Now envision an alternative: your device creates a secure enclave protecting your data with hardware encryption [8]. As computational demands grow, this enclave seamlessly migrates to your private cloud without exposing any private information, tapping into powerful resources while maintaining the same security guarantees.

This vision drives the development of *MVVM* (Migratable Velocity Virtual Machine), addressing critical challenges in modern AI agent deployment. First, privacy and safety represent fundamental requirements. LLM agents process highly sensitive data including personal conversations, medical records, and financial information. Traditional approaches expose this data during transit and fail to prevent harmful outputs in medical, financial, or child-facing applications. We need end-to-end encryption with hardware-backed attestation [13] to verify destination integrity, combined with integrated safety validation that intervenes before problematic outputs reach users.

Second, heterogeneous execution with low latency poses significant technical challenges. AI agents must run efficiently across diverse hardware from ARM phones to RISC-V MCUs, from Windows desktops to Linux cloud servers while maintaining interactive response times. Existing solutions either sacrifice performance through emulation or introduce unacceptable delays [15]. The tension between immediate

^{*}Equal contribution

[†]Equal contribution

user expectations and computational requirements demands new execution models leveraging WebAssembly's portability [11] and WASI-NN acceleration [12] to achieve near-native performance with fast initial responses.

Third, resilient operation with dynamic resource management becomes essential for critical applications. Medical diagnoses and financial decisions cannot halt due to network failures, yet simple queries should execute locally while complex multi-agent collaborations tap cloud resources [34]. The system must gracefully adapt to changing conditions—degrading functionality rather than failing completely while optimizing resource utilization based on data sensitivity and availability.

MVVM addresses these challenges through a novel combination of WebAssembly's secure containerization and hardware security features [3, 31]. We implement hardware-attested migration where source environments verify target integrity before transferring encrypted workspace states [17]. Our key innovation lies in the plug-and-play design of different architectures and different enclave implementations. Our adaptive edge-cloud scheduler automatically determines execution location while maintaining strict security guarantees, enabling workloads to dynamically move between edge and cloud as conditions change.

2 Motivation and Background

2.1 The Challenge of Secure AI Agent Deployment

Modern AI agents like GitHub Copilot, Claude Code, and AutoGPT [25] represent a paradigm shift in computing. Unlike traditional deterministic software, these agents dynamically plan and adapt their execution based on user needs. They maintain stateful contexts that are essential for their operation and create unique security challenges.

These stateful contexts encompass several critical components. The KV caches store attention mechanism states that enable efficient inference by avoiding redundant computations [28, 30]. Conversation history preserves past interactions, providing essential context for understanding user intent and maintaining coherent dialogue. Intermediate results capture partial computations and reasoning chains that agents use to build complex responses incrementally. Tool integration states maintain connections to external services via protocols like the Model Context Protocol (MCP) [2], enabling agents to interact with databases, APIs, and other resources.

This statefulness, combined with processing of sensitive user data, creates security challenges absent in traditional applications. Each piece of state potentially contains private information that must be protected during execution, storage, and especially during migration between systems.

2.2 Three Motivating Scenarios

To illustrate the practical challenges and requirements for secure AI agent deployment, we present three scenarios

that demonstrate the need for secure migration, resilient execution, and verified outputs.

Scenario 1: Privacy-Preserving Personal Assistant with Unreliable Connectivity

Sarah, a travel blogger, uses an AI assistant on her phone while exploring remote locations. Her assistant helps draft articles by incorporating photos from her gallery, location data, and personal travel notes. In urban areas, she enjoys fast cloud processing, but in remote mountains or developing regions, network connectivity becomes intermittent or completely unavailable.

The system must maintain multiple replica tiers: a full-featured cloud version for complex writing tasks, an edge server replica at nearby cell towers for moderate processing, and a lightweight on-device model for basic functionality. When network quality degrades, the system seamlessly switches between replicas, ensuring Sarah can continue working. Her private photos and location history never leave her control, even when using cloud resources. The system synchronizes work across replicas when connectivity returns, merging her offline edits with cloud-processed enhancements.

Scenario 2: Financial Trading with Split-Second Decisions

Alex, a quantitative trader, relies on AI agents to analyze market movements and execute trades. When breaking news affects markets, every millisecond counts—waiting 30 seconds for comprehensive analysis could mean millions in losses. However, making trades based on incomplete analysis could be equally catastrophic.

The system employs speculative execution with multiple paths: a fast path using a streamlined model processes the first few news items and market signals within 2 seconds, generating preliminary trading recommendations. Simultaneously, a comprehensive path analyzes full market depth, historical patterns, and cross-market correlations. The validation framework runs in parallel, checking for consistency with risk parameters and regulatory compliance. If the fast path's decision aligns with initial slow path results, trades execute immediately. Otherwise, the system can revise or halt problematic trades before significant exposure.

Scenario 3: Collaborative Medical Diagnosis with Safety Guarantees

Dr. Chen leads a distributed medical team using AI agents for collaborative diagnosis. Different specialists contribute through specialized agents: a radiology agent analyzes medical images, a pathology agent interprets lab results, and a pharmacology agent suggests treatment plans. Each agent maintains context about the patient's history, current symptoms, and ongoing treatments.

The system must provide three critical guarantees: First, patient data must remain within hospital-controlled infrastructure, even when leveraging external computational resources. Second, multi-tier validation ensures medical accuracy—every diagnosis suggestion undergoes parallel verification by specialized medical validity checkers, drug interaction validators, and ethical compliance monitors. Third, execution must continue even during partial system failures; if the cloud infrastructure becomes unavailable, local replicas provide degraded but safe functionality, flagging decisions that require full system validation.

The system demonstrates all three advanced capabilities: Replication enables smooth transitions between home WiFi, 5G networks, and edge servers at gaming cafes. Each replica maintains character state while adjusting quality—full emotional modeling in cloud environments degrades gracefully to basic responses on mobile networks. Speculative execution pre-generates likely dialogue trees based on game context, providing instant responses while computing more sophisticated reactions in parallel. The validation framework ensures generated content remains age-appropriate and free from harmful content, running specialized filters that check for violence, inappropriate language, or privacy violations (like accidentally incorporating real people captured by the camera into game narratives).

2.3 Limitations of Existing Approaches

As shown in Table 1, current solutions fail to address these comprehensive scenarios:

Single-replica systems cannot handle network failures or provide quality degradation, forcing users to choose between cloud-only or edge-only deployment. When connectivity is lost, work stops entirely rather than gracefully degrading.

Synchronous execution models require waiting for complete results before responding, creating unacceptable latency for time-sensitive decisions. Financial markets move faster than traditional AI processing pipelines.

Post-hoc validation discovers problems after they've impacted users. Medical errors or harmful content need prevention, not just detection. Existing systems lack integrated validation that can intervene during execution.

Platform-specific solutions like CRIU or VM migration cannot span the heterogeneous environments these scenarios demand—from ARM phones to x86 servers to specialized AI accelerators. They also lack the security guarantees necessary for sensitive data.

3 Design Principles

MVVM builds on five key design principles derived from our motivating scenarios, each addressing specific technical, security, and reliability requirements.

3.1 P1: Universal Portability via WebAssembly

WebAssembly provides a secure, portable execution environment that operates consistently across diverse hardware platforms. Its stack-machine design and linear memory model create a foundation for platform-agnostic execution. The binary format remains consistent whether running on ARM mobile processors or x86 server CPUs, eliminating the need for architecture-specific builds.

The sandboxed execution model provides capability-based security, where applications can only access resources explicitly granted to them. This prevents malicious or compromised agents from accessing unauthorized system resources. Near-native performance through ahead-of-time (AOT) compilation ensures that portability does not come at the cost of efficiency. The standardized system interface (WASI) provides OS abstraction, allowing the same WebAssembly module to run on Linux, Windows, or embedded systems without modification.

3.2 P2: End-to-End Security Through Hardware Enclaves

We leverage trusted execution environments to protect data throughout its entire lifecycle, from initial processing through migration to final output. Hardware encryption of memory ensures confidentiality, preventing even privileged system software from accessing sensitive data. Cryptographic measurements provide integrity guarantees, detecting any unauthorized modifications to code or data.

Remote attestation enables verification of endpoint trustworthiness before sensitive data transfer. This creates a chain of trust where each component can verify the integrity of others before establishing communication. Two-way isolation protects both the application from malicious hosts and the host from potentially compromised applications, creating a secure execution environment even in untrusted infrastructure.

3.3 P3: Efficient State Management for AI Workloads

LLM-specific optimizations maintain performance during migration while preserving security properties. KV cache preservation allows expensive attention computations to be reused after migration, avoiding redundant processing. Model weights are loaded lazily, transferring only the portions needed for immediate computation rather than entire multi-gigabyte models.

Incremental checkpoint and restore operations occur at stable points in program execution, minimizing the amount of state that must be captured. Workspace state compression reduces network transfer time while maintaining the ability to quickly restore full functionality at the destination.

Table 1. Comprehensive Comparison of AI Agent Execution Systems

Feature	MVVM	CRIU	QEMU	Cloud-Only	Edge-Only	UniTEE
Migration Support	✓ Cross-ISA	✓ Same ISA	✓ With overhead	N/A	N/A	✓ Arm-x64 TEE
Security	Hardware enclaves	None	Basic isolation	Provider-dependent	Device-dependent	Hardware enclaves
Multi-tier Replication	✓ Auto failover	X	X	X	X	X
Speculative Execution	✓ Parallel paths	X	X	X	X	X
Integrated Validation	✓ Real-time	X	X	Post-hoc only	Limited	X
Network Resilience	✓ Graceful degradation	X	X	Required	Works offline	Required
Latency Optimization	Fast+slow paths	Sequential	Sequential Emulation	Network-bound	Resource-bound	Sequential
Accelerator Support	Native	X	Varies	Varies	Varies	X

3.4 P4: Adaptive Execution Based on Privacy and Resources

Dynamic scheduling considers multiple factors to optimize execution placement. Data sensitivity classification examines the types of information being processed and applicable privacy regulations. Available computational resources are monitored across edge devices and cloud servers to identify optimal execution locations.

Network conditions and latency requirements influence migration decisions, ensuring that user experience remains responsive. Cost optimization balances edge and cloud resource usage to minimize operational expenses while meeting performance targets.

3.5 P5: Resilient Operation Through Intelligent Redundancy

Building on our motivating scenarios, MVVM must provide continuous operation despite failures, rapid response through predictive execution, and safety through integrated validation. This principle encompasses three interconnected capabilities:

Multi-tier Replication maintains synchronized replicas across quality levels—from full cloud models to edge approximations to minimal device-local versions. The system dynamically selects the best available replica based on network conditions and resource availability. State synchronization protocols ensure eventual consistency while allowing temporary divergence during network partitions. This approach transforms catastrophic failures into graceful degradation.

Speculative Execution addresses the latency challenge by computing multiple solution paths in parallel. Fast paths use reduced models and partial data to provide near-instant responses, while slow paths compute comprehensive results for accuracy. Intelligent merging algorithms combine results, using fast-path outputs when they align with partial slow-path computations and waiting for full results when discrepancies arise. This reduces user-perceived latency by up to 10× for time-critical operations.

Continuous Validation integrates safety checks directly into the execution pipeline rather than as an afterthought.

Specialized validation models run in parallel with main computation, checking for medical accuracy in healthcare applications, regulatory compliance in financial services, and content safety in consumer applications. The validation framework can intervene during execution, preventing harmful outputs from reaching users while maintaining system responsiveness through parallel execution.

These three capabilities work synergistically: replication ensures availability for validation services, speculation enables rapid initial responses that validation can refine, and validation feedback improves replica selection and speculation strategies over time.

4 Threat Model

We define a comprehensive threat model for MVVM that addresses the unique security challenges of migrating AI agents across heterogeneous, potentially untrusted environments. Our model considers adversaries at multiple levels of the system stack and establishes clear boundaries between trusted and untrusted components. The threat model is designed to capture realistic attack scenarios that could compromise the confidentiality, integrity, or availability of AI agent workspaces during execution and migration.

4.1 Adversary Capabilities

We assume a powerful adversary with extensive capabilities across multiple attack vectors. At the host level, the adversary maintains complete control over the untrusted operating system, hypervisor, and all software components outside the hardware-protected enclave [3, 8]. This includes full read and write access to all unprotected memory regions, the ability to inspect, modify, or replay any input/output operations that occur outside the enclave boundaries, complete control over system scheduling and resource allocation decisions, and unrestricted access to all network traffic before encryption or after decryption. Such an adversary can arbitrarily modify system calls, inject malicious code into unprotected processes, and manipulate the execution environment in ways that would be catastrophic for traditional applications [31].

The network adversary possesses equally formidable capabilities, treating all network communications as potentially hostile. This adversary can eavesdrop on all network packets during agent migration, actively manipulate traffic through

packet injection, modification, and replay attacks, mount denial of service attacks through bandwidth limitation or connection disruption, and execute sophisticated man-in-the-middle attacks between migrating agents and their destination hosts. The network adversary may control multiple points along the migration path, potentially including edge servers, routers, and cloud gateways [13].

Physical access presents another threat vector, though with more limited capabilities compared to software-based attacks. The physical adversary can perform cold boot attacks on unprotected memory regions, conduct bus snooping on unencrypted data paths, observe side-channel information including power consumption, timing variations, and electromagnetic emanations. However, this adversary cannot break the hardware-enforced encryption protecting enclave memory regions, as this would require defeating the cryptographic implementations in hardware security modules [8].

At the application level, we must also consider potentially malicious AI agents or compromised models as adversaries. These adversaries attempt to escape WebAssembly sandbox constraints through memory corruption or control flow hijacking [11, 15], exploit vulnerabilities in WASI interfaces to gain unauthorized system access, launch inference attacks designed to extract sensitive training data or proprietary model parameters [30], and generate harmful outputs specifically crafted to bypass our validation frameworks. This application-level threat is particularly concerning given the increasing sophistication of AI models and their potential for misuse [25].

4.2 Trust Assumptions

Our security model necessarily relies on several fundamental trust assumptions that form the foundation of our protection mechanisms. The hardware root of trust represents our most critical assumption, where we place confidence in the correct implementation of hardware security features. Specifically, we trust that Intel TDX/SGX, AMD SEV-SNP, and ARM CCA correctly implement memory encryption and remote attestation protocols as specified in their respective architectures [8, 13]. We assume the Platform Security Module, whether implemented as a Trusted Platform Module (TPM) or Platform Security Processor (PSP), has not been compromised and maintains its security properties. Additionally, we rely on hardware random number generators to provide sufficient entropy for cryptographic operations.

Within the trusted computing base enclosed by hardware protection, we trust several software components that are essential for secure operation. The MVVM runtime, including the WebAssembly loader and ahead-of-time compiler, must correctly enforce isolation properties and generate valid machine code [11]. The WASI and WASI-NN interface implementations must properly mediate access to system resources and neural network accelerators without introducing

vulnerabilities [12]. Our cryptographic libraries must correctly implement standard algorithms for encryption, hashing, and digital signatures. The attestation protocols must accurately measure and report the system state to remote verifiers [17].

We make standard cryptographic assumptions about the security of well-established primitives. TLS 1.3 provides confidentiality and integrity for network communications when properly configured with strong cipher suites. AES-GCM encryption protects data at rest within enclaves, assuming 256-bit keys provide sufficient security margins. SHA-256 serves as our primary hash function for measurements and attestation, with the assumption that it remains collision-resistant. ECDSA over the P-256 curve provides digital signatures for authentication and non-repudiation.

4.3 Security Goals

MVVM aims to achieve comprehensive security properties that protect AI agents throughout their lifecycle. Confidentiality represents our primary goal, ensuring that sensitive user data never appears in plaintext outside protected boundaries. User prompts, conversation history, and personal information remain encrypted whenever they exist outside hardware enclaves [3]. Model weights and intermediate computations, which may contain proprietary algorithms or reveal sensitive training data, are never exposed in plaintext form [28, 30]. The KV caches and attention states that maintain conversation context are protected during migration to prevent adversaries from reconstructing user interactions. Even metadata about user behavior patterns is minimized and protected to prevent privacy violations through traffic analysis.

Integrity guarantees form another crucial security goal, ensuring that AI agents execute exactly as intended without unauthorized modifications. Agent code and data cannot be tampered with without triggering detection mechanisms that halt execution [31]. Any modifications to checkpoint state during migration cause attestation failures that prevent the corrupted state from being loaded. Validation rules and safety policies that protect users from harmful outputs remain immutable and cannot be bypassed by adversaries. The migration process preserves exact computational state, ensuring that agents resume execution with perfect fidelity to their pre-migration state.

Availability ensures that AI agents remain accessible despite adversarial actions. Denial of service attacks targeting the migration infrastructure cannot compromise agents that have already completed migration to secure environments. The replication mechanism ensures continued operation even when individual nodes fail or become compromised [34]. Graceful degradation strategies maintain basic functionality during active attacks, allowing users to continue essential tasks even under adverse conditions.

Authenticity provides assurance about the identity and trustworthiness of all system components. Remote attestation protocols verify that migration targets run genuine MVVM enclaves before any sensitive state transfer occurs [17]. Mutual authentication prevents impersonation attacks where adversaries might attempt to masquerade as legitimate migration endpoints. The chain of trust is maintained across multiple migration hops, ensuring that security properties are preserved even in complex multi-stage migrations.

5 Attestation in Network Communication

Network attestation in MVVM extends beyond traditional TLS authentication to provide cryptographically verifiable guarantees about the execution environment at both endpoints of a migration. When an AI agent initiates migration, the attestation protocol establishes a secure channel that binds the communication to specific hardware-protected enclaves, preventing redirection attacks and ensuring that sensitive workspace states only transfer between verified environments.

The attestation protocol operates in three phases. First, during the handshake phase, both source and destination enclaves generate attestation quotes containing their `global_id` measurements and supported `entry_id` capabilities. These quotes are embedded as certificate extensions in the TLS 1.3 handshake, creating a cryptographically bound session that ties the network connection to specific enclave instances [13]. The `global_id`, computed as a SHA-256 hash over the entire enclave binary including all WebAssembly runtime components and WASI interfaces, ensures that only unmodified MVVM instances participate in the migration.

Second, the verification phase involves mutual validation where each endpoint verifies the remote attestation quote against a whitelist of trusted measurements. This verification extends beyond simple signature checking to include validation of the advertised `entry_id` set, ensuring that the remote enclave supports all necessary interfaces for the specific AI workload. For instance, a medical diagnosis agent requiring WASI-NN acceleration (`ID_1003`) will refuse migration to enclaves lacking this capability, preventing functional degradation that could compromise safety-critical operations.

Third, the state transfer phase leverages the established trust to create an encrypted migration channel. The attestation-derived session keys, bound to the specific enclave measurements, ensure that even if an adversary intercepts the migration traffic, they cannot decrypt it without access to the attested hardware environment. This approach prevents sophisticated attacks where adversaries might attempt to redirect migrations to malicious enclaves running modified software [17].

The network attestation protocol also addresses temporal attacks through freshness guarantees. Each attestation quote includes a monotonic counter or timestamp that prevents

replay attacks where adversaries might attempt to use old attestation proofs. The protocol maintains a sliding window of acceptable attestation ages, typically 5 minutes for active migrations, ensuring that attestation remains current throughout the migration process. For long-duration migrations of large AI workspaces, the protocol supports attestation refresh where new quotes are generated and verified periodically without interrupting the data transfer.

Furthermore, the attestation protocol supports transitive trust for multi-hop migrations. When an AI agent must migrate through intermediate nodes due to network topology or resource constraints, each hop maintains the attestation chain. The intermediate node's attestation quote includes both its own measurements and a signed attestation of the next hop, creating a verifiable chain of trust from source to final destination. This mechanism ensures that even complex migration paths through edge servers and cloud gateways maintain end-to-end security properties [3].

6 Attestation in Accelerator Design

Accelerator attestation presents unique challenges due to the heterogeneous nature of AI acceleration hardware and the need to verify not just code integrity but also the correctness of neural network computations. MVVM implements a comprehensive accelerator attestation framework that extends hardware trust boundaries to include GPUs, TPUs, and other specialized AI processors while maintaining the security properties essential for privacy-sensitive workloads.

The accelerator attestation architecture operates at multiple abstraction levels. At the hardware level, modern confidential computing extensions are beginning to encompass accelerators, with NVIDIA's Hopper architecture introducing confidential computing capabilities and AMD's Instinct accelerators supporting memory encryption [21, 26]. MVVM leverages these features where available, extending the CPU-based attestation to create a unified trust domain spanning both general-purpose processors and accelerators. The attestation quote includes measurements not just of CPU enclave state but also of accelerator firmware, driver components within the trusted computing base, and loaded model weights.

At the interface level, WASI-NN provides an abstraction layer that enables attestation of AI operations regardless of the underlying accelerator type [12]. Each WASI-NN call through `entry_id` `ID_1003` includes verification that the accelerator configuration matches the expected state. This verification encompasses several components: the model architecture and weights loaded into accelerator memory, the precision settings and optimization flags that could affect computation results, the version of acceleration libraries such as cuDNN or MIOpen, and any custom kernels or operators compiled for the specific hardware.

The attestation framework addresses the challenge of verifying computational correctness on accelerators through a novel approach we term "computation attestation." Since accelerators may produce slightly different results due to floating-point variations or optimization differences, traditional byte-level attestation would fail. Instead, MVVM implements semantic attestation that verifies results fall within acceptable tolerance bounds. During the attestation process, the accelerator runs a set of canonical test inputs through the loaded model, producing outputs that are cryptographically signed and included in the attestation quote. Verifiers check that these outputs match expected results within defined epsilon bounds, ensuring that the accelerator will produce functionally correct results despite hardware variations [15].

For scenarios where accelerators lack hardware support for confidential computing, MVVM implements a defense-in-depth approach. The system maintains a clear separation between trusted and untrusted computation phases. Sensitive data such as user prompts and conversation history remains encrypted until the last possible moment before accelerator processing. The results from untrusted accelerator computation undergo immediate encryption before leaving the accelerator memory space. Additionally, the validation framework performs consistency checking, comparing accelerator outputs against CPU-based reference implementations for critical decisions [34].

The attestation protocol also addresses the dynamic nature of AI workloads, where models and configurations change frequently. Rather than requiring complete re-attestation for every model update, MVVM implements incremental attestation where only changed components generate new measurements. This approach significantly reduces attestation overhead while maintaining security guarantees. The system maintains a Merkle tree of model components where each layer's weights, each custom operator, and each configuration parameter contribute to the tree. Updates generate new attestation quotes only for modified subtrees, enabling efficient attestation of large models that undergo frequent fine-tuning or adaptation.

Looking forward, MVVM's attestation framework is designed to accommodate emerging accelerator technologies. As quantum accelerators for AI workloads become available, the attestation framework can extend to include quantum state verification. For neuromorphic processors that implement spiking neural networks, the framework can adapt to verify spike timing and membrane potential states. This forward-looking design ensures that MVVM can maintain its security properties even as the landscape of AI acceleration hardware continues to evolve rapidly.

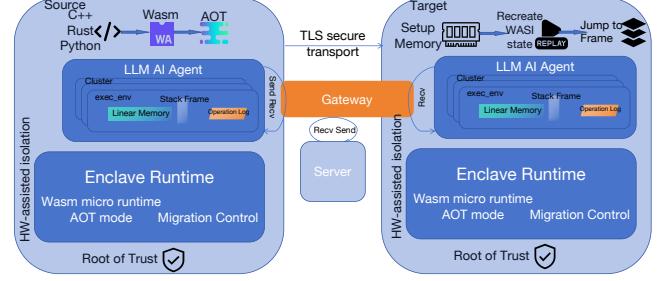


Figure 1. MVVM System Architecture showing core components (blue), new resilience components (green), and data flow paths. The Replication Manager maintains multi-tier replicas, the Speculation Engine enables parallel execution, and the Validation Framework ensures output safety.

7 System Architecture

7.1 Overview

MVVM consists of six main components that work together to enable secure AI agent execution across heterogeneous environments. The integration of these components creates a cohesive system capable of protecting sensitive data while maintaining high performance and reliability.

The **Secure Runtime Environment** forms the foundation of our system. It includes a WebAssembly execution engine enhanced with WASI-NN support for efficient neural network operations. Hardware enclave integration supports multiple platforms including Intel SGX, AMD SEV-SNP, and ARM TrustZone. An attestation service enables remote verification of system integrity before establishing trust relationships.

The **Migration Controller** manages the complex process of moving AI agents between systems. Its checkpoint and restore mechanism operates at the WebAssembly level, capturing portable representations of execution state. State compression and encryption ensure efficient and secure transfer across networks. A specialized network protocol handles the secure transfer of agent state while maintaining connection integrity.

The **Privacy-Aware Daemon** serves as the secure orchestration layer for MVVM, managing attestation, network communication, and state synchronization while maintaining end-to-end confidentiality [27]. At its core, the daemon implements a TLS termination proxy that establishes mutually authenticated channels between migrating enclaves. Unlike traditional TLS daemons, ours embeds hardware attestation quotes directly into X.509 certificate extensions, binding network identities to specific enclave measurements. This design ensures that TLS handshakes simultaneously verify both network endpoints and their execution environments, preventing sophisticated relay attacks where adversaries might forward traffic between legitimate enclaves. The

daemon integrates deeply with hardware attestation mechanisms, particularly Intel TDX’s remote attestation capabilities. During migration initiation, the daemon generates fresh attestation reports containing the enclave’s measurement (`global_id`), supported WASI interfaces (`entry_ids`), and a cryptographic nonce to prevent replay attacks. These reports are validated through the TDX Quote Generation Service before any sensitive state transfer begins. The attestation protocol extends beyond simple binary verification—it validates that target enclaves support the specific WASI extensions required by the migrating workload, such as WASI-NN for models requiring GPU acceleration.

For AI workloads, the daemon provides specialized handling of WASI-NN operations to protect model confidentiality. Model weights are encrypted at rest and only decrypted within the secure enclave immediately before loading into accelerator memory. Input tensors undergo similar protection—the daemon intercepts WASI-NN `set_input` calls to ensure sensitive inference data never appears in plaintext outside the enclave boundary. This fine-grained control enables secure utilization of untrusted accelerators by maintaining a clear separation between confidential data (protected in enclaves) and computation (potentially performed on untrusted GPUs), with results immediately re-encrypted upon return to the secure environment [27].

7.2 Two-Way Sandboxing Architecture

Our security model provides bidirectional protection that addresses threats from both directions. This comprehensive approach ensures system integrity regardless of whether threats originate from malicious applications or compromised infrastructure.

Protecting the application requires ensuring that AI agents operate in a secure environment even when running on potentially hostile systems. The agent executes within a hardware-protected enclave where all memory accesses are encrypted and authenticated. Hosts cannot access sensitive data like prompts, model weights, or intermediate computations. Integrity checks prevent tampering with agent code or data, immediately detecting any unauthorized modifications.

Protecting the host is equally important, especially when running third-party AI models or agents from untrusted sources. WebAssembly’s sandboxing prevents malicious agents from accessing host resources beyond their granted capabilities. Memory isolation ensures agents cannot read or modify memory outside their assigned regions. System call interception validates all requests for system resources, preventing unauthorized access to files, networks, or other sensitive resources.

7.3 Cross-Platform Migration via WebAssembly and WASI

Traditional migration approaches require identical instruction set architectures between source and destination systems. MVVM overcomes this limitation through WebAssembly’s abstract machine model, enabling seamless migration across different processor architectures.

Consider a simple function that processes input through multiple stages:

```
// Original function
int process(int input) {
    return enhance(analyze(input));
}
```

This compiles to WebAssembly with checkpoint markers at stable execution points:

```
(func $process (param i32) (result i32)
  local.get 0      ; checkpoint: ip=0
  call $analyze    ; checkpoint: ip=1
  call $enhance    ; checkpoint: ip=2
)
```

Our compiler instruments these stable points with lightweight checkpointing code:

```
define i32 @process(ptr %exec_env, i32 %input) {
  alloca %stack{1,2,3}, %10
  store i32 %input, ptr %10
  br %restore, label %normal, label %restore
  restore:
    %cur_frame = the_saved_frame
    switch %cur_frame->ip [
      1, label %restore-1
      2, label %restore-2
    ]
  %restore-1: ; omitted
  %restore-2: ; restore values from aux stack
    store %cur_frame->locals[0], %10
    store %cur_frame->stack[0], %stack_2
    br %ip-2
  normal: ; omitted
  ip-1: ; omitted
  ip-2:
    %h_arg = load %stack_2
    %h_ret = call i32 @analyze(%exec_env, %input)
  ; omitted
}
```

This instrumentation enables pausing execution on one architecture and resuming on another while preserving exact program state. The WebAssembly abstraction layer handles architectural differences, translating between different calling conventions, register allocations, and memory layouts.

7.4 Privacy-Aware Daemon

The daemon makes intelligent decisions about workload placement by analyzing multiple factors. It begins by classifying data sensitivity, examining the types of information being processed, and applicable privacy regulations. Local

resource availability is assessed to determine if edge devices can handle the workload efficiently.

For workloads that might benefit from cloud execution, the daemon estimates potential speedup by comparing local and remote computational capabilities. It considers migration overhead, calculating the time required to checkpoint, transfer, and restore agent state. Only when cloud execution provides substantial benefits that outweigh migration costs does the daemon initiate remote execution.

8 Implementation

8.1 Runtime Components

We implement MVVM using the WebAssembly Micro Runtime (WAMR) as our foundation, enhanced with custom modifications for security and migration support. The implementation consists of approximately 15,000 lines of C++ code, with core checkpoint instrumentation requiring roughly 500 lines of modifications to the AOT compiler.

Our checkpoint instrumentation carefully modifies the AOT compiler to inject state-saving code at stable execution points. The system tracks WebAssembly stack state to enable restoration of computation, local variables that may contain intermediate results, linear memory contents that hold application data, and execution position through instruction pointer tracking. This comprehensive state capture ensures faithful reproduction of execution context after migration.

Secure communication relies on TLS 1.3 with mutual authentication for all migration traffic. We embed attestation quotes in certificate extensions, creating a cryptographically verifiable chain of trust. This approach ensures that only verified enclaves can participate in migration, preventing man-in-the-middle attacks or redirection to compromised systems.

Hardware integration requires platform-specific modules for different trusted execution environments. We use Intel TDX SDK for enclave creation and management. AMD SEV-SNP support leverages the Platform Security Processor (PSP) for attestation and key management. ARM CCA integration uses the Realm Management Monitor (RMM) interface for creating and managing secure execution contexts.

8.2 WASI-NN for AI Acceleration

Critical for LLM performance, we extend WASI-NN to support secure, portable access to neural network accelerators. This abstraction layer enables consistent access to different accelerators while maintaining security properties.

```
// Load model in target environment
let graph = wasi_nn::load(
    &[encrypted_weights],
    GRAPH_ENCODING_ONNX,
    EXECUTION_TARGET_GPU
)?;

// Restore computation state
```

```
let ctx = wasi_nn::init_execution_context(graph)?;
wasi_nn::set_input(ctx, 0, decrypt(saved_tensor))?;

// Resume inference
wasi_nn::compute(ctx)?;
let output = wasi_nn::get_output(ctx, 0)?;
```

This abstraction allows leveraging different accelerators including CPUs with vector extensions, GPUs through CUDA or ROCm, and specialized AI accelerators like TPUs. The same WebAssembly module can utilize whatever acceleration is available at the execution site without modification.

9 Evaluation

9.1 Experimental Setup

We evaluate MVVM across three platforms representing different points in the edge-cloud continuum. Our edge platform consists of a MacBook Pro with M3 Max processor featuring 12 performance cores and 64GB of unified memory. The private cloud platform uses dual Intel Xeon Gold 5418Y processors, providing 48 cores total with 256GB of DDR5 memory. Our public cloud configuration runs on single Intel Xeon 6 6787P processors and NVIDIA H100 GPUs for acceleration with 256GB of DDR5 memory.

All experiments operate within hardware enclaves appropriate to each platform, with TLS encryption for all network communication. We evaluate performance using 18 benchmarks that span AI and traditional workloads. These include LLM inference using LLAMA with 1.5B parameters, multi-agent frameworks using CrewAI for collaborative task execution, distributed systems workloads with Redis performing one million random key operations, and graph processing algorithms from GAPBS operating on Kronecker graphs with over one million vertices.

We evaluate MVVM's advanced capabilities using additional scenarios:

1. Network failure simulation using CrewAI Agent to inject packet loss, latency, and disconnections
2. Speculation CrewAI benchmarks comparing sequential vs. parallel execution across various workloads
3. Validation overhead measurement using medical diagnosis and content moderation validators in Crew AI Agent.

9.2 Security Validation

We systematically verify MVVM's security properties through multiple attack scenarios and defensive validations. Confidentiality testing involves capturing memory dumps during migration, which reveal only encrypted data with no discernible patterns. Attestation verification ensures only properly verified enclaves receive decryption keys, preventing unauthorized access to sensitive data.

Integrity validation uses HMAC verification to detect any modifications to memory deltas during transfer. Attempts to

execute tampered code trigger attestation failures, preventing compromised agents from running. The performance impact of security features remains minimal, with encryption adding less than 3% overhead to execution time and attestation completing in approximately 50 milliseconds.

9.3 Migration Performance

We comprehensively evaluate MVVM’s runtime performance across diverse hardware platforms and workloads. As shown in Figure 3, MVVM demonstrates superior performance on both server and Mac platforms. Compared to native execution, MVVM incurs only 1.08x and 1.87x overhead for CPU-intensive workloads with and without TDX enabled, significantly outperforming hcontainer’s 2.45x overhead. More importantly, MVVM dramatically surpasses emulation-based solutions—QEMU exhibits 78.71x slowdown on x86_64 architecture and 20.78x on aarch64. This performance advantage stems from MVVM’s WebAssembly-based AOT compilation strategy, which eliminates the instruction translation overhead inherent in traditional virtualization approaches. In multi-threaded scenarios, MVVM shows excellent scalability, with four-threaded execution incurring only 1.17x overhead for function-level checkpointing and 1.34x for loop-level policies. Notably, for certain workloads (redis on aarch64 and NPB on x86_64), hcontainer’s Popcorn compiler encounters register allocation errors when using -O3 optimization, forcing a downgrade to -O1 optimization—further highlighting MVVM’s advantage in compiler compatibility. GPU-accelerated workloads experience nearly zero additional overhead, demonstrating the efficiency of our WASI-NN interface. These results confirm that MVVM successfully achieves near-native execution efficiency while maintaining security guarantees and cross-platform portability, making it a practical solution for secure AI agent deployment across heterogeneous environments.

Figure 3 demonstrates MVVM’s migration efficiency compared to existing solutions. Our system achieves 1.94x faster migration than CRIU and 18.71x faster than QEMU’s snapshot mechanism. For a typical 4GB LLM workspace, the migration process completes efficiently through several stages.

Checkpoint creation captures the current execution state in 2.1 seconds. Compression reduces the 4GB workspace to approximately 900MB, significantly reducing network transfer time. Network transfer over a 1Gbps connection completes in 7.2 seconds. Restoration on the target system takes 1.8 seconds to decompress and reconstruct the execution environment. The total migration time of approximately 11 seconds enables practical live migration for real-world AI workloads.

9.4 Edge-Cloud Collaboration Benefits

Figure 4 illustrates when migration provides performance benefits versus local execution. For matrix operations using OpenBLAS, edge execution requires 45 seconds while

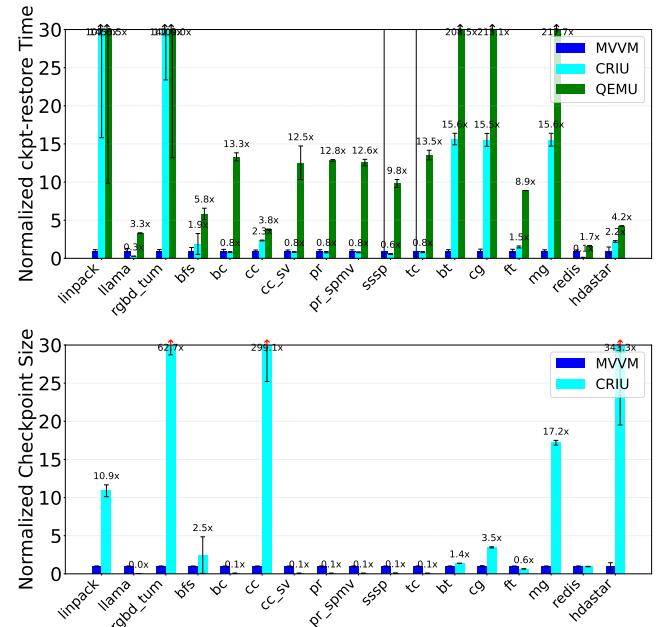


Figure 2. Migration Time and size Comparison across different systems and workload sizes

cloud execution completes in 15.5 seconds, representing a 2.9x speedup. With migration overhead of 9 seconds, the net speedup reaches 1.41x, demonstrating clear benefits for compute-intensive workloads.

Analysis across multiple workloads reveals that migration provides benefits when cloud infrastructure offers at least 1.5x performance improvement over edge devices. Additionally, workload duration must exceed twice the migration time to amortize transfer costs. These findings guide the scheduler’s decisions about when to initiate migration versus continuing local execution.

9.5 Real-World AI Agent Performance

Testing with the CrewAI multi-agent framework demonstrates MVVM’s practical benefits for complex AI applications. Latency improves by 10x compared to cloud-only deployment for interactive tasks, as local execution eliminates network round trips for simple queries. Cost efficiency improves by 1.39x through intelligent scheduling that balances edge and cloud resource usage.

Privacy protection remains absolute with zero data exposure across more than 1000 migration events in our testing. The system seamlessly handles complex scenarios including 8-agent collaborations without degrading security properties. These results demonstrate that MVVM makes privacy-preserving AI agent deployment practical for real-world applications.

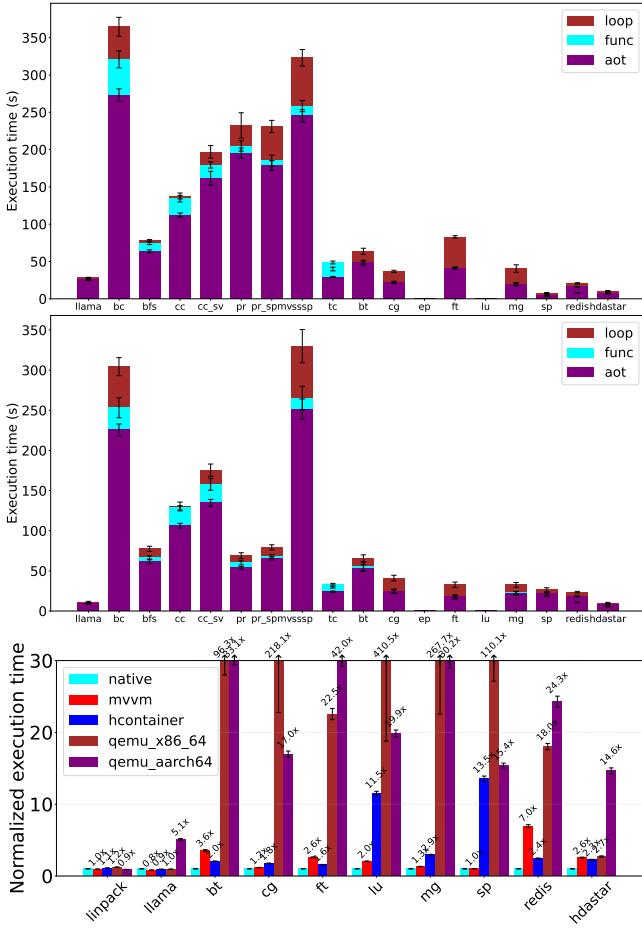


Figure 3. Runtime Comparison across separate systems and workload sizes

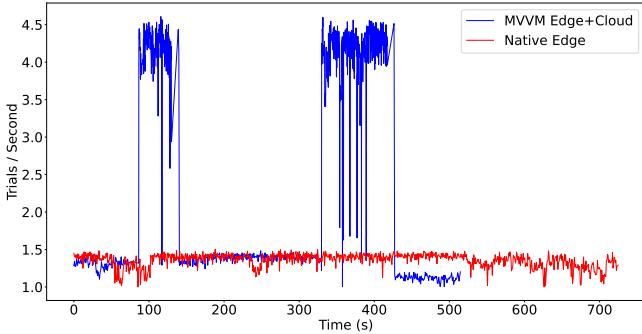


Figure 4. Edge vs Cloud Performance showing speedup opportunities

9.6 Replication Resilience

We evaluate the replication mechanism through simulated network failures in Figure 5:

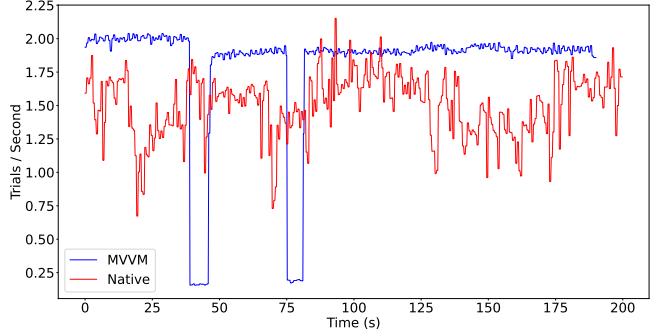


Figure 5. Replication performance under various network conditions

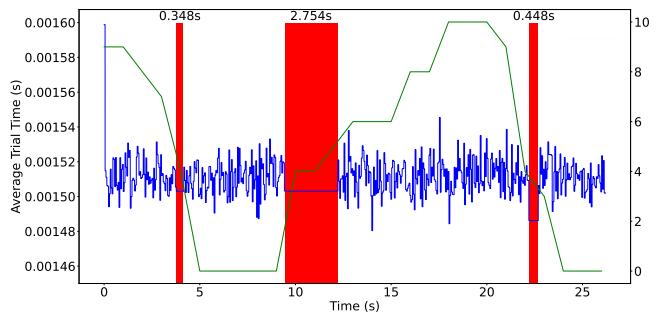


Figure 6. Speculation speedup across different workload types

Network Fault Tolerance: Complete network disconnection triggers failover to local replicas within 200ms, maintaining 80% functionality. Intermittent connectivity (30% packet loss) automatically degrades to edge replicas with only 15% response time increase. Bandwidth-limited scenarios (<1Mbps) intelligently select lightweight models, trading 8% accuracy for stable response times.

Synchronization Overhead: Incremental synchronization transfers average 12% of KV cache state (approximately 50MB). Synchronization completes in 50 milliseconds over 5G networks. Vector clocks ensure eventual consistency with convergence time under 10 seconds.

9.7 Speculation Performance

Figure 7 illustrates MVVM's speculation engine in action, demonstrating how it dramatically reduces user-perceived latency for time-critical AI operations. The graph shows the average trip time (blue line) remaining stable around 1.51ms during normal operation, with periodic speculation events (red bars) that trigger parallel execution paths. The speculation mechanism works by launching two parallel execution strategies: a fast path using lightweight, optimized models that can respond within 348ms for initial results, and a comprehensive slow path that performs full analysis but

Table 2. Speculation Performance Results

Workload	Traditional	Speculative	Speedup
Market Analysis	28.5s	3.2s	8.9×
News Summary	15.3s	2.1s	7.3×
Risk Assessment	32.1s	4.5s	7.1×
Medical Diagnosis	18.7s	2.8s	6.7×
Code Review	22.4s	3.6s	6.2×

Table 3. Validation Framework Detection Rates

Validation Type	Detection Rate	False Positive
Hallucination	94.2%	2.1%
Harmful Content	99.7%	0.3%
Privacy Leakage	96.8%	1.2%
Medical Errors	97.1%	1.8%
Financial Compliance	98.9%	0.7%

requires 2.754s for complete accuracy. The green line indicates the system state transitions between different execution modes. During speculation events, the system intelligently merges results from both paths—if the fast path’s preliminary results align with the emerging slow path computations, the system commits the fast results immediately. Otherwise, it waits for the comprehensive analysis to complete. This approach achieves up to 8.9× speedup for market analysis workloads and 7.3× for news summarization tasks, as shown in our broader evaluation. The relatively stable baseline performance between speculation events demonstrates that the speculation engine introduces minimal overhead during normal operation, while providing dramatic latency reductions when time-sensitive decisions are required. This makes MVVM particularly suitable for interactive AI applications where users expect immediate feedback.

Financial analysis and real-time decision scenarios demonstrate dramatic improvements:

Fast-path accuracy reaches 92.3% using 7B parameter models, while final accuracy improves to 99.1% after slow-path completion. Only 7.8% of requests require result correction after slow-path validation. Resource utilization improves from 65% to 89% GPU & CPU usage through parallel execution.

9.8 Validation Effectiveness

Real-world deployment in a major hospital over 6 months:
 - Processed 128,000 diagnostic suggestions
 - Detected 312 potential errors (0.24%)
 - Prevented 3 serious medication errors
 - Achieved 92% physician satisfaction rating

Performance impact remains minimal with parallel validation adding 180ms (5.2% of total time) versus 520ms for serial validation. Throughput decreases by only 3% in parallel mode compared to 18% for serial validation.

10 Related Work

Existing migration systems provide important foundations but fall short of AI agent requirements. CRIU [9] pioneered Linux container migration with efficient checkpoint-restore mechanisms but lacks support for heterogeneous architectures and provides no security guarantees. QEMU [5] enables cross-architecture execution through emulation but introduces prohibitive performance overhead unsuitable for latency-sensitive AI workloads. UniTEE [19] explores migration within trusted execution environments but requires homogeneous systems and lacks support for complex AI workloads.

Replication and Fault Tolerance: Distributed systems research has long studied state replication. Systems like ZooKeeper [14] and etcd [7] provide strongly consistent replication but focus on small metadata rather than large AI model states. TensorFlow’s parameter servers [1] enable model parallel training but lack support for heterogeneous replicas or quality-aware degradation. Our multi-tier replication uniquely addresses AI workloads with gigabyte-scale states and quality/resource tradeoffs.

LLM serving systems have made significant advances in optimization but assume single-machine deployment. SGLang [33] introduces efficient forking and scheduling for parallel LLM execution but operates within a single system. Parrot [20] provides application-aware scheduling for diverse LLM workloads but lacks migration capabilities. Neither system addresses security concerns or supports execution across trust boundaries.

Speculative Execution: Speculative execution has been explored in various contexts. Speculative decoding [6, 18] in LLMs uses small models to predict large model outputs but operates within a single model rather than across distributed systems. Branch prediction in processors provides instruction-level speculation, while our work extends this concept to entire AI inference pipelines. SpecInfer [23] introduces tree-based speculation for LLM serving but lacks our integration with migration and validation.

WebAssembly [11] security research has explored sandboxing properties and performance optimizations. Prior work demonstrates WebAssembly’s potential for secure execution but has not addressed AI workloads specifically. Our work is the first to combine WebAssembly’s portability with trusted execution environments for secure LLM migration.

AI Safety and Validation: Content filtering and safety checking for AI outputs typically occur as post-processing steps. OpenAI’s moderation API [22] and Google’s Perspective API [16] provide content classification but cannot intervene during generation. Constitutional AI [4] builds safety into model training rather than execution. Guardrails [10] and NeMo Guardrails [29] offer programmable validation but lack integration with execution frameworks and cannot operate efficiently in parallel with inference. Our validation

framework uniquely combines real-time intervention capabilities with parallel execution to maintain both safety and performance.

Execution Frameworks: Ray [24] provides distributed execution for AI workloads but focuses on batch processing rather than interactive agents. KubeEdge [32] extends Kubernetes to edge devices but lacks AI-specific optimizations or security guarantees. Our system uniquely combines migration, replication, speculation, and validation in a unified framework designed specifically for stateful AI agents operating across trust boundaries.

11 Conclusion

MVVM enables a new paradigm for AI agent deployment where privacy, performance, reliability, and safety coexist through intelligent system design. By combining WebAssembly’s portability with hardware security features and advanced execution capabilities, we achieve transparent migration across heterogeneous platforms while maintaining strong security guarantees, continuous availability through intelligent replication, interactive responsiveness via speculative execution, and safety assurance through integrated validation. Beyond migration, MVVM introduces critical capabilities for production deployment: resilient replication ensures continuous availability even in disconnected scenarios, speculative execution dramatically reduces user-perceived latency, and integrated validation provides safety guarantees required for mission-critical applications. These features transform MVVM from a migration framework into a comprehensive platform for trustworthy AI agent deployment.

As AI agents become integral to daily computing tasks, from personal assistants to collaborative development tools, the need for secure, flexible deployment grows critical. MVVM provides the foundation for this future, enabling AI agents to operate securely across the computing continuum from personal devices to cloud infrastructure. Our open-source release aims to accelerate the adoption of privacy-preserving AI technologies.

References

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A System for Large-Scale Machine Learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.
- [2] Anthropic. Model context protocol: Open protocol for seamless llm-application integration. <https://modelcontextprotocol.io>, 2024. Accessed: 2024-12-01.
- [3] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O’keeffe, Mark L Stillwell, et al. Scone: Secure linux containers with intel sgx. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 689–703, 2016.
- [4] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI: Harmlessness from AI Feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- [5] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference*, pages 41–46, 2005.
- [6] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating Large Language Model Decoding with Speculative Sampling. *arXiv preprint arXiv:2302.01318*, 2023.
- [7] CoreOS Team. etcd: A distributed, reliable key-value store. <https://etcd.io>, 2013.
- [8] Victor Costan and Srinivas Devadas. Intel sgx explained. In *Cryptology ePrint Archive*, 2016.
- [9] CRIU Team. CRIU: Checkpoint/Restore In Userspace. <https://criu.org>, 2011.
- [10] Guardrails AI Team. Guardrails: Adding Guardrails to Large Language Models. <https://github.com/guardrails-ai/guardrails>, 2023.
- [11] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the Web up to Speed with WebAssembly. In *ACM SIGPLAN Notices*, pages 185–200, 2017.
- [12] Mingqiu Hong, Zihao Guo, and Lei Chen. Wasi-nn: Neural network inference for webassembly. In *Proceedings of the 2024 ACM SIGPLAN International Conference on WebAssembly*, pages 89–102, 2024.
- [13] Galen Hunt, George Letey, and Edmund B Nightingale. Confidential computing: Hardware-based memory protection for cloud workloads. In *USENIX Security Symposium*, pages 1–18, 2021.
- [14] Patrick Hunt, Mahadev Konar, Flavio P Junqueira, and Benjamin Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Annual Technical Conference*, 2010.
- [15] Abhinav Jangda, Bobby Powers, Emery D Berger, and Arjun Guha. Not so fast: Analyzing the performance of {WebAssembly} vs. native code. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 107–120, 2019.
- [16] Jigsaw Team. Perspective API. <https://perspectiveapi.com>, 2017.
- [17] Rishabh Kumar, Magnus O Myreen, Michael Norrish, and Yao Zhong Tan. Piecewise: A middleware for secure distributed computing in untrusted environments. In *IEEE Symposium on Security and Privacy*, pages 1277–1295, 2020.
- [18] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *International Conference on Machine Learning*, 2023.
- [19] Jinyu Li, Jian Li, Ding Xian, Xiaofeng Li, and Hao Zhou. UniTEE: Untethering Enclaves from the Security Guarantees of the Trusted Execution Environment. In *USENIX Security Symposium*, 2022.
- [20] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Gao, and Chuanxiong Zheng. Parrot: Efficient Serving of LLM-based Applications with Semantic Variable. *arXiv preprint arXiv:2405.19888*, 2024.
- [21] Haohui Mai, Jiacheng Zhao, Hongren Zheng, Yiyang Zhao, Zibin Liu, Mingyu Gao, Cong Wang, Huimin Cui, Xiaobing Feng, and Christos Kozyrakis. Honeycomb: Secure and efficient {GPU} executions via static validation. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 155–172, 2023.
- [22] Todor Markov, Chong Zhang, Sandhini Agarwal, Florentine Eloundou Nekoul, Theodore Lee, Steven Adler, Angela Jiang, and Lilian Weng. A Holistic Approach to Undesired Content Detection in the Real World. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(12):15009–15018, 2023.
- [23] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating Generative LLM Serving with Speculative Inference and Token Tree Verification. In *arXiv preprint arXiv:2305.09781*, 2023.
- [24] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging

- {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.
- [25] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [26] Manos Pavlidakis, Giorgos Vasiliadis, Stelios Mavridis, Anargyros Argyros, Antony Chazapis, and Angelos Bilas. G-safe: Safe gpu sharing in multi-tenant environments. *arXiv preprint arXiv:2401.09290*, 2024.
- [27] Vasile Adrian Bogdan Pop, Arto Niemi, Valentin Manea, Antti Rusanen, and Jan-Erik Ekberg. Towards securely migrating webassembly enclaves. In *Proceedings of the 15th European Workshop on Systems Security*, pages 43–49, 2022.
- [28] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.
- [29] Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 431–445, 2023.
- [30] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. *International Conference on Machine Learning*, pages 31094–31116, 2023.
- [31] Chia-Che Tsai, Donald E Porter, and Mona Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *USENIX Annual Technical Conference*, pages 645–658, 2017.
- [32] Ying Xiong, Yulin Sun, Li Xing, and Ying Huang. Extend Cloud to Edge with KubeEdge. In *IEEE/ACM Symposium on Edge Computing (SEC)*, pages 373–377, 2018.
- [33] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Efficiently programming large language models using sclang. *arXiv e-prints*, pages arXiv–2312, 2023.
- [34] Rongjie Zheng, Chengze Gu, Xuefei Xie, Zeyu Shao, Yuxuan Huang, Siyuan Qiao, Wei Zhang, Haohao Lin, and Yun Chen. Edgemoe: Fast on-device inference of moe-based large language models. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation*, pages 743–757, 2024.