

Proactive Security for Large-Scale AI Inference: Adaptive Container Orchestration in Kubernetes

Akram Sheriff 

Cisco Systems
San Jose, CA, USA
isheriff@cisco.com

Zsolt Németh

R6 Security
Budapest, Hungary
zsolt@r6security.com

Ken Huang 

Distributedapps.ai
San Francisco, CA, USA
ken.huang@distributedapps.ai

Abstract—Static, long-lived AI inference deployments—such as NVIDIA NIM containers on Kubernetes—give adversaries stable footholds for reconnaissance, persistence, and lateral movement. This paper presents an AMTD technique (Adaptive NIMs), a Kubernetes-native Automated Moving Target Defense (AMTD) framework that continuously rotates inference containers, mutates runtime configurations, and couples telemetry with automated enforcement. Unlike detection-only runtime tools, the proposed AMTD technique treats eviction as a first-class control: compromised or long-lived instances are replaced from signed golden images before attackers can complete a kill chain.

We define a precise threat model for in-pod and in-node footholds, situate the proposed AMTD technique relative to prior AMTD work (including ADA and related microservice/serverless MTD), and evaluate four deployment regimes plus S1–S4 ablations on a multi-cloud Kubernetes testbed with repeated trials. Adaptive rotation raises mean kill-chain disruption to about 94% and reduces attacker dwell time to minutes, while keeping availability near 99.8% with roughly 1.7% compute overhead. Results show proactive orchestration is a practical enforcement layer for AI inference, complementary to runtime detection; optional modules are reported with measured incremental gains rather than assumed.

I. INTRODUCTION

The rapid growth of generative AI and large language models (LLMs) has shifted computational demand toward high-performance inference in containerized GPU clusters. Platforms such as NVIDIA NIMs enable unprecedented scale, but production deployments are often *static*: the same container image, identity, network endpoints, and GPU bindings persist for hours or days. That stability is convenient for operators and costly for defenders, because every hour without change is an extra hour for an attacker to map, probe, and exploit the serving stack.

Traditional cloud and container security emphasizes detection and patching after compromise. Runtime sensors can raise alerts, yet they seldom *guarantee* removal of an attacker’s foothold without human or scripted intervention. **Automated Moving Target Defense (AMTD)** takes the opposite stance: it continuously shifts the attack surface so that reconnaissance ages quickly and persistence is repeatedly invalidated [3]–[5]. Prior AMTD work has focused on networks, VMs, and general containers; its application to AI inference—where GPU utilization, warm-up cost, and latency SLOs dominate—remains limited.

In this paper we propose an **AMTD technique** (Adaptive NIMs), tailored to Kubernetes-hosted AI inference. Our contributions are as follows:

- 1) A precise threat model for AI inference on Kubernetes that defines in-pod footholds, in-node footholds, and lateral movement, and that clarifies what AMTD can and cannot mitigate.
- 2) A capability-oriented comparison (Table I) that explains four security capabilities—ephemeral rotation, configuration mutation, telemetry-driven triggers, and zero-downtime handoff—and how each improves security relative to static, detection-only, and prior rotation systems.
- 3) A Kubernetes-native architecture and four core randomization strategies (container re-randomization, ephemeral pod rotation, network/namespace mutation, and memory-layout randomization), including GPU-aware micro-shuffling and staggered warm-standby handoff.
- 4) An empirical evaluation with clearly motivated baselines—Static, Static+Runtime, Timed Rotation, and Adaptive AMTD—showing large gains in kill-chain disruption and dwell-time reduction at modest cost.

The remainder of this paper reviews related work (Section II), presents the threat model (Section III), details the proposed AMTD design and randomization mechanisms (Section IV), and reports experimental results (Section V).

II. BACKGROUND AND RELATED WORK

A. Automated Moving Target Defense

AMTD reduces an adversary’s ability to perform reconnaissance, persistence, and exploitation by continuously changing system configuration [3]. Classical techniques include network address randomization, instruction-set randomization, and virtual-machine migration [4], [5]. Surveys of moving-target methods in networking and cloud settings emphasize that dynamism raises attacker cost but must be balanced against availability and operational overhead [6], [7]. Dynamic microservice rotation and shuffling [8] and secure serverless workflow execution with environment replacement [9] are closer to containerized deployments, yet they do not address GPU-bound inference constraints such as model warm-up, MIG placement, or KV-cache continuity.

B. ADA: Ephemeral Rotation for AI Workloads on Kubernetes

ADA (Automated Moving Target Defense for AI Workloads via Ephemeral Infrastructure-Native Rotation in Kubernetes) [1] showed that continuous, automated rotation of AI workloads can disrupt attacker persistence: pods and related objects are rotated on a schedule so process memory, local credentials, and container identities are discarded and replaced from known-good images.

Three gaps remain for inference. First, ADA’s rotation is largely workload-agnostic and does not reason about GPU warm-up, Leader-Worker warm pools, or latency SLOs. Second, it emphasizes infrastructure churn more than detection-coupled enforcement with warm standbys. Third, it does not mutate hardening profiles (Seccomp, AppArmor) or GPU bindings at each cycle, leaving a partially stable surface between rotations. The proposed AMTD technique keeps ADA’s ephemeral-rotation insight but adds telemetry-weighted triggers, GPU-aware placement, and zero-downtime handoff.

C. AI Infrastructure Security and Static NIM Deployments

Generative AI systems inherit cloud-native risks and add model-specific threats: prompt injection, model extraction, poisoning, and exfiltration of proprietary weights or sensitive prompts. Container and isolation research has long shown that cloud-native stacks trade isolation strength for agility [2], [10]. NVIDIA NIMs package models as portable inference containers; in practice many deployments remain long-lived with stable digests, Services, and GPU bindings. Runtime tools such as Falco or CNAPP sensors improve visibility, but detection alone leaves a remediation window for escalation, lateral movement, or artifact extraction.

D. Orchestration Security, Detection-Only Defenses, and Closest MTD Systems

A second line of prior work hardens the *platform* rather than continuously relocating the workload. Kubernetes-oriented defenses emphasize admission control, least-privilege RBAC, NetworkPolicy isolation, and runtime sensors that raise alerts when a process, syscall, or network flow looks anomalous [2], [10]–[14]. Those controls are necessary, but they remain largely *reactive*: an in-pod foothold can persist until an operator (or a separately scripted playbook) evicts the pod. Classical and cloud MTD surveys likewise stress that effective moving-target designs must couple configuration diversity with operational constraints on availability and overhead [5]–[7]. Among executable MTD systems closest to ours, Xu et al. [8] optimize multi-dimensional replica/rotation schedules against graph lateral movement in microservice settings, and Liang et al. [9] replace serverless execution environments under makespan limits. Neither targets GPU-bound NIM inference with warm-standby cut-over, telemetry-accelerated eviction, or MIG-aware placement. Confidential-computing and stronger isolation layers (e.g., VM- or TEE-backed containers) can raise the cost of escape and in-node persistence, but they do not by

themselves invalidate a long-lived in-pod foothold or stolen in-memory credential; AMTD is therefore complementary to—not a substitute for—isolation and detection.

E. Positioning of This Work

Table I summarizes how the proposed AMTD technique differs along four capabilities: prior systems either detect without guaranteed eviction, or rotate without inference-aware adaptivity and configuration mutation. The proposed technique combines both, and it specializes the rotation envelope to inference SLOs that generic microservice or serverless MTD formulations do not encode.

III. THREAT MODEL

A. System and Defender Assumptions

We consider AI inference services deployed as containers (typically pods) on Kubernetes, in cloud or edge GPU clusters. The defender controls the orchestration plane sufficiently to create, terminate, and reschedule workloads, pull signed images from an immutable registry, and apply NetworkPolicies and admission controls. We assume the container registry and control plane are not fully compromised; if an adversary controls the API server or can push malicious “golden” images, rotation alone cannot restore integrity. Clients reach inference through Kubernetes Services (or an ingress/load balancer), so backend pod identity may change without changing the external URL. We further assume that signed image digests are verified at admission (e.g., via policy controllers) and that GPU nodes expose standard device-plugin interfaces used by the Rotation Engine for placement.

B. Terminology

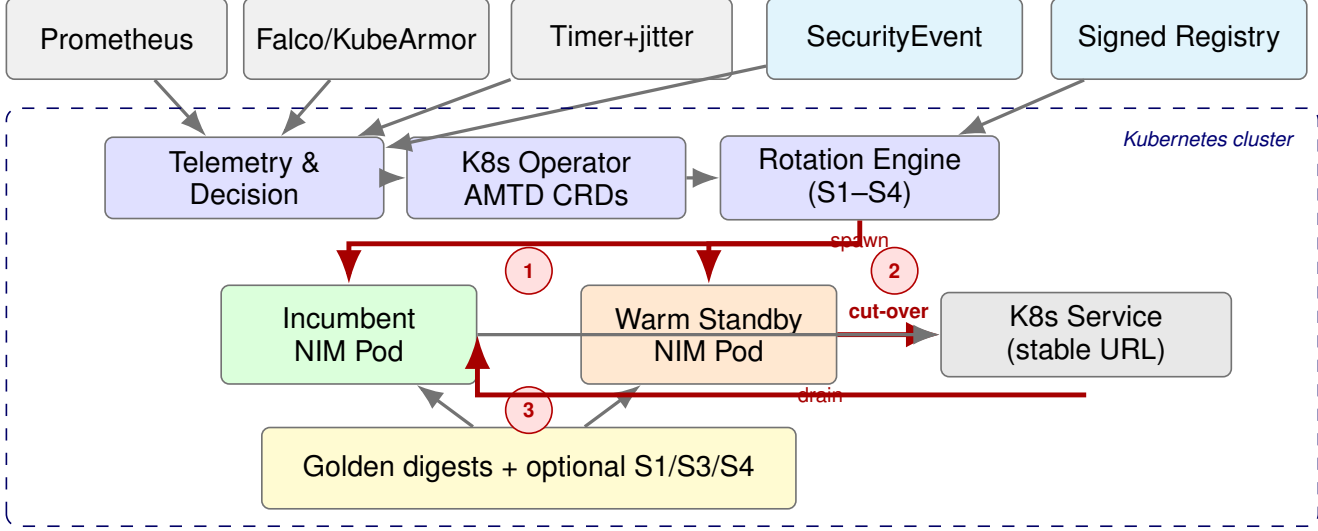
- **In-pod foothold:** Attacker-controlled code or state inside a compromised inference container—e.g., a reverse shell, stolen secrets in process memory, or a patched serving binary. Clean-image replacement removes it.
- **In-node foothold:** Host-level control that survives pod deletion (escape implant, tainted runtime, stolen kubelet credentials). Pod rotation alone does not clear it; AMTD shortens the window before escape succeeds.
- **Lateral movement:** Propagation via stolen ServiceAccount tokens, adjacent NIMs, or permissive NetworkPolicies; identity and endpoint churn raise cost.
- **Dwell time:** Duration an adversary retains a usable foothold before eviction.
- **Kill chain:** ATT&CK-aligned stages from reconnaissance through impact/exfiltration used in our evaluation.

C. Adversary Goals and Capabilities

The adversary may partially know the cluster architecture (common image names, Service DNS, GPU node pools) but lacks continuous, real-time visibility into ephemeral pod identities and placement after each rotation. The adversary can perform remote reconnaissance, exploit known container or application vulnerabilities, attempt container escape, and, once inside a pod, attempt persistence and lateral movement.

TABLE I
CAPABILITY COMPARISON. “YES” DENOTES A FIRST-CLASS, AUTOMATED MECHANISM. AMTD DENOTES THE PROPOSED TECHNIQUE.

Capability	Static NIM	Runtime detection	ADA [1]	AMTD (ours)
C1: Ephemeral container rotation	–	–	Yes	Yes
C2: Runtime configuration mutation	–	Partial	Limited	Yes
C3: Telemetry-driven / adaptive triggers	–	Alert only	Fixed / limited	Yes
C4: Zero-downtime staggered handoff for GPUs	–	–	Limited	Yes



Cut-over: (1) standby Ready with GPU/MIG; (2) Service endpoints shift; (3) terminate incumbent (wipe in-pod foothold). S2=core; S1/S3/S4 optional.

Fig. 1. Proposed AMTD architecture. Telemetry (metrics, alerts, jittered timers) drives the Operator and Rotation Engine, which pull signed NIM images and execute a numbered warm-standby cut-over behind a stable Service while optional S1/S3/S4 policies mutate runtime identity.

Primary goals are (i) long-lived persistence, (ii) theft or extraction of proprietary models, (iii) exfiltration of sensitive prompts or outputs, and (iv) availability disruption.

Out of scope for AMTD-as-enforcement are full control-plane compromise, supply-chain compromise of the signed golden image before it enters the registry, and purely algorithmic attacks that succeed in a single request without needing persistence (e.g., one-shot prompt injection that does not plant a foothold). Those threats require complementary controls (admission signing, model-level defenses, and least-privilege RBAC).

D. How AMTD Maps to These Threats

The proposed AMTD technique mitigates in-pod footholds by terminating and replacing containers from immutable images. It disrupts reconnaissance and lateral movement by changing pod IPs, identities, and optionally network and GPU bindings. It increases attack cost by forcing re-exploitation after each cycle. It does not, by itself, remediate established in-node footholds; instead it narrows the window in which escape and host persistence can be completed.

IV. PROPOSED AMTD DESIGN

A. Design Objectives

The proposed AMTD technique aims to (1) disrupt adversary persistence without violating inference latency and availability SLOs, (2) automate orchestration-level defense with minimal manual intervention, (3) integrate with existing Kubernetes and NIM deployment pipelines, and (4) adapt rotation frequency using live telemetry rather than timers alone.

B. Four Capabilities and Their Security Benefit

We treat the four rows of Table I as first-class capabilities.

a) *C1: Ephemeral container rotation.*: Inference pods are periodically replaced from signed golden images. Any in-pod foothold—malware in memory, backdoored processes, stolen tokens—is discarded with the old pod, so persistence cannot rely on a long-lived process tree.

b) *C2: Runtime configuration mutation.*: At each rotation, AMTD may reshuffle ports, Service backends, Sec-comp/AppArmor profiles, and selected RBAC or NetworkPol-

icy bindings within a safe envelope, so attackers cannot assume a fixed hardening layout across cycles.

c) *C3: Telemetry-driven adaptive triggers.*: Besides a baseline timer, the Telemetry & Decision Module consumes Prometheus metrics and runtime alerts (Falco, KubeArmor). Elevated anomaly scores accelerate rotation, shortening dwell immediately rather than waiting for the next fixed interval—a key ADA limitation.

d) *C4: Zero-downtime staggered handoff.*: Warm standbys and Service endpoint updates bring a replacement NIM to Ready before draining the old pod, staggering rotations across replicas so eviction remains forceful without trading away availability.

C. System Architecture

The proposed AMTD system comprises four components: (1) a **Kubernetes Operator** that reconciles rotation policy custom resources; (2) a **Rotation Engine** that executes re-deployments with randomized parameters under time-based, event-based, or hybrid strategies; (3) an **Immutable Image Registry** of signed NIM images with rollback; and (4) a **Telemetry & Decision Module** that correlates metrics and alerts into rotation urgency. Figure 1 shows the end-to-end control and data-plane cut-over path.

D. How Containers Are Rotated Randomly

AMTD does not restart the same long-lived container in place. Instead, it *replaces* inference pods from signed golden images while randomizing enough identity and placement attributes that an adversary cannot reuse prior reconnaissance or keep an in-pod foothold across cycles. Randomization applies at three layers: *when* a rotation fires, *which* replica is rotated, and *what* configuration the replacement receives.

a) *Triggers with temporal jitter.*: An AdaptiveMovingTargetDefense policy sets a base interval T (20–30 min in evaluation). The Rotation Engine samples jitter δ uniformly from $[-\alpha T, +\alpha T]$ ($\alpha \approx 0.1$ – 0.3), yielding deadline $T + \delta$ so attackers cannot synchronize to a fixed clock. Independently, Integration Backends translate runtime alerts into SecurityEvent objects that can force immediate rotation with a shorter grace period.

b) *Replica selection and staggered cut-over.*: Among Ready, non-draining replicas, AMTD selects the next victim uniformly under a concurrent-rotation budget so a quorum remains available. For each selected pod, a warm standby pulls the signed image, attaches a GPU/MIG slice, and reaches Ready; only then do Service endpoints shift and the incumbent terminate, wiping memory, local credentials, and in-pod implants.

c) *What is randomized on each replacement.*: Each cycle samples a fresh configuration from a defender envelope:

- **Identity**: new pod name/UID and, where permitted, cluster IP; Service DNS stays stable for clients.
- **Placement**: optional random reassignment among eligible GPU nodes or MIG partitions.

- **Network surface**: shuffled ports and regenerated NetworkPolicy/sidecar bindings within policy.
- **Hardening profile**: a randomly chosen admissible Seccomp/AppArmor profile from a signed allow-list.

Digests remain pinned; failed readiness aborts cut-over and retries rather than failing open. Timed Rotation retains only scheduled replacement; Adaptive AMTD enables the full envelope (Table III).

E. Core Randomization and Defense Strategies

The proposed AMTD technique organizes defense into four strategies. **S2 is the primary, fully measured mechanism**; S1, S3, and S4 are optional modules isolated via ablation (Section V-E).

a) *S1: Container re-randomization (optional).*: Mutates Seccomp/AppArmor, ports, and service-account mounts; optional request-path schema/length checks. Deep semantic anti-poisoning is out of scope for quantitative claims.

b) *S2: Ephemeral infrastructure rotation (core).*: Destroys and respawns pods from signed golden images with warm-standby cut-over—the main driver of dwell-time reduction and kill-chain disruption in Tables II–III.

c) *S3: Network and namespace mutation (optional).*: Regenerates NetworkPolicies, reshuffles Service backends, quarantines pods, and optionally hops short-lived namespaces. Default evaluation enables NetworkPolicy refresh and quarantine.

d) *S4: Memory layout randomization (optional).*: GPU/MIG rebinding is enabled by default. Disposable encrypted volumes and cross-capsule sharding exist in the prototype but are not claimed as fully stress-tested; ablation reports disabling MIG/device rebinding only.

S2 realizes C1; S1/S3 contribute to C2; C3 accelerates enabled modules; C4 makes S2 practical for GPUs.

F. Operator Behavior and Failure Modes

The operator reconciles AdaptiveMovingTargetDefense and SecurityEvent CRDs. Soak tests on a 20-node slice: **failed standby**→abort cut-over, keep incumbent, backoff (1.4% of attempts); endpoints update only after readiness gates; **registry pull timeout**→same path (pin digests; pull-through cache for large NIMs); **alert storms**→per-Deployment token-bucket limits on event-driven rotations.

V. EVALUATION

A. Experimental Setup

We evaluated the proposed AMTD technique on Kubernetes clusters spanning AWS and CoreWeave with approximately 400 mixed GPU/CPU nodes. The workload was an LLM served via an NVIDIA NIM container with a multi-replica Deployment behind a ClusterIP Service. Monitoring used Prometheus, Grafana, and KubeArmor. Rotations were staggered across replicas. Timed policies used a 20–30 minute base interval with jitter; adaptive policies combined that baseline with on-demand SecurityEvent triggers. Security results use $N=50$ trials per TTP per regime (Section V-C). Warm

standbys were sized to keep at least one Ready backend per model during cut-over.

B. Baselines and Motivation

We compare four deployment regimes and deliberately avoid “B0–B3” labels so the contribution is not conflated with controls:

- **Static:** Long-lived NIM pods with no rotation—today’s default production posture.
- **Static+Runtime:** Static pods plus Falco-style/CNAPP alerting without automated eviction.
- **Timed Rotation:** ADA-style fixed-interval ephemeral rotation without telemetry acceleration or full mutation [1].
- **Adaptive AMTD (proposed):** Hybrid timed+event-driven rotation with warm handoff, mutation, and GPU-aware placement.

Timed Rotation is not a bit-exact ADA reimplementation; it isolates adaptivity (C3) and mutation (C2) beyond “rotation vs. none.” Xu et al. [8] and Liang et al. [9] are related SOTA classes (microservice/serverless MTD) but do not target GPU NIM warm-standby cut-over on our stack.

C. Attack Methodology, Scoring, and Repeated Trials

a) *TTPs.*: Each trial runs one ATT&CK-aligned playbook (Metasploit + custom kubectl helpers): **T1** in-pod RCE/reverse shell and retain foothold; **T2** steal ServiceAccount tokens/secrets and list API objects; **T3** lateral move to a peer pod or writable namespace; **T4** attempt GPU memory/weight dump or bulk egress exfiltration. Table III maps these to kill-chain stages.

b) *Scoring and repetition.*: Attacker success means the stage goal completes within 30 minutes before remediation; *disrupted* means rotation/quarantine removes the foothold first or NetworkPolicy/RBAC denies the step. Disruption rate = disrupted/total trials. Dwell time = beacon→foothold loss. Per regime and TTP we run $N=50$ independent trials (800 security trials total), reseeding jitter; operational metrics use 72-hour soaks at 100 req/s. We report mean±sample std (and may cite 95% CI $\bar{x} \pm 1.96 s/\sqrt{N}$). Red-team operators were blind to the next jittered rotation deadline; defenders used only telemetry and policy already available to the Operator.

D. Results

Table II summarizes outcomes as mean±std over repeated trials. Adaptive AMTD reaches $94.1 \pm 2.3\%$ disruption versus $41.8 \pm 4.1\%$ Static and $62.7 \pm 3.6\%$ Static+Runtime, with $99.81 \pm 0.04\%$ availability and $1.7 \pm 0.3\%$ cost overhead. Timed Rotation achieves $86.9 \pm 2.8\%$ disruption, confirming ADA-like churn; the proposed AMTD technique’s adaptivity and enabled mutation modules close most of the residual gap. Mean dwell falls from unbounded (Static) or 9.4 ± 3.1 days (detection-only, until manual response in our script) to 6.2 ± 2.4 min (Timed) and 4.8 ± 1.9 min (AMTD). Rotation cut-over latency remains sub-second on average (0.81 ± 0.21 s), dominated by readiness gating rather than API thrash.

TABLE II
OPERATIONAL AND SECURITY COMPARISON (MEAN±STD). AMTD DENOTES THE PROPOSED TECHNIQUE.

Metric	Static	+Runtime	Timed	AMTD
Rot. latency (s)	N/A	N/A	0.72 ± 0.18	0.81 ± 0.21
Availability (%)	99.84 ± 0.03	99.83 ± 0.03	99.80 ± 0.05	99.81 ± 0.04
Cost overhead (%)	0	6.2 ± 2.1	1.3 ± 0.2	1.7 ± 0.3
Disruption (%)	41.8 ± 4.1	62.7 ± 3.6	86.9 ± 2.8	94.1 ± 2.3
Dwell (if foothold)	Unbounded	9.4 ± 3.1 d	6.2 ± 2.4 min	4.8 ± 1.9 min

TABLE III
ATTACKER SUCCESS RATE (% , MEAN±STD, $N=50$ /STAGE; LOWER IS BETTER).

Stage	Static	+Run.	Timed	AMTD
Reconnaissance	82.1 ± 3.4	67.6 ± 3.9	29.8 ± 3.1	9.7 ± 2.2
Weaponization	85.7 ± 3.1	59.9 ± 4.0	27.2 ± 2.9	8.1 ± 2.0
Delivery	76.8 ± 3.6	54.2 ± 3.8	19.1 ± 2.7	6.2 ± 1.8
Exploitation	71.2 ± 3.9	52.0 ± 3.7	15.3 ± 2.5	5.1 ± 1.6
Persistence	93.6 ± 2.4	61.1 ± 3.5	26.2 ± 2.8	5.2 ± 1.7
Priv. escalation	81.8 ± 3.2	52.3 ± 3.6	18.4 ± 2.6	3.3 ± 1.4
Defense evasion	84.7 ± 3.0	54.1 ± 3.7	27.9 ± 3.0	5.0 ± 1.6
Lateral movement	76.9 ± 3.5	43.2 ± 3.9	14.1 ± 2.4	4.1 ± 1.5
Impact / exfil.	71.8 ± 3.8	35.2 ± 3.4	8.2 ± 2.1	2.4 ± 1.1

Table III details stage-wise attacker success. Static deployments leave most paths viable; runtime detection helps but leaves large residual success; Timed Rotation cuts rates sharply; Adaptive AMTD drives most stages to single digits, especially post-compromise steps that need lasting footholds. Figure 2 plots the same comparison.

E. Ablation of Strategies S1–S4

Starting from Adaptive AMTD, we disable one module at a time (Table IV). Removing S2 (no pod respawn; alerts only) collapses disruption toward the Static+Runtime regime, confirming ephemeral rotation as the dominant factor. Disabling adaptive triggers (C3) while keeping S2 recovers Timed-like performance ($86.9 \pm 2.8\%$). Turning off S1 hardening/port mutation or S3 NetworkPolicy/quarantine yields modest but consistent losses, largest on T3 lateral movement. Disabling S4 MIG/device rebinding mainly affects T4 impact success. Optional namespace hopping and encrypted-volume modes were tested on a subset ($N=20$); they did not change disruption beyond one standard deviation, so we do not claim measured benefit for those sub-features. Overall, the ablation supports a deployment guideline: enable S2+C4 first for GPU inference, then layer C3 for alert-driven acceleration, and enable S1/S3/S4 when the threat model emphasizes lateral movement or device-level residual risk.

TABLE IV
ABLATION ON DISRUPTION RATE (% , MEAN±STD, $N=50$ POOLED TTPS).

Configuration	Disruption
Adaptive AMTD (full default)	94.1 ± 2.3
w/o S1 (no hardening/port mutation)	91.2 ± 2.5
w/o S3 (no NetworkPolicy refresh/quarantine)	89.4 ± 2.7
w/o S4 (no MIG/device rebinding)	92.0 ± 2.4
w/o C3 (timed S2 only)	86.9 ± 2.8
w/o S2 (detection+alert only)	63.1 ± 3.5

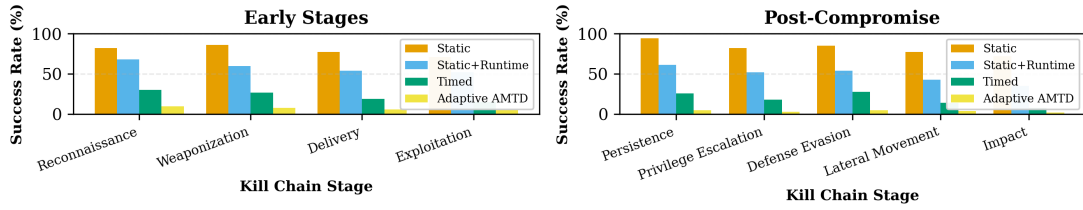


Fig. 2. Attack success by defense across kill-chain stages (Adaptive AMTD <10% everywhere).

F. Discussion and Limitations

Detection and enforcement are complementary: S2+C3 explain most measured gains, while S1/S3/S4 add smaller, TTP-specific increments (largest on lateral movement and impact). Models with warm-up beyond about 60s need careful staggering and larger standby pools; noisy sensors can pressure the event-driven rate limiter; established in-node footholds still require host remediation. Our TTPs are representative of common container kill chains but are not exhaustive of novel zero-days. Optional namespace-hop and encrypted-volume modes remain engineering features pending larger dedicated studies. In production terms, enable S2+C4 first for GPU inference availability, then C3 once alert quality is acceptable, and finally S1/S3/S4 when lateral movement or device co-residency is in scope. AMTD does not replace supply-chain signing, prompt-level defenses, or node attestation; those controls address threats outside the in-pod dwell-time problem studied here. Cost overhead remains dominated by warm-standby GPU reservation rather than control-plane thrash.

VI. CONCLUSION

We presented the proposed AMTD technique for large-scale AI inference. With a precise threat model, explicit randomization mechanics, failure-mode handling, repeated-trial evaluation, and S1–S4 ablation, adaptive container orchestration raises kill-chain disruption to about 94% and shrinks dwell time to minutes at roughly 1.7% cost overhead and 99.8% availability—complementing, rather than replacing, runtime detection. Proactive, inference-aware orchestration is thus a practical enforcement layer for Kubernetes-hosted NIMs: eviction becomes an automated control rather than a manual incident-response step. Operators can pair existing sensors with policy-driven rotation so a successful exploit does not buy a long-lived foothold on expensive GPU backends. Adaptivity and warm-standby handoff are what separate Adaptive AMTD from timer-only rotation; optional S1/S3/S4 modules should be enabled selectively against lateral movement and device-binding risks. Future work includes deeper native integration with Kubernetes and Slurm, stronger isolation and measurement of optional S3/S4 sub-features, risk-score-driven rotation policies, and joint evaluation with confidential-container runtimes where in-node escape is the dominant residual risk.

ACKNOWLEDGMENT

The authors thank colleagues at Cisco, R6 Security, and Distributedapps.ai for feedback on the Adaptive NIMs threat model, rotation-policy design, and evaluation methodology, and anonymous reviewers of earlier drafts for improving the baselines and ablation clarity. Any remaining errors are our own.

REFERENCES

- [1] A. Sheriff, K. Huang, Z. Németh, and M. Nakhjiri, “ADA: Automated Moving Target Defense for AI Workloads via Ephemeral Infrastructure-Native Rotation in Kubernetes,” arXiv:2505.23805, 2025.
- [2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” *Commun. ACM*, vol. 59, no. 5, pp. 50–57, 2016.
- [3] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*. Springer, 2011.
- [4] E. Al-Shaer, “Toward Network Configuration Randomization for Moving Target Defense,” in *Moving Target Defense*. Springer, 2011, pp. 153–159.
- [5] B. C. Ward, S. R. Gomez, R. W. Skowrya, D. Bigelow, J. N. Martin, J. W. Landry, and H. Okhravi, “Survey of Cyber Moving Targets, Second Edition,” MIT Lincoln Laboratory TR-1228, 2018.
- [6] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, “A Survey of Moving Target Defenses for Network Security,” *IEEE Commun. Surv. Tutor.*, vol. 22, no. 3, pp. 1909–1941, 2020.
- [7] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, “Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense,” *IEEE Commun. Surv. Tutor.*, vol. 22, no. 1, pp. 709–745, 2020.
- [8] H. Xu, G. Cheng, X. Yang, W. Liu, D. Zhou, and W. Guo, “Multi-Dimensional Moving Target Defense Method Based on Adaptive Simulated Annealing Genetic Algorithm,” *Electronics*, vol. 13, no. 3, art. no. 487, 2024.
- [9] H. Liang, S. Zhang, X. Liu, G. Cheng, H. Ma, and Q. Wang, “SMWE: A Framework for Secure and Makespan-Oriented Workflow Execution in Serverless Computing,” *Electronics*, vol. 13, no. 16, art. no. 3246, 2024.
- [10] I. Mavridis and H. Karatza, “Combining Containers and Virtual Machines to Enhance Isolation and Extend Functionality on Cloud Computing,” *Future Gener. Comput. Syst.*, vol. 94, pp. 674–696, 2019.
- [11] M. Souppaya, J. Morello, and K. Scarfone, “Application Container Security Guide,” NIST Special Publication 800-190, 2017.
- [12] T. Combe, A. Martin, and R. Di Pietro, “To Docker or Not to Docker: A Security Perspective,” *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 54–62, 2016.
- [13] S. Sultan, I. Ahmad, and T. Dimitriou, “Container Security: Issues, Challenges, and the Road Ahead,” *IEEE Access*, vol. 7, pp. 52976–52996, 2019.
- [14] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, 2014.