Adaptive Side-Channel Attack Mitigation for Cloud Security via Model Predictive Control and Deep Learning

Jiashen Liu, Xin Yang, Hang Xue, Xin Li, Zhongjie Ba, Shanchen Pang.

***Abstract*—The widespread co-location of virtual machines (VMs) in cloud environments exposes systems to cache-based side-channel attacks, which exploit micro-architectural resource contention to compromise confidentiality. Existing defenses largely emphasize detection or static mitigation, but they rarely offer adaptive optimization strategies that simultaneously enhance security and resource efficiency. This paper proposes an adaptive side-channel attack mitigation framework that integrates deep learning–based attack risk prediction with model predictive control (MPC)–driven VM migration optimization. The framework consists of three modules: a data collection module that collects real-time hardware performance counter (HPC) events and resource utilization metrics; a deep learning module trained on HPC datasets to predict VM attack risks; and an MPC module that forecasts system dynamics and computes adaptive migration strategies. By formulating a multi-objective optimization problem, the framework effectively balances attack risk mitigation, power consumption, migration overhead, and resource utilization. Extensive experiments under representative attack scenarios, including Flush+Flush, Flush+Reload, Prime+Probe, Meltdown, and Spectre variants, show that the proposed approach significantly improves attack detection accuracy and reduces vulnerability, while maintaining efficient resource scheduling. To the best of our knowledge, this is the first work that couples deep learning with MPC for adaptive and predictive VM migration against side-channel attacks in cloud environments.**

***Index Terms*—Cloud Computing, Side-Channel Attacks, Virtual Machine Migration, Deep Learning**

I. Introduction

Cloud computing has become a cornerstone of modern information infrastructure due to its scalability, flexibility, and cost efficiency. To maximize resource utilization, public cloud providers frequently co-locate virtual machines (VMs) belonging to different tenants on the same physical server. Although virtualization ensures logical isolation, the sharing of micro-architectural resources—most notably CPU caches—introduces severe security vulnerabilities. Cache-based side-channel attacks exploit subtle timing variations in shared resources to infer sensitive information across VMs, threatening the confidentiality of cloud services.

Conventional defenses against side-channel attacks primarily rely on static isolation mechanisms (e.g., cache partitioning, scheduling-based isolation) or runtime detection techniques (e.g., anomaly detection, performance counter monitoring). While effective in constrained scenarios, these approaches suffer from several limitations: static defenses often lead to significant performance overhead and resource underutilization, whereas detection-based methods typically act in a reactive manner and cannot provide proactive protection. The increasing sophistication and persistence of such attacks underscore the need for adaptive mitigation frameworks that can dynamically balance system security with efficient resource scheduling.

To address this challenge, we propose an adaptive mitigation framework that leverages deep learning for attack risk prediction and model predictive control (MPC) for dynamic optimization of VM migration strategies. The framework comprises three tightly coupled components. **Data Collection Module:** continuously monitors hardware performance counter (HPC) events along with CPU and memory utilization. **Moe-based Deep Learning Module:** employs a classifier trained on HPC datasets to provide accurate, real-time probability estimates of VM attack risks. **MPC Decision Module:** integrates risk predictions with resource utilization to forecast system evolution and compute adaptive migration strategies that minimize risks while maintaining system efficiency.

The contributions of this paper are threefold:

- Mixture of Experts (MoE)–Enhanced Deep Learning: We introduce a Mixture of Experts–based deep learning module to improve the accuracy and scalability of attack risk prediction. By dynamically selecting specialized expert networks according to input features, the MoE architecture captures heterogeneous attack patterns and resource utilization behaviors, thereby achieving more precise and robust predictions under diverse side-channel attack scenarios.
- Adaptive and Multi-Objective Optimization: We develop the first predictive migration optimization framework that couples deep learning–based risk prediction with model predictive control (MPC), enabling proactive and adaptive virtual machine migration decisions based on both current system states and forecasted future risks.
- Multi-Objective Optimization: We formulate a novel multi-objective optimization model that simultaneously

This work was supported by the National Key R&D Program of China (2023YFB2904000, 2023YFB2904001), the Shandong Provincial Natural Science Foundation (ZR2024QF156). *(Corresponding author: Xin Yang.)*

Xin Yang, Shanchen Pang is with the Qingdao Institute of Software College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, China, and with Jiaxing Research Institute, Zhejiang University, Jiaxing 314031, China, and also with Shandong Key Laboratory of Intelligent Oil & Gas Industrial Software, Qingdao, 266580, China. Email:xiny@upc.edu.cn.

Zhongjie Ba is with the State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou 310027, China. E-mail: zhongjieba@zju.edu.cn.

TABLE I
REPRESENTATIVE CACHE-BASED SIDE-CHANNEL ATTACKS AND REFERENCE IMPLEMENTATIONS

Attack Type	Attack Principle	Reference Source Code
Flush+Reload	Exploits cache timing characteristics by measuring the cache access time of other virtual machines, enabling the theft of sensitive data like encryption keys.	[1]
Flush+Flush	Repeatedly flushes the cache and measures access time to infer whether data has been accessed. The attack leverages the cache storage competition mechanism.	[2]
Prime+Probe	The attacker inserts "interfering data" into the shared cache and monitors cache latency when the target virtual machine accesses the data, revealing target behavior.	[3]
Meltdown	Exploits CPU microarchitectural vulnerabilities to bypass memory access controls and steal memory from other virtual machines. It manipulates out-of-order execution to access unauthorized memory.	[4]
Spectre v1	Exploits branch prediction mechanisms by causing incorrect predictions in speculative execution to leak sensitive data.	[4]
Spectre v2	Exploits indirect branch prediction vulnerabilities and cache timing to leak sensitive data during speculative execution.	[5]
Spectre v4	Similar to Spectre v2, but leverages more advanced speculative execution vulnerabilities to access and leak isolated memory content.	[4]

minimizes attack risks, migration overhead, and power consumption while improving resource utilization, providing a tunable balance between security and operational efficiency.

Overall, this study introduces a proactive and adaptive approach to mitigating side-channel attacks in cloud computing, thereby contributing both theoretical insights and practical solutions toward secure, efficient, and intelligent cloud infrastructures.

II. BACKGROUND AND RELATED WORK

III. DATASET CONSTRUCTION

A. Attack Reproduction

To systematically characterize cache-based side-channel attacks in virtualized cloud environments, we reproduced a set of representative attack techniques that exploit shared micro-architectural resources to extract sensitive information. Specifically, we implemented and executed seven well-studied attack variants—Flush+Reload, Flush+Flush, Prime+Probe, Meltdown, Spectre v1, Spectre v2, and Spectre v4—based on publicly available reference implementations. The brief principles and corresponding references for these attacks are summarized in Table I.

The experimental environment was established on a physical host equipped with an AMD Ryzen 7 5800H processor, running a virtualized infrastructure with a Ubuntu 18.04 guest VM configured with 4 GB of memory. To ensure dataset diversity and generalizability, we simulated three workload intensity levels—low, medium, and high—using the sysbench benchmarking tool. For each workload condition, all seven attacks were executed, and a no-attack control group was included for baseline comparison.

B. HPC Data Collection and Feature Selection

To capture fine-grained system behavior during both attack and normal executions, we employed the Linux `perf` tool to collect hardware performance counter (HPC) events at 0.1-second intervals. A total of 20 HPC events were initially recorded, covering micro-architectural behaviors such as cache accesses and misses, branch execution, and CPU cycles, as listed in Table II.

TABLE II
INITIAL HARDWARE PERFORMANCE COUNTER (HPC) EVENTS COLLECTED VIA `perf`

Perf	Hardware Event	Hardware Cache Event
Event	1.branch-instructions 2.branch-misses 3.cache-misses 4.cache-references 5.cpu-cycles 6.instructions 7.stalled-cycles-backend 8.stalled-cycles-frontend	9.L1-dcache-load-misses 10.L1-dcache-loads 11.L1-dcache-prefetches 12.L1-icache-loads 13.L1-icache-load-misses 14.branch-load-misses 15.branch-loads 16.dTLB-load-misses 17.dTLB-loads 18.iTLB-load-misses 19.iTLB-loads

TABLE III
SELECTED HPC EVENTS AFTER FEATURE SELECTION

Perf	Hardware Event	Hardware Cache Event
Event	1.branch-instructions 2.cache-misses 3.cache-references 4.cpu-cycles 5.instructions 6.stalled-cycles-frontend	7.L1-dcache-load-misses 8.L1-dcache-loads 9.branch-loads 10.dTLB-loads

To reduce redundancy and highlight the most informative indicators of attack activity, we applied a Random Forest–based feature selection procedure. Events were ranked by their relative importance, and the top nine features demonstrating the strongest discriminative power for distinguishing attack from normal conditions were retained. The selected HPC events are summarized in Table III.

C. Dataset Composition and Statistics

The final dataset consists exclusively of the nine selected HPC events. For each workload condition (low, medium, and high), we collected 25,000 samples for each of the seven attack types and 35,000 samples for the no-attack baseline, resulting

in 210,000 samples per workload condition. Across all three workloads, the dataset comprises a total of 630,000 samples, stored as individual CSV files.

This large-scale dataset provides a rich characterization of micro-architectural signatures associated with side-channel attacks under varying workload intensities. It serves as the foundation for training and evaluating the deep learning–based attack risk prediction model in our proposed framework.

This dataset, containing rich micro-architectural signatures across multiple attack variants and workload conditions, provides the foundation for the subsequent methodology. Specifically, it is used to train the deep learning model for real-time attack risk prediction and to drive the MPC optimization model for adaptive VM migration.

IV. Methodology

Building on the constructed dataset, we design a hybrid methodology that integrates deep learning and Model Predictive Control (MPC) to achieve adaptive attack mitigation in cloud environments. The deep learning component leverages the collected HPC events to predict attack risks in real time, while the MPC component incorporates these predictions into a multi-objective optimization framework for VM migration. Together, these two components form the core of our adaptive mitigation strategy.

A. Deep Learning–Based Risk Prediction

B. MPC Optimization Model

In large-scale cloud environments, virtual machine (VM) migration policies must jointly optimize security, resource efficiency, and operational overhead. To this end, we formulate a Model Predictive Control (MPC) model that dynamically determines VM placement and migration strategies. The objective is to minimize a weighted composite cost function that integrates co-residency attack risk, power consumption, migration overhead, and resource imbalance.

Model Predictive Control (MPC) is employed to capture the sequential and dynamic nature of VM migration decisions in cloud environments.Unlike static optimization, MPC formulates the migration problem over a finite prediction horizon $[t, t+W]$, where future attack risks, resource utilization, and migration costs are jointly predicted.

At each time step, the controller solves an optimization problem to obtain a sequence of candidate migration actions. Only the first action is executed, after which the system state is updated and the optimization is repeated in a receding horizon manner.

This design enables the migration policy to adaptively respond to evolving attack behaviors and workload dynamics while maintaining long-term optimality.

1) Objective Function: The total cost at time t is defined as:

$$C(t) = \alpha \cdot C_{Cr}(t) + \beta \cdot C_{Pc}(t) + \gamma \cdot C_{Mc}(t) + \delta \cdot C_{Rc}(t), \quad (1)$$

where α, β, γ, and δ denote weighting coefficients for the four cost components. The optimization problem is solved over a sliding time horizon $[t_1, t_W]$ to capture both short-term dynamics and long-term performance.

a) Co-residency Risk $C_{Cr}(t)$: The co-residency risk measures the probability of adversarial VMs co-located with victim VMs on the same physical host. It is defined as:

$$C_{Cr}(t) = \frac{\sum_{i \in M} \left(C^i_{Cr}(t)\right)^2}{scale}, \quad (2)$$

with

$$C^i_{Cr}(t) = \sum_{a<b} T^i_{ab}(t), \quad (3)$$

where $C^i_{Cr}(t)$ denotes the cumulative co-residency risk of server i at time t, and $T^i_{ab}(t)$ represents the accumulated co-residency duration between VMs a and b. Its evolution is defined as:

$$T^i_{ab}(t) = \begin{cases} T^i_{ab}(t-1) + \Delta t, & \text{if } x^i_a(t) \cdot x^i_b(t) = 1 \ \wedge \ m_{ab}(t) = 1 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where $x^i_k(t)$ indicates whether VM k is hosted on server i, $p_k(t)$ denotes whether VM k is malicious, and $m_{ab}(t) = |p_a(t) - p_b(t)|$ represents the security state difference between VM a and VM b. The normalization factor $scale = \left(\frac{|N|}{|N_0|}\right)^2$ ensures comparability across different VM densities.

b) Power Consumption $C_{Pc}(t)$: The power consumption cost is expressed as:

$$C_{Pc}(t) = |M| \cdot single_server_load, \quad (5)$$

where $|M|$ denotes the number of active servers, and $single_server_load$ is the baseline energy consumption per active server.

c) Migration Cost $C_{Mc}(t)$: The migration cost captures the overhead associated with VM relocation:

$$C_{Mc}(t) = \sum_{k \in N} C^{mig}_k(t) \cdot mig_k(t), \quad (6)$$

where $mig_k(t)$ is a binary variable indicating whether VM k is migrated at time t. The migration cost for VM k is defined as:

$$C^{mig}_k(t) = \nu + \mu \cdot mem_k(t), \quad (7)$$

with ν and μ denoting fixed and memory-dependent coefficients, respectively, and $mem_k(t)$ representing the memory utilization of VM k.

d) Resource Imbalance $C_{Rc}(t)$: The resource imbalance cost measures load heterogeneity among servers:

$$C_{Rc}(t) = \frac{1}{|M|-1} \sum_{i \in M} \Bigg[w_{cpu} \Big(\sum_{k \in N} x^i_k(t) \cdot cpu_k(t) - CPU(t) \Big)^2 + w_{mem} \Big(\sum_{k \in N} x^i_k(t) \cdot mem_k(t) - MEM(t) \Big)^2 \Bigg]. \quad (8)$$

where $cpu_k(t)$ and $mem_k(t)$ denote the CPU and memory utilization of VM k, respectively. $CPU(t)$ and $MEM(t)$ represent the average CPU and memory loads across all servers, while w_{cpu} and w_{mem} weight their relative importance.

TABLE IV
SYMBOL TABLE FOR MODEL PREDICTIVE CONTROL (MPC) FORMULATION

Symbol	Description	Type
t	Discrete time step	Time index
Δt	Time interval	Constant
W	Length of sliding time window	Constant
$[t_1, t_W]$	Optimization time horizon	Time interval
M	Set of active servers	Set
N	Set of virtual machines	Set
$\|M\|$	Number of active servers	Scalar
$\|N\|$	Number of virtual machines	Scalar
$C(t)$	Total cost at time t	Objective
$\alpha, \beta, \gamma, \delta$	Weight coefficients for cost components	Parameters
$C_{Cr}(t)$	Co-residency risk cost	Cost component
$C_{Pc}(t)$	Power consumption cost	Cost component
$C_{Mc}(t)$	Migration cost	Cost component
$C_{Rc}(t)$	Resource imbalance cost	Cost component
$C^i_{Cr}(t)$	Risk level of server i at time t	Intermediate
$T^i_{ab}(t)$	Risk co-residency time of VMs a and b on server i at t	Matrix
$m_{ab}(t)$	Security state difference: $m_{ab}(t) = \|p_a(t) - p_b(t)\|$	Binary
$x^i_k(t)$	Binary: VM k on server i at t	Binary var
$p_k(t)$	Binary: VM k is attacker at t	Binary state
$mig_k(t)$	Binary: VM k is migrated at t	Binary var
ss_load	Base power per server (single_server_load)	Constant
$C^{mig}_k(t)$	Migration cost of VM k at t	Cost
ν	Fixed migration cost coefficient	Parameter
μ	Memory-dependent migration cost coefficient	Parameter
$mem_k(t)$	Memory consumption of VM k at t	Resource
$cpu_k(t)$	CPU consumption of VM k at t	Resource
w_{cpu}	Weight for CPU imbalance	Parameter
w_{mem}	Weight for memory imbalance	Parameter
$CPU(t)$	Avg CPU: $\frac{1}{\|M\|}\sum_i \sum_k x^i_k cpu_k$	Resource
$MEM(t)$	Avg memory: $\frac{1}{\|M\|}\sum_i \sum_k x^i_k mem_k$	Resource
max_serv	Max allowed servers (max_servers)	Constraint
max_res	Max resource competition (max_resource_competition)	Constraint

The optimization is performed over a sliding time horizon $[t_1, t_W]$, such that the cumulative cost is minimized:

$$\min_{\{x^i_k(\tau), mig_k(\tau)\}^{t_W}_{\tau=t_1}} \sum_{\tau=t_1}^{t_W} C(\tau), \tag{9}$$

subject to the system constraints defined below.

2) Model Constraints: To ensure that the virtual machine migration decisions maintain both security and resource efficiency in the cloud environment, the following constraints are applied in the MPC model:

a) Server Capacity Constraint:

$$|M| \leq max\_servers \tag{10}$$

b) Resource Constraint:

$$\sum_{k \in N} x^i_k(t) \cdot cpu_k(t) < 1, \quad \forall i \in\in M \tag{11}$$

$$\sum_{k \in N} x^i_k(t) \cdot mem_k(t) < 1, \quad \forall i \in M \tag{12}$$

c) VM Placement Constraint:

$$\sum_{i \in M} x^i_k(t) = 1, \quad \forall k \in N \tag{13}$$

d) Binary Variables Constraint:

$$x^i_k(t) \in \{0, 1\}, \quad \forall k \in N, \forall i \in M \tag{14}$$

$$p_k(t) \in \{0, 1\}, \quad \forall k \in N \tag{15}$$

These constraints guarantee that each VM is allocated to exactly one active server, the number of servers does not exceed the maximum limit, and resource imbalance remains within acceptable bounds, while adversarial states are explicitly considered.

V. FRAMEWORK DESIGN

To achieve adaptive and efficient mitigation of cache-based side-channel attacks in cloud environments, we design an integrated framework consisting of three modules: (i) a Data Collection Module that continuously monitors low-level system events and workload statistics, (ii) a MoE-based Deep Learning Module that predicts attack probabilities, and (iii) an MPC Decision Module that performs dynamic optimization of VM migration strategies. The interactions among these modules enable closed-loop detection, prediction, and mitigation.

A. Data Collection Module

The Data Collection Module is responsible for monitoring system-level signals that capture both micro-architectural behaviors and workload dynamics. Specifically, we leverage the Linux `perf` tool and Python `utils` libraries to collect hardware performance counter (HPC) events from each virtual machine at an interval of 0.1 seconds. At the same time, CPU utilization and memory usage of all VMs are also recorded at the same sampling rate.

The framework adopts a fixed-length sliding window of 5 seconds as the basic unit of observation. Within each window, 50 samples of CPU utilization, memory utilization and HPC event counts are collected per VM. Upon completion of a window, the collected features are dispatched to different downstream modules: HPC events are fed into the MoE-based Deep Learning Module for attack probability estimation, and workload statistics are forwarded to the MPC Decision Module for resource prediction and optimization.

B. MoE-based Deep Learning Module

The MoE-based Deep Learning Module processes the HPC event sequences collected over the 5-second window. For each VM, the module aggregates the 50 HPC samples to generate a single probability score in the range $[0, 1]$, indicating the likelihood that the VM is executing a side-channel attack during the observation window. This aggregation provides a compact representation of each VM's adversarial behavior, which is then supplied to the MPC Decision Module.

The MoE architecture enhances model generalization across heterogeneous workloads and attack patterns. Each expert network specializes in capturing different micro-architectural signatures, and a gating network dynamically weights their contributions to produce a robust and accurate attack probability estimate.

C. MPC Decision Module

The MPC Decision Module is responsible for determining adaptive VM migration strategies that minimize future system costs while maintaining security and resource efficiency. It consists of three key steps: (i) future system state prediction, (ii) MPC-based objective formulation, and (iii) heuristic-based optimization.

1) Future System State Prediction: To evaluate potential migration strategies, the module predicts the evolution of both workload and attack conditions over a sliding time horizon. The CPU and memory utilization of each VM are forecasted using time-series models: ARIMA models capture linear temporal correlations in CPU usage, while LSTM networks capture nonlinear dependencies in memory utilization. These predictions provide per-VM resource demand profiles for the upcoming horizon.

Concurrently, the probability of each VM being an attacker is derived directly from the current time window, as the probability values provided by the MoE-based deep learning module are assumed to persist over the prediction horizon. By combining predicted workloads with attack probabilities, the module obtains a comprehensive forecast of the system state, which serves as input for the optimization step.

2) MPC Formulation: The controller constructs a multi-objective cost function that balances: (1) co-residency attack risk, (2) power consumption, (3) migration overhead, and (4) resource imbalance. For each candidate migration strategy, the predicted future CPU/memory loads and attack probabilities are used to simulate system evolution, allowing the module to compute the expected cost over the horizon. The controller adopts a receding horizon policy: only the first-step migration decisions are executed, while subsequent steps are re-optimized in the next time window.

3) Optimization via PBHM: Exact optimization of the MPC objective is computationally intractable in large-scale cloud environments. To achieve tractable performance while retaining safety and efficiency, we employ the *Priority-Based Heuristic Migration* (PBHM) method. PBHM sequentially applies three priorities:

1) **Attacker Aggregation:** gather potential attacker VMs together to minimize co-residency with normal VMs. Candidate target servers are selected based on attacker density and cumulative cohabitation time with normal VMs. Migrations are accepted only if CPU and memory capacity constraints are satisfied. After completing attacker aggregation, if the migration limit is reached, the predicted cost of the current mapping is compared to the best cost; if worse, the mapping is reverted to the original.
2) **Server Consolidation:** reduce the number of active servers by redistributing normal VMs from lightly loaded servers to other servers without attackers.Placement proceeds in a round-robin fashion, starting from servers with the largest normal VM populations. As in the first step, each candidate migration is validated against server CPU and memory constraints, and the mapping is reverted if the maximum migration count is reached and the cost increases.
3) **Load Balancing:** migrate low-load normal VMs from heavily loaded servers to lightly loaded servers to smooth CPU and memory utilization across the cluster. This step is performed only if migration limits have not been reached.

PBHM enforces migration limits proportional to the VM population and systematically validates each candidate mapping against resource constraints. The final mapping that minimizes the predicted MPC cost is applied. The procedure is summarized in Algorithm 1.

VI. Experiment

VII. Conclusion and Future Work

References

[1] RazorBest, "flush-reload-gpg-rsa," https://github.com/RazorBest/flush-reload-gpg-rsa, 2014.
[2] isec tugraz, "flush_flush," https://github.com/isec-tugraz/flush_flush, 2016.
[3] kreelsama, "prime-probe-aes," https://github.com/kreelsama/prime-probe-aes, 2020.
[4] v lavrentikov, "meltdown-spectre," https://github.com/v-lavrentikov/meltdown-spectre, 2020.
[5] Eugnis, "spectre-attack," https://github.com/Eugnis/spectre-attack, 2018.

Algorithm 1 Priority-Based Heuristic Migration (PBHM)

Input: Current VM-server mapping $vm\_pm\_mapping$, attacker probabilities p_k, CPU/mem usage
Output: Updated VM-server mapping
$new\_mapping \leftarrow vm\_pm\_mapping$, $migration\_count \leftarrow 0$
Determine $max\_migrations$ based on VM population
Identify attacker and normal VMs
Compute current cluster risk
if cluster risk exceeds threshold **then**
  Attacker Aggregation:
    Attempt to migrate attackers to reduce co-residency
    Respect server CPU/mem constraints
    If $migration\_count \geq max\_migrations$, evaluate predicted cost
    Revert mapping if cost worsens
end if
if $migration\_count < max\_migrations$ **then**
  Server Consolidation:
    Migrate normal VMs to reduce active servers
    Respect server CPU/mem constraints
    If $migration\_count \geq max\_migrations$, evaluate predicted cost
    Revert mapping if cost worsens
end if
if $migration\_count < max\_migrations$ **then**
  Load Balancing:
    Migrate low-load normal VMs from high-load to low-load servers
    Respect server CPU/mem constraints
end if
return $new\_mapping$