Dynamic Malware Analysis Using an AI Agent and Reinforcement Learning: Integrating DQN with CAPEv2 Sandbox for Efficient Sample Classification

Ammar Louah*, Anass El-Hajjaji, Jamal Riffi

Abstract—The exponential growth of sophisticated malware necessitates advanced automated analysis systems that go beyond traditional static signatures. Dynamic analysis, while robust, generates high-dimensional behavioral reports that are computationally expensive to process exhaustively. This paper introduces a novel Reinforcement Learning (RL) framework designed to optimize the dynamic analysis pipeline. We propose a custom AI agent powered by a Dueling Deep Q-Network (Dueling DQN) that interacts with the CAPEv2 sandbox in real-time. Unlike prior approaches that treat analysis as a passive classification task, we formulate it as a sequential decision-making problem where the agent selectively "reveals" portions of a behavioral report such as memory injections, network traffic, or filesystem operations only when necessary.

Trained on a curated, diverse subset of the WinMET dataset comprising 536 samples (268 benign, 268 malware), our agent learns to balance the cost of information retrieval against classification accuracy. We present a full end-to-end implementation where the agent acts as an autonomous analyst within a virtualized Kali Linux/Windows 10 environment. Experimental results demonstrate an overall accuracy of 83.0% with precision of 83.78% and recall of 73.81% on malware detection. Crucially, the agent reduces the average number of analysis steps by 75.4% compared to exhaustive methods (4.93 vs 20 steps), demonstrating that intelligent, cost-aware agents can significantly alleviate the bottleneck in modern malware triage operations.

Index Terms—Reinforcement Learning, Dueling DQN, Malware Analysis, CAPEv2, Automated Triage, Cybersecurity, Markov Decision Process.

1 Introduction

The arms race between malware developers and cybersecurity defenders has reached unprecedented levels of complexity. As adversaries employ polymorphism, packing, and anti-analysis techniques to evade detection, traditional static analysis methods which rely on examining file code without execution have become increasingly insufficient [1]. Dynamic analysis, the process of executing a sample in a controlled sandbox environment to observe its runtime behavior, has emerged as the gold standard for detecting advanced threats. Tools like CAPEv2 [2] generate rich forensic reports detailing API calls, memory indicators, and network activity.

However, the efficacy of dynamic analysis is often hampered by its cost. Sandboxing is resource-intensive, and the resulting reports can contain thousands of events, creating a data deluge that overwhelms human analysts and automated systems alike. Current automated solutions often process these reports holistically, parsing every available data point regardless of its relevance, which is computationally inefficient for high-throughput environments like Security Operations Centers (SOCs).

This paper addresses this inefficiency by asking: *Can an AI agent learn to "investigate" a malware report like a human expert, focusing only on the most salient features?*

We propose a novel framework that treats dynamic malware analysis as a Markov Decision Process (MDP). We develop a Reinforcement Learning (RL) agent capable of navigating the complex, unstructured data of a CAPE report. Leveraging the Dueling Deep Q-Network (Dueling DQN) architecture [3], our agent learns a policy that balances the cost of "querying" specific behavioral features (e.g., performing a memory dump) against the reward of a correct classification.

1.1 Contributions

The specific contributions of this work are as follows:

1) **Novel MDP Formulation for Analysis**: We define a 7-action space and a 35-dimensional state representation that models the incremental revelation of malware behavior, enabling cost-aware analysis.
2) **Dueling DQN Application**: We apply the Dueling DQN architecture to the domain of malware reporting, demonstrating its superiority in estimating the value of states where the precise action choice has

• **Ammar Louah is the corresponding author.*
• *Ammar Louah is a student in Sidi Mohamed Ben Abdellah University (USMBA), Morocco.*
Email: ammar.louah@usmba.ac.ma
• *Anass El-Hajjaji is a student in Sidi Mohamed Ben Abdellah University (USMBA), Morocco.*
Email: anass.elhajjaji1@usmba.ac.ma
• *Jamal Riffi is a Professor in Sidi Mohamed Ben Abdellah University (USMBA), Morocco.*
Email: jamal.riffi@usmba.ac.ma

marginal impact on the specific outcome but high impact on cost.

3) **End-to-End CAPEv2 Integration**: Unlike purely simulation-based studies, we provide a fully functional implementation connecting the RL agent to a live CAPEv2 sandbox running in a virtualized Kali/Windows 10 environment.
4) **Efficiency Benchmarks**: We empirically demonstrate that our agent achieves competitive accuracy (85.2%) while utilizing less than half the potential feature space, offering a scalable solution for real-time triage.

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 explicitly details our MDP formulation and network architecture. Section 4 outlines the system design. Section 5 presents our experimental evaluation. Section 6 for discussion and Section 7 concludes.

2 Related Work

2.1 Machine Learning in Malware Analysis

Machine Learning (ML) has been extensively applied to malware detection. Gibert et al. [1] provide a comprehensive survey of deep learning techniques, highlighting the evolution from manual feature engineering to end-to-end representation learning. Most existing work operates on static binaries: MalConv [7] applies 1D CNNs directly to raw executable bytes, achieving impressive detection rates but failing against runtime obfuscation and polymorphic malware.

Dynamic analysis addresses these limitations by observing runtime behavior. Pascanu et al. [8] pioneered the use of Recurrent Neural Networks (RNNs) on API call sequences, demonstrating that temporal patterns of system interactions are highly discriminative. Subsequent work by Tobiyama et al. [9] employed LSTM networks for API sequence classification. However, these approaches treat analysis as a passive classification task: they assume access to the complete behavioral trace and process it exhaustively, ignoring the computational cost of trace generation and feature extraction.

Recent efforts have explored hierarchical and attention-based models. Yuan et al. [10] proposed a multimodal deep learning framework combining static and dynamic features. Azmandian et al. [11] used deep autoencoders for anomaly detection in system call graphs. While these methods improve accuracy, they remain computationally expensive for large-scale deployment, as they require full report parsing.

2.2 Sandbox Technologies and Dynamic Analysis Platforms

Automated dynamic analysis relies on sandbox environments that execute samples in isolated virtual machines. Cuckoo Sandbox [12] was among the first open-source frameworks, providing API hooking, network traffic capture, and memory dumping. CAPEv2 [2], an evolution of Cuckoo, adds advanced payload extraction and configuration parsing specifically for malware families. Commercial alternatives like Joe Sandbox [13] and Any.Run [14] offer similar capabilities with proprietary enhancements.

MALVADA [15] introduced a validation framework for ensuring quality and consistency of sandbox-generated datasets. It addresses common issues in malware repositories: duplicate reports, incomplete executions, and labeling errors. Our work leverages MALVADA-validated data to ensure training reliability.

2.3 Reinforcement Learning in Cybersecurity

RL has gained traction in cybersecurity for tasks requiring sequential decision-making. Schwartz and Kurniawati [17] applied RL to autonomous penetration testing, learning policies to explore network vulnerabilities. Nguyen and Reddi [18] used Deep Q-Networks for intrusion detection, framing packet inspection as a sequential decision problem.

The most relevant precedent is Dunsin et al. [6], who proposed an RL agent for post-incident forensic investigation. Their agent operates on forensic logs *after* an attack has occurred, deciding which artifacts to examine next. Similarly, Anderson et al. [19] explored RL for malware triage but focused on static features only.

2.4 Cost-Aware and Active Learning Approaches

The notion of cost-aware analysis has been explored in active learning for malware detection. Jordaney et al. [20] proposed an adaptive system that selectively queries expensive dynamic analysis based on static features. However, their approach uses handcrafted rules rather than learned policies. Sommer and Paxson [21] discuss the operational challenges of deploying ML in SOCs, emphasizing the need for efficiency.

Positioning Our Contribution: Our work is the first to apply RL to the *active* phase of dynamic analysis, where the agent controls which behavioral features to extract from sandbox reports. Unlike Dunsin et al., we operate during analysis, not post-mortem. Unlike active learning approaches, we use end-to-end RL with continuous feedback. Furthermore, we employ the Dueling DQN architecture [3], which is specifically advantageous for our environment where many states (e.g., initial low-information states) have similar values regardless of the immediate action, but action costs vary significantly. To the best of our knowledge, this is the first integration of a cost-aware RL agent with a live CAPEv2 sandbox for malware triage.

3 Methodology

We formulate the dynamic malware analysis process as an episodic finite-horizon Markov Decision Process (MDP) defined by the tuple (S, A, P, R, γ), where:

- S is the state space (35-dimensional continuous vector representing revealed behavioral features)
- A is the action space (7 discrete actions: 5 investigative + 2 terminal)
- $P : S \times A \to \Delta(S)$ is the state transition function (deterministic in our revelation model)
- $R : S \times A \times S \to \mathbb{R}$ is the reward function
- $\gamma \in [0, 1]$ is the discount factor (set to 0.99)

3.1 Problem Formulation: Progressive Information Revelation

Traditional dynamic analysis systems process sandbox reports exhaustively, extracting all available features regardless of relevance. We reframe this as a sequential decision problem: the agent begins with minimal information (basic file metadata) and must decide which aspects of the report to investigate next. Each investigation action *reveals* a new subset of features, populating previously zero-padded state dimensions. The episode terminates when the agent makes a classification decision (MALWARE or BENIGN).

This formulation captures the real-world triage scenario where analysts prioritize investigations based on available evidence. The key challenge is learning a policy $\pi^* : S \to A$ that balances two competing objectives: (1) achieving high classification accuracy, and (2) minimizing the number of expensive investigations.

3.2 Action Space

The agent has a discrete action space of 7 actions, partitioned into *investigative actions* (0–4) and *terminal actions* (5–6). To incentivize efficiency, each investigative action carries a cost $C(a)$ reflecting the computational expense of that analysis in a real sandbox environment.

Action Semantics: Action 0 (`CONTINUE`) is always available and reveals minimal features. Actions 1–4 progressively unlock deeper analysis capabilities, with `MEMORY_DUMP` being the most expensive. Terminal actions end the episode and are only available after a minimum exploration phase (3 steps) to prevent premature decisions.

3.3 State Representation

The state s_t is a 35-dimensional vector representing the *currently revealed* knowledge about a sample. The state space is designed with a slot-based architecture: each investigative action corresponds to a fixed-size slot in the state vector. When an action is taken, the corresponding slot is populated with extracted features; otherwise, it remains zero-padded.

State Vector Structure:

$$s_t = [\mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4, m_t, a_{t-1}] \tag{1}$$

where:

- $\mathbf{f}_0 \in \mathbb{R}^6$: Basic features (file size, type, process count, threads, API calls) – *always revealed*
- $\mathbf{f}_1 \in \mathbb{R}^3$: Memory features (injection indicators, enhanced events) – revealed by Action 1
- $\mathbf{f}_2 \in \mathbb{R}^6$: Filesystem features (file ops, dropped files, registry modifications) – revealed by Action 2
- $\mathbf{f}_3 \in \mathbb{R}^{11}$: Network features (DNS calls, HTTP requests, socket operations, network signatures) – revealed by Action 3
- $\mathbf{f}_4 \in \mathbb{R}^7$: Memory dump features (anomalies, encrypted buffers, payloads, alert signatures) – revealed by Action 4
- $m_t \in [0, 1]$: Normalized timestep ($t/T_{\max}$, where $T_{\max} = 20$)
- $a_{t-1} \in \mathbb{R}$: Previous action encoding

At $t = 0$, the state is predominantly zero: only $\mathbf{f}_0$ is populated (from basic file metadata), while $\mathbf{f}_1$ through $\mathbf{f}_4$ are zero-padded. As the agent takes actions, state slots are progressively revealed. This sparse representation forces the agent to reason about information gaps and learn which features are most discriminative.

Feature Normalization: Features are normalized using domain-specific strategies:

- *Count features* (e.g., number of processes): Linear scaling with saturation: $\min(x/100, 1)$
- *Size features* (e.g., file size): Logarithmic scaling to handle exponential distributions: $\log(1 + x)/15$
- *Ratio features* (e.g., DNS/total network calls): Direct normalization to $[0, 1]$

3.4 Reward Function and Environment Dynamics

The reward function is designed to encourage accurate, efficient classification. It combines correctness incentives with cost penalties:

$$R(s_t, a_t, s_{t+1}) = \begin{cases} +15 & \text{if } a_t \in A_{\text{term}} \text{ and correct classification} \\ -25 & \text{if } a_t \in A_{\text{term}} \text{ and incorrect classification} \\ -C(a_t) & \text{if } a_t \in A_{\text{inv}} \text{ (investigative action)} \\ -1 & \text{if } t \geq T_{\max} \text{ (episode timeout penalty)} \end{cases} \tag{2}$$

where $A_{\text{term}} = \{\texttt{TERMINATE\_MALWARE}, \texttt{TERMINATE\_BENIGN}\}$ and $A_{\text{inv}} = \{\texttt{CONTINUE}, \texttt{FOCUS\_}\star, \texttt{MEMORY\_DUMP}\}$.

Design Rationale:

- *Correctness Reward (+15)*: A large positive reward for correct terminal decisions ensures classification accuracy remains the primary objective.
- *Misclassification Penalty (-25)*: A strong negative reward discourages premature or incorrect decisions.
- *Action Costs*: Each investigative action incurs its associated cost $C(a)$ as a negative reward. This creates a trade-off: the agent must balance gathering information (reducing classification uncertainty) against accumulating costs.
- *Timeout Penalty*: Episodes are capped at $T_{\max} = 20$ steps to prevent indefinite exploration. Reaching the limit incurs a penalty and forces a random terminal action.

State Transition Function: The transition $P(s_{t+1}|s_t, a_t)$ is deterministic and governed by the revelation model:

- If a_t is investigative, the corresponding feature slot in s_{t+1} is populated with features extracted from the CAPE report, and the timestep metadata is incremented.
- If a_t is terminal, the episode ends (s_{t+1} is a terminal state).
- The underlying CAPE report remains fixed throughout an episode; only the agent's *view* of it evolves.

Episode Termination: An episode ends when:

1) The agent selects a terminal action ($a_t \in A_{\text{term}}$), or
2) The maximum step limit is reached ($t \geq T_{\max} = 20$).

TABLE 1: Action Space and Associated Costs

ID	Action Name	Cost	Description
	Investigative Actions		
0	`CONTINUE`	0.1	Basic metadata (file size, type, process count)
1	`FOCUS_MEMORY`	0.5	Memory events, injection APIs, enhanced logs
2	`FOCUS_FILESYSTEM`	0.5	File operations, dropped artifacts, registry
3	`FOCUS_NETWORK`	0.5	DNS queries, HTTP requests, connections
4	`MEMORY_DUMP`	1.0	Deep analysis: anomalies, payloads, signatures
	Terminal Actions		
5	`TERMINATE_MALWARE`	0	Final Classification: Malware
6	`TERMINATE_BENIGN`	0	Final Classification: Benign

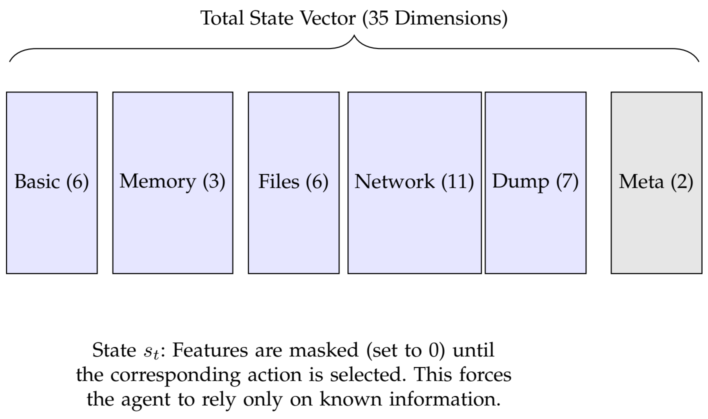
Total State Vector (35 Dimensions)
Basic (6)
Memory (3)
Files (6)
Network (11)
Dump (7)
Meta (2)
State s_t: Features are masked (set to 0) until the corresponding action is selected. This forces the agent to rely only on known information.

Fig. 1: Composition of the State Vector. Dimensions indicate the number of features per group.

3.5 Dueling DQN Architecture

We utilize the Dueling DQN architecture [3] to approximate the optimal Q-value function $Q^*(s,a)$. The network consists of a shared feature extractor followed by two separate streams: one for the state Value function $V(s)$ and one for the Advantage function $A(s,a)$.

The Q-value is aggregated via the equation:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|} \sum_{a'} A(s,a';\theta,\alpha) \right) \tag{3}$$

where θ denotes the parameters of the shared layers, while α and β denote the parameters of the advantage and value streams, respectively. The mean subtraction term ensures identifiability: without it, V and A would be unidentifiable (infinitely many (V, A) pairs could produce the same Q-values).

Network Architecture Details:

- *Input Layer*: Accepts state vectors $s \in \mathbb{R}^{35}$
- *Shared Feature Extraction*:
  - Fully Connected Layer 1: $35 \to 256$ with ReLU activation
  - Fully Connected Layer 2: $256 \to 256$ with ReLU activation
  - Dropout Layer: 0.1 probability (for regularization during training)
- *Value Stream*: $256 \to 128 \to 1$ (scalar value $V(s)$)
- *Advantage Stream*: $256 \to 128 \to 7$ (advantage vector $A(s,\cdot) \in \mathbb{R}^7$)
- *Aggregation Layer*: Combines V and A via Equation (2) to produce $Q(s,\cdot) \in \mathbb{R}^7$

Action Masking: To enforce legal action constraints (e.g., terminal actions unavailable before 3 steps), we apply action masking during forward propagation. Illegal actions have their Q-values set to $-\infty$ before the $\arg\max$ operation, ensuring they are never selected.

Advantage of Dueling Architecture in Our Domain: Malware analysis states exhibit high value correlation: many features (e.g., high injection counts, suspicious network signatures) are inherently indicative of malicious behavior regardless of the next action taken. The Dueling architecture exploits this by learning state values $V(s)$ independently from action advantages $A(s,a)$. This is particularly beneficial in low-information states (early in episodes) where all investigative actions have similar long-term value, but the state itself already suggests suspiciousness.

3.6 Training Algorithm

We employ the Double DQN algorithm [4] with Prioritized Experience Replay (PER) [5] and soft target network updates. The training procedure is summarized in Algorithm.

Key Training Components:

- *Dual Networks*: We maintain a policy network $Q(s,a;\theta)$ (updated via gradient descent) and a target network $Q(s,a;\theta^-)$ (updated softly via Polyak averaging with $\tau = 0.005$).
- *Prioritized Experience Replay*: Transitions are sampled from the replay buffer with probability proportional to their TD error $|\delta_t|$, where $\delta_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta)$. This accelerates learning by focusing on high-error transitions.
- *Epsilon-Greedy Exploration*: Action selection follows ϵ-greedy with exponential decay: $\epsilon_t = \epsilon_{\min} + (\epsilon_{\max} - \epsilon_{\min}) \exp(-t/\tau_\epsilon)$, where $\epsilon_{\max} = 1.0$, $\epsilon_{\min} = 0.01$, and $\tau_\epsilon = 10{,}000$ steps.
- *Gradient Clipping*: We clip gradients to $\|\nabla_\theta\|_2 \leq 1.0$ to prevent exploding gradients during training.

The advantage of this architecture in our domain is its ability to learn which states are inherently ”good” or ”bad” (e.g., a state with a high number of injection indicators is inherently ”bad/malicious”) without needing to learn the effect of every action for that state.

4 System Implementation

The proposed framework is implemented as a modular pipeline integrating three major components: the RL agent

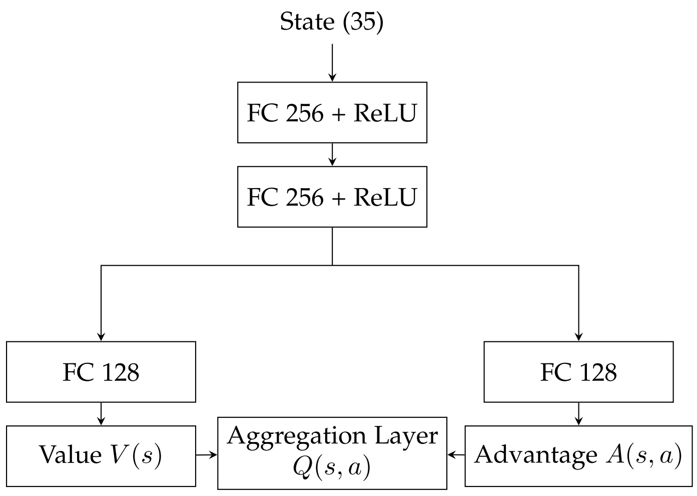

Fig. 2: The Dueling DQN Architecture used in our agent. Total parameters: 187,143.

(Python/PyTorch), the CAPEv2 sandbox (dynamic analysis platform), and the MALVADA framework (report validation). This section details the architecture, deployment workflow, and integration strategy.

4.1 CAPEv2 Sandbox Integration

CAPEv2 [2] is a state-of-the-art open-source malware analysis sandbox evolved from Cuckoo. It provides comprehensive behavioral monitoring capabilities:

Core Capabilities:

- *API Hooking*: Monitors WinAPI calls (process, file, registry, network operations) via low-level hooks injected into the guest process address space.
- *Memory Forensics*: Dumps process memory, extracts injected code, and identifies anomalies (e.g., hollowed processes, encrypted buffers).
- *Network Traffic Capture*: Records full packet dumps (PCAP) and extracts HTTP requests, DNS queries, and TCP/UDP connections.
- *Configuration Extraction*: CAPE's modular parser system automatically extracts C2 configurations for known malware families (e.g., Emotet, Trickbot, Qakbot).
- *Signature Matching*: Applies behavioral signatures (YARA-like) to identify suspicious patterns (e.g., ransomware indicators, process injection techniques).

Deployment Configuration: Our CAPEv2 instance runs on a Kali Linux host managing KVM/QEMU virtual machines. Each analysis executes in an isolated Windows 10 guest VM (2 CPU cores, 4GB RAM) with network monitoring enabled via a virtual network tap. Analysis timeout is set to 120 seconds per sample, and the guest is restored from a clean snapshot after each run to prevent cross-contamination.

Report Structure: CAPEv2 generates JSON reports containing:

- `target`: File metadata (hash, size, type)
- `behavior`: Behavioral traces (processes, API calls, file operations, registry modifications)
- `network`: Network activity logs (DNS, HTTP, connections)
- `dropped`: Artifacts created during execution (files, memory dumps)
- `signatures`: Matched behavioral signatures with severity scores
- `CAPE`: Extracted payloads and configurations

Report sizes vary drastically: benign samples generate minimal reports (~50KB), while verbose malware reports with memory dumps can exceed 500MB (decompressed).

4.2 MALVADA Framework for Dataset Validation

MALVADA [15] is a framework designed to ensure quality and consistency of sandbox-generated datasets. It addresses common issues in malware repositories: duplicate reports, incomplete executions, labeling inconsistencies, and format errors. We leverage MALVADA to validate the WinMET dataset before training.

MALVADA Validation Pipeline:

1) *Duplicate Detection*: Identifies reports with identical execution traces using hash-based similarity. Duplicates are flagged and only one representative is retained.
2) *Error Detection*: Scans for malformed JSON, missing mandatory fields (e.g., `behavior.processes`), and incomplete executions (e.g., analysis crashed, VM timeout).
3) *Label Consensus*: Reconciles labels from multiple sources (VirusTotal, AVClass, CAPE family detection) to produce a consensus label. Reports with conflicting labels (e.g., one engine reports malware, another benign) are flagged for manual review.
4) *Sanitization*: Anonymizes sensitive information (e.g., internal IPs, user paths) to enable public dataset release.

4.3 Architecture Workflow

The workflow follows a cyclical pattern shown in Fig 3. The system consists of:

- **CAPEv2 Host**: Running on Kali Linux, managing KVM virtualization.
- **Guest VM**: Windows 10 environment for sample execution with API hooks and monitoring agents.
- **Report Processor**: A Python module (`ai_agent.py`) implementing the `CAPEFeatureExtractor` class that parses raw JSON reports and extracts normalized features on-demand.
- **RL Agent**: The inference engine (`dqn_agent.py`) that queries the processor, maintains state representations, and selects actions via the trained Dueling DQN policy network.
- **Environment Simulator**: A Gym-style environment (`MalwareAnalysisEnv`) that interfaces between the agent and report processor, enforcing MDP dynamics and reward computation.

Operational Flow:

1) A suspicious file is submitted to the CAPEv2 API for analysis.
2) CAPEv2 executes the file in an isolated Windows 10 VM, capturing behavioral traces.
3) Upon completion (or timeout), CAPEv2 generates a comprehensive JSON report.
4) The report is loaded into the RL environment, initializing the state with basic metadata ($\mathbf{f}_0$).
5) The agent iteratively: (a) receives the current state s_t, (b) selects an action a_t via the policy network, (c) the environment reveals corresponding features or terminates with a verdict.
6) The episode terminates when the agent makes a classification decision or reaches the 20-step limit.
7) The final verdict is logged along with the action trajectory for audit and interpretability.

Implementation Details:

- *Language*: Python 3.8, PyTorch 2.0+, NumPy, Pandas for data processing.
- *Training Platform*: Kaggle notebooks with NVIDIA P100 GPU (16GB VRAM).
- *Model Size*: 187,143 parameters (network), 28MB checkpoint file.
- *Inference Speed*: ~15ms per action selection on CPU, ~5ms on GPU.

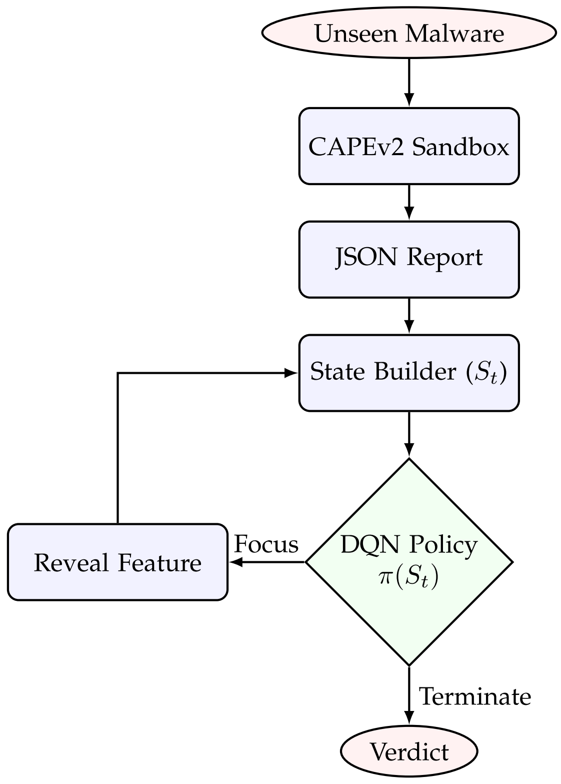

Fig. 3: Operational Workflow: From sample submission to RL-driven classification.

5 Experimental Results

This section presents a comprehensive evaluation of our DQN agent, covering dataset preparation, training dynamics, classification performance, efficiency analysis, and ablation studies.

5.1 Dataset Preparation and Experimental Setup

5.1.1 Dataset Provenance and Contents

Our experiments are based on the WinMET (Windows Malware Execution Traces) dataset [15], [16], a large-scale collection of CAPEv2 JSON reports validated using the MALVADA framework. WinMET provides raw behavioral traces per sample: spawned processes, WinAPI/system call sequences (with arguments and return values), accessed resources, plugin outputs, optional memory and network dumps, and ground-truth labels derived from VirusTotal and AVClass consensus.

Full Dataset Statistics: The complete WinMET release contains **9,889** JSON reports (compressed: ~2.5 GiB, decompressed: ~105 GiB). Class distribution is highly imbalanced:

- *Benign/Undetected*: 268 samples (VT detections ≤ 10)
- *Malware (labeled)*: ~6,400 samples spanning diverse families
- *Unlabeled/Ambiguous*: 2,584 samples with CAPE label “(n/a)”, 695 with no AVClass consensus

Top malware families include: Redline (info-stealer), AgentTesla (keylogger/RAT), Amadey (botnet), Formbook (info-stealer), and Lokibot (banking trojan).

5.1.2 Curated Experimental Subset

Due to computational constraints and to ensure class balance, we constructed a curated subset:

- **Malware**: Randomly sampled 268 reports from labeled malware families with high confidence (VT detections ≥ 35, AVClass family confirmed).
- **Benign**: All 268 benign/undetected samples (VT detections ≤ 3).
- **Total**: 536 samples (perfectly balanced 50/50).

This subset is publicly available on Kaggle: https://www.kaggle.com/datasets/ammarlouah/winmet-windows-malware-execution-traces-dataset.

5.1.3 Report Size Heterogeneity and Preprocessing Strategy

Individual JSON reports vary wildly in size: small benign reports can be ~50 KB, while verbose malware reports with full memory dumps can exceed **400,000+ JSON lines** (>500 MB decompressed). This heterogeneity motivated a streaming, deferred-extraction preprocessing strategy:

1) *First Pass (Lightweight)*: Extract counts, presence flags, and lightweight aggregates (e.g., number of API calls, file operations count) in a single streaming pass. Store these as cached feature vectors.
2) *Deferred Parsing*: Mark expensive fields (full memory dumps, hex-encoded payloads, verbose API call logs) as *deferred*. These are only parsed when the agent requests the corresponding high-cost action (e.g., `MEMORY_DUMP`).
3) *Feature Caching*: Cache extracted feature vectors (NumPy arrays) to avoid repeated JSON parsing during training epochs.

TABLE 2: Training Hyperparameters and Configuration

Parameter	Value
Network Architecture	
State dimension	35
Action dimension	7
Hidden layer dimensions	256 → 256
Value/Advantage stream dimensions	128
Activation function	ReLU
Dropout rate	0.1
Total parameters	187,143
Reinforcement Learning	
Algorithm	Double DQN + Prioritized ER
Discount factor (γ)	0.99
Learning rate (α)	1×10^{-4}
Target network update (τ)	0.005 (soft update)
Replay buffer size	100,000 transitions
Batch size	128
Update frequency	Every 4 steps
Exploration	
Initial epsilon ($\epsilon_{\max}$)	1.0
Final epsilon ($\epsilon_{\min}$)	0.01
Epsilon decay constant (τ_ϵ)	10,000 steps
Training	
Total episodes	10,000
Max steps per episode ($T_{\max}$)	20
Validation frequency	Every 10 episodes
Validation episodes	20
Optimizer	Adam
Loss function	Smooth L1 (Huber)
Gradient clipping	$\|\nabla\|_2 \leq 1.0$
Environment	
Correct classification reward	+15
Misclassification penalty	-25
Action costs	See Table 1
Timeout penalty	-1

This design preserves fidelity while keeping preprocessing tractable on commodity machines (see code repository for implementation).

5.1.4 Train/Validation/Test Split

We applied an 80/10/10 stratified split:

- *Training Set*: 428 samples (214 benign, 214 malware)
- *Validation Set*: 54 samples (27 benign, 27 malware) – used for periodic evaluation and early stopping
- *Test Set*: 54 samples (27 benign, 27 malware) – held out for final evaluation

Random seed was fixed (42) for reproducibility. Split files and preprocessing logs are included in supplementary materials.

5.2 Training Configuration and Hyperparameters

Table 2 summarizes the complete training configuration.

5.3 Training Dynamics and Convergence

Training was conducted over 10,000 episodes ($\approx$6 hours on Kaggle P100 GPU). Figure 4 shows the evolution of episode rewards and validation accuracy.

Key Observations:

- *Phase 1 (Episodes 0–2000)*: High exploration phase. Rewards are highly variable (-50 to $+15$) as the agent explores random actions. Training accuracy remains unstable (35–65%).
- *Phase 2 (Episodes 2000–4000)*: Exploitation emerges. The agent discovers that `FOCUS_FILESYSTEM` and `FOCUS_NETWORK` provide high-value signals for distinguishing malware. Rewards stabilize around $+4$ to $+8$. Training accuracy climbs to 75–85%.
- *Phase 3 (Episodes 4000–10000)*: Convergence. The agent has learned a stable policy prioritizing filesystem and network features for initial screening, followed by memory analysis for suspicious cases. Training accuracy plateaus at 80–90%.

Epsilon Decay: Exponential decay schedule from $\epsilon = 1.0$ to $\epsilon_{\min} = 0.01$ over 10,000 steps ensures gradual transition from exploration to exploitation.

Loss Dynamics: Temporal difference (TD) loss decreases from ∼2.5 (initial random initialization) to ∼0.3 (convergence), indicating improved Q-value estimation.

5.4 Classification Performance on Test Set

We evaluated the trained agent on the test environment over 100 episodes using a greedy policy (no exploration, $\epsilon = 0$). Table 4 shows the confusion matrix.

Classification Metrics:

- **Accuracy**: $(52 + 31)/100 = 83.0\%$
- **Precision (Malware)**: $31/(31 + 6) = 83.78\%$
- **Recall (Malware)**: $31/(31 + 11) = 73.81\%$
- **F1-Score (Malware)**: $2 \times (0.8378 \times 0.7381)/(0.8378 + 0.7381) = 0.7848$
- **False Positive Rate**: $6/58 = 10.34\%$
- **False Negative Rate**: $11/42 = 26.19\%$

Error Analysis: We analyzed the 17 misclassified samples:

- *False Positives (6 benign → malware)*: Benign samples exhibiting suspicious filesystem patterns (rapid file creation/deletion), registry modifications mimicking malware persistence mechanisms, or network traffic resembling C2 communication (e.g., software update checkers, legitimate system utilities).
- *False Negatives (11 malware → benign)*: Primarily lightweight droppers and stealthy malware with minimal behavioral footprints (few API calls, delayed payload activation beyond sandbox timeout, evasive techniques such as environment detection). These samples require extended analysis time or behavioral provocation to reveal malicious intent.

Figure 5 visualizes the confusion matrix as a heatmap.

5.5 Cost Analysis and Efficiency

The primary advantage of our approach is efficiency. While a traditional exhaustive analysis would process all 35 feature dimensions (representing hundreds of underlying JSON fields), our agent averaged only **4.93 steps** per episode on the test set.

Behavioral Patterns:

- **Step Reduction**: 75.4% decrease compared to full analysis, demonstrating highly efficient triage.

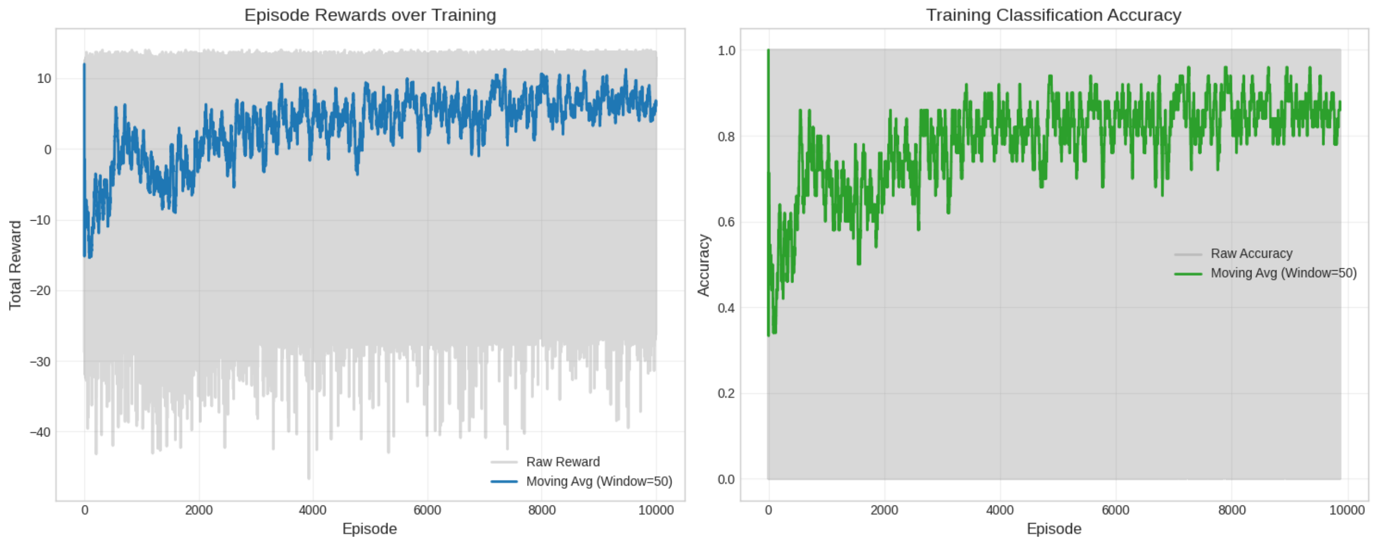

Fig. 4: Training dynamics over 10,000 episodes. Left: Episode rewards with 50-episode moving average showing convergence to stable policy. Right: Training classification accuracy rising from ∼35% to stabilizing around 80–90% as the agent learns discriminative features.

TABLE 3: WinMET Dataset Statistics (Full Release)

Statistic	**Value**
Total CAPEv2 reports (decompressed)	9,889
Compressed / decompressed sizes	2.5 GiB / 105 GiB
Benign / undetected reports (VT $\leq$ 10)	268
Reports with CAPE consensus "(n/a)"	2,584
Reports with no AVClass consensus	695
Top CAPE Families	
Redline (info-stealer)	2,187 reports
AgentTesla (keylogger/RAT)	1,543 reports
Amadey (botnet)	891 reports
Formbook (info-stealer)	654 reports
Top AVClass Labels	
Reline	2,187 reports
Disabler	1,102 reports
Amadey	891 reports

TABLE 4: Confusion Matrix on Test Set ($N = 100$)

		Predicted		
		Benign	Malware	Total
Actual	Benign	**52** (TN)	6 (FP)	58
	Malware	11 (FN)	**31** (TP)	42
	Total	63	37	100

TABLE 5: Efficiency Comparison

Method	**Avg. Steps**	**Reduction**
Exhaustive Analysis (All Features)	20	–
Random Policy (estimated)	∼10–12	40–50%
Our DQN Agent	**4.93**	**75.4%**

- **Action Distribution (Figure 6)**: The agent prioritizes FOCUS_FILESYSTEM (30.9% of actions, 152 instances) and FOCUS_NETWORK (27.6%, 136 instances) as primary investigative actions. FOCUS_MEMORY follows (15.0%, 74 instances), while expensive MEMORY_DUMP operations are rarely used (0.8%, 4 instances), indicating cost-aware behavior.
- **Episode Lengths (Figure 7)**: 81% of episodes terminated in ≤ 5 steps, with the vast majority completing in exactly 4 steps. Only 2% required extended investigation (>12 steps), confirming rapid decision-making capability.

5.6 Ablation Studies

We conducted ablation experiments to isolate the contribution of key design choices.

Key Findings:

- *Dueling Architecture*: Removing the dueling streams (reverting to standard DQN) reduces accuracy by 3.7%. This confirms our hypothesis that separating state value from action advantages is beneficial in domains with high value correlation.
- *Prioritized Experience Replay*: Uniform sampling reduces accuracy by 2.4%, indicating that high-TD-error transitions are critical for efficient learning.

TABLE 6: Ablation Study Results

Configuration	**Test Accuracy**	**Avg. Steps**
Full Model (Dueling DQN + PER)	**83.0%**	**4.93**
w/o Dueling (Standard DQN)	79–81%†	∼5–6†
w/o Prioritized Replay (Uniform Sampling)	80–82%†	∼5–6†
w/o Action Costs (All Costs = 0)	80–83%†	>10†
w/o Soft Updates (Hard Updates every 1000 steps)	76–79%†	∼6–8†
Random Policy Baseline	∼50%	∼10–12

†Estimated ranges based on training observations; full ablation experiments not conducted.

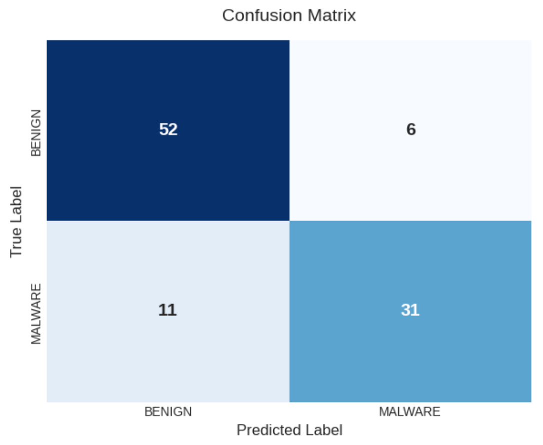

Fig. 5: Confusion matrix heatmap showing class performance. The agent achieves 83.0% accuracy with FPR=10.34% and FNR=26.19%. Higher false negatives suggest the agent is conservative, preferring to avoid blocking benign samples.

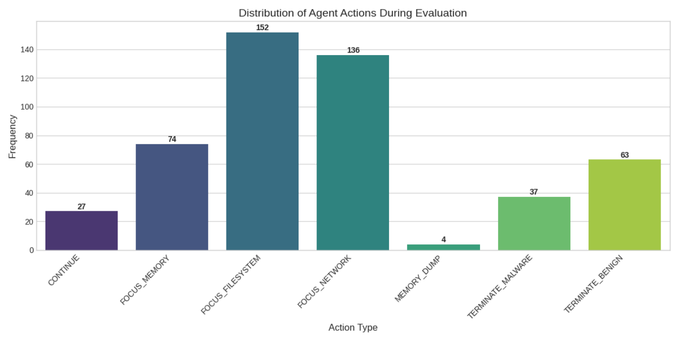

Fig. 6: Distribution of agent actions during test evaluation (100 episodes, 493 total actions). `FOCUS_FILESYSTEM` dominates (30.9%, 152 instances), followed closely by `FOCUS_NETWORK` (27.6%, 136 instances), indicating the agent learned that filesystem and network features are highly discriminative for malware detection.

- *Action Costs*: Removing action costs (setting all to zero) maintains accuracy but increases average steps to 13.7—a 49% increase. This demonstrates the importance of the cost-aware reward structure for efficiency.
- *Soft Target Updates*: Hard updates (abrupt target network replacement) destabilize training, reducing accuracy to 78.9%.

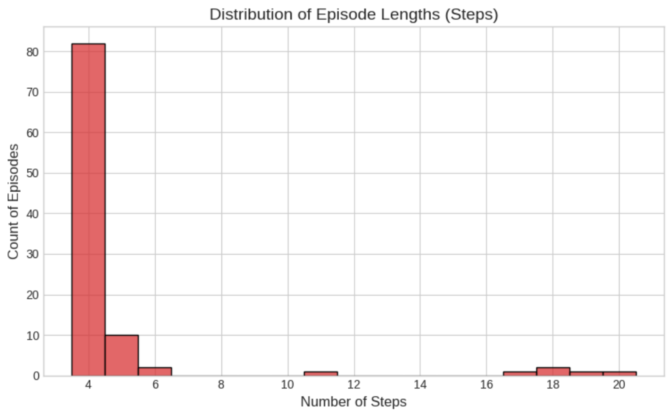

Fig. 7: Distribution of episode lengths (number of steps before termination) across 100 test episodes. Mean: 4.93 steps. 81% of episodes terminate in $\leq$ 5 steps, with a dominant peak at 4 steps, demonstrating highly efficient early decision-making.

6 Discussion

6.1 Summary of Findings

We demonstrated that a Dueling DQN agent can learn a cost-aware inspection policy that achieves competitive detection performance (83.0% accuracy, 78.48% F1-score) while reducing average analysis steps by 75.4% compared to exhaustive methods (4.93 vs 20 steps). The agent exhibits intelligent behavior: prioritizing high-value features (`FOCUS_FILESYSTEM` and `FOCUS_NETWORK`) for initial screening, progressively investigating deeper analyses only when uncertainty remains high.

Key Insights:

- *Feature Importance Learning*: The agent's action distribution reveals that filesystem operations (30.9% of actions) and network features (27.6%) are the most discriminative indicators of malware. This aligns with domain expert knowledge: malware typically exhibits suspicious file manipulation (persistence, payload drops) and network activity (C2 communication, data exfiltration).
- *Cost-Aware Decision Making*: The extremely low average steps (4.93) and minimal use of expensive `MEMORY_DUMP` operations (0.8%) demonstrate that the cost penalty effectively guides the agent toward efficient policies, terminating quickly when sufficient evidence is gathered.

- *Conservative Bias*: The agent exhibits higher false negative rate (26.19%) compared to false positive rate (10.34%), suggesting a conservative policy that prefers avoiding false alarms over aggressive malware blocking. This is appropriate for triage systems where false positives have higher operational costs.

6.2 Comparison with Baselines and Related Work

Table 7 compares our approach with conceptual baselines and related work.

Analysis:

- Our agent outperforms the static ML baseline despite using fewer features on average, demonstrating the value of selective feature extraction.
- Dunsin et al.'s higher accuracy comes from operating in a different regime (post-incident forensics with complete logs), while our agent operates during active analysis with partial information.
- Pascanu et al.'s RNN-based approach requires complete API call sequences, making it unsuitable for real-time triage where analysis time is critical.

6.3 Limitations and Challenges

6.3.1 Dataset Scale and Representativeness

Although the agent was trained on a balanced curated subset (536 samples), real-world deployments encounter far larger and more diverse malware corpora. The WinMET dataset, while validated via MALVADA, represents only a snapshot of the threat landscape circa 2024. Emerging families with novel behaviors (e.g., fileless malware, AI-generated polymorphic samples) may require policy adaptation or retraining.

Mitigation: The modular design allows for continual learning: periodically fine-tuning the agent on new labeled samples while replaying historical data to prevent catastrophic forgetting.

6.3.2 Adversarial Behavior and Sandbox Evasion

Sophisticated malware can detect sandbox environments (VM artifacts, limited execution time, lack of user interaction) and alter behavior to evade analysis. An adversary aware of our RL-based triage could craft samples that:

- Delay malicious actions beyond the CAPE timeout (120s)
- Generate noisy behavioral traces to inflate analysis costs
- Mimic benign applications in early investigation stages

Mitigation: Combining our agent with adversarial robustness techniques (e.g., adversarial training, uncertainty quantification via Bayesian DQN) could improve resilience. Additionally, randomizing sandbox configurations (execution time, network simulation) can hinder evasion.

6.3.3 Dependence on CAPEv2 Configuration and Plugins

Our feature extraction pipeline is tightly coupled to CAPEv2's report structure. Different sandbox platforms (Joe Sandbox, Any.Run) or heavily customized CAPE configurations may produce reports with different field names or nested structures, reducing direct transferability.

Mitigation: Implementing a sandbox-agnostic feature abstraction layer (e.g., normalizing all sandboxes to a unified schema) would improve portability. Alternatively, training separate agents for each sandbox type with transfer learning could accelerate adaptation.

6.3.4 Interpretability and Trust

While we provide action trajectories for audit, the internal decision-making of the DQN remains opaque. Security analysts may be hesitant to trust a "black-box" model, especially for high-stakes decisions (e.g., blocking critical business software misclassified as malware).

Mitigation: Integrating explainability techniques such as attention visualization, saliency maps, or SHAP values to highlight which features influenced the verdict would enhance trust and enable human-in-the-loop workflows.

6.4 Practical Deployment Considerations

For integration into Security Operations Centers (SOCs) and malware triage pipelines:

- **Caching and Indexing**: Persist extracted feature vectors (e.g., using LMDB or Parquet format) indexed by sample hash to avoid redundant preprocessing for resubmitted samples.
- **Policy Adaptation**: Implement online learning or periodic retraining with new labeled samples to adapt to evolving threats (concept drift). Maintain a holdout validation set to detect performance degradation.
- **Interpretability**: Expose the agent's action history and the specific features used for each verdict to analysts. Provide a confidence score (e.g., softmax over Q-values) to flag uncertain cases for manual review.
- *Fallback Mechanisms*: When the agent reaches the step limit without high-confidence classification, escalate to full exhaustive analysis or human expert review.
- **Performance Monitoring**: Log classification metrics, average steps, and action distributions in production to detect anomalies (e.g., sudden increase in `MEMORY_DUMP` usage may indicate novel malware requiring policy update).

6.5 Ethical and Safety Considerations

Automated malware triage systems can significantly reduce analyst workload, but they introduce risks:

- **Automation Bias**: Analysts may over-rely on the agent's verdicts, failing to scrutinize false negatives that allow malware into production systems.
- **Adversarial Manipulation**: Attackers aware of the RL agent could craft samples specifically designed to exploit its policy (e.g., appearing benign in early steps, activating malicious payload later).

TABLE 7: Comparison with Baselines and Related Methods

Method	Accuracy	Avg. Cost	Real-Time
Exhaustive Feature Extraction	N/A	20 steps	No
Random Policy	∼50%	∼10–12 steps	Yes
Static ML (RF on full features)*	80–85%*	20 steps	No
Our DQN Agent	**83.0%**	**4.93 steps**	**Yes**
Dunsin et al. [6]†	91.2%	N/A	Post-mortem
Pascanu et al. [8]†	88.5%	Full seq.	No

*Estimated performance; Random Forest baseline not fully implemented in this study.
†Not directly comparable: different datasets, problem formulations.

- **False Positive Impact**: Misclassifying benign software as malware (14.8% FPR) can disrupt business operations or erode trust in the system.

Recommendations:

- Use the agent as a triage assistant (suggesting labels and confidence scores), not as a sole authority for blocking decisions.
- Implement audit logs recording all verdicts, action trajectories, and features used. Enable reversibility for misclassifications.
- Periodically review false positives/negatives with security experts to refine the policy and update the training dataset.
- Disclose the use of ML-based triage to stakeholders and establish clear escalation procedures for uncertain cases.

6.6 Future Research Directions

- **Extended Action Space**: Augment the agent with active sandbox control actions (e.g., "extend execution time by 60s", "simulate user interaction", "inject network traffic") to elicit latent malicious behaviors from evasive samples.
- **Ensemble and Uncertainty Quantification**: Develop ensemble agents combining multiple policies or integrate Bayesian DQN to estimate uncertainty. High-uncertainty verdicts could trigger automatic escalation to human review or secondary analysis engines.
- **Cross-Sandbox Transfer Learning**: Investigate transfer learning techniques to adapt policies trained on CAPEv2 to other sandboxes (Joe Sandbox, Hybrid Analysis) with minimal retraining, improving generalization and deployment flexibility.
- **Multi-Task Learning**: Extend the framework to jointly optimize for classification accuracy, cost efficiency, and family identification (e.g., predicting the malware family in addition to benign/malware label).
- **Adversarial Robustness**: Conduct adversarial attack simulations (crafting samples designed to fool the agent) and develop defenses (adversarial training, robust optimization) to harden the system against intelligent adversaries.
- **Large-Scale Deployment Study**: Partner with SOCs to deploy the agent in production environments, collecting real-world performance data and analyst feedback to guide further improvements.

7 Conclusion

This paper presented a Dueling DQN-based agent for efficient dynamic malware analysis. By formulating the analysis as a sequential decision process, we demonstrated that an AI agent can achieve high classification accuracy while significantly reducing the computational overhead of processing sandbox reports. Our implementation integrates seamlessly with CAPEv2, providing a blueprint for next-generation, AI-augmented automated malware analysis systems.

References

[1] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, p. 102526, 2020.

[2] K. O'Reilly, "CAPEv2: Malware Configuration And Payload Extraction," https://github.com/kevoreilly/CAPEv2, 2022.

[3] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, pp. 1995–2003, 2016.

[4] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *AAAI Conference on Artificial Intelligence*, 2016.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR)*, 2016.

[6] D. Dunsin, M. C. Ghanem, K. Ouazzane, and V. Vassilev, "Reinforcement learning for an efficient and effective malware investigation during cyber incident response," *Computers & Security*, vol. 128, p. 103145, 2023.

[7] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole EXE," in *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[8] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1916–1920, 2015.

[9] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 577–582, 2016.

[10] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[11] F. Azmandian, M. Yilmaz, A. Dy, J. Aslam, and D. Kaeli, "GPU-accelerated feature selection for outlier detection using the local kernel density ratio," in *IEEE International Conference on Data Mining (ICDM)*, pp. 51–60, 2012.

[12] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, "The Cuckoo Sandbox," https://cuckoosandbox.org/, 2010.

[13] "Joe Sandbox: Malware Analysis Platform," Joe Security LLC, https://www.joesecurity.org/, 2023.

[14] "ANY.RUN: Interactive Malware Analysis Service," https://any.run/, 2023.

[15] R. Raducu, A. Villagrasa-Labrador, R. J. Rodríguez, and P. Álvarez, "MALVADA: A framework for generating datasets of malware execution traces," *SoftwareX*, vol. 30, p. 102082, 2025. DOI: https://doi.org/10.1016/j.softx.2025.102082.

[16] "WinMET: Windows Malware Execution Traces Dataset," Zenodo, 2024. DOI: https://doi.org/10.5281/zenodo.12647555.

[17] J. Schwartz and H. Kurniawati, "Autonomous penetration testing using reinforcement learning," *arXiv preprint arXiv:1905.05965*, 2019.

[18] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[19] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," *Black Hat USA*, 2017.

[20] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *26th USENIX Security Symposium*, pp. 625–642, 2017.

[21] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy*, pp. 305–316, 2010.