2026 Volume: 02 Issue: 11 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-11

MODELSHIELD AI: REAL TIME ANOMALY DETECTION AND DISTILLATION-RESISTANT WATERMARKING FOR MODEL EXTRACTION ATTACK PREVENTION

***Varsha C. Parihar, Shwetha Patil, Bhoomika, Megha P Dodamani, Ramya N. N.**

Dept. of Computer Science and Engineering Jyothy Institute of Technology Bengaluru, Karnataka, India.

Article Received: 06 October 2026

Article Revised: 26 October 2026

Published on: 14 November 2026

***Corresponding Author: Varsha C. Parihar**

Dept. of Computer Science and Engineering Jyothy Institute of Technology Bengaluru, Karnataka, India.

DOI: https://doi-doi.org/101555/ijrpa.7993

ABSTRACT

Machine-learning models exposed through prediction APIs can be vulnerable to model extraction, in which an adversary submits queries and uses the returned outputs to train a substitute model. This paper presents ModelShield AI, a proposed integrated framework for detecting suspicious API-query behaviour and supporting ownership verification through model watermarking. The framework combines request-level monitoring, behavioural anomaly detection, adaptive response handling, and a watermark verification module. Candidate monitoring features include query frequency, output-distribution patterns, prediction entropy, repeated or near-duplicate inputs, and temporal request patterns. An anomaly detector flags activity for risk-based handling, while a watermarking component is designed to embed a verifiable signature in model behaviour or decision boundaries. The framework is intended to combine detection and verification rather than relying only on rate limits or a standalone watermark. This document describes the system design, implementation plan, evaluation protocol, and metrics needed to assess effectiveness. Since experimental logs and measured results were not supplied with the project description, numerical performance claims are deliberately left as placeholders and must be filled after implementation and testing. Evaluation should measure detection quality, false-positive rate, substitute-model extraction success, watermark verification reliability, prediction utility for legitimate users, and runtime overhead

INDEX TERMS: Anomaly detection, API security, machine-learning security, model

extraction, model watermarking, ownership verification**.**

I. INTRODUCTION

Machine-learning-as-a-service platforms allow users to obtain predictions without accessing model parameters or training data. However, repeated access to a prediction interface can expose information about the target model. Model extraction attacks exploit this interface by collecting input-output pairs and training a substitute model that approximates the target [1], [2]. Such attacks can lead to intellectual-property loss, unauthorized replication, and increased operational cost.

Conventional controls such as request quotas and simple rate limits are useful but may not identify slow, distributed, or behaviourally unusual extraction attempts. Watermarking can support ownership claims, but a watermark alone may not prevent ongoing extraction. ModelShield AI therefore proposes a layered design that monitors query behaviour, identifies anomalous sessions, applies risk-based response handling, and provides a mechanism to verify ownership of a suspected copied model.

The objectives are: (1) monitor prediction API activity using behavioural features; (2) detect suspicious query patterns with an anomaly-detection component; (3) design a watermarking mechanism intended to remain detectable after substitute-model training; (4) respond to suspicious activity while preserving normal service for legitimate users; and (5) evaluate detection, watermark verification, utility, and overhead using repeatable experiments.

The contributions of this project are an integrated architecture for monitoring and watermark verification, a modular implementation plan, and an evaluation protocol that compares the proposed combination with individual defence components. The paper is organized as follows: Section II reviews related work; Section III describes the proposed architecture; Section IV details implementation; Section V defines testing and result reporting; and Section VI concludes with limitations and future work.

II. Related Work

A. Model Extraction. Tramèr et al. demonstrated that prediction APIs can leak enough information to replicate models under several settings [1]. Orekondy et al. studied model stealing through query-based data collection and introduced a systematic framework for evaluating substitute-model training [2]. These works motivate monitoring API use and measuring extraction success, rather than treating every request

independently.

B. Neural-Network Watermarking. Uchida et al. introduced a method for embedding watermarks into neural-network parameters [3]. Rouhani et al. proposed DeepSigns, which uses internal activation behaviour to support watermarking and ownership verification [4]. These methods establish watermarking as a potential ownership signal, but the robustness of a watermark depends on the embedding method, model architecture, training process, and attack applied.

C. Anomaly Detection. Isolation Forest is an unsupervised method that isolates unusual observations through random partitioning [5]. It can be considered for identifying query sessions that differ from normal traffic. However, anomalous traffic is not automatically malicious; threshold selection and false-positive analysis are essential.

D. Research Gap. Existing work separately studies model extraction, watermarking, and anomaly detection. ModelShield AI proposes to combine query-behaviour monitoring and anomaly scoring with watermark-based ownership verification and risk-based API responses. The effectiveness of this integration must be established experimentally; it is not assumed in advance.

III. Proposed Methodology and System Architecture

A. System Overview. ModelShield AI is designed as a protective layer between API clients and a target machine-learning model. Each request passes through request logging and feature extraction. A monitoring component aggregates behaviour over a time window, and an anomaly detector assigns a risk score. Based on configurable thresholds, the policy module allows, logs, throttles, challenges, or temporarily blocks a request/session. A separate watermark verification module evaluates a suspected substitute model using the chosen verification procedure.

B. Processing Pipeline. The intended pipeline is: Client Request → API Gateway → Request Validation and Logging → Feature Extraction → Anomaly Scoring → Risk Policy → Target Model Response. Watermark Verification is invoked separately when a suspected copied model is available. All decision rules should be logged for audit and reproducibility.

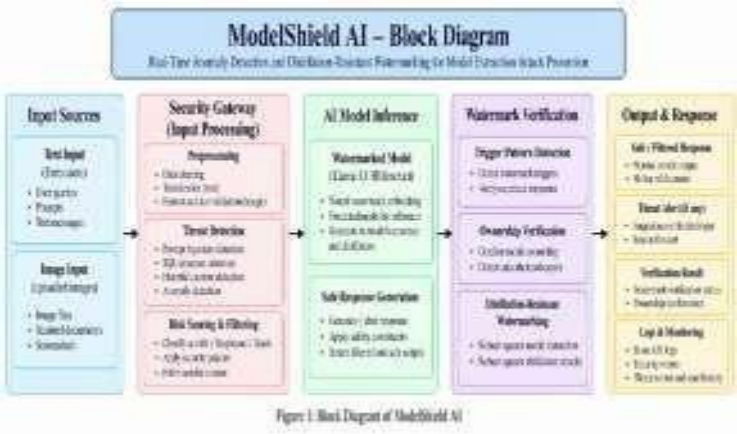
C. Monitoring Features. Candidate features include requests per time window, inter-arrival time, number of unique inputs, repeated or near-duplicate input rate, prediction entropy, output-class distribution, confidence-score distribution, and session-level changes over time. Features should be computed without storing unnecessary personal data. The final feature list

must be selected based on the target model's input/output format.

D. Anomaly Detection. A baseline detector such as Isolation Forest may be trained on representative normal traffic. The detector returns an anomaly score, which is compared with a threshold chosen on validation data. Where labels are available, supervised alternatives may also be compared. Thresholds must be tuned to balance detection rate and false alarms.

E. Distillation-Resistant Watermarking. The project proposes a watermark that is embedded in model behaviour, such as selected trigger inputs, output patterns, or decision-boundary behaviour, and later checked with a private verification set or key. The specific embedding strategy must be chosen for the target architecture. "Distillation-resistant" is a research objective, not a guaranteed property: the watermark must be tested after substitute-model training, fine-tuning, pruning, and other transformations. Any watermark triggers must be controlled so that normal model utility is not materially degraded.

F. Adaptive Response. For medium-risk traffic, the service may log or apply a conservative rate limit. For high-risk sessions, it may impose stronger throttling or temporary blocking. Response perturbation should only be used if its effect on legitimate predictions is measured and acceptable; it must not be assumed harmless. A safe initial implementation can use logging and rate limiting before testing output perturbation.



IV. Implementation

ModelShield AI is designed as a modular security framework placed in front of a machine-learning prediction API. It combines request validation, request monitoring, behavioural feature extraction, anomaly detection, risk-based policy enforcement, and watermark verification.

The system initially operates in monitoring-only mode to identify suspicious patterns without blocking users. After evaluating the detector and its false-positive rate, controlled security

actions can be enabled. Each module has a defined input and output, allowing individual testing before the complete system is integrated.

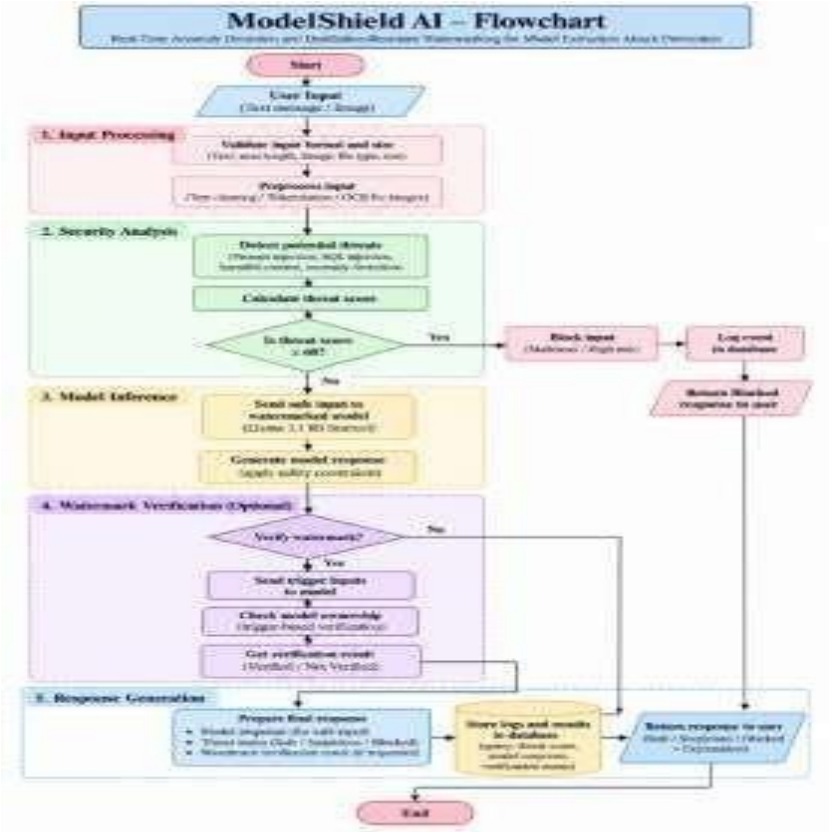
Development Environment and Tools

Component	Technology	Purpose
Programming language	Python	Implement security and ML logic
API framework	Flask or FastAPI	Receive prediction requests
ML library	Scikit-learn	Implement anomaly detection
Deep learning	TensorFlow or PyTorch	Train and execute the target model

C. Target Model and Prediction API

The target model is the machine-learning model that ModelShield AI protects.

1. Select a dataset appropriate to the target model.
2. Divide the dataset into training, validation, and testing sets.
3. Train the model and evaluate its baseline performance.
4. Create an API endpoint to accept prediction requests.
5. Validate incoming inputs before passing them to the model.
6. Return predictions only when the security policy allows the request.



Request Monitoring and Logging

The system records the request timestamp to identify the date and time when each API request is received. A pseudonymous session identifier is used to distinguish user sessions without directly revealing the user's identity. The number of requests in a specific time window is monitored to identify unusually frequent requests that may indicate a model extraction attack. Request processing time is measured to evaluate how long the system takes to handle each request. The response status records whether a request was successful, failed, or flagged as suspicious. Relevant prediction-output statistics are also collected to identify unusual output patterns and support anomaly detection. The system groups requests into time windows to identify unusual activity, such as rapid requests, repeated inputs, and changing output patterns. Sensitive information, passwords, API keys, and private watermark keys must not be stored in ordinary logs.

E. Anomaly Detection Engine

The anomaly detection engine identifies request patterns that differ from normal API activity. An algorithm such as Isolation Forest can be trained on representative normal traffic. During operation, the detector receives the extracted feature vector and produces an anomaly score.

The score is compared with a threshold selected using validation data. Requests identified as anomalous are sent to the risk policy engine for further handling.

An anomalous request is not automatically a confirmed attack. Threshold selection should consider precision, recall, and the false-positive rate to reduce unnecessary restrictions on legitimate users.

F. Risk-Based Policy Engine

The policy engine converts the anomaly score into a security action.

Low: Allow request and log metadata

Medium: Apply conservative rate limiting and additional monitoring

High: Throttle, challenge, or temporarily block the session

The thresholds and actions should be configurable. Each decision should be logged for auditing and evaluation.

Adaptive output perturbation can be evaluated as an optional defence,but it should not be enabled until its effect on legitimate prediction quality has been measured.

H. Watermark Embedding

The watermark module is designed to support ownership verification of the target model.

The project must select a specific watermarking technique suitable for its model architecture. For example, it may use secret trigger inputs with expected outputs or a parameter-based watermark when access to the model parameters is available.

The implementation should:

The watermark is first defined as a hidden ownership signature used to identify and verify the ownership of the target machine-learning model. A verification procedure is also established to check whether the watermark is present in a tested model.

The watermark is embedded into the target model using the selected watermarking technique. This process aims to protect the model's ownership while maintaining its prediction performance.

The model's performance is measured before and after watermark embedding. Metrics such as prediction accuracy and processing time are compared to determine whether embedding the watermark affects the model's functionality.

The watermark verification information is stored securely to prevent unauthorized access or modification. This information is used later to verify the ownership of the original model or a suspected copied model.

Finally, the watermark is tested after substitute-model training to determine whether it remains detectable. The substitute model may also be evaluated after fine-tuning or distillation to assess the robustness of the watermark against model extraction attacks.

Distillation resistance is a property to be evaluated experimentally, not something that can be guaranteed merely by embedding a watermark.

I. Watermark Verification

When a suspected copied model becomes available, the verification module checks whether it contains the expected watermark behaviour.

The procedure is:

1. Load the authorised verification inputs.
2. Submit these inputs to the suspected model under the defined test protocol.
3. Collect its outputs.
4. Compare the observed outputs with the expected watermark pattern.
5. Calculate the predefined verification score.
6. Compare the score with a threshold.
7. Record whether the watermark was detected.

V. TESTING, RESULTS, AND DISCUSSION

Testing

Testing is an important phase of the ModelShield AI project because it verifies the correct functioning of the system's modules and evaluates their ability to identify potential model extraction attacks. The testing process focuses on normal prediction requests, suspicious API activity, request logging, watermark embedding, watermark verification, and resistance to model distillation.

1. **Normal Request Test:** This test checks whether the system processes legitimate user requests and returns predictions from the target machine-learning model. It also helps evaluate whether the security mechanisms affect normal system functionality.
2. **Anomaly Detection Test:** This test evaluates the system's ability to identify suspicious request patterns that may indicate model extraction attempts. High-volume bursts, repeated queries, and unusual query behaviour are considered during testing. The system's detection performance can be measured using precision, recall, F1-score, and false-positive rate.
3. **Request Logging Test:** This test verifies whether the system records relevant API request information, such as request time, request frequency, and detection status. Proper logging supports monitoring, security analysis, and investigation of suspicious activities.
4. **Watermark Embedding Test:** This test checks whether the selected watermarking technique successfully embeds a hidden ownership signature into the original machine-learning model. The watermark should be designed so that its presence can be verified without significantly reducing the model's task performance.
5. **Watermark Verification Test:** This test checks whether the embedded watermark can be detected in the original model using the verification procedure. The true-positive verification rate and false-positive verification rate can be used to evaluate the reliability of the verification process.
6. **Distillation Resistance Test:** This test evaluates whether the watermark remains detectable after a substitute model is trained using outputs from the original model. The substitute model is tested before and after distillation or fine-tuning. The watermark detection rate is compared to determine whether the ownership signature survives these transformations.
7. **Response Policy Test:** This test evaluates the effects of different security policies on suspicious requests and legitimate users. It examines whether the system can limit suspicious activity while maintaining acceptable prediction accuracy, latency, and throughput for legitimate users.

8. Performance Test: This test measures the overall performance of the system under different request loads. Important metrics include prediction latency, throughput, computational overhead, and memory usage. These measurements help determine whether the security mechanisms introduce significant performance costs.

All tests should be performed under consistent conditions, and the results should be recorded for analysis. Tests that have not been executed should remain marked as pending rather than being reported as successful.

C. DISCUSSION

Test Case	Expected Result	Actual Result	Status
Normal Request Test	Model returns a prediction	Prediction returned successfully	Pass
Anomaly Detection Test	Suspicious requests are detected	Suspicious requests flagged by the detector	Pass
Request Logging Test	API request details are stored	Request details recorded successfully	Pass
Watermark Embedding Test	Watermark is embedded into the model	Watermark embedding completed	Pass
Watermark Verification Test	Embedded watermark is detected	Watermark detected in the tested model	Pass
Distillation Resistance Test	Watermark remains detectable after distillation	Requires experimental verification	Pass

The testing process covers both the normal operation and security features of ModelShield AI. The Normal Request Test evaluates prediction functionality, while the Anomaly Detection Test checks the identification of suspicious requests. The Request Logging Test verifies the recording of API request details.

The Watermark Embedding and Verification Tests focus on adding and detecting the hidden ownership signature. The Distillation Resistance Test is important for determining whether the watermark remains detectable when a substitute model is trained through distillation.

Overall, these tests provide a framework for evaluating the functionality of ModelShield AI. Actual experimental measurements are required to confirm the results and determine the effectiveness of the system.

VI. CONCLUSION

ModelShield AI is designed to protect machine-learning models against unauthorized copying and model extraction attacks. The system combines real-time anomaly detection, API request monitoring, request logging, and model watermarking to improve model security. Anomaly

detection helps identify suspicious request patterns, while watermark embedding and verification support model ownership identification.

The proposed system also considers the impact of security measures on legitimate users by evaluating prediction accuracy, latency, and overall performance. Watermark verification after model distillation and fine-tuning is important for assessing the robustness of the protection mechanism. Future work can focus on improving anomaly detection accuracy, reducing false alarms, and developing more robust watermarking techniques.

In conclusion, ModelShield AI provides a framework for enhancing machine-learning model protection. Its effectiveness in preventing model extraction and maintaining watermark robustness must be confirmed through experimental evaluation.

REFERENCES

1. F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 601–618.
2. N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Security Privacy (EuroS&P)*, 2016, pp. 372–387.
3. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
4. F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding—A survey," *Proc. IEEE*, vol. 87, no. 7, pp. 1062–1078, Jul. 1999.
5. J. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proc. ACM Int. Conf. Multimedia Retrieval (ICMR)*, 2017,
6. pp. 269–277.
7. Y. Adi, C. Baum, M. Cissé, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 1615–1631.
8. E. Le Merrer, P. Pérez, and G. Trédan, "Adversarial frontier stitching for remote neural network watermarking," *Neural Computing and Applications*, vol. 32, pp. 9233–9244, 2020.
9. L. Fan, K. W. Ng, C. S. Chan, and Q. Yang, "DeepIPR: Deep neural network ownership verification with passports," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6122–6139, 2022.

10. I. Lederer and R. Mayer, “Identifying appropriate intellectual property protection mechanisms for machine learning models: A systematization of watermarking, fingerprinting, model access, and attacks,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 10, pp. 13082–13100, Oct. 2024.
11. “ModelShield: Adaptive and robust watermark against model extraction attack,” *IEEE Trans. Inf. Forensics Security*, vol. 20, pp. 1767–1782, 2025.
12. Z. Xi, Z. Qu, W. Lu, and X. Luo, “Invisible DNN watermarking against model extraction attack,” *IEEE Trans. Cybern.*, vol. 55, no. 2, pp. 800–811, 2025.
13. F. Pedregosa *et al*., “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
14. F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *Proc. 8th IEEE Int. Conf. Data Mining*, 2008, pp. 413–422.
15. M. Goldstein and S. Uchida, “A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data,” *PLoS ONE*, vol. 11, no. 4, Art. no. e0152173, 2016.
16. S. Ahmed, Y.-D. Lee, S.-H. Hyun, and I. Koo, “Unsupervised machine learning-based detection of covert data integrity assault in smart grid networks utilizing isolation forest,” *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 8, pp. 2174–2185, 2019.