Model Research Report: AI-Based Cyber Security Threat Prediction

Prepared for: AI Threat Prediction Agent Project

Executive Summary

This document compares candidate machine learning and deep learning models for building an AI-based cyber-security threat prediction system. It evaluates models on accuracy, latency, interpretability, data requirements, robustness, scalability, and operational complexity. Recommendations include which model families to prioritize for different use-cases (real-time detection, offline forensic analysis, graph-based risk scoring), and an implementation roadmap for experimentation and production deployment.

Objectives & Success Metrics

Objectives:

- Predict cyber-security threats/attacks or anomalous behavior on hosts and networks.

- Provide risk scores, actionable alerts, and concise explanations to SOC analysts.

Success Metrics:

- Detection metrics: Precision, Recall, F1-score, AUC-PR (preferred for imbalanced data).

- Operational metrics: Latency (ms), throughput (events/sec), false positive rate, alerts per 1k hosts.

- Business metrics: Mean time to detect (MTTD), SOC workload reduction, cost of false positives/negatives.

Data Assumptions

- Data sources: Network logs (NetFlow), host logs (Syslog, EDR), authentication logs, firewall/IDS alerts, threat intel feeds.

- Typical characteristics: highly imbalanced (few malicious events), noisy, time-series/sequential nature, graph relationships (hosts, users, IPs).

- Labeling: partial/weak labels from alerts, SOC triage, sandbox verdicts.

Candidate Model Families

I. Classical Machine Learning (ML) Models

- Logistic Regression: simple, interpretable, low latency. Works well with engineered features and balanced datasets.

- Random Forest / Gradient Boosted Trees (XGBoost / LightGBM): strong baselines for tabular features, robust to feature scaling, good accuracy, moderate interpretability (feature importances, SHAP).

- SVM / One-Class SVM: anomaly detection when labeled negatives are scarce. Scales poorly to very large datasets.

- Isolation Forest / LOF: unsupervised anomaly detection, effective for outlier detection on tabular features.

II. Deep Learning (DL) Models

- Fully Connected Neural Networks (MLP): for richer nonlinear interactions in tabular features; needs more data and careful regularization.

- Recurrent Neural Networks (LSTM/GRU): for sequential/time-series modeling (user/host event sequences). Good at capturing temporal patterns but can be slower at inference.

- Transformer-based models: powerful for sequences, can model long-range dependencies. More computationally expensive but state-of-the-art for sequence representation.

- Autoencoders / Variational Autoencoders (VAE): unsupervised anomaly detection for reconstructive models on normal behavior.

- Graph Neural Networks (GNNs): model relationships between hosts, users, IPs, and services for propagation/lateral movement detection. Excellent for graph-structured data; more complex to build and serve.

- Contrastive / Self-Supervised Models: learn representations from unlabeled logs which can be fine-tuned for downstream tasks.

Model Capability Comparison

Model	Accuracy (potential)	Latency	Data Required	Interpretability	Robustness to Drift	Ease of Deployment	Best Use-case
Logistic Regression	Low-Medium	Very Low	Low	High	Low-Medium	High	Simple baselines, early-warning

Random Forest / XGBoost	Medium-High	Low-Medium	Medium	Medium (SHAP)	Medium	High	Tabular features, alert scoring
Isolation Forest / LOF	Medium	Low	Low	Medium	Low-Medium	High	Unsupervised anomaly detection
SVM / One-Class SVM	Medium	Medium-High	Low	Low	Low	Medium	Small datasets, anomaly detection
MLP (NN)	Medium-High	Medium	Medium-High	Low	Medium	Medium	Nonlinear tabular interactions
LSTM / GRU	High	Medium-High	High	Low	Medium	Medium	Sequential event detection
Transformer (sequence)	Very High	High	Very High	Low	Medium-High	Low-Medium	Long-range sequence patterns
Autoencoder / VAE	Medium-High	Medium	High	Low	Low-Medium	Medium	Unsupervised anomaly detection
GNN (e.g., GAT, GraphSAGE)	Very High	Medium-High	High	Low-Medium	High	Low-Medium	Graph-based risk/lateral movement detection
Contrastive / SSL	High (with fine-tune)	Medium-High	High (pretraining)	Low	High	Medium	Representation learning for few-shot settings

Which Model to Choose and Why

Guiding principle: align model family to the problem, available data, latency requirements, and operational constraints. Below are recommendations by use-case.

A. Low-latency, real-time detection (e.g., inline alert scoring, streaming)

- Preferred: Lightweight ML models (Logistic Regression, LightGBM/XGBoost) or optimized MLPs.

- Why: They offer a strong balance of accuracy, low inference latency, and easier explainability. Feature-engineered tabular inputs perform well in streaming contexts.

- Notes: Use online feature store (Redis), model quantization or optimized inference (ONNX, Triton) for speed.

B. Sequential behavior / user-host event modeling (e.g., lateral movement, command sequences)

- Preferred: LSTM/GRU or Transformer-based sequence models; consider lightweight Transformers or distilled models for faster inference.

- Why: They capture temporal dependencies and sequence patterns that classical ML cannot without heavy feature engineering.

- Notes: If latency is critical, consider hybrid approach: use sequence model offline to produce embeddings, then a lightweight classifier for online scoring.

C. Graph-based risk scoring (e.g., lateral propagation, supply-chain interactions)

- Preferred: GNNs (GraphSAGE, GAT) for relational reasoning.

- Why: Graph models naturally model relationships and propagation effects; very effective for multi-hop attack detection.

- Notes: Build a periodic graph construction pipeline and precompute node embeddings for online lookup to reduce latency.

D. Unsupervised anomaly detection (rare events, unknown threats)

- Preferred: Autoencoders, Isolation Forests, or One-Class models.

- Why: They detect deviations from learned normal behavior where labels are scarce.

- Notes: Combine unsupervised scores with supervised classifiers in an ensemble.

Best Overall Approach (Practical Recommendation)

There is no single "best" model for all aspects; a hybrid, ensemble-based architecture gives the best trade-offs:

- Use a tiered/ensemble system:

  1) Lightweight supervised model (XGBoost/LightGBM) for low-latency, high-precision scoring on engineered features.

  2) Sequence model (Transformer or LSTM) or precomputed sequence embeddings for deeper temporal patterns.

  3) GNN-based risk scorer for graph effects and lateral movement.

  4) Unsupervised anomaly detectors (Autoencoder/Isolation Forest) to catch novel threats.

- Fuse outputs in a Decision Agent that weights models according to confidence, asset criticality, and recent model performance. This design maximizes accuracy, resilience, interpretability (via SHAP on tabular), and operational speed (by prioritizing fast models for initial scoring).

Efficiency, Speed, and Accuracy Trade-offs

- Classical ML (XGBoost/LightGBM): fast training (relative), very fast inference (optimized), strong accuracy on tabular data with moderate feature engineering. Good interpretability (SHAP).

- Deep Learning (Transformers, GNNs): higher accuracy for sequences/graphs but heavier compute and more data needed. Use pretraining and distillation to reduce cost and latency.

- Ensemble/hybrid: best predictive quality but increases deployment complexity. Mitigate with model orchestration and prioritized inference (fast path then slow path).

Practical Experimentation Roadmap

Phase 0: Data preparation & feature store

- Build ingestion for one log source, create offline dataset, compute baseline features.

Phase 1: Baseline models

- Train Logistic Regression and XGBoost baseline, evaluate on holdout, produce confusion matrix and PR curve.

Phase 2: Unsupervised detectors

- Train Isolation Forest and Autoencoder on 'normal' windows, evaluate detection recall and false positives.

Phase 3: Sequence & graph models

- Train LSTM/Transformer on event sequences; build a small graph and experiment with GraphSAGE for node embeddings.

Phase 4: Ensemble & Decision Agent

- Build fusion logic, validation, and shadow deployment. Monitor operational metrics and SOC feedback.

Phase 5: Optimization & Deployment

- Optimize models (quantization, ONNX), set up serving (Triton/TF-Serving/fastAPI), and integrate with Feature Store and SOAR.

Implementation Notes & Best Practices

- Start with strong feature engineering and solid baselines; many security problems are feature-solved.

- Use cross-validation and time-aware splits (avoid leakage across time).

- Address class imbalance via focal loss, resampling, or cost-sensitive learning.

- Track model and data versions; implement monitoring for drift and performance regression.

- Design human-in-the-loop workflows and conservative auto-action policies initially.

References & Further Reading

- MITRE ATT&CK framework for mapping techniques and explainability.

- Papers on GNNs for security and Transformers for log analysis (search for LogBERT, DeepLog, GraphSAGE).

- Practical resources: Feast (feature store), ONNX/Triton for model serving, LightGBM/XGBoost documentation.