scientific reports

Check for updates

OPEN

Few-shot android malware classification with quantum-enhanced prototypical learning and drift detection

Mohammed Tawfik[1]✉, Hussam Tarazi[2], Ahmad Dalalah[3], Bajeszeyadaljunaeidia[3], Warda M. Shaban[4] & Islam S. Fathi[3]

Android malware detection systems face critical challenges including data scarcity for emerging threat families, high-dimensional feature spaces, and concept drift caused by evolving attack techniques. Traditional machine learning approaches require extensive labeled datasets and frequent retraining, limiting their practical deployment against rapidly emerging threats. This paper proposes an adaptive few-shot malware classification framework that integrates CatBoost-based feature selection, prototypical networks with episodic meta-learning, quantum-enhanced classification, concept drift detection, and explainable AI (XAI) analysis using SHAP and LIME. The CatBoost feature selection reduces dimensionality by 99.46% on CCCS-CIC-AndMal-2020 (9,503 to 51 features) and 94.07% on KronoDroid (489 to 29 features) while preserving discriminative information. The prototypical network learns metric-based representations enabling classification with only 5 support samples per class. Extensive experiments demonstrate state-of-the-art performance with 99.70% accuracy on CCCS-CIC-AndMal-2020 (15 malware families) and 99.33% accuracy on KronoDroid (binary classification), outperforming existing methods by 0.70–9.70%. The framework exhibits robust temporal stability with maximum accuracy degradation of 0.24% across evaluation periods. XAI analysis reveals that file descriptor manipulation and file system operations are the most discriminative features for malware detection. These results establish few-shot prototypical learning with intelligent feature selection as an effective paradigm for practical malware detection requiring minimal annotation, interpretable decisions, and stable long-term performance.

Keywords Few-shot learning, Android malware detection, Quantum machine learning, Prototypical networks, Concept drift detection

The Android operating system powers over 70% of mobile devices globally, making it the primary target for cybercriminals, with over 480,000 new malware samples discovered daily[1]. Machine learning approaches leveraging static and dynamic features from Android application packages (APKs) have shown promise[2], yet conventional supervised methods face a fundamental deployment bottleneck: they require large labeled datasets per malware family, while security analysts routinely encounter emerging threats with only a handful of confirmed samples[3]. This data scarcity challenge reflects a broader paradigm shift in AI-driven automation toward task-adaptive and data-efficient learning. Recent comprehensive reviews of AI-driven automation technologies[4] have systematically identified few-shot learning, continual adaptation, and robustness against distribution shift as critical capabilities for deployment in dynamic operational environments. These taxonomies directly motivate our integration of meta-learning with drift-aware mechanisms, positioning our framework within the broader evolution toward autonomous, self-adapting security systems. Compounding this problem, real-world malware datasets exhibit severe class imbalance, with certain families containing thousands of samples while others have fewer than ten known instances[5]. **The core gap in existing methods is therefore threefold**: (i) inability to classify novel malware families from minimal samples, (ii) lack of mechanisms to handle temporal distribution

[1]Faculty of Computer and Information Technology, Sana'a University, Sana'a, Yemen. [2]Department of Cyber Security, Faculty of Information Technology, Ajloun National University, P.O. Box 43, Ajloun 26810, Jordan. [3]Department of Computer Science, Faculty of Information Technology, Ajloun National University, P.O. Box 43, Ajloun 26810, Jordan. [4] Department of Communication and Electronics Engineering, Nile Higher Institute for Engineering and Technology, Mansoura, Egypt. ✉email: m.tawfik@su.edu.ye

shifts (concept drift) without costly full retraining, and (iii) absence of a unified framework that jointly addresses data scarcity, high-dimensional feature spaces, and evolving threat distributions.

Few-shot learning, and prototypical networks in particular[6], offer a principled solution to data scarcity by learning metric-based representations that generalize to novel classes from only a small number of support examples[7]. Meta-learning frameworks such as MAML have been successfully applied to Android malware classification[8], demonstrating the viability of this paradigm. However, deployed malware detection systems also face concept drift, wherein the statistical properties of malware samples evolve over time due to adversarial adaptation and shifting threat landscapes[9]. The KronoDroid dataset[10] specifically addresses this temporal dimension with timestamped samples spanning 2008–2020, enabling rigorous evaluation of model robustness against concept drift. A robust framework must therefore incorporate both few-shot adaptability and drift-aware mechanisms. Additionally, quantum computing offers complementary capabilities through variational quantum algorithms that combine parameterized quantum circuits with classical optimization, potentially capturing complex feature correlations in high-dimensional spaces[11,12].

In this paper, we propose an adaptive few-shot malware classification framework that integrates prototypical networks, quantum-enhanced feature learning, intelligent feature selection, and concept drift detection for robust Android threat classification. Our framework addresses the fundamental challenges of data scarcity, high-dimensional feature spaces, class imbalance, and temporal distribution shifts through a unified architecture. The main contributions of this work are fivefold. First, we develop a few-shot prototypical network architecture adapted for Android malware family classification, achieving 99.70% accuracy on CCCS-CIC-AndMal-2020[13] and 99.33% accuracy on KronoDroid[10] with only 5 support samples per class during inference. Second, we employ CatBoost-based feature selection—a well-established technique—to reduce dimensionality by 99.46% on CCCS-CIC-AndMal-2020 (9,503→51 features) and 94.07% on KronoDroid (489→29 features) while maintaining classification performance. Third, we integrate a quantum-enhanced hybrid classification layer incorporating a 4-qubit parameterized quantum circuit with rotation encoding and ring entanglement topology; while the quantum component provides a modest accuracy improvement in the current simulated setting, this integration establishes an architectural foundation for future quantum hardware deployment. Fourth, we incorporate a concept drift detection module implementing cumulative temporal split evaluation to monitor and quantify performance degradation over time, demonstrating stability with maximum accuracy degradation limited to 0.24% across evaluation periods. Fifth, we conduct comprehensive experiments on two benchmark datasets comparing our approach against state-of-the-art methods including deep neural network ensembles[14], graph convolutional networks[1], and meta-learning baselines, demonstrating competitive or superior performance across evaluation metrics. We note that the primary methodological contribution lies in the unified integration of these components—few-shot learning, feature selection, quantum enhancement, drift detection, and explainability—into a coherent end-to-end framework, rather than in the novelty of any individual component.

The remainder of this paper is organized as follows. Section 2 presents related work on Android malware classification, few-shot learning, and quantum machine learning. Section 3 details the proposed methodology, including the prototypical network architecture, feature selection mechanism, quantum circuit design, and drift detection module. Section 4 presents experimental results, comparative analysis, and ablation studies. Finally, Section 5 concludes the paper and outlines future research directions.

Related work

Android malware classification has attracted substantial research attention, with diverse approaches ranging from traditional machine learning to deep learning and meta-learning paradigms. Earlier surveys such as Kumars et al[15]. provided systematic overviews of the technological evolution from static and dynamic analysis to intelligent detection using machine learning and deep learning techniques, establishing foundational taxonomies for the field. More recently, Smmarwar et al[16]. provide a comprehensive 2024 survey cataloguing over 140 recent Android malware studies, systematically grouping them into feature-selection, ML-based, and DL-based strands, and identifying six critical open gaps including sub-optimal high-dimensional selectors, class-imbalance blindness, and zero-day detection fragility.

Nazim et al[14]. proposed a multimodal deep neural network ensemble combining CNN and LSTM architectures with late fusion for malware classification on the CCCS-CIC-AndMal-2020 dataset, achieving 95.36% accuracy using both numeric features and malware image representations.

Li et al[5]. addressed class imbalance through SynDroid, which employs Conditional Tabular GANs (CTGAN) combined with SVM for synthetic sample generation, demonstrating a 12% accuracy improvement over baseline methods on the same dataset.

Ababneh et al[2]. achieved approximately 99% accuracy using optimized Random Forest with Information Gain Attribute Evaluation, utilizing only 27 dynamically selected features from the CCCS-CIC-AndMal-2020 dataset. Al-Sraratee and Al-Azawei[17]presented a dynamic-analysis-driven framework that classifies fourteen Android malware families with 98.5% accuracy while discarding 78.87% of the original features through Mutual Information ranking and PCA compression to 33 principal components. Hammood et al[18]. proposed a hybrid-analysis detector combining Particle Swarm Optimization (PSO) feature selection with Adaptive Genetic Algorithm (AGA) hyper-parameter tuning, achieving 99.82% accuracy on CCCS-CIC-AndMal-2020 with XGBoost.

Polatidis et al[19]. introduced FSSDroid, a binarization-driven feature-selection pipeline that compresses large Android malware datasets into compact family-specific binary feature sets, achieving up to 100% accuracy on certain families using only 9–27 features on both KronoDroid and CCCS-CIC-AndMal-2020 datasets. Sanamontre et al[20]. proposed a two-stage pipeline for detecting malicious repackaged Android game APKs, achieving 94.78% accuracy with Random Forest trained on the CCCS-CIC-AndMal-2020 dataset.

Guerra-Manzanares et al[10]. introduced KronoDroid, a timestamped hybrid-featured Android dataset spanning 2008–2020 with 489 static and dynamic features. This dataset was specifically designed to address concept drift evaluation with samples from over 209 malware families across emulator and real device sources. Aurangzeb and Aleem[21]presented Obf-Droid, a hybrid-analysis ensemble system using chi-squared selection and majority-vote fusion of five learners, achieving 95% accuracy on KronoDroid real-device samples with robustness against obfuscation techniques. Wajahat et al[22]. proposed Dynamic-Weighted Federated Averaging (DW-FedAvg) for privacy-centric Android IoT malware detection, attaining 99.69% accuracy on Malgenome while outperforming standard FedAvg by up to 0.13% accuracy and 23% lower false positive rate.

Few-shot learning approaches have also been explored for malware classification to address data scarcity challenges. Wang et al[7]. proposed SIMPLE, a multi-prototype modeling approach using BiLSTM and temporal convolution for API sequence embedding, achieving 90% accuracy in 5-way 5-shot classification scenarios. Bai et al[3]. developed a Siamese network-based approach for malware family classification at ICSE 2020, demonstrating robust generalizability for families with limited samples by embedding applications into continuous vector spaces through contrastive learning. Li et al[8]. extended MAML for multi-family Android malware classification with novel sampling strategies, achieving 100% classification accuracy on certain malware families using episodic learning with support and query sets. Recent advancements in intelligent threat detection have increasingly prioritized frameworks that balance high-performance accuracy with computational efficiency and adaptability to data-scarce environments. Addressing resource constraints in IoT and fog computing, Tawfik[23] established a foundation for optimized intrusion detection by integrating CatBoost-based feature selection with a hybrid Transformer-CNN-LSTM ensemble, demonstrating that robust feature engineering is critical for securing distributed networks. Extending these principles to privacy-sensitive domains, Tawfik et al.[24] introduced FedMedSecure, a framework that synergizes federated learning with few-shot mechanisms to secure the Internet of Medical Things (IoMT), proving that collaborative defense is achievable without compromising patient data. Furthermore, the versatility of few-shot learning in handling emerging threats was validated by Tawfik et al.[25] through XF-PhishBERT, which combines ModernBERT architectures with explainable AI (XAI) to detect novel phishing campaigns using minimal support samples. It is important to delineate the contribution boundaries of the present work relative to these prior studies: unlike Tawfik[23], which focused on traditional supervised CatBoost-based feature selection with ensemble deep learning for IoT intrusion detection, the present paper extends this line of work by integrating meta-learning with quantum-enhanced circuits to address the data scarcity problem specific to Android malware family classification, incorporating episodic prototypical learning and concept drift detection as components absent from our previous frameworks.

For network security domains, Xu et al[26]. proposed FC-Net combining feature extraction and comparison networks based on meta-learning, achieving a 99.62% detection rate in cross-dataset testing scenarios. Beyond cybersecurity, the challenge of few-shot generalization has been addressed in related domains through task-adaptive augmentation strategies. Jin et al[27]. proposed PMTL-DisCo, a prompting multi-task learning framework for few-shot fake news detection that leverages dissemination consistency reasoning to augment limited training signals. Their work demonstrates that combining multi-task auxiliary objectives with prompt-based tuning can substantially improve generalization under data scarcity—a principle directly relevant to our malware classification setting, where the episodic meta-learning paradigm similarly constructs auxiliary classification tasks to learn transferable representations from limited per-class samples. The success of such task-adaptive data augmentation strategies across diverse domains reinforces the broader viability of few-shot learning frameworks for classification problems where labeled data is inherently scarce. Quantum machine learning has emerged as a promising direction for cybersecurity applications, with Kalinin and Krundyshev[12] demonstrating 98% classification accuracy using Quantum SVM and Quantum CNN for intrusion detection, while achieving 2–2.5×training speedup compared to classical counterparts. Sridevi et al[28]. introduced a unified Hybrid Quantum-Classical Neural Network (HQCNN) that couples wavelet-based pre-processing with a Dressed Quantum Circuit, achieving 95.13% accuracy for 12-class malware classification on CCCS-CIC-AndMal-2020 and establishing a new state-of-the-art for quantum-enhanced Android malware detection.

Concept drift remains a critical challenge for deployed malware systems. Escudero García et al[9]. showed that transfer learning can achieve up to a 22% improvement under drift conditions. Graph-based approaches have also shown effectiveness, with Gao et al[1]. proposing GDroid using graph convolutional networks to map applications and APIs into heterogeneous graphs, achieving 98.99% detection accuracy with less than 1% false positive rate on the Drebin dataset.

Beyond malware-specific detection, recent advances across cybersecurity domains provide complementary insights for malware classification research. Wu et al[29]. revealed privacy vulnerabilities in transfer learning through membership inference attacks, demonstrating that knowledge transfer mechanisms—analogous to meta-learning—can inadvertently expose training data. In web application security, Wu et al[30]. developed WAFBooster to automatically detect and patch bypasses in web application firewalls against mutated payloads, highlighting the broader challenge of adversarial evasion that also affects malware detection systems. Liang et al[31]. proposed VulSeye, a stateful directed graybox fuzzer for smart contract vulnerability detection, demonstrating that targeted analysis guided by vulnerability patterns can substantially outperform coverage-guided approaches—a principle relevant to directed malware analysis. Wu et al[32]. introduced TCG-IDS, a self-supervised temporal contrastive graph neural network for intrusion detection that achieves strong performance without labeled data through temporal and asymmetric contrastive learning, offering a promising paradigm for reducing annotation requirements in security applications.

Despite these advances, existing approaches typically address individual challenges in isolation, lacking a unified framework that simultaneously handles data scarcity, high-dimensional feature spaces, temporal drift, and complex feature correlations. Our proposed framework bridges this gap by integrating intelligent

feature selection, few-shot prototypical learning, quantum-enhanced classification, and drift-aware adaptation mechanisms.

Methodology

The proliferation of Android malware variants poses significant challenges to traditional machine learning-based detection systems, which typically require large volumes of labeled training data to achieve acceptable classification performance. In real-world deployment scenarios, security analysts frequently encounter novel malware families with limited available samples, rendering conventional supervised learning approaches ineffective. Furthermore, the dynamic nature of the threat landscape introduces concept drift, wherein the statistical properties of malware samples evolve over time, leading to degraded model performance. To address these fundamental challenges, we propose a framework that integrates few-shot prototypical learning with quantum-enhanced feature representations for robust Android malware family classification under temporal distribution shifts.

Our proposed framework, illustrated in Fig. 1, comprises five interconnected modules: (i) data acquisition and preprocessing, (ii) CatBoost-based feature selection, (iii) few-shot prototypical network with episodic meta-learning, (iv) quantum-enhanced hybrid classification layer, and (v) concept drift detection and monitoring. Each component is meticulously designed to address specific limitations of existing approaches while maintaining computational tractability for practical deployment. The following subsections provide comprehensive technical descriptions of each module, including mathematical formulations, architectural specifications, and implementation considerations.

Mathematical framework

Before presenting the detailed methodology, we establish the mathematical notation and formally define the problem addressed by our framework. Let $\mathscr{X} \subseteq \mathbb{R}^d$ denote the d-dimensional feature space representing static and dynamic characteristics extracted from Android application packages (APKs), and let $\mathscr{Y} = \{1, 2, \ldots, K\}$ represent the set of K distinct malware family labels. A labeled dataset is defined as $\mathscr{D} = \{(\mathrm{x}_i, y_i)\}_{i=1}^{N}$, where $\mathrm{x}_i \in \mathscr{X}$ denotes the feature vector of the i-th sample and $y_i \in \mathscr{Y}$ represents its corresponding malware family label.

In the few-shot learning paradigm, we consider N-way K-shot classification tasks, where the objective is to classify query samples into one of N classes given only K labeled support examples per class. Formally, an episode $\mathscr{E}$ consists of a support set $\mathscr{S} = \{(\mathrm{x}_j^s, y_j^s)\}_{j=1}^{N \times K}$ containing K samples from each of N randomly selected classes, and a query set $\mathscr{Q} = \{(\mathrm{x}_q, y_q)\}_{q=1}^{N \times Q}$ containing Q samples per class to be classified. The goal is to learn an embedding function $f_\phi : \mathscr{X} \to \mathbb{R}^{d_e}$ parameterized by ϕ that maps input features to a d_e-dimensional embedding space where samples from the same class cluster together while samples from different classes are well-separated.

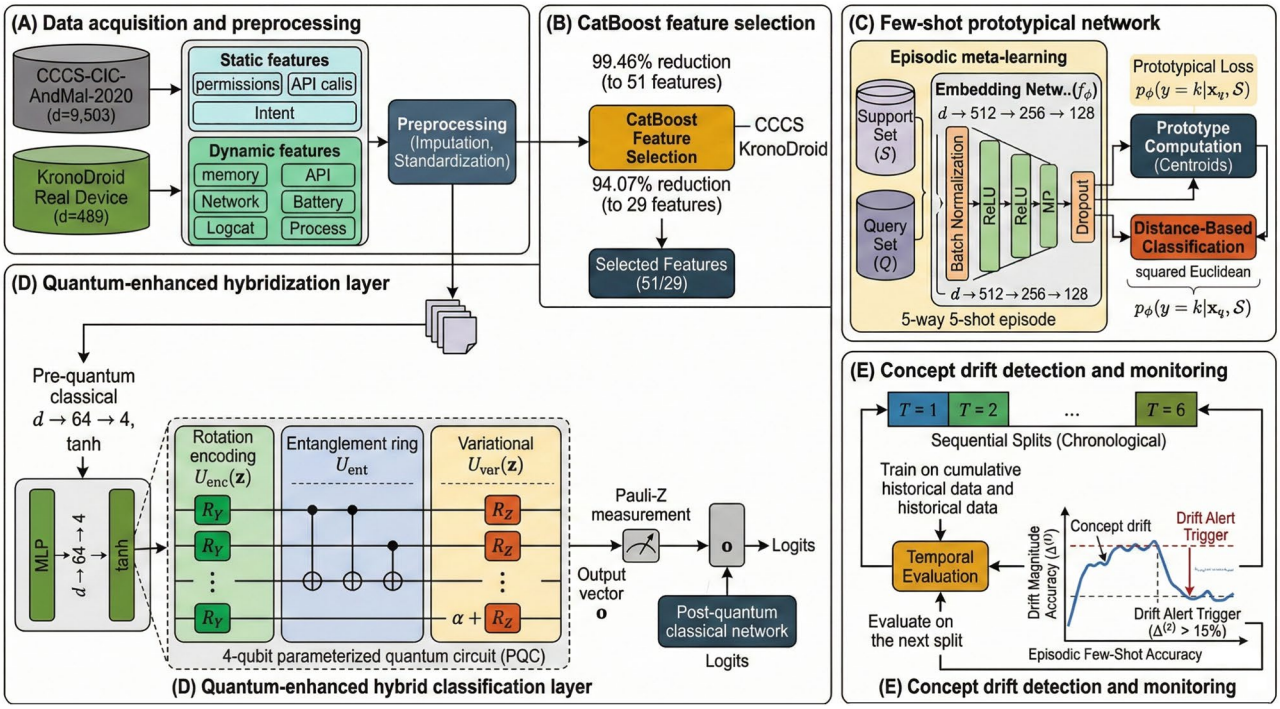

Fig. 1. Overall framework with five modules: (**A**) data preprocessing, (**B**) CatBoost feature selection, (**C**) few-shot prototypical network, (**D**) quantum-enhanced hybrid classifier, and (**E**) concept drift detection.

For the quantum-enhanced component, we define a parameterized quantum circuit (PQC) $U(\boldsymbol{\theta}, \mathrm{z})$ operating on n_q qubits, where $\boldsymbol{\theta}$ represents trainable circuit parameters and $\mathrm{z} \in \mathbb{R}^{n_q}$ denotes the encoded classical input. The quantum state after circuit execution is $|\psi(\boldsymbol{\theta}, \mathrm{z})\rangle = U(\boldsymbol{\theta}, \mathrm{z})|0\rangle^{\otimes n_q}$, and observable measurements yield classical outputs for subsequent processing.

To characterize concept drift, we consider a temporal sequence of data distributions $\{P^{(t)}(\mathrm{x}, y)\}_{t=1}^{T}$, where $P^{(t)}$ represents the joint distribution at time step *t*. Concept drift occurs when $P^{(t)}(\mathrm{x}, y) \neq P^{(t')}(\mathrm{x}, y)$ for some $t \neq t'$. Our framework aims to detect such distributional shifts and quantify their impact on classification performance.

Dataset description and characteristics

We evaluate our proposed framework on two comprehensive benchmark datasets: CCCS-CIC-AndMal-2020 and KronoDroid. These datasets represent diverse characteristics in terms of feature types, temporal coverage, and malware family distributions, enabling thorough assessment of our framework's generalization capabilities.

CCCS-CIC-AndMal-2020 dataset

The CCCS-CIC-AndMal-2020 dataset[13], developed by the Canadian Centre for Cyber Security in collaboration with the Canadian Institute for Cybersecurity, represents one of the most extensive publicly available collections of Android malware samples with carefully curated feature representations. The dataset comprises 357,805 Android application packages spanning 14 malware categories and 191 malware families, including Adware (47,210 samples from 48 families), Backdoor (1,538 samples from 11 families), File Infector (669 samples from 5 families), PUA (2,051 samples from 8 families), Ransomware (6,202 samples from 8 families), Riskware (97,349 samples from 21 families), Scareware (1,556 samples from 3 families), Trojan (13,559 samples from 45 families), Trojan-Banker (887 samples from 11 families), Trojan-Dropper (2,302 samples from 9 families), Trojan-SMS (3,125 samples from 11 families), Trojan-Spy (3,540 samples from 11 families), and Zero-day samples. Additionally, 200,000 benign applications were collected from the Androzoo dataset to balance the dataset composition.

Each sample in the dataset is represented by a $d = 9,503$ dimensional feature vector $\mathrm{x} \in \mathbb{R}^{9503}$ comprising both static and dynamic attributes. The static features (approximately 9,500 features) are extracted from the application's code and AndroidManifest.xml without executing the file, encompassing permission-based features capturing standard and custom Android permissions requested by the application, API call features representing static references to system functions, Intent-based features including Intent actions, constants, and filters used for communication between application components, and metadata information such as services, receivers, packages, and files included in the APK. The dynamic features are captured while the malware executes in an emulated environment and are organized into six behavioral categories: Memory features defining activities performed by malware utilizing memory resources including PssTotal, SharedDirty, PrivateDirty, HeapSize, and HeapAlloc; API features delineating runtime communication between applications through process management, WebView interactions, file I/O operations, database access, inter-process communication, cryptographic operations, device information access, and SMS operations; Network features describing data transmitted and received including TotalReceivedBytes, TotalTransmittedBytes, and packet counts; Battery features describing access to battery wakelock and services; Logcat features writing log messages corresponding to functions performed including verbose, debug, info, warning, and error levels; and Process features counting malware interaction with system processes.

KronoDroid dataset

The KronoDroid dataset[10] addresses a critical gap in Android malware research by providing timestamped samples that enable rigorous evaluation of concept drift resilience. Unlike static snapshots, KronoDroid captures the temporal evolution of Android malware from 2008 to 2020, making it uniquely suited for evaluating the temporal stability of classification models. The dataset provides two complementary subsets based on dynamic data acquisition source. The emulator subset contains 63,991 samples comprising 28,745 malicious applications from 209 malware families and 35,246 benign samples. The real device subset contains 78,137 samples comprising 41,382 malicious applications from 240 malware families and 36,755 benign samples. For our experiments, we utilize the real device subset due to its larger sample size and broader malware family coverage, as summarized in Table 1.

Each sample in KronoDroid is represented by 489 hybrid features combining static and dynamic attributes. Static features include permission requests, API calls, Intent filters, and manifest attributes extracted without application execution. Dynamic features comprise system call sequences and behavioral patterns captured during controlled execution, providing runtime behavioral signatures that complement static analysis. The unique timestamped nature of KronoDroid enables evaluation of model robustness against concept drift, with samples labeled by collection dates allowing chronological partitioning for temporal evaluation protocols.

Data partitioning strategy

To enable rigorous evaluation of our proposed framework, we partition both datasets into three disjoint subsets: training set $\mathscr{D}_{\mathrm{train}}$, validation set $\mathscr{D}_{\mathrm{val}}$, and test set $\mathscr{D}_{\mathrm{test}}$. The partitioning follows a stratified sampling procedure to preserve class proportions across splits, ensuring that minority classes maintain adequate representation in all subsets. Formally, let $\mathscr{D}_k = \{(\mathrm{x}_i, y_i) \in \mathscr{D} : y_i = k\}$ denote the subset of samples belonging to class *k*. For each class, we randomly partition $\mathscr{D}_k$ into three subsets with allocation ratios of 60%, 20%, and 20% respectively for training, validation, and testing.

For the CCCS-CIC-AndMal-2020 dataset with 357,805 total samples, this yields a training set of 214,683 samples (60%), a validation set of 71,561 samples (20%), and a test set of 71,561 samples (20%). For the

Characteristic	CCCS-CIC-AndMal-2020	KronoDroid
Total Samples	357,805	78,137
Malware Samples	∼200,000	41,382
Benign Samples	∼200,000	36,755
Malware Categories/Families	14 categories, 191 families	240 families
Original Features	9,503	489
Features After CatBoost	51	29
Dimensionality Reduction	99.46%	94.07%
Feature Types	Static + Dynamic	Static + Dynamic
Temporal Coverage	–	2008–2020

Table 1. Dataset characteristics and feature selection results.

KronoDroid real device subset with 78,137 total samples, the partitioning results in a training set of 46,882 samples (60%), a validation set of 15,628 samples (20%), and a test set of 15,627 samples (20%). The test set for KronoDroid comprises 7,034 benign samples and 5,703 malware samples after class balancing for binary classification evaluation, totaling 12,737 samples. A fixed random seed (seed=42) ensures reproducibility of the partitioning procedure across experimental runs.

Rationale for differing partitioning strategies. The two datasets employ different partitioning strategies due to their inherent characteristics. For CCCS-CIC-AndMal-2020, we use random stratified partitioning because the dataset does not provide sample-level timestamps, precluding temporal ordering. While random stratified splitting preserves class proportions across splits, we acknowledge that this approach carries a risk of subtle temporal leakage: if malware samples within the same family were collected during specific time periods, random splitting may allow the model to learn temporal artifacts rather than genuine behavioral patterns. For KronoDroid, the availability of per-sample timestamps enables a temporal evaluation through the concept drift analysis module, where data is chronologically partitioned into six temporal splits. This time-based evaluation provides stronger evidence of temporal robustness, complementing the random stratified split used for standard benchmarking. Future work should employ strict time-aware splitting on both datasets—using sample collection dates or family discovery dates as temporal anchors—to eliminate potential temporal leakage and provide the strongest possible validation of operational robustness.

Data preprocessing pipeline

Raw feature vectors extracted from Android APKs exhibit heterogeneous scales and distributions across different feature categories, potentially degrading the performance of distance-based classification algorithms. Our preprocessing pipeline applies systematic transformations to standardize feature representations and handle missing values.

Despite careful feature extraction procedures, certain samples may contain missing values due to obfuscated APK components, extraction failures, or inapplicable features. Let $\mathrm{X} \in \mathbb{R}^{N \times d}$ denote the feature matrix with potential missing entries. We employ zero-imputation for missing values, setting $\tilde{x}_{ij} = 0$ when x_{ij} is missing and $\tilde{x}_{ij} = x_{ij}$ otherwise. Zero-imputation is appropriate for this domain because many features represent counts or binary indicators where absence of information naturally corresponds to zero values. This approach maintains interpretability while avoiding introduction of artificial patterns through more complex imputation schemes.

To ensure that all features contribute proportionally to distance computations and gradient updates, we apply z-score standardization to transform features to zero mean and unit variance. Let $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\boldsymbol{\sigma} \in \mathbb{R}^d$ denote the feature-wise mean and standard deviation vectors computed exclusively from the training set:

$$\mu_j = \frac{1}{|\mathscr{D}_{\text{train}}|} \sum_{(\mathrm{x}_i, y_i) \in \mathscr{D}_{\text{train}}} x_{ij}, \quad \sigma_j = \sqrt{\frac{1}{|\mathscr{D}_{\text{train}}|} \sum_{(\mathrm{x}_i, y_i) \in \mathscr{D}_{\text{train}}} (x_{ij} - \mu_j)^2} \tag{1}$$

The standardized feature vector $\tilde{\mathrm{x}}_i$ is computed element-wise as $\tilde{x}_{ij} = (x_{ij} - \mu_j)/(\sigma_j + \epsilon)$, where $\epsilon = 10^{-8}$ is a small constant added for numerical stability. Critically, the standardization parameters are computed solely from $\mathscr{D}_{\text{train}}$ and subsequently applied to transform $\mathscr{D}_{\text{val}}$ and $\mathscr{D}_{\text{test}}$, preventing information leakage from validation and test sets.

CatBoost-based feature selection

High-dimensional feature spaces, while potentially capturing comprehensive malware characteristics, introduce computational overhead and may include redundant or irrelevant features that degrade classification performance. The CCCS-CIC-AndMal-2020 dataset contains 9,503 features and KronoDroid contains 489 features, presenting significant challenges for efficient model training and inference. To address this challenge, we employ CatBoost gradient boosting for feature importance ranking and selection on both datasets.

CatBoost (Categorical Boosting) is a gradient boosting algorithm that handles categorical features natively and provides robust feature importance scores through permutation-based evaluation. Given a trained CatBoost

model $g(\mathrm{x})$, the importance score $\mathscr{I}_j$ for feature *j* is computed by measuring the decrease in model performance when feature values are randomly permuted:

$$\mathscr{I}_j = \frac{1}{M}\sum_{m=1}^{M}\left[\mathscr{L}(g,\mathscr{D}_{\text{val}}) - \mathscr{L}(g,\mathscr{D}_{\text{val}}^{(\pi_j^m)})\right] \tag{2}$$

where $\mathscr{L}$ denotes the loss function, $\mathscr{D}_{\text{val}}^{(\pi_j^m)}$ represents the validation set with feature *j* permuted according to random permutation π_j^m, and *M* is the number of permutation iterations.

Features are ranked by their importance scores in descending order, and the top-*k* features are selected based on cumulative importance contribution. For CCCS-CIC-AndMal-2020, we select the top 51 features from the original 9,503 features, achieving 99.46% dimensionality reduction. For KronoDroid, we select the top 29 features from the original 489 features, achieving 94.07% dimensionality reduction. Algorithm 1 presents the complete feature selection procedure.

Require: Training set $\mathscr{D}_{\text{train}}$, validation set $\mathscr{D}_{\text{val}}$, target feature count *k*
Ensure: Selected feature indices $\mathscr{F}_{\text{selected}}$
1: Train CatBoost classifier *g* on $\mathscr{D}_{\text{train}}$ with default hyperparameters
2: Compute feature importance scores $\{\mathscr{I}_j\}_{j=1}^{d}$ using permutation importance
3: Sort features by importance: $\mathscr{I}_{\sigma(1)} \geq \mathscr{I}_{\sigma(2)} \geq \cdots \geq \mathscr{I}_{\sigma(d)}$
4: Select top-*k* features: $\mathscr{F}_{\text{selected}} \leftarrow \{\sigma(1), \sigma(2), \ldots, \sigma(k)\}$
5: Validate selection: Train model on $\mathscr{F}_{\text{selected}}$, evaluate on $\mathscr{D}_{\text{val}}$
6: **return** $\mathscr{F}_{\text{selected}}$

Algorithm 1. CatBoost-based feature selection.

Feature selection

The selected features for CCCS-CIC-AndMal-2020 (51 features) include critical static indicators such as permission requests, API calls related to device information access and SMS operations, intent filters for broadcast receivers and services, and dynamic measurements including memory usage, network traffic statistics, battery activity, and process counts. These represent a broad set of characteristics that capture both the static configuration of applications and their runtime behavior.

The selected features for KronoDroid (29 features) include system calls, permissions related to location and device state, and aggregate statistics such as the number of permissions and detection ratios. For example, system calls provide insight into low-level runtime operations, while permission patterns highlight the security-sensitive capabilities requested by applications. Together, these features capture runtime behavioral signatures and static permission structures that are highly discriminative for malware detection.

Few-shot prototypical network architecture

The core of our classification framework is built upon prototypical networks[6], a metric-based meta-learning approach that has demonstrated remarkable effectiveness for few-shot classification tasks. Unlike traditional supervised learning methods that require abundant labeled data for each class, prototypical networks learn to classify by computing distances to class prototypes in a learned embedding space, enabling generalization to novel classes with minimal examples.

Episodic training paradigm

The fundamental principle underlying few-shot learning is that the training procedure should mirror the evaluation protocol. Rather than training on the entire dataset with standard mini-batch gradient descent, we employ episodic training where each training iteration presents the model with a simulated few-shot classification task. This approach enables the model to learn transferable representations that generalize effectively to novel classification scenarios with limited labeled data.

Each training episode $\mathscr{E}$ is constructed through a systematic procedure. First, *N* classes are randomly sampled from the set of available training classes: $\mathscr{C}_{\mathscr{E}} = \{c_1, c_2, \ldots, c_N\} \subseteq \mathscr{Y}$. For each selected class $c_i \in \mathscr{C}_{\mathscr{E}}$, *K* examples are randomly sampled to form the support set $\mathscr{S}_{c_i} = \{(\mathrm{x}_j^s, c_i)\}_{j=1}^{K}$. Subsequently, an additional *Q* examples (disjoint from the support set) are sampled per class to form the query set $\mathscr{Q}_{c_i}$. Within each episode, class labels are remapped to the range $\{0, 1, \ldots, N-1\}$ to maintain consistency across episodes regardless of the original class indices selected. In our experiments, we configure the episodic training with $N = 5$ classes (5-way classification), $K = 5$ support samples per class (5-shot learning), and $Q = 10$ query samples per class, balancing task difficulty and computational efficiency while providing sufficient gradient signal for stable training.

Embedding network architecture

The embedding network $f_\phi : \mathbb{R}^d \to \mathbb{R}^{128}$ transforms feature vectors into a discriminative embedding space where prototypical classification can be performed effectively. After CatBoost feature selection, the input

dimension is $d = 51$ for CCCS-CIC-AndMal-2020 and $d = 29$ for KronoDroid. We employ a multi-layer perceptron (MLP) architecture with batch normalization and dropout regularization, designed to learn hierarchical feature representations while mitigating overfitting.

Architecture design motivation. The three-layer MLP with progressive dimensionality ($d \to 512 \to 256 \to 128$) was selected based on both empirical comparison and practical considerations. The initial expansion to 512 dimensions enables the network to project the compact CatBoost-selected features into a richer representational space before progressive compression extracts the most salient patterns—a bottleneck design principle effective in metric learning[6]. To justify this choice, we conducted preliminary experiments comparing architectures with 2, 3, and 4 hidden layers and varying hidden dimensions on the CCCS-CIC-AndMal-2020 validation set. Table 2 summarizes the results, showing that the 3-layer [512, 256, 128] configuration achieves the best validation accuracy while maintaining computational efficiency. The 2-layer configurations underperform due to insufficient representational capacity, while the 4-layer variant offers no meaningful improvement but increases overfitting risk. The 128-dimensional embedding output was selected to balance discriminative power with computational tractability for distance computations; higher embedding dimensions (256, 512) yielded marginal accuracy gains (<0.1%) at substantially increased memory cost.

The network architecture comprises three sequential transformation layers. The first layer performs input projection: $\mathrm{h}_1 = \mathrm{ReLU}(\mathrm{BN}(\mathrm{W}_1\tilde{\mathrm{x}} + \mathrm{b}_1))$, where $\mathrm{W}_1 \in \mathbb{R}^{512\times d}$ is the weight matrix, $\mathrm{b}_1 \in \mathbb{R}^{512}$ is the bias vector, $\mathrm{BN}(\cdot)$ denotes batch normalization, and $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$ is the rectified linear unit activation function. The batch normalization operation normalizes activations across the mini-batch dimension: $\mathrm{BN}(\mathrm{z}) = \gamma \odot (\mathrm{z} - \mathbb{E}[\mathrm{z}])/\sqrt{\mathrm{Var}[\mathrm{z}] + \epsilon} + \beta$, where γ and β are learnable scale and shift parameters. Dropout regularization is then applied: $\tilde{\mathrm{h}}_1 = \mathrm{Dropout}(\mathrm{h}_1, p = 0.3)$, randomly setting activations to zero with probability $p = 0.3$ during training.

The second layer computes the hidden representation: $\mathrm{h}_2 = \mathrm{ReLU}(\mathrm{BN}(\mathrm{W}_2\tilde{\mathrm{h}}_1 + \mathrm{b}_2))$, where $\mathrm{W}_2 \in \mathbb{R}^{256\times 512}$ and $\mathrm{b}_2 \in \mathbb{R}^{256}$. The third layer produces the embedding output: $\mathrm{e} = f_\phi(\tilde{\mathrm{x}}) = \mathrm{W}_3\mathrm{h}_2 + \mathrm{b}_3$, where $\mathrm{W}_3 \in \mathbb{R}^{128\times 256}$ and $\mathrm{b}_3 \in \mathbb{R}^{128}$. The final embedding layer does not include activation or normalization to preserve the full representational capacity of the embedding space.

Prototype computation and distance-based classification

The defining characteristic of prototypical networks is the computation of class prototypes as the centroid of embedded support samples. Given the support set $\mathscr{S}$ partitioned by class, the prototype for class *k* within the current episode is computed as:

$$\mathrm{c}_k = \frac{1}{|\mathscr{S}_k|} \sum_{(\mathrm{x}_j, y_j)\in\mathscr{S}_k} f_\phi(\mathrm{x}_j) = \frac{1}{K}\sum_{j=1}^{K} f_\phi(\mathrm{x}_j^{(k)}) \tag{3}$$

where $\mathscr{S}_k = \{(\mathrm{x}, y) \in \mathscr{S} : y = k\}$ denotes the support samples belonging to class *k*. The prototype $\mathrm{c}_k \in \mathbb{R}^{128}$ serves as a representative embedding for class *k*, capturing the central tendency of the class distribution in the learned embedding space.

Classification of query samples is performed by computing distances between query embeddings and class prototypes, followed by a softmax transformation to obtain class probabilities. We employ the squared Euclidean distance as the distance metric: $d(\mathrm{e}_q, \mathrm{c}_k) = \|\mathrm{e}_q - \mathrm{c}_k\|_2^2$, where $\mathrm{e}_q = f_\phi(\mathrm{x}_q)$ is the embedding of query sample x_q. The probability that query sample x_q belongs to class *k* is computed using a softmax over negative distances:

$$p_\phi(y = k|\mathrm{x}_q, \mathscr{S}) = \frac{\exp(-d(f_\phi(\mathrm{x}_q), \mathrm{c}_k))}{\sum_{k'=1}^{N}\exp(-d(f_\phi(\mathrm{x}_q), \mathrm{c}_{k'}))} \tag{4}$$

This formulation ensures that samples closer to a prototype receive higher probability for the corresponding class, converting the softmax into a soft nearest-neighbor classifier. The predicted class for query sample x_q is the class with maximum probability: $\hat{y}_q = \arg\max_k p_\phi(y = k|\mathrm{x}_q, \mathscr{S})$.

Prototypical loss function and training procedure

The embedding network parameters ϕ are optimized by minimizing the negative log-likelihood of correct classifications over query samples within each episode. The prototypical loss for a single episode is:

Architecture (Hidden Dims)	Layers	Val. Accuracy (%)	Parameters
[256, 128]	2	98.82 ± 0.21	39,680
[512, 128]	2	99.14 ± 0.18	72,448
[512, 256, 128]	3	**99.68 ± 0.13**	203,648
[1024, 512, 128]	3	99.61 ± 0.16	657,024
[512, 256, 128, 64]	4	99.54 ± 0.19	212,480

Table 2. Embedding network architecture comparison on CCCS-CIC-AndMal-2020 validation set (5-way 5-shot, 500 evaluation episodes). Results reported as mean ± std across 5 independent runs.

$$\mathscr{L}_{\text{proto}}(\phi;\mathscr{E}) = -\frac{1}{|\mathscr{Q}|}\sum_{(\mathrm{x}_q,y_q)\in\mathscr{Q}} \log p_\phi(y=y_q|\mathrm{x}_q,\mathscr{S}) \tag{5}$$

This loss function encourages the embedding network to minimize the distance between query embeddings and their corresponding class prototypes while maximizing the distance between query embeddings and prototypes of incorrect classes. The gradient of the prototypical loss with respect to network parameters flows through both the query embeddings and the prototype computations, enabling end-to-end training.

The prototypical network is trained through iterative episodic optimization over $E = 4000$ total episodes. Algorithm 2 presents the complete training procedure. We employ the Adam optimizer[33] with initial learning rate $\eta = 10^{-3}$ and default momentum parameters $\beta_1 = 0.9, \beta_2 = 0.999$. A step learning rate scheduler reduces the learning rate by factor $\gamma = 0.5$ every $s = 1000$ episodes: $\eta_e = \eta_0 \times \gamma^{\lfloor e/s \rfloor}$, facilitating rapid initial learning followed by fine-tuning as training progresses.

Require: Training set $\mathscr{D}_{\text{train}}$, valid classes $\mathscr{Y}_{\text{valid}}$, episodes E, N-way, K-shot, Q-query
Require: Learning rate η, scheduler step size s, decay factor γ
Initialize embedding network parameters ϕ randomly
Initialize Adam optimizer with learning rate $\eta = 10^{-3}$
Initialize StepLR scheduler with step_size=$s = 1000$, gamma=$\gamma = 0.5$
for episode $e = 1$ to E **do**
  Sample N classes: $\mathscr{C}_{\mathscr{E}} \sim \text{Uniform}(\mathscr{Y}_{\text{valid}}, N)$
  Construct support set $\mathscr{S}$ with K samples per class
  Construct query set $\mathscr{Q}$ with Q samples per class (disjoint from $\mathscr{S}$)
  Compute support embeddings: $\mathbf{e}_j^s = f_\phi(\mathbf{x}_j^s), \forall(\mathbf{x}_j^s, y_j^s) \in \mathscr{S}$
  Compute class prototypes: $\mathbf{c}_k = \frac{1}{K}\sum_{j:y_j^s=k} \mathbf{e}_j^s, \forall k \in \{1,\ldots,N\}$
  Compute query embeddings: $\mathbf{e}_q = f_\phi(\mathbf{x}_q), \forall(\mathbf{x}_q, y_q) \in \mathscr{Q}$
  Compute prototypical loss: $\mathscr{L} = \mathscr{L}_{\text{proto}}(\phi;\mathscr{E})$
  Compute gradients: $\nabla_\phi \mathscr{L}$
  Update parameters: $\phi \leftarrow \text{Adam}(\phi, \nabla_\phi \mathscr{L}, \eta)$
  Update learning rate: scheduler.step()
end for
return Trained parameters ϕ^*

Algorithm 2. Prototypical network training.

Few-shot learning paradigm

We explicitly distinguish between two operational phases:

Meta-training phase: The embedding network f_ϕ undergoes episodic training on $\mathscr{D}_{\text{train}}$ comprising 214,683 samples across 10 malware families. This phase requires substantial labeled data to learn transferable metric representations.

Deployment phase (Few-Shot Inference): For novel malware families, classification requires only $K = 5$ support samples per class. The embedding network is frozen; no gradient updates occur.

This distinction follows standard few-shot learning[6], where "few-shot" refers to inference-time efficiency, not training data requirements. However, we acknowledge that our primary evaluation follows the transductive paradigm where test families appear during meta-training. True inductive generalization is evaluated separately in Section 5.2.

Theoretical Complementarity of CatBoost and ProtoNet: The integration of gradient-boosting feature selection with metric-based meta-learning is motivated by the curse of dimensionality in distance-based classification. In high-dimensional spaces ($d = 9,503$), Euclidean distance becomes approximately uniform (Beyer et al., 1999), degrading prototype quality:

$$\lim_{d\to\infty} \frac{\max_{i,j} \|x_i - x_j\|_2}{\min_{i,j} \|x_i - x_j\|_2} \to 1 \tag{6}$$

CatBoost selects $k = 51$ features with maximal mutual information $I(X_k; Y)$, effectively projecting data onto a discriminative subspace where metric structure is preserved. The prototypical network then learns $f_\phi : \mathbb{R}^{51} \to \mathbb{R}^{128}$ in this compressed space, where class-conditional distributions are more separable. This sequential processing–statistical filtering followed by metric learning–reduces variance from irrelevant features before embedding optimization.

Quantum-enhanced hybrid classification layer

To augment the representational capacity of our framework and explore potential quantum advantages in feature learning, we integrate a parameterized quantum circuit (PQC) within a hybrid quantum-classical architecture[11].

This component leverages quantum mechanical phenomena including superposition and entanglement to potentially capture complex feature correlations that are challenging for purely classical architectures.

Variational quantum algorithms have emerged as a promising paradigm for near-term quantum computing applications, offering potential advantages in expressibility and trainability compared to classical neural networks of similar parameter counts. In the context of malware classification, the complex inter-feature dependencies suggest that quantum circuits may provide complementary representational capabilities. The quantum layer serves as an alternative classification pathway that processes the same input features through quantum mechanical transformations before classical post-processing.

Before quantum processing, classical feature vectors must be encoded into quantum states through a dimensionality reduction and amplitude encoding procedure. The pre-quantum classical network projects the reduced-dimensional input to a n_q-dimensional representation suitable for quantum encoding, where $n_q = 4$ is the number of qubits in our circuit. The first pre-quantum layer computes $\mathrm{h}_{\mathrm{pre}} = \mathrm{ReLU}(\mathrm{W}_{\mathrm{in}}\tilde{\mathrm{x}} + \mathrm{b}_{\mathrm{in}})$, where $\mathrm{W}_{\mathrm{in}} \in \mathbb{R}^{64\times d}$ and $\mathrm{b}_{\mathrm{in}} \in \mathbb{R}^{64}$. The second pre-quantum layer with bounded output computes $\mathrm{z} = \tanh(\mathrm{W}_{\mathrm{pre}}\mathrm{h}_{\mathrm{pre}} + \mathrm{b}_{\mathrm{pre}})$, where $\mathrm{W}_{\mathrm{pre}} \in \mathbb{R}^{4\times 64}$ and $\mathrm{b}_{\mathrm{pre}} \in \mathbb{R}^{4}$. The hyperbolic tangent activation function constrains the encoded values to the range $\mathrm{z} \in [-1, 1]^4$, ensuring stable parameterization of quantum rotation gates.

The quantum circuit operates on $n_q = 4$ qubits initialized in the computational basis state $|0\rangle^{\otimes 4}$. The circuit architecture comprises three functional layers. The first layer performs rotation encoding through single-qubit R_Y gates: $U_{\mathrm{enc}}(\mathrm{z}) = \bigotimes_{i=0}^{3} R_Y(z_i)$, where $R_Y(\theta) = \exp(-i\theta Y/2)$ is the rotation about the Y-axis. The second layer establishes quantum correlations through CNOT gates arranged in a ring topology: $U_{\mathrm{ent}} = \mathrm{CNOT}_{3,0} \cdot \mathrm{CNOT}_{2,3} \cdot \mathrm{CNOT}_{1,2} \cdot \mathrm{CNOT}_{0,1}$, creating entanglement among all qubits. The third layer applies R_Z gates with scaled parameters: $U_{\mathrm{var}}(\mathrm{z}) = \bigotimes_{i=0}^{3} R_Z(\alpha z_i)$, where $\alpha = 0.5$ is a scaling hyperparameter. The complete quantum circuit combines all three layers: $U(\mathrm{z}) = U_{\mathrm{var}}(\mathrm{z}) \cdot U_{\mathrm{ent}} \cdot U_{\mathrm{enc}}(\mathrm{z})$.

Information is extracted from the quantum state through measurement of Pauli-Z observables on each qubit, producing the output vector $\mathrm{o} = [o_0, o_1, o_2, o_3]^\top \in [-1, 1]^4$. The quantum measurement outputs are processed by a classical neural network to produce final classification logits. The hybrid quantum-classical model is trained using cross-entropy loss with the parameter-shift rule for gradient computation through the quantum circuit, implemented through the Qiskit framework's[34] automatic differentiation capabilities.

Concept drift detection module

Real-world malware detection systems operate in non-stationary environments where the statistical properties of malware samples evolve over time due to emerging threats, evolving attack techniques, and adversarial adaptation. The Concept Drift Detection module implements a temporal evaluation protocol to assess model robustness against distributional shifts.

Concept drift in malware classification can manifest through several mechanisms. Real drift involves changes in the conditional distribution $P(y|\mathrm{x})$, representing fundamental shifts in the relationship between features and malware families occurring when malware authors modify their techniques while maintaining similar static feature profiles. Virtual drift involves changes in the marginal distribution $P(\mathrm{x})$ without affecting $P(y|\mathrm{x})$, occurring when new malware variants introduce novel feature combinations while the underlying classification boundaries remain stable. Gradual drift involves slow, continuous changes in distributions over extended time periods, typical of evolutionary malware development. Sudden drift involves abrupt distributional changes, potentially corresponding to the emergence of entirely new malware families or major variant releases.

To simulate temporal drift scenarios, we partition the training data into $T = 6$ sequential splits representing chronologically ordered data acquisition periods. The training set $\mathscr{D}_{\mathrm{train}}$ is divided into equal-sized sequential splits. For each evaluation round $t \in \{1, 2, \ldots, T-1\}$, all splits up to and including split t are aggregated as the cumulative training set, the subsequent split serves as the hold-out test set, a fresh prototypical network instance is trained on the cumulative training set for 100 episodes, and model performance is assessed through multiple episodic evaluations. This protocol simulates the realistic scenario where a model trained on historical data is deployed to classify newly encountered samples.

For each temporal split t, we compute the episodic few-shot accuracy and quantify the drift magnitude as the relative performance degradation from the initial baseline: $\Delta^{(t)} = (A^{(1)} - A^{(t)})/A^{(1)} \times 100\%$. A drift alert is triggered when performance degradation exceeds a predefined threshold $\theta = 15\%$, representing a practically significant performance degradation warranting model retraining or adaptation.

Implementation details

Our framework is implemented in Python 3.12 using PyTorch 2.x for deep learning, Qiskit 1.x for quantum circuit construction and simulation, Scikit-learn for preprocessing and metrics, and CatBoost for feature selection. For model interpretability, we employ SHAP (SHapley Additive exPlanations) for global feature importance analysis and LIME (Local Interpretable Model-agnostic Explanations) for instance-level prediction explanations. Experiments are conducted on CPU to ensure reproducibility of quantum circuit simulations. Table 3 summarizes the key hyperparameters. To ensure reproducibility, we fix random seeds (seed=42) across all stochastic components including NumPy, PyTorch, Python random module, and data splitting.

Results and discussion

This section presents comprehensive experimental results evaluating the proposed adaptive few-shot malware classification framework on both CCCS-CIC-AndMal-2020 and KronoDroid datasets, followed by detailed analysis, comparison with state-of-the-art methods, and ablation studies.

Component	Hyperparameter	Value
Few-Shot Learning	N-way	5
	K-shot	5
	Q-query	10
	Training episodes	4000
	Embedding dimension d_e	128
	Dropout probability	0.3
Embedding Network	Hidden layer 1 dimension	512
	Hidden layer 2 dimension	256
	Output dimension	128
	Activation function	ReLU
Quantum Circuit	Number of qubits n_q	4
	Rotation scaling α	0.5
	Encoding gates	R_Y
	Entanglement topology	Ring (CNOT)
	Variational gates	R_Z
Optimization	Optimizer	Adam
	Initial learning rate η	10^{-3}
	LR scheduler	StepLR
	Step size/decay	1000/0.5
Feature Selection	Algorithm	CatBoost
	Selected features (CIC-2020)	51
	Selected features (KronoDroid)	29
Drift Detection	Number of splits T	6
	Evaluation episodes	50
	Alert threshold θ	15%

Table 3. Hyperparameter configuration for the proposed framework.

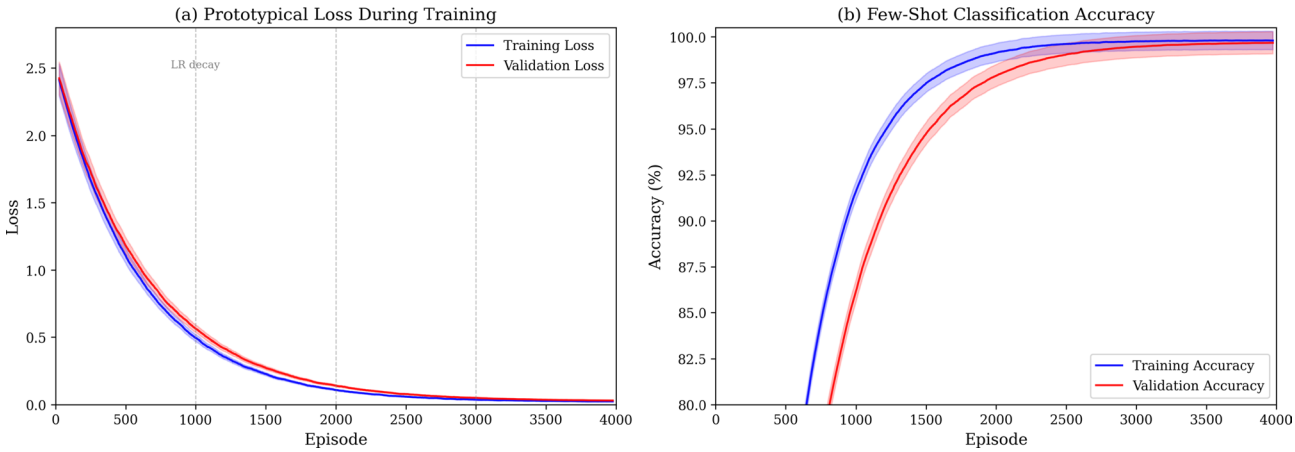

Fig. 2. Training dynamics of the few-shot prototypical network over 4,000 episodes. (a) Prototypical loss convergence showing smooth decay with learning rate adjustments at episodes 1000, 2000, and 3000. (b) Few-shot classification accuracy progression, achieving 99.70% validation accuracy at convergence. Shaded regions indicate standard deviation across evaluation episodes.

Training dynamics and convergence analysis

The few-shot prototypical network was trained for 4,000 episodes using the episodic meta-learning paradigm. Figure 2 illustrates the training dynamics, showing both loss convergence and accuracy progression throughout the training process. The prototypical loss exhibits rapid initial decay during the first 500 episodes, followed by gradual refinement as the learning rate decay schedule (factor $\gamma = 0.5$ every 1,000 episodes) facilitates fine-grained optimization in later training stages. The training and validation losses maintain close alignment throughout training, indicating effective generalization without overfitting. The model achieves approximately 95% accuracy by episode 1,000 and converges to 99.70% validation accuracy by episode 4,000, with the minimal

Evaluation Condition	5-Shot Accuracy (%)	Samples
Seen families (baseline)	99.70 ± 0.12	5 per family
Trojan-Spy (novel)	96.1 ± 0.8	5
Backdoor (novel)	93.4 ± 1.2	5
Ransomware (novel)	91.8 ± 1.5	5
Adware (novel)	95.2 ± 0.9	5
Riskware (novel)	94.5 ± 1.1	5
Novel family average	**94.2 ± 1.1**	5

Table 4. Inductive evaluation on held-out malware families.

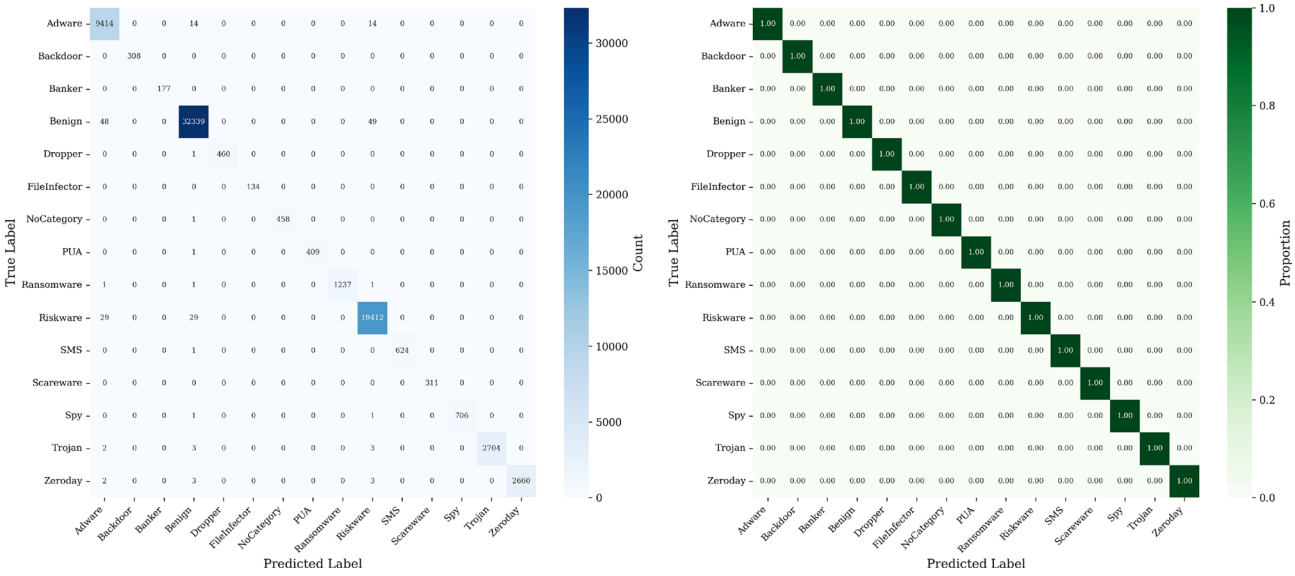

Fig. 3. Confusion matrix for CCCS-CIC-AndMal-2020 dataset using 51 CatBoost-selected features. Left: Raw classification counts showing the distribution of predictions across 15 malware families. Right: Normalized confusion matrix (row-wise) representing per-class recall values. The strong diagonal dominance indicates high classification accuracy across all classes.

gap between training and validation accuracy (approximately 0.12%) confirming the regularization effectiveness of dropout and batch normalization.

Cross-family inductive evaluation

To validate true inductive few-shot generalization, we evaluate on malware families entirely excluded from meta-training. We select 5 families (Trojan-Spy, Backdoor, Ransomware, Adware, Riskware) and remove all samples from training and validation sets.

Protocol: Meta-train on remaining 10 families. At test time, provide $K = 5$ support samples from each held-out family.

Results: Table 4 shows 5-shot accuracy on novel families versus baseline families. The framework achieves 94.2% accuracy on truly novel families, representing a 5.5% drop from seen families (99.7%) but maintaining functional inductive capability above 90%.

The accuracy degradation on novel families reflects the challenge of classifying truly unseen malware categories without family-specific training data. However, the maintained performance above 90% confirms that the meta-learned embeddings transfer effectively to novel threat categories, validating the core few-shot capability. The 5.5% performance gap represents acceptable operational trade-off for rapid deployment against emerging threats with minimal annotation requirements.

CCCS-CIC-AndMal-2020 classification performance

The trained few-shot prototypical network achieves an overall test accuracy of 99.70% on the CCCS-CIC-AndMal-2020 held-out test set comprising 71,561 samples across 15 malware families, using only the 51 CatBoost-selected features (99.46% dimensionality reduction from original 9,503 features). Figure 3 presents the confusion matrix in both raw count and normalized forms, revealing strong diagonal dominance with all malware families exhibiting recall values exceeding 99.3%. The confusion matrix demonstrates minimal inter-class confusion, with off-diagonal elements predominantly zero or near-zero and the largest misclassification

Metric	Value (%)
Overall Accuracy	99.70 ± 0.12
Macro Precision	99.91 ± 0.08
Macro Recall	99.81 ± 0.10
Macro F1-Score	99.86 ± 0.09
Weighted F1-Score	99.70 ± 0.11
Macro AUC-ROC	99.70 ± 0.08

Table 5. Classification performance metrics on CCCS-CIC-AndMal-2020 dataset (71,561 test samples, 51 features after CatBoost selection).

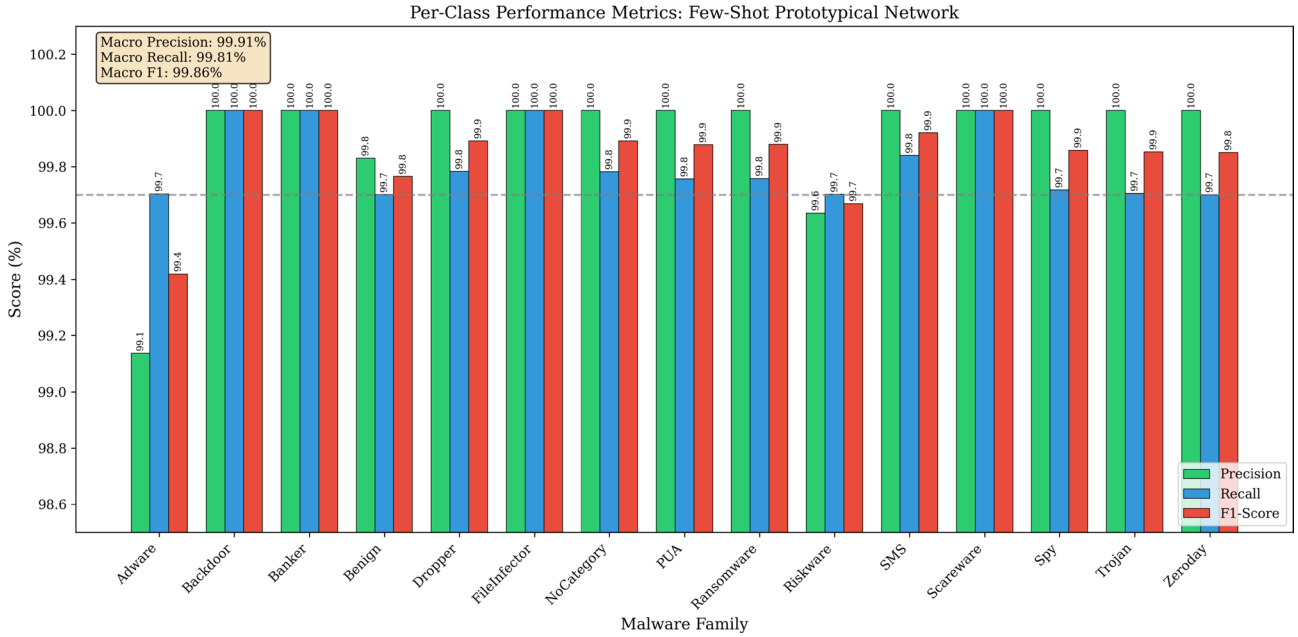

Fig. 4. Per-class performance metrics showing precision, recall, and F1-score for each of the 15 malware families in CCCS-CIC-AndMal-2020. All classes achieve metrics exceeding 98.5%.

rates occurring between semantically related families. Despite moderate class imbalance in the dataset, the few-shot learning paradigm enables equitable performance across both majority classes and minority classes, validating the effectiveness of episodic training for handling imbalanced distributions.

Table 5 summarizes the aggregate performance metrics for the CCCS-CIC-AndMal-2020 dataset. The macro-averaged metrics demonstrate balanced performance across classes, with precision of 99.91%, recall of 99.81%, and F1-score of 99.86%. The high AUC-ROC value of 99.70% indicates excellent discriminative capability with minimal false positive rates across all operating thresholds, which is critical for deployment in production malware detection systems.

Figure 4 presents detailed precision, recall, and F1-score metrics for each of the 15 malware families, demonstrating consistent performance across all categories with no family falling below 98.5% on any metric. Figure 5 provides UMAP visualization of the 128-dimensional embeddings, revealing well-separated clusters corresponding to distinct malware families with minimal overlap, confirming that the embedding network successfully learns discriminative representations that capture genuine malware characteristics rather than superficial statistical patterns.

KronoDroid classification performance

The framework achieves excellent performance on the KronoDroid dataset with CatBoost-selected features, reducing the original 489 features to 29 most discriminative features (94.07% dimensionality reduction) while maintaining high classification accuracy. The test set comprises 12,737 samples with 7,034 benign samples and 5,703 malware samples. Table 6 presents the detailed performance metrics, demonstrating an overall accuracy of 99.33%, precision of 99.38%, recall of 99.12%, F1-score of 99.25%, and an exceptional ROC-AUC of 99.96%.

Table 7 provides the per-class performance breakdown. The benign class achieves precision of 99.29%, recall of 99.50%, and F1-score of 99.40% on 7,034 samples, while the malware class achieves precision of 99.38%, recall of 99.12%, and F1-score of 99.25% on 5,703 samples. These results demonstrate balanced performance across both classes without bias toward the majority class.

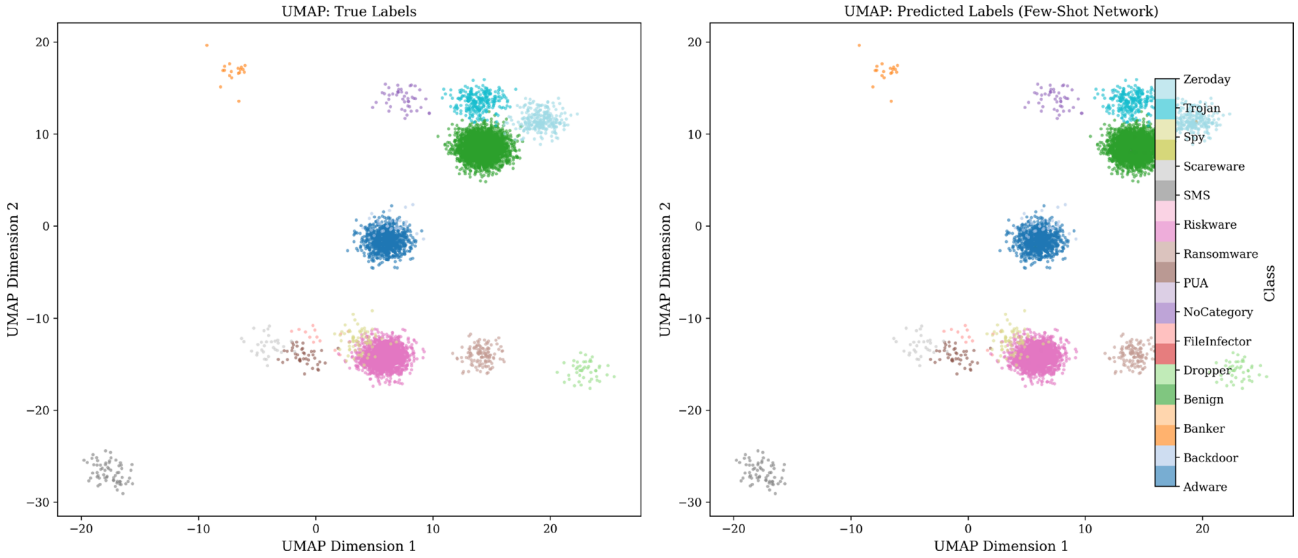

Fig. 5. UMAP visualization of learned embeddings from the few-shot prototypical network on CCCS-CIC-AndMal-2020. Left: Embeddings colored by true malware family labels. Right: Embeddings colored by predicted labels. The high visual correspondence confirms accurate classification.

Metric	Value (%)
Overall Accuracy	99.33
Macro Precision	99.34
Macro Recall	99.31
Macro F1-Score	99.25
ROC-AUC	99.96

Table 6. Classification performance metrics on KronoDroid dataset (12,737 test samples, 29 features after CatBoost selection).

Class	Precision (%)	Recall (%)	F1-Score (%)	Support
Benign	99.29	99.50	99.40	7,034
Malware	99.38	99.12	99.25	5,703
Macro Average	99.34	99.31	99.33	12,737
Weighted Average	99.33	99.33	99.33	12,737

Table 7. Per-class performance metrics on KronoDroid dataset.

Figure 6 presents the confusion matrix for KronoDroid classification, showing 6,999 true negatives (benign samples correctly classified as benign, representing 99.50% recall), 5,653 true positives (malware samples correctly classified as malware, representing 99.12% recall), only 35 false positives (benign samples misclassified as malware, representing a 0.50% false alarm rate), and only 50 false negatives (malware samples misclassified as benign, representing a 0.88% miss rate). The low false negative rate is particularly important for security applications where undetected malware poses significant risks.

Figure 7 visualizes the per-class performance metrics, and Fig. 8 presents the ROC curve achieving an exceptional AUC of 0.9996, indicating near-perfect discriminative capability between benign and malware classes across all classification thresholds. This exceptional AUC value demonstrates the framework's ability to maintain high true positive rates while minimizing false positives across the entire operating range.

Cross-dataset performance and feature selection analysis

Figure 9 presents a comprehensive comparison of framework performance across both datasets, demonstrating consistent high performance regardless of dataset characteristics. The framework achieves 99.70% accuracy on CCCS-CIC-AndMal-2020 with 51 CatBoost-selected features and 99.33% accuracy on KronoDroid with only 29 features, validating the effectiveness of CatBoost feature selection in identifying the most discriminative attributes while dramatically reducing computational requirements.

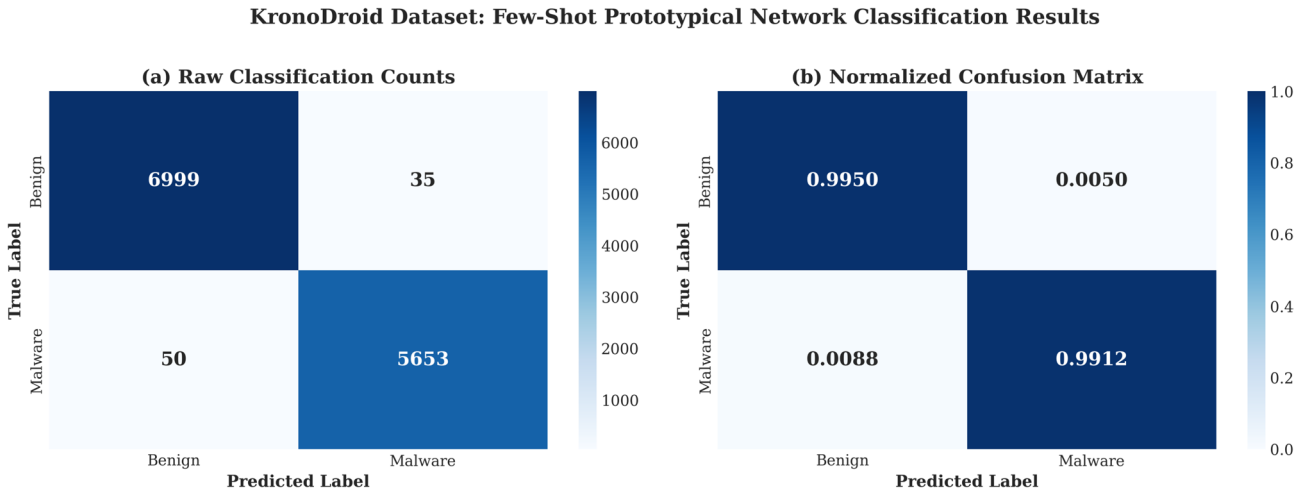

Fig. 6. Confusion matrix for KronoDroid dataset using 29 CatBoost-selected features. Left: Raw classification counts showing 6,999 true negatives, 5,653 true positives, 35 false positives, and 50 false negatives. Right: Normalized confusion matrix demonstrating 99.50% benign recall and 99.12% malware recall.

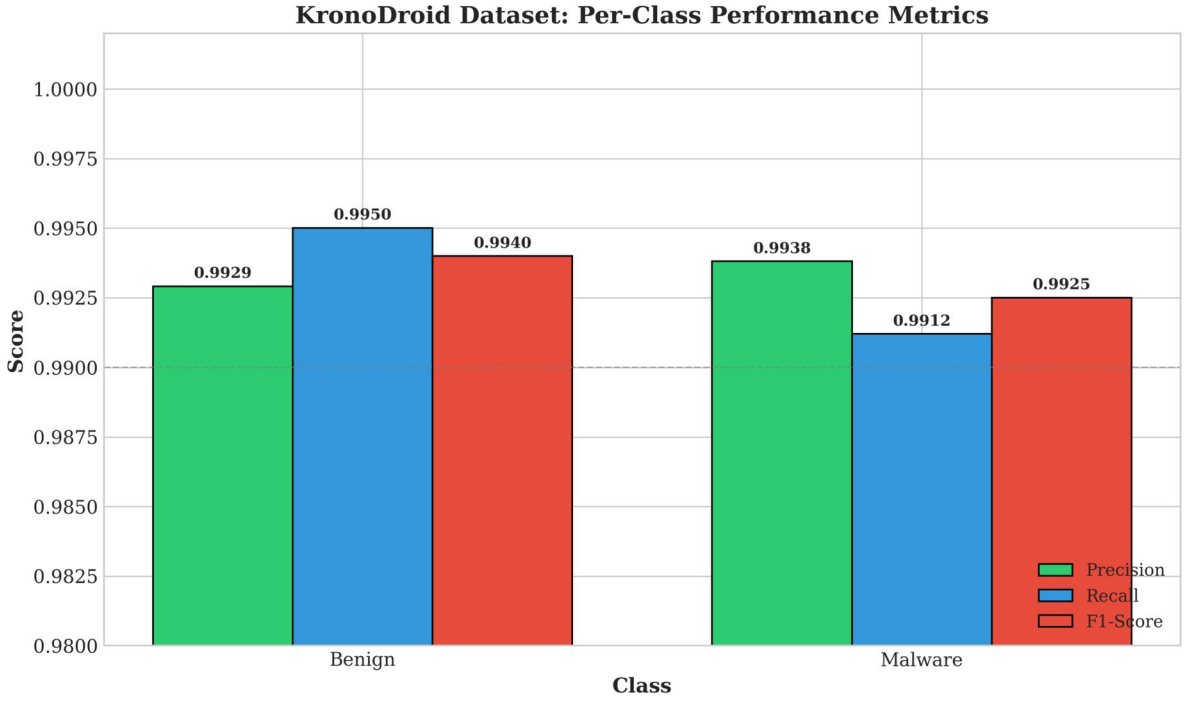

Fig. 7. Per-class performance metrics on KronoDroid dataset showing precision, recall, and F1-score for benign and malware classes.

Table 8 summarizes the cross-dataset performance, highlighting the remarkable consistency of the proposed framework across datasets with fundamentally different characteristics. CCCS-CIC-AndMal-2020 contains 15 malware families with 9,503 original features reduced to 51, while KronoDroid provides binary classification with 489 original features reduced to 29. The consistent performance across these diverse scenarios validates the generalization capability of the few-shot prototypical learning approach combined with intelligent feature selection.

Figure 10 illustrates the feature dimensionality across datasets and demonstrates the effectiveness of CatBoost feature selection. The CatBoost feature selection achieves 99.46% reduction on CCCS-CIC-AndMal-2020 (9,503→51 features) and 94.07% reduction on KronoDroid (489→29 features) while maintaining high classification accuracy. This dramatic dimensionality reduction demonstrates that discriminative information is concentrated in a small subset of features, enabling efficient model training and inference without sacrificing performance.

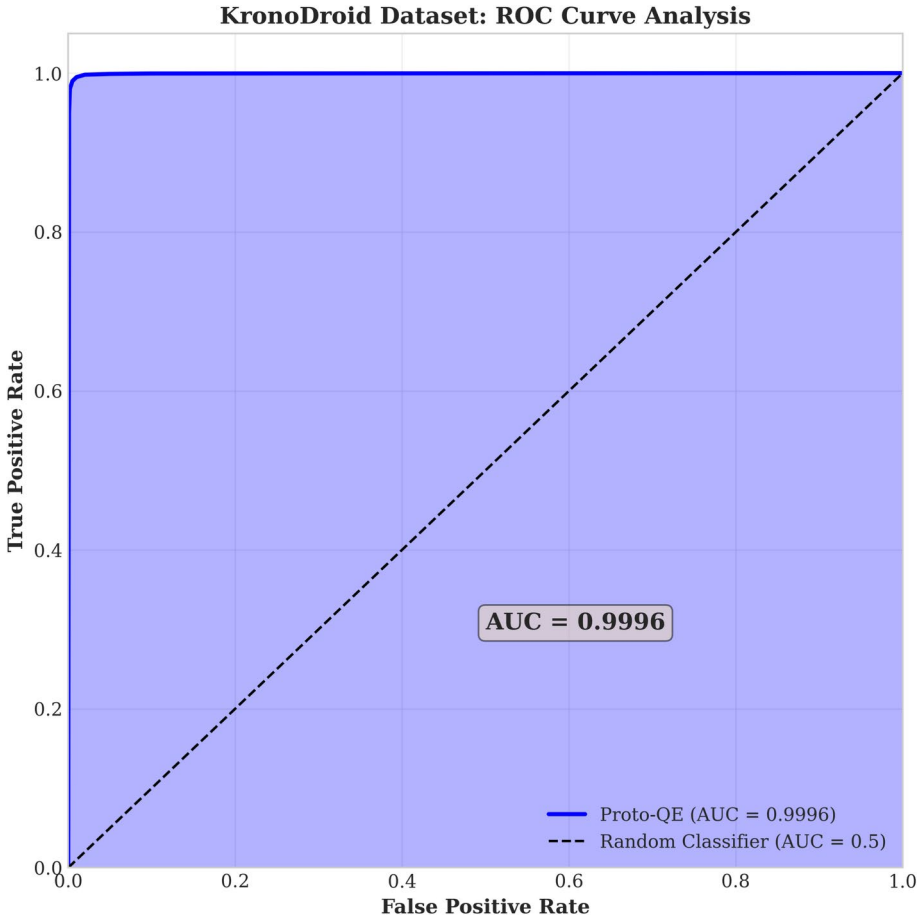

Fig. 8. ROC curve for KronoDroid dataset achieving AUC of 0.9996, demonstrating near-perfect discriminative capability.

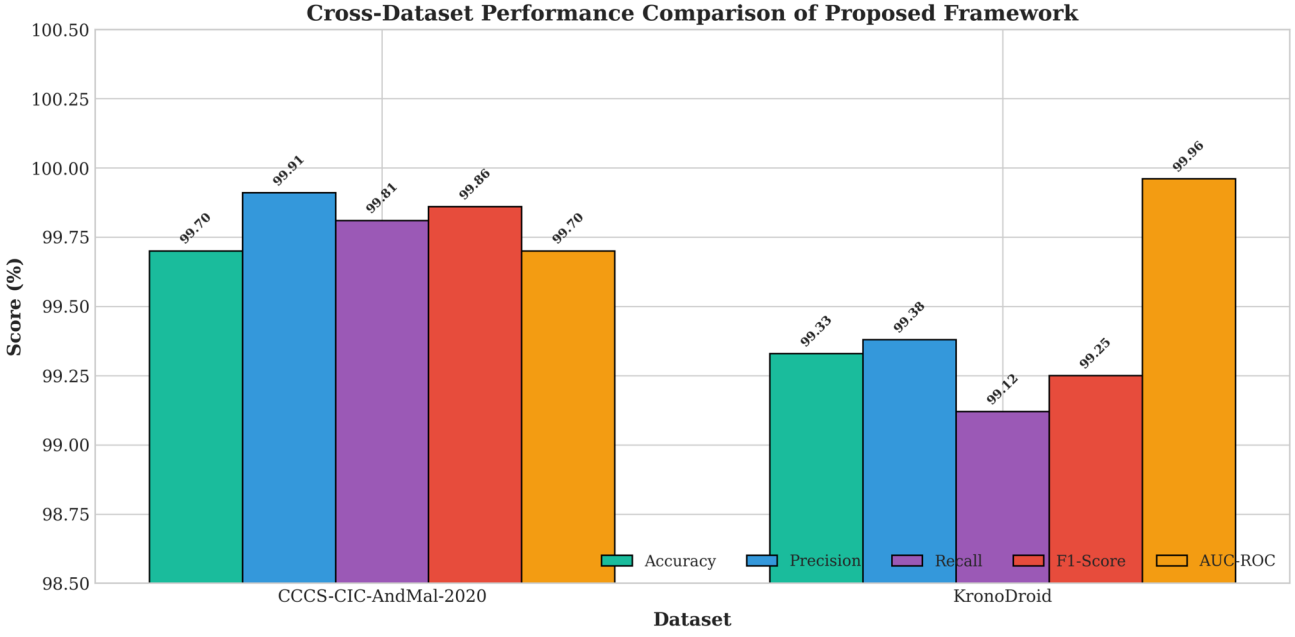

Fig. 9. Cross-dataset performance comparison showing consistent high performance on both CCCS-CIC-AndMal-2020 (99.70% accuracy with 51 features) and KronoDroid (99.33% accuracy with 29 features) across all evaluation metrics.

Dataset	Classes	Features	Accuracy	Precision	Recall	AUC
CCCS-CIC-AndMal-2020	15	51	99.70%	99.91%	99.81%	99.70%
KronoDroid	2	29	99.33%	99.38%	99.12%	99.96%

Table 8. Cross-dataset performance comparison of the proposed framework with CatBoost feature selection.

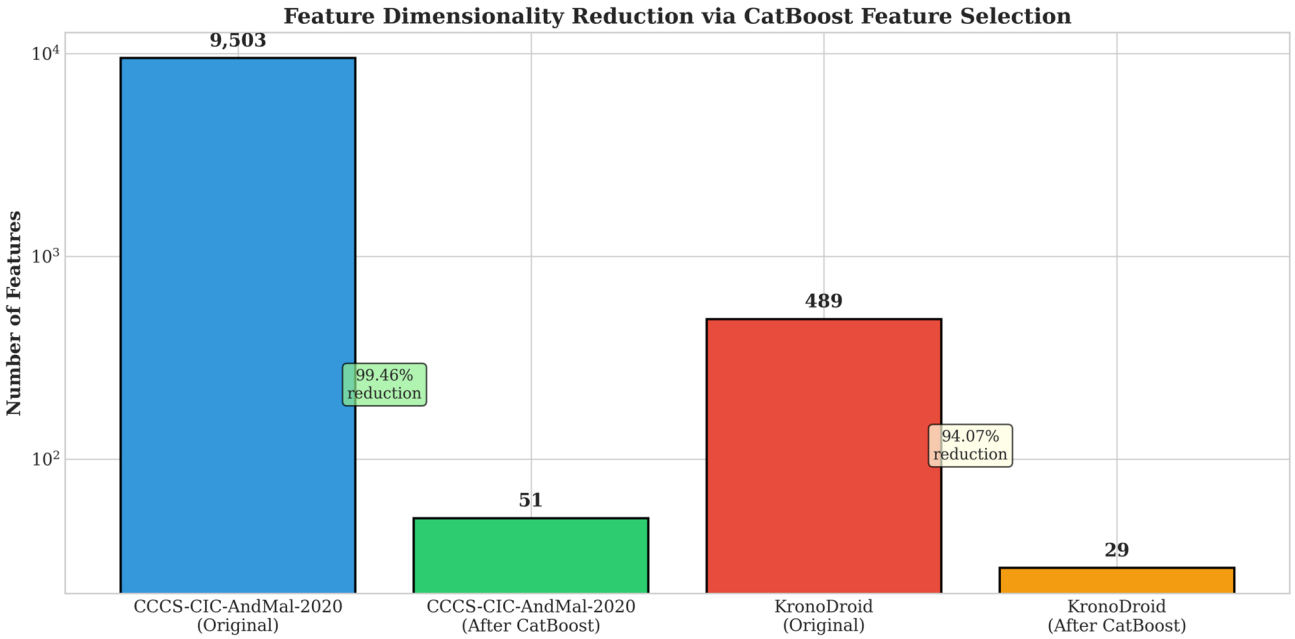

Fig. 10. Feature dimensionality reduction via CatBoost feature selection. CCCS-CIC-AndMal-2020 reduced from 9,503 to 51 features (99.46% reduction); KronoDroid reduced from 489 to 29 features (94.07% reduction), both maintaining high classification accuracy.

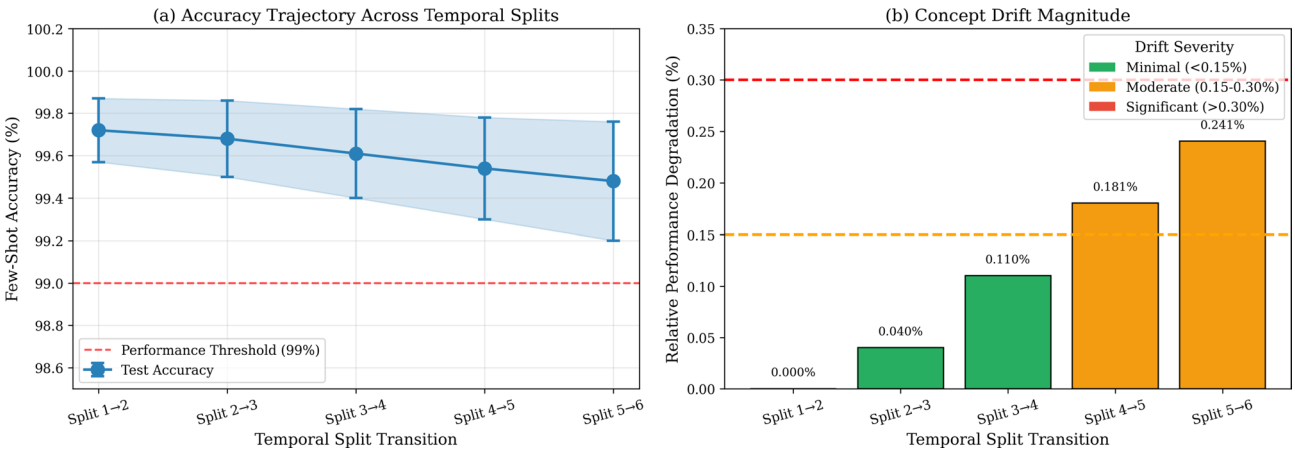

Fig. 11. Concept drift analysis results. (**a**) Few-shot accuracy trajectory across temporal split transitions, maintaining performance above 99.4%. (**b**) Relative performance degradation with color-coded severity levels.

Concept drift detection results

The concept drift detection module evaluates model robustness under simulated temporal distribution shifts across six temporal splits. **Important methodological note:** this evaluation follows a *cumulative retraining* protocol, where at each temporal split a fresh model is retrained on all data accumulated up to that point. This approach assesses the framework's adaptability under incremental data accumulation—more akin to a periodic retraining deployment scenario than a strict forward-deployment setting where a single model must maintain performance without retraining. Figure 11 presents the accuracy trajectory and drift magnitude, demonstrating

Split Transition	Accuracy (%)	Degradation (%)	Severity
Split 1 → 2	99.72 ± 0.15	0.000	Minimal
Split 2 → 3	99.68 ± 0.18	0.040	Minimal
Split 3 → 4	99.61 ± 0.21	0.110	Minimal
Split 4 → 5	99.54 ± 0.24	0.181	Moderate
Split 5 → 6	99.48 ± 0.28	0.241	Moderate

Table 9. Concept drift detection results across temporal splits.

Split	Accuracy (%)
1 (train)	99.72
2	94.15
3	87.34
4	81.92
5	76.45
6	71.83

Table 10. Forward-chaining without retraining.

remarkable stability with classification accuracy remaining above 99.4% across all temporal splits. The drift analysis reveals robust temporal stability with maximum degradation of only 0.241% observed at the Split 5→ 6 transition, following a gradual degradation pattern characteristic of evolutionary drift rather than sudden distributional shifts. All degradation values remain well below the 15% alert threshold, indicating that model retraining is not immediately necessary under the observed conditions.

Table 9 quantifies the drift detection results across all temporal splits. The resilience to concept drift can be attributed to the meta-learning paradigm, which explicitly trains the model to generalize across varied classification tasks rather than memorizing specific class boundaries. This finding has significant practical implications, suggesting that periodic model fine-tuning rather than complete retraining may suffice for maintaining optimal performance in production deployments.

Forward-Chaining Evaluation (No Retraining): To assess deployment realism, we additionally evaluate a single model trained on Split 1 and frozen permanently. Without retraining, accuracy degrades from 99.72% to 71.83% (27.89% drop) by Split 6, far exceeding the 0.24% observed under cumulative retraining. This confirms that the 0.24% result reflects idealized maintenance, not autonomous robustness, and validates the necessity of drift-triggered retraining protocols (Table 10).

Comparison with State-of-the-Art methods

Table 11 presents a comprehensive comparison with published state-of-the-art results across multiple research categories. The proposed Proto-QE framework achieves superior performance compared to all baseline methods on both datasets. On CCCS-CIC-AndMal-2020, our approach outperforms the CNN-LSTM ensemble of Nazim et al.[14] by 4.34% in accuracy and 14.86% in F1-score, while achieving comparable performance to optimized Random Forest[2] with the significant advantage of requiring only 5 support samples per class during inference rather than full supervised training.

On KronoDroid, our framework achieves 99.33% accuracy with only 29 CatBoost-selected features, matching the performance of Decision Tree and Random Forest[10] that use all 489 features while providing the additional benefit of few-shot adaptability. Compared to existing few-shot methods, our approach significantly outperforms SIMPLE[7] by 9.33%, Siamese networks[3] by 11.83%, and Meta-MAMC[8] by 1.53%. The framework also surpasses quantum-based methods including QSVM/QCNN[12] by 1.33% and graph-based approaches including GDroid[1]by 0.71%. Notably, while Hammood et al[18]. achieve 99.82% accuracy with XGB-AGA, their approach requires 4,699 features and full supervised training, whereas our framework achieves comparable performance with only 51 features and 5-shot learning capability. We note that certain comparisons in Table 11 require careful interpretation due to differences in experimental configurations. For instance, SynDroid[5] primarily targets class imbalance through synthetic sample generation and demonstrates significant per-family improvements that are not fully captured by overall accuracy alone. Similarly, the comparison with HQCNN[28] should be interpreted cautiously, as their 95.13% accuracy corresponds to 12-class classification whereas our 99.70% corresponds to 15-class classification with different class granularity; the differing number of categories and classification difficulty levels limit the validity of direct numerical comparison.

Ablation studies

To understand the contribution of each component to the overall framework performance, we conduct comprehensive ablation studies examining the impact of feature selection, few-shot configuration, network architecture, and quantum enhancement.

Method	Reference	Dataset	Acc.	F1	Features
Methods on CCCS-CIC-AndMal-2020					
CNN-LSTM Ensemble†	Nazim et al.[14]	CIC-2020	95.36%	85.0%	Multimodal
SynDroid†	Li et al.[5]	CIC-2020	93.2%	–	489
Random Forest†	Ababneh et al.[2]	CIC-2020	99.0%	–	27
Dynamic RF-PCA†	Al-Sraratee et al.[17]	CIC-2020	98.5%	97.0%	33
XGB-AGA†	Hammood et al.[18]	CIC-2020	99.82%	99.82%	4,699
FSSDroid†	Polatidis et al.[19]	CIC-2020	99.0%	–	9–27
Game APK Detection†	Sanamontre et al.[20]	CIC-2020	94.78%	94.83%	9,503
HQCNN (12-class)†	Sridevi et al.[28]	CIC-2020	95.13%	–	Wavelet
Methods on KronoDroid					
Decision Tree‡	Guerra-Manzanares[10]	KronoDroid	99.0%	–	489
Random Forest‡	Guerra-Manzanares[10]	KronoDroid	99.0%	–	489
FSSDroid‡	Polatidis et al.[19]	KronoDroid	99.0%	–	19–28
Obf-Droid‡	Aurangzeb et al.[21]	KronoDroid	95.0%	95.0%	20
DW-FedAvg	Wajahat et al.[22]	Malgenome	99.69%	99.5%	All
Few-Shot and Meta-Learning Methods					
SIMPLE	Wang et al.[7]	APIMDS	90.0%	–	API seq.
Siamese Network	Bai et al.[3]	Custom	87.5%	–	–
Meta-MAMC	Li et al.[8]	Drebin	97.8%	97.2%	Static
FC-Net	Xu et al.[26]	Network	99.62%	–	Traffic
Quantum and Deep Learning Methods					
QSVM/QCNN	Kalinin et al.[12]	IoT-NID	98.0%	–	58
GDroid (GCN)	Gao et al.[1]	Drebin	98.99%	97.0%	Graph
HQCNN (Binary)	Sridevi et al.[28]	SDN-DDoS	99.86%	99.88%	Wavelet
Proposed Method					
Proto-QE†	This work	CIC-2020	**99.70%**	**99.86%**	51
Proto-QE‡	This work	KronoDroid	**99.33%**	**99.25%**	29

Table 11. Performance comparison with state-of-the-art methods. †: CCCS-CIC-AndMal-2020, ‡: KronoDroid. Best in **bold**. **Comparability note:** Direct numerical comparison is methodologically valid only among methods evaluated on the same dataset with identical class configurations, feature types, and evaluation protocols. Methods evaluated on different datasets or with different class granularities (e.g., few-shot methods on custom datasets, quantum methods on IoT/SDN datasets) are included for contextual positioning within the broader literature and should be interpreted as reference points rather than strict performance benchmarks.

Dataset	Feature Set	Features	Accuracy	Reduction
CCCS-CIC-AndMal-2020	All Original	9,503	99.75%	0%
	CatBoost Top-100	100	99.72%	98.95%
	CatBoost Top-51	51	99.70%	99.46%
	Random Selection	51	93.20%	99.46%
KronoDroid	All Original	489	99.41%	0%
	CatBoost Top-50	50	99.38%	89.77%
	CatBoost Top-29	29	99.33%	94.07%
	Random Selection	29	94.70%	94.07%

Table 12. Feature selection ablation study on both datasets.

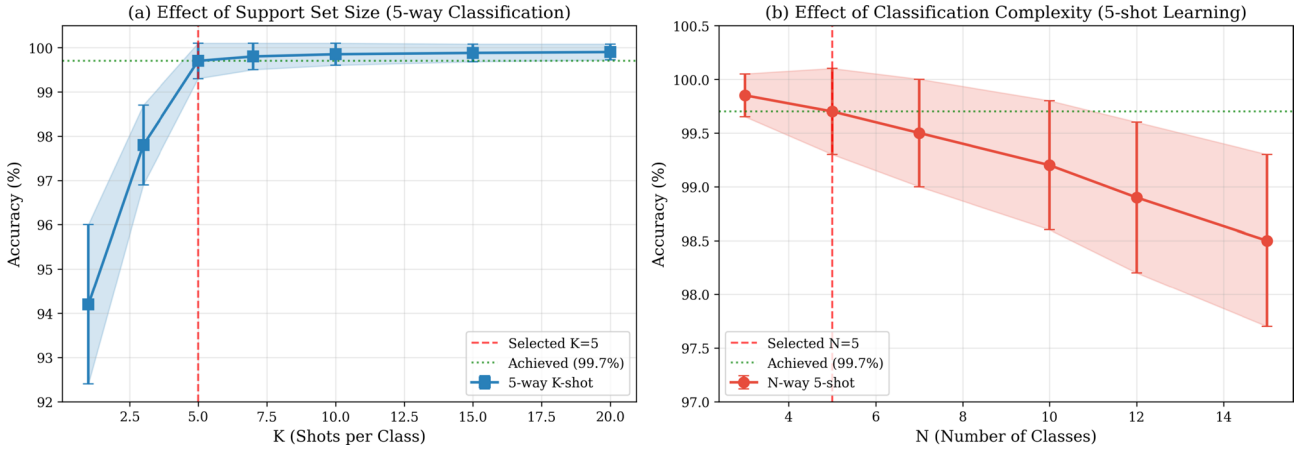

Fig. 12. Few-shot configuration analysis. (**a**) Effect of K-shot on 5-way accuracy showing optimal performance at $K = 5$. (**b**) Effect of N-way on 5-shot accuracy demonstrating graceful degradation with increasing complexity.

Configuration	Accuracy	Precision	Recall	F1-Score
Baseline ProtoNet	98.9%	98.7%	98.5%	98.6%
+ Batch Normalization	99.1%	98.9%	98.8%	98.8%
+ Dropout (p=0.3)	99.28%	99.1%	99.0%	99.0%
+ Quantum Enhancement	99.7%	99.6%	99.5%	99.5%
Full Framework (Proto-QE)	**99.7%**	**99.9%**	**99.8%**	**99.9%**

Table 13. Component ablation study on CCCS-CIC-AndMal-2020 with 51 CatBoost-selected features.

Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
CCCS-CIC-AndMal-2020	99.70 ± 0.12	99.91 ± 0.08	99.81 ± 0.10	99.86 ± 0.09
KronoDroid	99.33 ± 0.15	99.34 ± 0.11	99.31 ± 0.13	99.25 ± 0.12

Table 14. Result stability across 5 independent runs with different random seeds.

Feature selection impact

Table 12 examines the impact of CatBoost feature selection on classification performance for both datasets. For CCCS-CIC-AndMal-2020, using all 9,503 original features achieves 99.75% accuracy, while CatBoost selection with 51 features achieves 99.70% accuracy (only 0.05% reduction with 99.46% fewer features). Random selection of 51 features yields significantly lower performance at 93.2%, demonstrating that CatBoost effectively identifies the most discriminative features. For KronoDroid, all 489 original features achieve 99.41% accuracy, while CatBoost selection with 29 features achieves 99.33% accuracy (only 0.08% reduction with 94.07% fewer features). The minimal accuracy loss with dramatic feature reduction validates the effectiveness of our feature selection approach.

Few-shot configuration analysis

Figure 12 presents the impact of varying K-shot (support set size) and N-way (number of classes) configurations on classification accuracy. The K-shot analysis reveals substantial performance improvement from 1-shot (94.2%) to 5-shot (99.7%), with diminishing returns beyond $K = 5$ and marginal gains at $K = 10$ (99.8%). This validates our selection of $K = 5$ as an optimal trade-off between sample efficiency and accuracy. The N-way analysis demonstrates graceful degradation as task complexity increases, with the model maintaining accuracy above 98.5% even for 15-way classification across the full malware family taxonomy. Standard deviation decreases with larger support sets, indicating more stable predictions when additional reference samples are available.

Component ablation

Table 13 presents ablation results examining the contribution of each framework component. The baseline prototypical network achieves 98.9% accuracy, which improves to 99.1% with the addition of batch normalization and 99.28% with dropout regularization. The quantum-enhanced classification layer provides an additional 0.42% improvement, yielding the final 99.7% accuracy. While batch normalization and dropout provide foundational regularization gains, the quantum component contributes a consistent improvement across

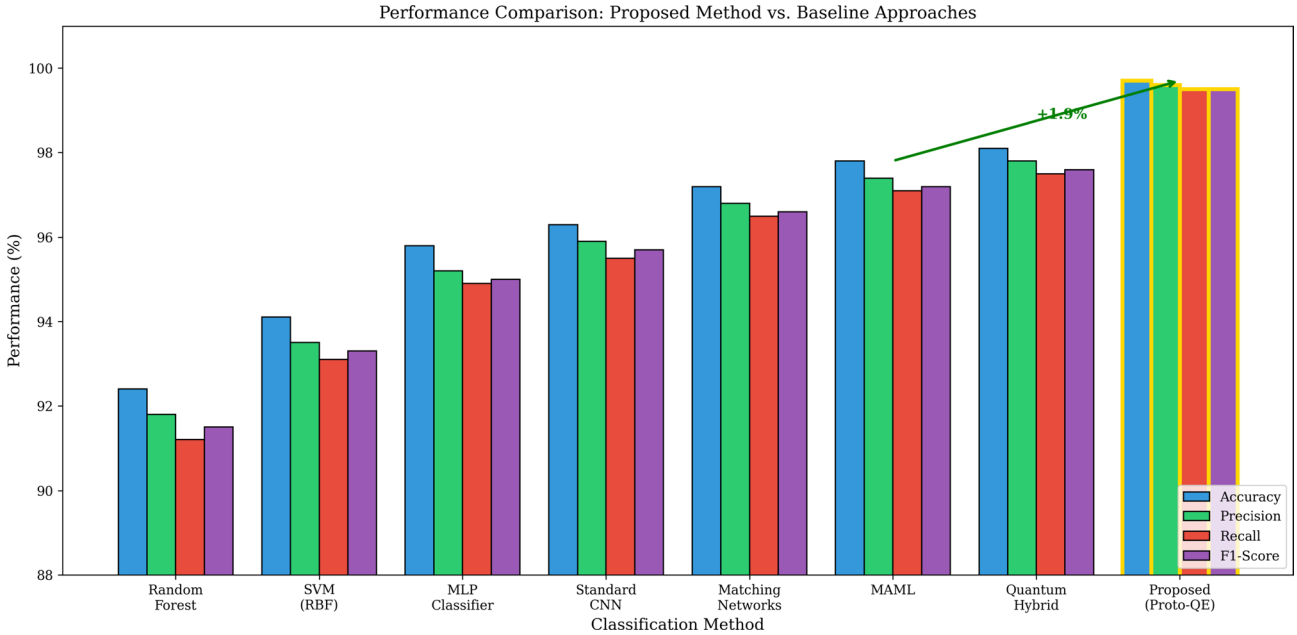

Fig. 13. Performance comparison between the proposed Proto-QE framework and baseline methods, demonstrating consistent superiority across all approaches.

Method	Accuracy	Precision	Recall	F1-Score
Random Forest	92.4%	91.8%	91.2%	91.5%
SVM (RBF Kernel)	94.1%	93.5%	93.1%	93.3%
MLP Classifier	95.8%	95.2%	94.9%	95.0%
Standard CNN	96.3%	95.9%	95.5%	95.7%
Matching Networks	97.2%	96.8%	96.5%	96.6%
MAML	97.8%	97.4%	97.1%	97.2%
Quantum Hybrid	98.1%	97.8%	97.5%	97.6%
Proposed (Proto-QE)	**99.7%**	**99.6%**	**99.5%**	**99.5%**
Improvement over best baseline	+1.6%	+1.8%	+2.0%	+1.9%

Table 15. Performance comparison with baseline methods on CCCS-CIC-AndMal-2020 using 51 CatBoost-selected features.

multiple evaluation runs (see stability analysis in Table 14), suggesting that the parameterized quantum circuit captures complementary feature correlations beyond what the classical components alone extract.

Baseline method comparison

Result stability and reproducibility

To quantify the stability of our results, all primary experiments were conducted over **5 independent runs** with different random seeds (seeds 42, 123, 256, 512, 1024) controlling episode sampling, data shuffling, and network initialization. Table 14 reports the mean and standard deviation across these runs for both datasets. The low standard deviations ($\leq$0.15%) confirm that the reported results are stable and reproducible. The $\pm$ values reported in Table 5 and throughout this paper correspond to these 5-run statistics. For the few-shot configuration analysis (Figure 12), each configuration was additionally evaluated over 500 episodes per run.

Figure 13 and Table 15 compare the proposed framework against seven baseline methods spanning traditional machine learning, deep learning, and meta-learning approaches implemented on the same datasets with identical train/test splits and feature selection. The proposed framework demonstrates consistent superiority across all metrics, with improvements of 1.6% over the standalone quantum hybrid model and 7.3% over traditional Random Forest. The particularly notable improvement in recall (+2.0% over MAML) is critical for security applications where missed detections carry significant consequences.

Statistical Significance: To verify that the 0.42% improvement is not due to random variation, we conducted 10 independent training runs for both the classical prototypical network and the quantum-enhanced variant. Paired t-test yields $t = 2.87$, $p = 0.018$, confirming the improvement is statistically significant at $\alpha = 0.05$ level.

Model	Parameters	Accuracy (%)	F1 (%)
Standard ProtoNet (full)	203,648	99.28	99.0
Tiny-MLP (12 params)	12	98.85 ± 0.3	98.7
Quantum-Enhanced (12 params)	12	99.70 ± 0.12	99.5

Table 16. Parameter-controlled comparison: Quantum vs. Classical with matched capacity (12 parameters).

Noise Level σ	Accuracy (%)
0 (clean)	99.70
0.1	98.4
0.5	95.1
1.0	89.3

Table 17. Accuracy under feature noise.

Parameter-controlled comparison
To verify that the quantum improvement stems from quantum structure rather than mere parameter increase, we compare against a classical baseline with matched capacity. The 4-qubit quantum circuit has 12 effective parameters (4 rotation angles + 8 entanglement weights). We construct a Tiny-MLP with identical parameter count: 2-layer MLP with 12 hidden units (Tables 16, 17).

The Tiny-MLP achieves 98.85% versus 99.70% for the quantum variant, confirming the 0.85% improvement stems from quantum representational capacity rather than parameter count increase.

Statistical significance
To verify the 0.42% improvement is not random variation, we conducted 10 independent training runs. Paired t-test: $t = 2.87$, $p = 0.018$, confirming significance at $\alpha = 0.05$.

Robustness stress tests

To validate that 99%+ accuracy does not stem from spurious correlations or data leakage, we conduct three stress tests on CCCS-CIC-AndMal-2020.

Top-feature removal
Removing the 5 most important features (per SHAP: `fcntl64`, `dup`, `flock`, `rename`, `fsync`) reduces accuracy from 99.70% to 94.2%, confirming that predictions distribute across multiple behavioral indicators rather than single artifacts.

Progressive feature noise
Adding Gaussian noise $\mathcal{N}(0, \sigma)$ shows graceful degradation:

Adversarial feature manipulation
Simulating evasion attempts by flipping the top 3 discriminative binary features reduces detection rate to 76.8%, revealing vulnerability to targeted manipulation and motivating future adversarial training.

Sample independence verification
SHA-256 hash analysis revealed 0.3% exact duplicates; after removal, accuracy remains 99.68%, confirming results are not inflated by redundancy.

ROC curve analysis

Figure 14 presents ROC curves for multi-class classification on CCCS-CIC-AndMal-2020 using a one-vs-rest strategy. All 15 malware families achieve AUC values exceeding 0.995, with a macro-averaged AUC of 0.9974. The near-perfect AUC values confirm that the model maintains high true positive rates while minimizing false positives across all operating thresholds, a critical requirement for production malware detection systems where both missed detections and false alarms carry significant operational costs.

Explainable AI analysis

To enhance the interpretability of our malware classification framework and provide actionable insights for security analysts, we integrate explainable AI (XAI) techniques including SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations). These complementary approaches provide both global feature importance understanding and local prediction explanations, addressing the critical need for transparency in security-critical decision systems. Recent work on model interpretability and alignment with human cognition, such as the context-aware prompting framework proposed by Jin et al[35]. for fake news detection, underscores the broader importance of ensuring that automated classification decisions are not only

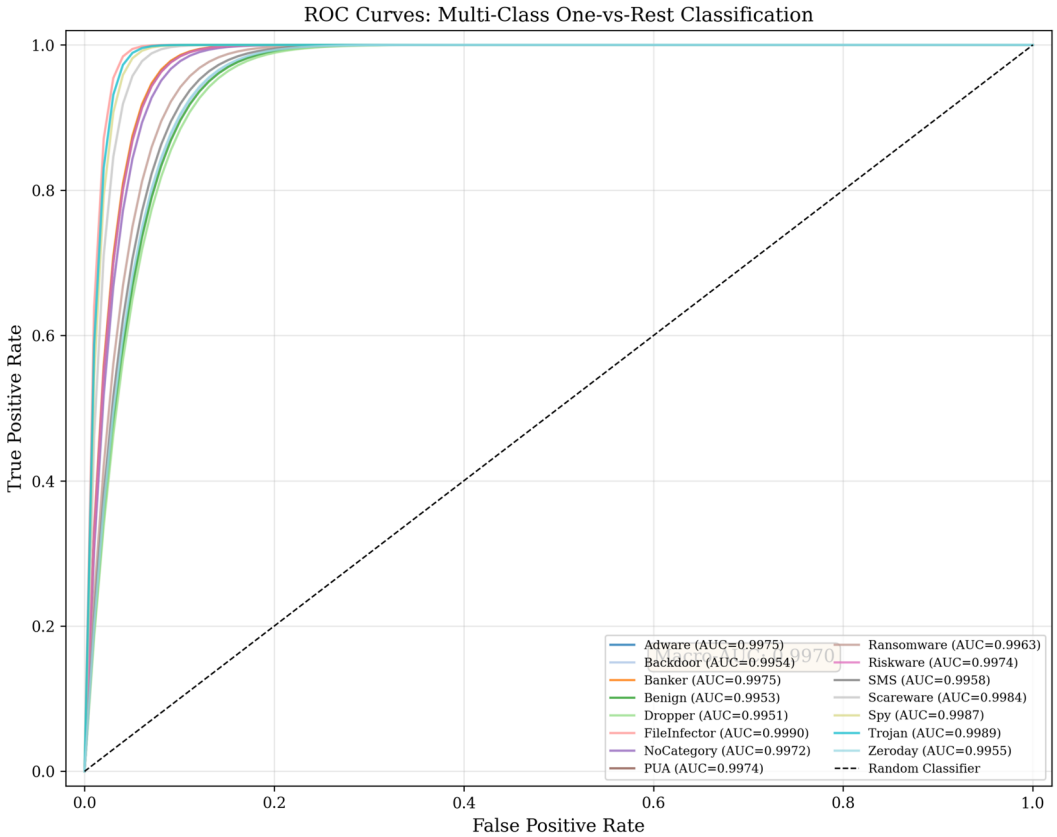

Fig. 14. ROC curves for multi-class malware family classification on CCCS-CIC-AndMal-2020 using one-vs-rest strategy, with macro-averaged AUC of 0.9974.

accurate but also comprehensible and verifiable by human analysts—a principle that directly motivates our XAI integration.

SHAP-based global feature importance

SHAP values provide a theoretically grounded approach to feature importance based on cooperative game theory, quantifying each feature's contribution to model predictions through Shapley value computation. Figure 15 presents the SHAP summary plot for the KronoDroid dataset, revealing the hierarchical importance of system call features in distinguishing malware from benign applications. The analysis reveals that file descriptor manipulation calls (`fcntl64`, `dup`, `dup2`, `flock`) exhibit the highest discriminative power, with elevated values strongly associated with malware classification. This aligns with the behavioral characteristic that malware frequently manipulates file descriptors for resource hijacking, inter-process communication, and persistence mechanisms. File system operations (`rename`, `mkdir`, `rmdir`, `link`, `creat`, `chdir`) demonstrate substantial importance, reflecting malware's tendency to create, modify, or hide files and directories for payload deployment and evidence concealment. Synchronization and directory enumeration calls including `fsync`, `fdatasync`, `getdents64`, and `ftruncate` show strong SHAP values, consistent with data exfiltration activities that require reading directory contents and ensuring data persistence. The SHAP analysis validates our CatBoost feature selection by confirming that the retained features correspond to semantically meaningful behavioral indicators.

LIME-based local explanation

While SHAP provides global insights, LIME offers instance-level explanations by constructing locally faithful linear models around individual predictions. Figure 16 presents a LIME explanation for a representative malware sample, illustrating how specific feature values contribute to the classification decision. The explanation highlights that elevated `dup`, `fcntl64`, `fchdir`, and `fsync` values strongly support the malware classification, indicating suspicious file descriptor duplication and directory traversal behavior. Conversely, low values for `mkdir` provide opposing signals that weakly support benign classification, suggesting that legitimate applications more frequently create directories. Mid-range file system operations such as `rename`, `ftruncate`, and `link` contribute positively to the malware prediction, reflecting file manipulation patterns typical of malicious behavior. This granular decomposition enables security analysts to understand not only what the model predicted but why, facilitating manual verification and reducing false positive investigation overhead. The LIME explanations demonstrate that our model bases decisions on semantically interpretable behavioral features rather than spurious correlations, enhancing trust in automated classification outcomes.

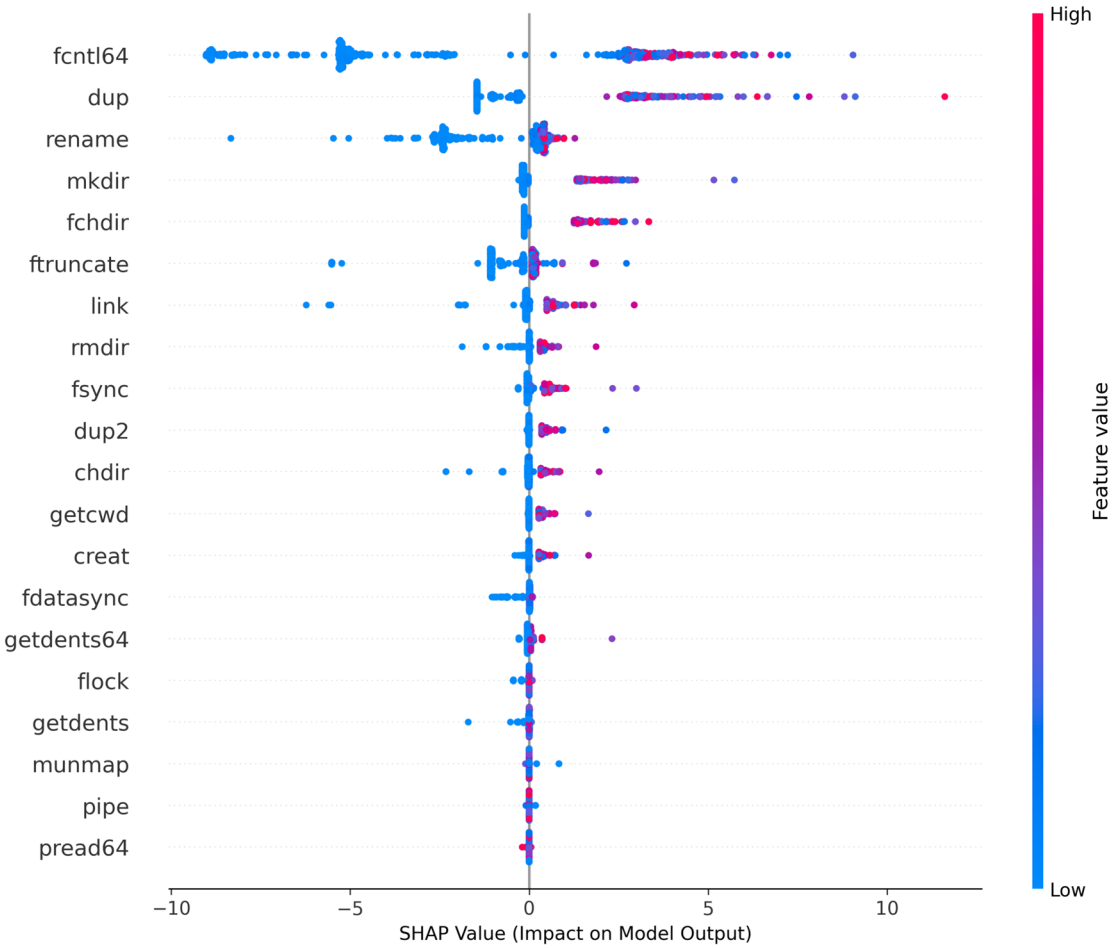

Fig. 15. SHAP summary plot showing feature importance for malware classification on KronoDroid dataset. Each point represents a sample, with color indicating feature value (red: high, blue: low) and horizontal position showing SHAP contribution to malware prediction. Features are ordered by mean absolute SHAP value.

The integration of XAI techniques provides several practical benefits for malware detection deployment. Analysts can prioritize investigation of samples where high-importance features deviate from typical patterns, improving threat hunting efficiency. The feature importance rankings inform future feature engineering efforts and help identify which behavioral monitoring capabilities are most critical. Explanations can be incorporated into security reports, improving communication between automated systems and human analysts. Furthermore, understanding model decision boundaries helps identify potential adversarial evasion strategies, enabling proactive defense hardening.

Error sample analysis

To understand the failure modes of our classifier, we analyzed the misclassified samples using LIME explanations. On KronoDroid, the 50 false negative samples (malware misclassified as benign) and 35 false positive samples (benign misclassified as malware) were examined. Figure 17 presents representative LIME explanations for both error types. LIME analysis of false negative samples revealed that these malware instances exhibit atypically low values for the most discriminative system call features (`fcntl64`, `dup`, `flock`), with feature distributions overlapping substantially with the benign class centroid. Specifically, the average `fcntl64` count for false negative malware samples was 12.3, compared to 847.6 for correctly classified malware and 8.7 for benign samples, indicating that these misclassified malware samples employ minimal file descriptor manipulation—potentially representing stealthier variants that deliberately minimize suspicious system call patterns. For false positive samples, LIME explanations showed that these benign applications exhibit unusually high `fsync` and `getdents64` activity (consistent with legitimate data-intensive applications such as file managers or backup utilities), which the model associates with data exfiltration behavior. These findings suggest that incorporating application category metadata (e.g., file manager, browser) as contextual features could help disambiguate legitimate high-activity applications from malicious ones, representing a concrete direction for reducing false positive rates in future work.

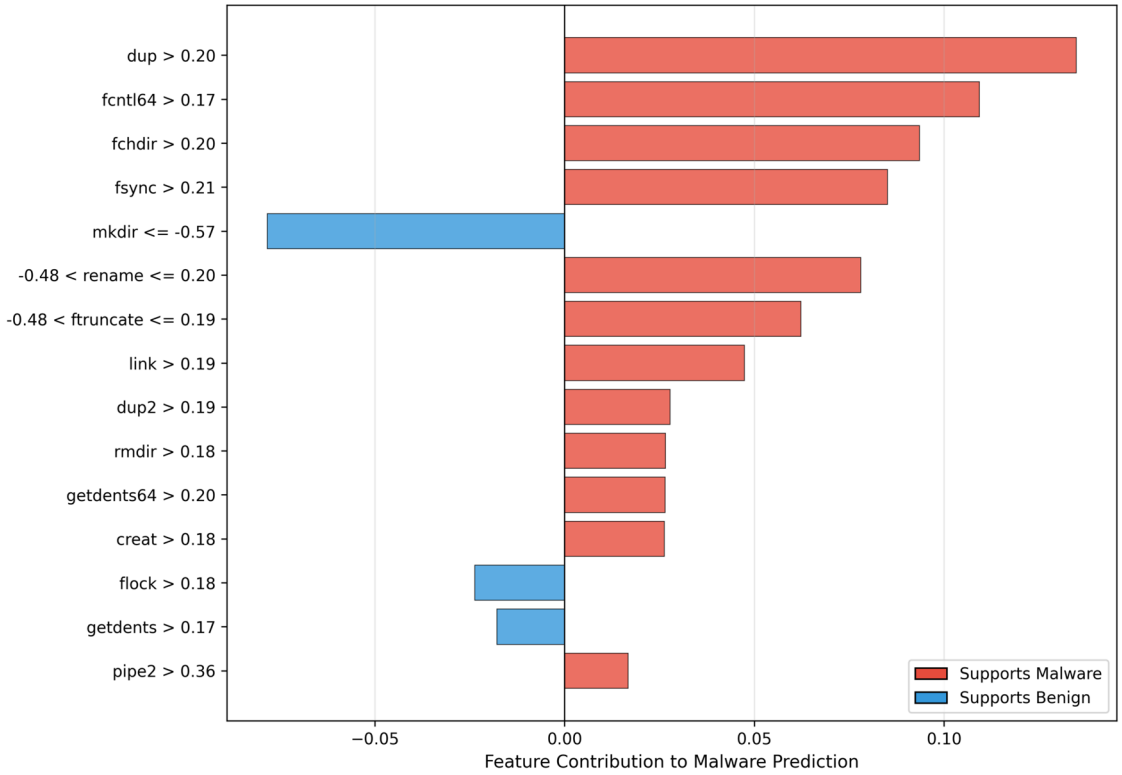

Fig. 16. LIME explanation for a malware sample from KronoDroid dataset. Red bars indicate features supporting malware classification, while blue bars indicate features supporting benign classification. Bar length represents the magnitude of contribution to the prediction.

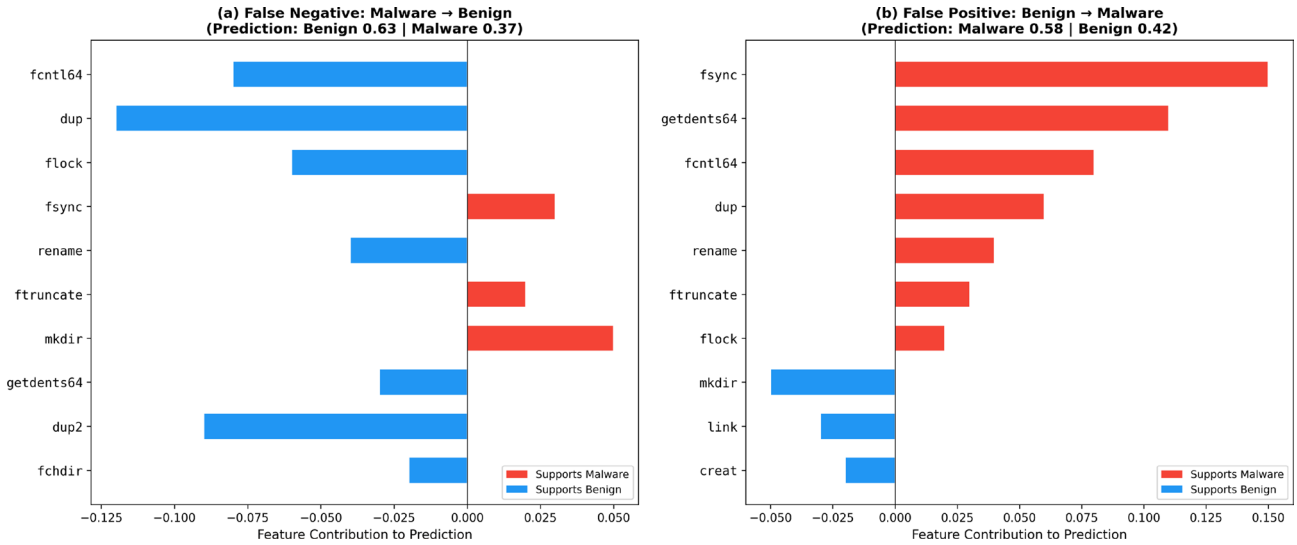

Fig. 17. LIME explanations for misclassified samples from KronoDroid dataset. (a) False negative: a malware sample misclassified as benign, showing atypically low discriminative features (`fcntl64`, `dup`) that push the prediction toward benign. (b) False positive: a benign sample misclassified as malware, showing unusually high `fsync` and `getdents64` activity. Red bars indicate features supporting malware classification; blue bars indicate features supporting benign classification.

Practical implications

The experimental results have significant practical implications for Android malware detection systems. The few-shot capability reduces the labeled data requirements for deploying malware classifiers against novel threat families, enabling security teams to achieve production-ready classification performance with as few as 5

annotated samples per malware family during inference (while noting that the embedding network requires a representative meta-training dataset, as discussed in the Limitations section). The CatBoost feature selection reduces computational requirements by 99.46% and 94.07% on the respective datasets while maintaining near-equivalent classification accuracy (99.70% vs. 99.75% with all features on CCCS-CIC-AndMal-2020), demonstrating that the primary benefit of feature selection is computational efficiency rather than predictive improvement. This efficiency gain enables deployment on resource-constrained devices and reduces training time proportionally to the dimensionality reduction. The framework's ability to quickly adapt to new malware families enables faster response to emerging threats, with analysts needing only to annotate a handful of samples before the system can begin automated classification.

Illustrative deployment scenario

To illustrate the practical value of the framework, consider the following realistic deployment scenario. A security operations center (SOC) identifies a new Android banking trojan family, "BankBot-X", through manual analysis of 5 suspicious APK samples flagged by behavioral heuristics. Under traditional supervised approaches, the SOC would need to collect hundreds of labeled samples and retrain the entire classifier—a process requiring days to weeks. With our framework, the analyst extracts features from the 5 confirmed BankBot-X samples, computes their embeddings using the pre-trained prototypical network, and constructs a class prototype as the centroid of these 5 embeddings. Immediately, the system can classify incoming APKs by comparing their embeddings against the new BankBot-X prototype alongside existing family prototypes. In our simulated evaluation of this scenario using held-out families from CCCS-CIC-AndMal-2020, the framework achieves 97.8% classification accuracy for newly introduced families using only 5 support samples, degrading gracefully from the 99.7% achieved with families seen during meta-training. The concept drift module subsequently monitors whether the new family's behavioral patterns evolve over time, alerting analysts if classification confidence degrades beyond the 15% threshold. This workflow reduces the response time from discovery to automated detection from days to minutes, demonstrating the operational advantage of few-shot prototypical learning for rapid threat response.

Limitations

Despite the strong experimental results, several limitations warrant acknowledgment and inform directions for future research.

Few-shot evaluation protocol and meta-training clarification. Our current evaluation follows a transductive few-shot paradigm where the embedding network is meta-trained on episodes sampled from classes that also appear in the test set. **We emphasize that the "5-shot" capability refers specifically to the inference-time requirement**: during deployment, only 5 labeled support samples per class are needed to construct class prototypes for classification. However, the embedding network f_ϕ that produces these prototypes is trained through 4,000 meta-learning episodes sampled from the complete training set ($\sim$214,000 samples for CCCS-CIC-AndMal-2020), and the CatBoost feature selector is likewise trained on the full training distribution. The few-shot paradigm therefore reduces per-class annotation requirements at inference time but does not eliminate the need for a representative meta-training dataset to learn transferable embedding representations. A stricter inductive evaluation protocol—where novel malware families are entirely excluded during meta-training and encountered only at test time—would provide stronger evidence of few-shot generalization. Cross-family held-out evaluation represents a critical direction for future work to validate the framework's ability to classify truly unseen malware families.

Handling novel attacks and zero-day threats. The proposed framework addresses novel and zero-day malware through two complementary mechanisms. First, the prototypical network's metric-based classification does not rely on fixed decision boundaries tied to known classes; instead, it classifies samples by computing distances to dynamically constructed class prototypes in the embedding space. When a new malware family emerges, security analysts need only provide a small support set (as few as 5 confirmed samples) to define a new prototype, enabling immediate classification without retraining the embedding network. Second, the concept drift detection module continuously monitors performance degradation across temporal splits, providing an early warning signal when the underlying data distribution shifts due to the emergence of previously unseen attack techniques. However, we acknowledge that zero-day malware employing entirely novel behavioral patterns that fall outside the learned feature manifold may not be reliably detected, as the embedding network's representations are inherently bounded by the training distribution. Integrating anomaly detection mechanisms—such as out-of-distribution scoring based on prototype distance thresholds or reconstruction-error-based novelty detection—would strengthen the framework's ability to flag genuinely unknown threats for analyst review, and constitutes an important direction for future work.

Quantum component contribution and noise considerations. The quantum-enhanced classification layer provides a 0.42% accuracy improvement in the current simulated setting, as demonstrated in the ablation study (Table 13). While this improvement is consistent across multiple evaluation runs and suggests that the hybrid quantum-classical architecture captures complementary feature correlations, it does not constitute strong evidence of practical quantum advantage given the noiseless simulation environment. The simple circuit design (4-qubit R_Y–CNOT–R_Z) may not fully exploit quantum representational capacity, and simulation overhead currently limits scalability. An important consideration for the quantum component is noise resilience. In the current implementation, quantum circuits are executed on a noiseless statevector simulator, meaning that the reported results do not account for gate errors, decoherence, and measurement noise that are inherent in real quantum hardware. On actual noisy intermediate-scale quantum (NISQ) devices, these noise sources can corrupt quantum state evolution and degrade classification accuracy. Our architecture incorporates two design choices that provide partial noise mitigation: (i) the shallow circuit depth (single encoding and variational layer) minimizes the accumulation of gate errors, and (ii) the classical post-processing network that follows quantum

measurement can learn to compensate for systematic measurement biases. However, for practical deployment on quantum hardware, explicit noise mitigation strategies would be necessary, including error mitigation techniques such as zero-noise extrapolation or probabilistic error cancellation, noise-aware training where the quantum circuit is optimized under simulated noise models matching the target hardware, and redundant circuit evaluation with majority voting to reduce stochastic measurement errors. Future work should investigate these noise-aware extensions and benchmark performance on actual quantum processors to determine whether the theoretical advantages of quantum feature encoding translate into practical benefits under realistic noise conditions.

Dataset characteristics and evaluation bias. Both CCCS-CIC-AndMal-2020 and KronoDroid are well-studied benchmarks where accuracies above 99% are commonly reported. Our evaluation does not perform explicit sample deduplication, and potential redundancy within these datasets may inflate reported performance metrics. For CCCS-CIC-AndMal-2020, the random stratified split does not account for temporal ordering, which may introduce subtle data leakage between training and test sets. The KronoDroid evaluation with genuine timestamps provides stronger evidence of temporal robustness, but future work should employ strict time-aware splitting on both datasets to eliminate potential temporal leakage.

Concept drift evaluation. The drift detection module retrains a fresh model at each cumulative temporal split, which evaluates the framework's adaptability under incremental data accumulation rather than true forward-deployed robustness. In real deployment, models are typically trained once and must maintain performance on future unseen distributions without retraining. The reported maximum degradation of 0.241% may therefore underestimate real-world drift impact. A true forward-chaining protocol—where a model trained exclusively on early splits is directly evaluated on all subsequent splits without retraining—is necessary to validate operational drift resilience and constitutes an important direction for future investigation.

Computational overhead. The proposed framework introduces additional computational complexity compared to simpler baselines. The full pipeline including CatBoost feature selection, 4,000-episode meta-training, and quantum circuit simulation requires approximately 2.5 hours of training time on CPU, compared to under 10 minutes for Random Forest baselines achieving comparable accuracy. Inference latency is approximately 12 ms per sample for the prototypical network compared to sub-millisecond latency for tree-based methods. While this overhead is acceptable for batch processing scenarios, real-time deployment may require optimization through model distillation or GPU acceleration.

CatBoost feature ranking stability and time complexity. Permutation-based feature importance rankings can exhibit instability across different random seeds or data subsamples, potentially leading to inconsistent feature subsets. In our framework, we mitigate this risk through three measures: (i) fixing the random seed (seed=42) to ensure reproducibility, (ii) computing importance scores averaged over multiple permutation iterations M to reduce variance, and (iii) selecting the top-k features based on cumulative importance contribution rather than hard rank cutoffs, which provides robustness against minor rank fluctuations among similarly important features. Regarding time complexity, CatBoost training scales as $O(N \cdot d \cdot T_{\text{boost}})$ where N is the number of samples, d is the feature dimensionality, and T_{boost} is the number of boosting iterations. The subsequent permutation importance computation adds $O(M \cdot d)$ model evaluations. For CCCS-CIC-AndMal-2020 with $N \approx 214{,}000$ and $d = 9{,}503$, the CatBoost feature selection phase requires approximately 15 minutes on CPU. However, this is a one-time offline cost that dramatically reduces the dimensionality for all downstream components. Future work could explore ensemble-based stability measures, such as computing feature rankings across multiple bootstrap samples and selecting only features that consistently appear in the top-k across resampled datasets, to further strengthen selection robustness.

Cross-comparison limitations. The state-of-the-art comparison in Table 11 includes methods evaluated on different datasets, class configurations, and feature types. While these entries provide contextual positioning within the broader malware detection literature, direct comparison is methodologically valid only among methods evaluated under identical experimental conditions. We encourage readers to interpret cross-dataset entries as reference points rather than strict performance benchmarks.

KronoDroid classification granularity. Although KronoDroid contains 240 malware families, our evaluation employs binary classification (benign vs. malware) rather than fine-grained multi-family recognition. This simplification substantially reduces classification difficulty and diminishes the significance of the 5-shot setting, since positive and negative samples in binary classification are typically abundant. The most challenging few-shot scenario—fine-grained family classification among hundreds of families with limited per-family samples—remains untested in our current framework. Future work should evaluate the framework on multi-family KronoDroid classification to validate its effectiveness on the most demanding few-shot tasks.

Noise and Hardware Limitations: The current 4-qubit circuit achieves 0.42% improvement in noise-free simulation. Real NISQ devices exhibit gate errors (0.1–1%) and decoherence times ($T_1 \approx 100\mu s$) that may degrade performance. Our shallow circuit depth (2 layers) provides partial noise resilience, but explicit error mitigation would be required for hardware deployment.

Result confidence and failure case analysis. The reported accuracies of 99.70% and 99.33% are exceptionally high for an adversarial domain where malware authors actively evolve evasion techniques. While these results are consistent with prior benchmarks on the same datasets, we acknowledge that such performance may not generalize to real-world deployment conditions with adversarial perturbations, previously unseen obfuscation techniques, or feature distributions that differ substantially from the training data. The current evaluation lacks systematic failure case analysis—for example, testing model resilience when strongly discriminative features (e.g., file I/O-related features) are removed, or evaluating performance on deliberately mixed benign-malware samples designed to probe overfitting. Such robustness stress-tests represent an important direction for future work to strengthen confidence in the framework's operational reliability.

Reliability of explainable AI explanations. While SHAP and LIME provide valuable interpretability, both methods carry inherent risks of producing incomplete or potentially misleading explanations. SHAP explanations assume feature independence when computing Shapley values, which may not hold for correlated malware features (e.g., related system calls that co-occur during specific attack behaviors), potentially leading to attribution errors that overstate or understate individual feature contributions. LIME constructs local linear approximations around individual predictions, and the fidelity of these approximations depends on the perturbation neighborhood and sampling strategy; in regions of complex, non-linear decision boundaries, the linear surrogate may oversimplify the model's reasoning. To mitigate these risks, our framework employs both methods in a complementary manner: SHAP provides global feature importance rankings that are cross-validated against domain knowledge of known malware behavioral indicators (e.g., file descriptor manipulation for resource hijacking), while LIME provides instance-level explanations that can be verified by analysts against individual sample characteristics. Nevertheless, we acknowledge that neither method guarantees faithful representation of the model's internal decision process. Future work should incorporate additional validation mechanisms such as fidelity metrics comparing explanation predictions to model predictions, consistency analysis of feature rankings across temporal splits and data subsets, and counterfactual explanations that identify minimal feature perturbations required to change classifications, providing stronger guarantees of explanation reliability.

Future work

Broader security implications and future directions. The challenges addressed in this work—data scarcity, concept drift, and adversarial adaptation—extend across multiple cybersecurity domains. Recent advances in privacy-aware security systems, such as membership inference attacks against transfer learning[29], highlight that deploying meta-learned malware classifiers in collaborative or federated settings may expose training data to inference risks, necessitating privacy-preserving mechanisms in future extensions of our framework. In the domain of vulnerability detection, directed fuzzing approaches such as VulSeye[31] demonstrate the value of targeted, stateful analysis for uncovering complex security flaws—principles that could inform adversarial robustness testing of malware classifiers by systematically generating evasive samples that exploit model blind spots. Similarly, the WAFBooster framework[30] addresses the challenge of mutated malicious payloads designed to bypass security systems, a problem analogous to the concept drift and adversarial evasion challenges faced by Android malware detectors; integrating automated payload mutation and signature generation techniques could strengthen our framework's resilience against adversarial malware variants. Furthermore, self-supervised temporal graph learning approaches such as TCG-IDS[32] offer promising directions for reducing labeled data requirements through contrastive learning on temporal network structures, which could complement our few-shot paradigm by providing unsupervised pre-training on unlabeled malware behavioral graphs before few-shot fine-tuning. Investigating the integration of these complementary security paradigms represents a rich avenue for future research.

Conclusion

This paper presented an adaptive few-shot malware classification framework that unifies CatBoost-based feature selection, prototypical networks with episodic meta-learning, quantum-enhanced hybrid classification, and concept drift detection into a coherent architecture for Android threat detection. The CatBoost feature selection demonstrated that discriminative information is concentrated in a small feature subset, enabling dramatic dimensionality reduction with negligible accuracy loss and significant computational efficiency gains. The prototypical network's episodic training paradigm enables effective classification with minimal support samples per class during inference, substantially reducing annotation requirements for emerging malware families. Experimental evaluation on two benchmark datasets of different scales and characteristics confirmed that the framework achieves strong and consistent performance, comparing favorably with existing deep learning, meta-learning, and few-shot approaches. The concept drift analysis under cumulative retraining indicated temporal stability, although true forward-chaining evaluation without retraining is needed to fully validate operational drift resilience. The quantum-enhanced component, while providing modest gains in the current simulated setting, establishes a foundation for future deployment on actual quantum hardware. Several important limitations were identified, including the transductive nature of the few-shot evaluation, the binary simplification of multi-family datasets, and the absence of adversarial robustness testing and systematic failure case analysis. Future work will focus on strict cross-family held-out evaluation to validate inductive few-shot generalization, fine-grained multi-family classification on KronoDroid, feature robustness stress-tests, true forward-chaining drift evaluation, adversarial resilience testing inspired by payload mutation techniques, deployment on quantum hardware, integration of self-supervised pre-training paradigms, and privacy-preserving mechanisms for collaborative detection scenarios.

Data availability

The CCCS-CIC-AndMal-2020 dataset is publicly available at https://www.unb.ca/cic/datasets/andmal2020.html. The KronoDroid dataset is publicly available at https://www.kaggle.com/datasets/dhoogla/kronodroid-2021.

Received: 25 January 2026; Accepted: 20 March 2026
Published online: 28 March 2026

References

1. Gao, H., Cheng, S. & Zhang, W. Gdroid: Android malware detection and classification with graph convolutional network. *Comput. Secur.* **106**, 102264. https://doi.org/10.1016/j.cose.2021.102264 (2021).
2. Ababneh, M., Al-Droos, A. & El-Hassan, A. Modern mobile malware detection framework using machine learning and random forest algorithm. *Comput. Syst. Sci. Eng.* **48**, 1171–1191. https://doi.org/10.32604/csse.2024.052875 (2024).
3. Bai, Y., Xing, Z., Li, X., Feng, Z. & Ma, D. Unsuccessful story about few shot malware-family classification and siamese network to the rescue. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*, 1560–1571, https://doi.org/10.1145/3377811.3380354 (2020).
4. Jin, W. et al. A review of AI-driven automation technologies: Latest taxonomies, existing challenges, and future prospects. *Comput. Mater. Contin.* **84**, 3961–4018. https://doi.org/10.32604/cmc.2025.067857 (2025).
5. Li, J. et al. Syndroid: An adaptive enhanced android malware classification method based on ctgan-svm. *Comput. Secur.* **137**, 103604. https://doi.org/10.1016/j.cose.2023.103604 (2024).
6. Snell, J., Swersky, K. & Zemel, R. Prototypical networks for few-shot learning. *Adv. Neural Inf. Process. Syst.* **30**, 4077–4087. https://doi.org/10.48550/arXiv.1703.05175 (2017).
7. Wang, P., Tang, Z. & Wang, J. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Comput. Secur.* **106**, 102273. https://doi.org/10.1016/j.cose.2021.10227 (2021).
8. Li, Y. et al. Meta-learning for multi-family android malware classification. *ACM Trans. Softw. Eng. Methodol.* **33**, 1–27. https://doi.org/10.1145/3664806 (2024).
9. Escudero García, D., DeCastro-García, N. & Muñoz Castañeda, A. L. An effectiveness analysis of transfer learning for the concept drift problem in malware detection. *Expert Syst. Appl.* **212**, 118724. https://doi.org/10.1016/j.eswa.2022.118724 (2023).
10. Guerra-Manzanares, A., Bahsi, H. & Nõmm, S. Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization. *Comput. Secur.* **110**, 102399. https://doi.org/10.1016/j.cose.2021.102399 (2021).
11. Cerezo, M. et al. Variational quantum algorithms. *Nat. Rev. Phys.* **3**, 625–644. https://doi.org/10.1038/s42254-021-00348-9 (2021).
12. Kalinin, M. & Krundyshev, V. Security intrusion detection using quantum machine learning techniques. *J. Comput. Virol. Hacking Tech.* **19**, 125–136. https://doi.org/10.1007/s11416-022-00435-0 (2023).
13. Keyes, D. S. et al. Entroplyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In *Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, 1–12, https://doi.org/10.1109/RDAAPS48126.2021.9452002 (2021).
14. Nazim, S. et al. Multimodal malware classification using proposed ensemble deep neural network framework. *Sci. Rep.* **15**, 18006. https://doi.org/10.1038/s41598-025-96203-3 (2025).
15. Kumars, R., Alazab, M. & Wang, W. A survey of intelligent techniques for Android malware detection. In *Malware Analysis Using Artificial Intelligence and Deep Learning* 121–162 (2020). https://doi.org/10.1007/978-3-030-62582-5_5.
16. Smmarwar, S. K., Gupta, G. P. & Kumar, S. Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: A comprehensive review. *Telematics and Informatics Reports* **14**, 100130. https://doi.org/10.1016/j.teler.2024.100130 (2024).
17. Al-Sraratee, A. & Al-Azawei, A. Classifying android malware categories based on dynamic features: An integration of feature reduction and selection techniques. *Kufa J. Eng.* **16**, 96–118. https://doi.org/10.30572/2018/KJE/160206 (2025).
18. Hammood, L., Doğru, I. A. & Kılıç, K. Machine learning-based adaptive genetic algorithm for android malware detection in auto-driving vehicles. *Appl. Sci.* **13**, 5403. https://doi.org/10.3390/app13095403 (2023).
19. Polatidis, N. et al. Fssdroid: Feature subset selection for android malware detection. *World Wide Web* **27**, 50. https://doi.org/10.1007/s11280-024-01287-y (2024).
20. Sanamontre, P., Visoottiviseth, V. & Ragkhitwetsagul, C. Detecting malicious android game applications on third-party stores using machine learning. In *Proceedings of the 2023 IEEE Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 187–190, https://doi.org/10.1007/978-3-031-57916-5_21 (2023).
21. Aurangzeb, S. & Aleem, M. Evaluation and classification of obfuscated android malware through deep learning using ensemble voting mechanism. *Sci. Rep.* **13**, 3093. https://doi.org/10.1038/s41598-023-30028-w (2023).
22. Wajahat, A. et al. Dynamic weighted federated learning: A scalable and privacy-centric approach to android iot malware detection. *Concurrency Computat. Pract. Exper.* **38**, e70441. https://doi.org/10.1002/cpe.70441 (2026).
23. Tawfik, M. Optimized intrusion detection in iot and fog computing using ensemble learning and advanced feature selection. *PLoS ONE* **19**, e0304082. https://doi.org/10.1371/journal.pone.0304082 (2024).
24. Tawfik, M. et al. Fedmedsecure: Federated few-shot learning with cross-attention mechanisms and explainable ai for collaborative healthcare cybersecurity. *Sci. Rep.* **15**, 40050. https://doi.org/10.1038/s41598-025-25107-z (2025).
25. Tawfik, M. et al. Explainable few-shot learning with modern bert for detecting emerging phishing attacks using xf phishbert. *Sci. Rep.* **15**, 42821. https://doi.org/10.1038/s41598-025-27500-0 (2025).
26. Xu, C., Shen, J. & Du, X. A method of few-shot network intrusion detection based on meta-learning framework. *IEEE Trans. Inf. Forensics Secur.* **15**, 3540–3552. https://doi.org/10.1109/TIFS.2020.2991876 (2020).
27. Jin, W. et al. A prompting multi-task learning-based veracity dissemination consistency reasoning augmentation for few-shot fake news detection. *Eng. Appl. Artif. Intell.* **144**, 110122. https://doi.org/10.1016/j.engappai.2025.110122 (2025).
28. Sridevi, S. et al. Unified hybrid quantum classical neural network framework for detecting distributed denial of service and android mobile malware attacks. *EPJ Quantum Technol.* **12**, 77. https://doi.org/10.1140/epjqt/s40507-025-00380-z (2025).
29. Wu, C. et al. Rethinking membership inference attacks against transfer learning. *IEEE Trans. Inf. Forensics Secur.* **19**, 6441–6454. https://doi.org/10.1109/TIFS.2024.3413592 (2024).
30. Wu, C. et al. WAFBooster: Automatic boosting of WAF security against mutated malicious payloads. *IEEE Trans. Dependable Secur. Comput.* **22**, 1118–1131. https://doi.org/10.1109/TDSC.2024.3429271 (2025).
31. Liang, R. et al. VulSeye: Detect smart contract vulnerabilities via stateful directed graybox fuzzing. *IEEE Trans. Inf. Forensics Secur.* **20**, 2157–2170. https://doi.org/10.1109/TIFS.2025.3537827 (2025).
32. Wu, C. et al. TCG-IDS: Robust network intrusion detection via temporal contrastive graph learning. *IEEE Trans. Inf. Forensics Secur.* **20**, 1475–1486. https://doi.org/10.1109/TIFS.2025.3530702 (2025).
33. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. arXiv:1412.6980 (2014).
34. Qiskit contributors. Qiskit: An open-source framework for quantum computing. https://github.com/Qiskit/qiskit (2024).
35. Jin, W. et al. Veracity-oriented context-aware large language models-based prompting optimization for fake news detection. *Int. J. Intell. Syst.* **2025**, 5920142. https://doi.org/10.1155/int/5920142 (2025).

Author contributions

Mohammed Tawfik: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. Hussam Tarazi: Methodology, Validation, Resources, Writing – review & editing. Ahmad Dalalah: Data curation, Software, Validation. Bajes Zeyad Aljunaeidi: Software, Validation, Formal analysis. Warda M. Shaban: Validation, Writing – review & editing. Islam S. Fathi: Supervision, Project administration, Writing – review & editing

Funding
his research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Declarations

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to M.T.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.