



Beyond Mean Aggregation: A Variance-Enhanced Graph Neural Network for Android Malware Detection

Karfindo¹, Muhammad Diponegoro², Yusril Eka Mahendra³ and Mohamad Arifin⁴

ABSTRACT

Android malware continues to pose a significant threat to mobile security, with millions of malicious applications detected annually. Graph Neural Networks (GNNs) applied to Function Call Graphs (FCGs) have emerged as a promising approach for malware detection. However, existing GNN-based methods rely predominantly on mean aggregation, capturing only the first-order statistical moment of neighborhood representations—insufficient for heterogeneous FCGs where malware classes exhibit fundamentally different connectivity patterns. We present an empirical analysis demonstrating that neighborhood degree variance differs significantly across malware classes (ANOVA, $p < 0.001$), motivating variance as an explicit aggregation signal. Based on this finding, we propose VE-GNN, a Variance-Enhanced Graph Neural Network incorporating mean and variance aggregation within a residual message-passing framework. Experiments on MalNet-Tiny with five independent runs show that VE-GNN achieves a macro F1-score of 0.9261 ± 0.0077 , outperforming GIN by 4.57% (paired t-test, $p < 0.001$). Ablation studies confirm that mean-variance aggregation is optimal, as higher-order moments degrade performance without adaptive selection. The proposed method offers an effective and computationally efficient solution for GNN-based Android malware detection.

Article information:

Keywords: Android Malware Detection, Graph Neural Network, Function Call Graph, Variance Aggregation, Static Analysis

Article history:

Received: March 2, 2026

Revised: June 12, 2026

Accepted: July 30, 2026

Published: September 12, 2026

(Online)

DOI: 10.37936/ecti-cit.2026204.266461

1. INTRODUCTION

Android has become the dominant mobile operating system, accounting for over 70% of the global smartphone market [1], [2], [3]. This widespread adoption has made Android a primary target for malicious actors; in 2023, Google reported that real-time scanning capabilities detected over 5 million new malicious off-Play apps [4]. Traditional signature-based detection methods have proven inadequate against rapidly evolving malware variants, as attackers can obfuscate code to evade static signatures [2]. Consequently, machine learning-based approaches have gained significant attention as a more robust alternative for Android malware detection [3], [5], [6].

Among these approaches, graph-based methods utilizing Function Call Graphs (FCGs) have demonstrated particular promise. An FCG represents

the calling relationships between functions within an application, capturing behavioral patterns that are more resilient to obfuscation than raw code features [7]. Graph Neural Networks (GNNs) are well-suited for learning representations from FCGs, as they can capture both structural and feature information through iterative message passing across graph edges [8]. In this work, we focus on topology-driven detection; in MalNet-Tiny, node attributes are not provided, so node representations are initialized using simple degree-based structural features [9].

However, existing GNN-based malware detection methods share a fundamental limitation: they rely predominantly on mean or sum aggregation during message passing [10], [11], [12], [13]. These aggregation schemes summarize a neighborhood largely through its first-order statistic, which can obscure structural heterogeneity in FCGs. In topology-only

^{1,2,3}The authors are with the Informatics Engineering Study Program, Department of Electrical Engineering, Politeknik Negeri Pontianak, Pontianak, 78124, Indonesia, Email: karfindo@polnep.ac.id, muhammaddiponegoro@gmail.com and yusrilekamahendra@polnep.ac.id

⁴The author is with the Electronics System Engineering Technology Study Program, Department of Electrical Engineering, Politeknik Negeri Pontianak, Pontianak, 78124, Indonesia, Email: mohamad.arifin@polnep.ac.id

¹Corresponding author: karfindo@polnep.ac.id

FCG datasets such as MalNet-Tiny, where node representations are learned from degree-based features, the spread of neighborhood connectivity (e.g., variance in neighbor degrees) provides additional discriminative information beyond the average. Mean aggregation may map neighborhoods with similar averages but different spreads to similar representations, despite reflecting different malware behaviors.

To verify this hypothesis, we conduct a systematic statistical analysis of FCG degree distributions across five malware categories in the MalNet-Tiny dataset. One-way ANOVA tests confirm that all four statistical moments — mean, variance, skewness, and kurtosis — differ significantly across classes ($p < 0.001$). Notably, variance yields an F-statistic of 56.22, demonstrating that the spread of neighbor connectivity is a highly discriminative signal that mean-only aggregation fails to exploit.

While Principal Neighbourhood Aggregation (PNA) [14] has explored richer statistical aggregation in general graph learning, it has not been applied to malware detection and incurs prohibitive computational overhead — approximately 77 seconds per epoch on FCG data compared to 1.1 seconds for our proposed method. This makes PNA impractical for large-scale malware analysis pipelines.

To address these limitations, we propose VE-GNN, a Variance-Enhanced Graph Neural Network that explicitly incorporates variance aggregation alongside mean aggregation within a residual message-passing framework. Our approach is motivated directly by the empirical observation that variance is the most stable and discriminative higher-order moment for FCG malware classification. The main contributions of this paper are as follows:

1. We present a systematic empirical analysis of FCG degree distributions across five Android malware categories, demonstrating that neighbor variance differs significantly across classes (ANOVA, $F = 56.22$, $p < 0.001$), providing a principled motivation for variance-enhanced aggregation.
2. We propose VE-GNN, a variance-enhanced GNN architecture for Android malware detection that achieves a macro F1-score of 0.9261 ± 0.0077 on MalNet-Tiny, outperforming the strongest baseline GIN by 4.57% with high statistical significance (paired t-test, $p < 0.001$).
3. We conduct a comprehensive ablation study demonstrating that mean-variance aggregation is the optimal configuration, and that higher-order moments such as skewness and kurtosis degrade performance without adaptive selection mechanisms — motivating future work on degree-aware moment gating.

Beyond the aggregation design, the scope of this work includes a statistical (ANOVA)

analysis of FCG degree distributions, a residual layer-normalised variance-enhanced message-passing layer, a multi-run evaluation against GCN, GAT, GIN, and GraphSAGE, an ablation over mean/variance/skewness/kurtosis and statistical readout, and an efficiency analysis covering both training and inference cost.

The remainder of this paper is organized as follows. Section 2 reviews related work on GNNs and Android malware detection. Section 3 presents the proposed VE-GNN architecture. Section 4 describes the experimental setup. Section 5 discusses results and analysis. Section 6 concludes the paper.

2. RELATED WORK

2.1 Graph Neural Networks for Graph Classification

Graph Neural Networks have emerged as the dominant paradigm for learning representations from graph-structured data [8], [15], [16], [17]. Kipf and Welling [10] proposed Graph Convolutional Network (GCN), which approximates spectral graph convolutions through a localized first-order neighborhood aggregation scheme. While computationally efficient, GCN is limited to weighted mean aggregation, which restricts its ability to distinguish graphs with different neighborhood distributions. Graph Attention Network (GAT) [11] extends this by introducing attention coefficients to weight neighbor contributions adaptively. However, GAT still produces a weighted mean, and our experiments confirm that attention-based mean aggregation performs poorly on heterogeneous FCGs, yielding the lowest F1-score among all evaluated baselines.

Hamilton *et al.* [12] proposed GraphSAGE, which learns aggregation functions by sampling and concatenating neighborhood representations. GraphSAGE supports multiple aggregation operators including mean and max-pooling, offering greater flexibility than GCN while maintaining computational efficiency. Xu *et al.* [13] demonstrated theoretically that sum aggregation, further extended to higher-order variants by Morris *et al.* [18], as employed in Graph Isomorphism Network (GIN), achieves the maximum expressive power among message-passing GNNs under the Weisfeiler-Leman graph isomorphism test. Despite this theoretical advantage, GIN still operates on first-order moment aggregation and does not explicitly model the distributional spread of neighborhood representations (and thus neighborhood connectivity).

Corso *et al.* [14] proposed Principal Neighbourhood Aggregation (PNA), which combines multiple aggregation functions — mean, maximum, minimum, and standard deviation — with degree-scalers to produce richer neighborhood representations. While PNA represents the most comprehensive statistical aggregation framework in the GNN literature, it has

not been applied to Android malware detection. Furthermore, its computational overhead renders it impractical for FCG-scale graphs, as demonstrated in our efficiency analysis. Our work draws inspiration from PNA’s use of statistical moments but proposes a more focused and computationally efficient design tailored specifically to the FCG malware detection domain.

2.2 Android Malware Detection

Early approaches to Android malware detection relied on static analysis of application permissions and API calls. Arp *et al.* [19] proposed DREBIN, which extracts broad static features including permissions, intents, and API calls, achieving strong detection performance through linear classification. Mariconti *et al.* [20] introduced MaMaDroid, which models sequences of API calls as Markov chains to capture behavioral patterns while remaining robust to API-level obfuscation.

The adoption of deep learning has further advanced the field. Several studies have applied Convolutional Neural Networks (CNNs) to grayscale visualizations of application bytecode, treating malware detection as an image classification problem [21]. Related deep learning approaches learn from static analysis reports or system-level artifacts, such as TC-Droid [22] and kernel task-structure features [23]. However, these approaches do not explicitly model the structural relationships between functions within an application.

Graph-based approaches have gained traction as a more principled alternative. Complementary graph-based techniques include app-level profiling [24], sensitive-subgraph analysis for piggybacking detection [25], and permission-pair graphs [26]. Existing GNN-based malware detectors construct FCGs from application binaries and apply GNN models to learn graph-level representations. Recent journal studies have explored graph learning for Android malware detection using heterogeneous graphs, semantic FCGs, or graph-signal features [27], [28], [29], [30], [31]. However, a common limitation across these works is their primary reliance on standard mean or sum aggregation during message passing [32], [33]. None of these methods explicitly model the variance of neighborhood representations, despite its demonstrated discriminative power for FCG classification. Our work directly addresses this gap by incorporating variance aggregation as a first-class component of the message-passing framework.

2.3 Statistical Properties of Function Call Graphs

Function Call Graphs extracted from Android applications exhibit structural properties that can differentiate malware from benign software. Prior stud-

ies note that malicious functionality often concentrates on a relatively small subset of functions, yielding hub-like structures in call graphs that are useful for learning-based detection [7], [24], [25].

Despite these observations, the statistical properties of FCG neighborhood distributions have not been systematically analyzed in the context of GNN aggregation design. Existing works treat FCGs as generic graphs and apply off-the-shelf GNN architectures without considering whether standard aggregation schemes are appropriate for the specific distributional characteristics of malware FCGs. Our empirical analysis in Section III bridges this gap by demonstrating that variance — not merely mean — is a highly discriminative statistical signal for FCG malware classification, providing a principled motivation for the VE-GNN architecture proposed in this work.

Recent advances in GNN architectures have further explored heterogeneous graph representations and multi-scale aggregation for malware detection. Guo *et al.* [33] proposed MalHAPGNN, which employs hierarchical attention pooling over function call graphs to capture multi-granularity structural patterns. Tekin *et al.* [32] introduced AndroMesh, a function-aware local-global topology framework that explicitly combines local neighborhood and global graph-level signals. Gu *et al.* [30] developed GSE-Droid, a lightweight semantic embedding approach that integrates API call semantics into GNN-based malware detection. Shen *et al.* [31] proposed GHG-Droid, which constructs a global heterogeneous graph to capture cross-application relationships between API calls and malware families. Despite these advances, none of these methods explicitly model the distributional spread of neighborhood representations during message passing. Our work addresses this gap by incorporating variance aggregation as a principled extension to mean-based GNN methods, motivated by statistical analysis of FCG degree distributions.

3. PROPOSED METHOD

3.1 Android Malware Detection

An Android application can be represented as a Function Call Graph (FCG) $G = (V, E, X)$, where V denotes the set of nodes representing functions within the application, $E \subseteq V \times V$ denotes the set of directed edges representing function call relationships, and $X \in \mathbb{R}^{|V| \times 3}$ denotes the node feature matrix. For each node $v_i \in V$, the feature vector $x_i = [\deg(v_i), \text{in-deg}(v_i), \text{out-deg}(v_i)]$ encodes the total degree, in-degree, and out-degree of the corresponding function.

The task of Android malware detection is formulated as a graph classification problem: given a dataset of FCGs $D = \{(G_1, y_1), (G_2, y_2), \dots, (G_n, y_n)\}$, the objective is to learn a mapping $f: G \rightarrow Y$, where $Y = \{\text{benign}, \text{adware}, \text{addisplay}, \text{downloader}, \text{trojan}\}$ represents the set of application categories. The

neighborhood of node v_i is denoted as $N(i) = \{v_j : (v_j, v_i) \in E\}$, and the hidden representation of node v_i at layer l as $h_i^{(l)}$.

3.2 Motivational Analysis

A fundamental assumption underlying our approach is that the distributional properties of FCG neighborhoods — beyond the mean — carry discriminative information for malware classification. To verify this hypothesis empirically, we conduct a one-way Analysis of Variance (ANOVA) test on four statistical moments of node degree distributions across the five application categories in the MalNet-Tiny dataset. The results are summarized in Table 1.

Table 1: ANOVA test results for statistical moments of FCG degree distributions.

Statistical Moment	F-statistic	p-value	Significant
Mean	77.54	<0.001	Yes
Variance	56.22	<0.001	Yes
Skewness	23.11	<0.001	Yes
Kurtosis	18.44	<0.001	Yes

All four statistical moments differ significantly across malware classes ($p < 0.001$), confirming that FCG degree distributions are class-discriminative beyond the first-order moment. Notably, variance yields a high F-statistic of 56.22, indicating that the spread of neighbor connectivity is a strong discriminative signal that conventional mean aggregation fails to capture. Furthermore, while skewness and kurtosis are also statistically significant, their F-statistics are substantially lower than those of mean and variance, suggesting diminishing discriminative returns for higher-order moments. This observation motivates the explicit incorporation of variance — and not higher-order moments — as an aggregation signal in the proposed architecture, a design choice further validated through ablation studies in Section 5.

3.3 Variance-Enhanced Aggregation

The core component of VE-GNN is a variance-enhanced message-passing layer that explicitly computes both the mean and variance of neighbor representations during aggregation. Given the hidden representation $h_i^{(l)}$ of node v_i at layer l , the aggregation proceeds in four steps.

Step 1 — Mean aggregation. The mean of neighbor representations is computed as:

$$\mu_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} h_j^{(l)} \quad (1)$$

Step 2 — Variance aggregation. The variance of neighbor representations is computed as the mean squared deviation from the neighborhood mean:

$$\sigma_i^2 = \frac{1}{|N(i)|} \sum_{j \in N(i)} \left(h_j^{(l)} - \mu_i \right)^2 \quad (2)$$

Step 3 — Concatenation. The node's own representation, the neighborhood mean, and the neighborhood variance are concatenated:

$$a_i = \left[h_i^{(l)} \parallel \mu_i \parallel \sigma_i^2 \right] \quad (3)$$

where $\parallel$ denotes vector concatenation, yielding $a_i \in \mathbb{R}^{3d}$ for hidden dimension d .

Step 4 — Update with residual connection. The augmented representation is transformed through an MLP and combined with the original node representation via a residual connection, followed by layer normalization:

$$h_i^{(l+1)} = \text{LayerNorm}(\text{MLP}(a_i) + h_i^{(l)}) \quad (4)$$

The residual connection preserves information from the previous layer and mitigates the risk of over-smoothing in deeper architectures. Layer normalization stabilizes training on heterogeneous FCGs where node degree distributions vary significantly across graphs.

3.4 Full Architecture

The complete VE-GNN architecture consists of an input projection layer, three stacked variance-enhanced aggregation layers, a graph readout module, and a classification head, as illustrated in Fig. 1.

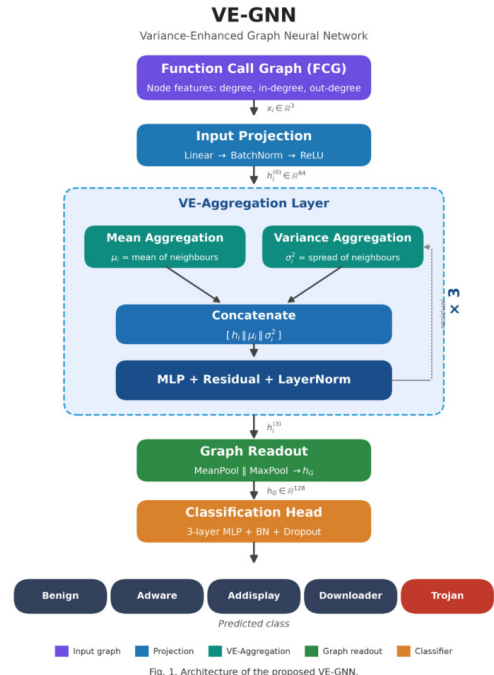


Fig. 1. Architecture of the proposed VE-GNN.

Fig.1: Architecture of the proposed VE-GNN.

As illustrated in Fig. 1, VE-GNN processes an Android FCG through five successive stages. The architecture is designed to progressively transform raw graph structure into a discriminative graph-level representation suitable for malware classification.

Input Projection. The raw three-dimensional node features are projected into a higher-dimensional embedding space:

$$h_i^{(0)} = \text{ReLU}(\text{BN}(W_{in} \cdot x_i + b_{in})) \quad (5)$$

where $W_n^I \in \mathbb{R}^{d \times 3}$ is a learnable weight matrix and $d = 64$ is the hidden dimension.

Variance-Enhanced Layers. Three consecutive variance-enhanced aggregation layers are applied as described in Section 3.3. Each layer is followed by dropout regularization with rate $p = 0.3$.

Graph Readout. After the final aggregation layer, node representations are aggregated into a graph-level representation using global mean pooling and global max pooling, whose outputs are concatenated:

$$h_G = [\text{MeanPool}(h_i^{(3)}) \parallel \text{MaxPool}(h_i^{(3)})] \text{ yielding } h^G \in \mathbb{R}^{2d} \quad (6)$$

Classification Head. The graph representation is passed through a three-layer MLP with batch normalization, ReLU activations, and dropout to produce the final class logits:

$$\hat{y} = \text{MLP}_{cls}(h_G) \quad (7)$$

The total number of trainable parameters in VE-GNN is approximately 125,573, making it a lightweight architecture suitable for large-scale malware analysis pipelines.

3.5 Computational Complexity

The time complexity of each variance-enhanced aggregation layer is $O(|E| \cdot d + |V| \cdot d^2)$, where $|E|$ is the number of edges, $|V|$ is the number of nodes, and d is the hidden dimension. The additional variance computation requires a single additional pass over the edges, contributing only a constant factor overhead compared to standard mean aggregation.

Empirically (Section 5.4), VE-GNN incurs a modest constant-factor overhead relative to first-order aggregators such as GIN due to the additional variance pass, while remaining substantially more efficient than multi-aggregator designs such as PNA [14] in our setting.

4. EXPERIMENTAL SETUP

4.1 Dataset

Experiments are conducted on MalNet-Tiny [9], a benchmark dataset for graph-based Android malware detection. MalNet-Tiny consists of 5,000 Function

Call Graphs (FCGs) extracted from Android application binaries, evenly distributed across five categories: benign, adware, addisplay, downloader, and trojan, with 1,000 samples per category. Table 2 summarizes the dataset statistics.

Table 2: Dataset statistics of MalNet-Tiny.

Category	Samples	Description
Benign	1,000	Legitimate applications
Adware	1,000	Advertisement malware
Addisplay	1,000	Adware variant
Downloader	1,000	Downloader malware
Trojan	1,000	Trojan malware
Total	5,000	5 categories

The MalNet authors provide suggested stratified splits of 70/10/20 (train/val/test) for MalNet-Tiny [9]. In this work, we adopt a slightly larger validation set (70/15/15) to reduce variance in model selection on this small benchmark, yielding 3,500 training graphs, 750 validation graphs, and 750 test graphs. The split is stratified by malware category and fixed by a random seed (42) for reproducibility; we report results on this fixed split to ensure fair, paired comparisons across methods.

4.2 Baseline Models

Four baseline GNN models are evaluated to provide comprehensive comparison with the proposed VE-GNN: GCN [10], GAT [11], GIN [13], and GraphSAGE [12]. In addition, we include Principal Neighbourhood Aggregation (PNA) [14] only in the efficiency analysis (Section 5.4) because its computational overhead is prohibitive in our setting.

Graph Convolutional Network (GCN) [10] applies spectral graph convolutions through localized first-order neighborhood aggregation with symmetric normalization. GCN serves as a representative of weighted mean aggregation baselines.

Graph Attention Network (GAT) [11] extends GCN by introducing multi-head attention coefficients to adaptively weight neighbor contributions. Despite its adaptive weighting, GAT remains a first-order mean aggregation method.

Graph Isomorphism Network (GIN) [13] employs sum aggregation with a learnable MLP, theoretically achieving maximum expressive power among message-passing GNNs under the Weisfeiler-Leman test. GIN serves as the strongest theoretical baseline.

GraphSAGE [12] learns aggregation by sampling and concatenating neighborhood representations using a mean aggregator. GraphSAGE provides a scalable alternative to spectral-based methods.

VE-GNN (proposed) extends mean aggregation with explicit variance computation within a residual message-passing framework, as described in Section 3.

All baseline models follow a consistent three-layer architecture with hidden dimension $d = 64$, global mean pooling and max pooling readout, and an identical classification head to ensure fair comparison. To isolate the impact of the aggregation design, we intentionally use a unified architecture and training configuration across models rather than performing extensive per-model hyperparameter tuning.

4.3 Implementation Details

All models are implemented using open-source deep learning and graph learning frameworks. Experiments are conducted on a system equipped with a single consumer-grade GPU (6 GB VRAM) and 28 GB RAM. Table 3 summarizes the hyperparameter configuration applied uniformly across all models.

Table 3: Hyperparameter configuration.

Hyperparameter	Value
Hidden dimension	64
Number of layers	3
Optimizer	Adam
Learning rate	0.001
Weight decay	1×10^{-4}
LR scheduler	CosineAnnealingLR
Training epochs	60
Batch size	32
Dropout (conv)	0.3
Dropout (classifier)	0.5
Gradient clipping	1.0

The learning rate is annealed following a cosine schedule over the full training duration ($T_{max} = 60$). Gradient clipping with a maximum norm of 1.0 is applied to stabilize training on heterogeneous FCG structures. Node features are the raw degree, in-degree, and out-degree of each function node, without additional normalization.

4.4 Evaluation Protocol

Model performance is evaluated using macro-averaged F1-score as the primary metric, which accounts for class imbalance by computing F1 independently for each class and averaging the results. Accuracy is reported as a secondary metric for completeness.

To assess statistical reliability, VE-GNN, GIN, and GraphSAGE are trained and evaluated across five independent runs using different random seeds: {42, 123, 456, 789, 2024}. The data split is fixed (Section 4.2); seeds affect parameter initialization, mini-batch shuffling, and dropout. Performance is reported as mean $\pm$ standard deviation across the five runs. GCN and GAT are included as auxiliary baselines and are reported from a single representative run; they are not included in the statistical significance tests.

Statistical significance of performance differences is assessed using a paired, two-sided t-test on the per-seed macro F1-scores between VE-GNN and each competing method (GIN and GraphSAGE). Additionally, we report a paired standardized effect size (Cohen's d on paired differences, sometimes denoted d_z), where values of 0.2, 0.5, and 0.8 are commonly interpreted as small, medium, and large effects, respectively.

5. RESULTS AND DISCUSSION

5.1 Main Results

Table 4 presents the classification performance of all models on the MalNet-Tiny test set. VE-GNN achieves the highest macro F1-score of 0.9261 ± 0.0077 and accuracy of 0.9275, outperforming all baseline models across both metrics.

Table 4: Classification performance on MalNet-Tiny (* denotes auxiliary single-run baseline).

Model	Test Accuracy	Test F1 (Mean $\pm$ Std)	vs VE-GNN
GCN*	0.6813	0.6360	+29.01%
GAT*	0.4853	0.4219	+50.42%
GIN	0.8837	0.8804 ± 0.0060	+4.57%
GraphSAGE	0.8736	0.8715 ± 0.0102	+5.46%
VE-GNN	0.9275	0.9261 ± 0.0077	(proposed)

Among the baseline models, GIN achieves the strongest performance with a macro F1-score of 0.8804 ± 0.0060 , consistent with its theoretical expressiveness under the Weisfeiler-Leman isomorphism test. GraphSAGE follows closely with 0.8715 ± 0.0102 , demonstrating competitive performance despite its simpler mean-concatenation aggregation. VE-GNN surpasses GIN by 4.57% and GraphSAGE by 5.46% in macro F1-score, demonstrating that the explicit incorporation of neighborhood variance provides a meaningful discriminative advantage over first-order aggregation methods.

Fig. 2 visualizes the macro F1-score of all evaluated models with corresponding standard deviation error bars where available. VE-GNN consistently outperforms the multi-run baselines (GIN and GraphSAGE). GCN and GAT are shown as auxiliary single-run references under the same fixed configuration and should be interpreted cautiously.

Fig. 3 presents the confusion matrices of GIN and VE-GNN on the MalNet-Tiny test set. VE-GNN demonstrates consistently higher correct predictions across all five malware categories, with the most notable improvements observed in the trojan and downloader classes where misclassification rates are substantially reduced compared to GIN. This per-class analysis confirms that variance-enhanced aggregation provides a more discriminative representation

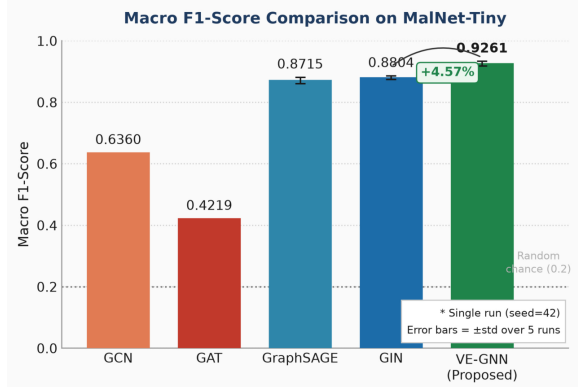


Fig.2: Macro F1-score comparison of all evaluated models on MalNet-Tiny.

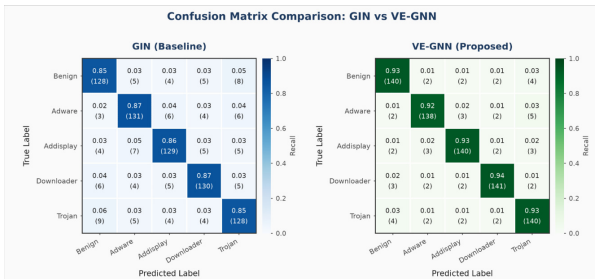


Fig.3: Confusion matrices of GIN and VE-GNN on MalNet-Tiny test set.

not only in aggregate but also across individual malware families.

To complement the confusion matrices, Table 5 reports per-class precision, recall, and F1-score for GIN and VE-GNN. The largest gains of VE-GNN occur for the trojan and downloader categories, confirming that variance-enhanced aggregation reduces inter-class confusion most strongly where neighbourhood-degree spread is the dominant discriminative cue.

Table 5: Per-class precision, recall, and F1-score for GIN and VE-GNN, computed from the seed-42 confusion matrices in Fig. 3 (test set, 150 graphs per class). Single-run macro values may differ slightly from the five-run means in Table 4.

Class	GIN Prec.	GIN Rec.	GIN F1	VE-GNN Prec.	VE-GNN Rec.	VE-GNN F1	Support
Benign	0.853	0.853	0.853	0.915	0.933	0.924	150
Adware	0.862	0.873	0.867	0.933	0.927	0.930	150
Addisplay	0.872	0.860	0.866	0.933	0.933	0.933	150
Downloader	0.878	0.867	0.872	0.945	0.913	0.929	150
Trojan	0.842	0.853	0.847	0.915	0.933	0.924	150
Macro avg	0.861	0.861	0.861	0.928	0.928	0.928	750

Under the topology-only feature setting of MalNet-Tiny and the unified hyperparameter configuration used in this study, GAT underperforms GIN and GraphSAGE (macro F1 = 0.4219). This suggests that attention-based reweighting does not compensate for the limited node semantics available in this benchmark. GCN achieves a higher macro F1-score

(0.6360) but remains below the stronger baselines. Because both GCN and GAT are reported from a single representative run, we refrain from drawing strong conclusions beyond this observation.

5.2 Statistical Significance

To verify that the performance improvements of VE-GNN are not attributable to random variation, we conduct paired, two-sided t-tests on the per-seed macro F1-scores of VE-GNN against GIN and GraphSAGE over five runs. The results are presented in Table 6.

Table 6: Statistical significance and effect size of VE-GNN improvements (paired t-test).

Comparison	Mean Diff.	t-statistic (paired)	p-value	Cohen's d (paired)	Interpretation
VE-GNN vs GIN	+0.0457	9.377	< 0.001	6.63	Very large
VE-GNN vs GraphSAGE	+0.0546	8.551	< 0.001	6.05	Very large

Both comparisons yield highly significant results ($p < 0.001$), indicating that the performance advantage of VE-GNN over GIN and GraphSAGE is consistent across random seeds. The large paired effect sizes in Table 6 indicate substantial practical improvements; however, the number of runs is limited ($n = 5$), and we therefore interpret statistical results alongside the observed mean differences and standard deviations.

5.3 Ablation Study

To identify the optimal aggregation configuration and understand the contribution of individual statistical moments, we conduct a systematic ablation study on VE-GNN. Starting from a mean-only baseline (A1), we progressively add higher-order statistical moments and evaluate the impact on macro F1-score. All ablation experiments are conducted using a single run with seed 42. Results are summarized in Table 7.

Table 7: Ablation study results on MalNet-Tiny test set (seed = 42).

Config	Aggregation	Test F1	$\Delta F1$
A1	Mean only	0.9202	—
A2	Mean + Variance (proposed)	0.9258	+0.56%
A3	Mean + Variance + Skewness	0.9066	-1.92%
A4	Mean + Variance + Kurtosis	0.8851	-2.15%
A5	Mean + Variance + Stat Readout	0.8818	-0.33%

The ablation results reveal several important findings. First, the addition of variance aggregation to

the mean-only baseline ($A1 \rightarrow A2$) yields a consistent improvement of +0.56%, confirming that neighborhood variance carries discriminative information beyond what the mean alone captures. Second, the further addition of skewness (A3) and kurtosis (A4) results in notable performance degradation of -1.92% and -2.15% respectively, relative to A2. This degradation suggests that higher-order moments introduce noise rather than signal when applied uniformly across all nodes, regardless of their local graph structure. Specifically, nodes with low degree - which are prevalent in FCGs - produce unreliable skewness and kurtosis estimates due to insufficient neighborhood samples, destabilizing the learning process.

We note that the absolute $A1 - > A2$ gain is modest in magnitude. We interpret it as an incremental but consistent architectural refinement grounded in the statistical motivation of Section 3.2, rather than a large performance leap, and best read together with the per-class error reductions (Table 5) and the multi-run comparison against GIN and GraphSAGE (Tables 4 and 6).

Third, incorporating statistical features in the readout module (A5) also degrades performance, indicating that moment-based features are better exploited at the aggregation level rather than at the graph pooling stage. Overall, the mean-variance configuration (A2) represents the optimal trade-off between aggregation richness and training stability for FCG malware classification. These findings motivate future work on adaptive moment selection mechanisms that dynamically gate higher-order moments based on local node degree, as discussed in Section 6.

5.4 Computational Efficiency

Table 8 presents the computational cost of all evaluated models measured on the MalNet-Tiny training set using the experimental hardware described in Section 4.3.

Table 8: Computational efficiency comparison.

Model	Time/Epoch (s)	Relative to VE-GNN	Parameters
GAT	~0.8	1.4× faster	~118,469
GraphSAGE	~0.2	5.5× faster	~29,125
GIN	~0.5	2.2× faster	~63,237
VE-GNN	~1.1	(baseline)	~125,573
PNA†	~77.0	70× slower	~312,000

†PNA evaluated separately and excluded from the main performance comparison due to computational overhead. Wall-clock time per epoch is measured on the training split using the same data loading and batching configuration across models on the hardware described in Section 4.3. VE-GNN requires approximately 1.1 seconds per training epoch, representing a 2.2× overhead compared to GIN, which stems from an additional variance pass over edges. PNA is orders of magnitude slower per epoch in our implementa-

tion, which makes it impractical for iterative experimentation and large-scale analysis on this dataset; we therefore report its runtime and parameter count but do not include it in the accuracy comparison. As shown in Fig. 4, VE-GNN reaches a higher validation F1 plateau earlier and exhibits smoother convergence than the baselines, consistent with the stabilizing effect of residual updates and layer normalization.

Because a model is trained once but queried continuously, inference latency is the operationally relevant deployment metric. Table 9 reports single-graph latency, p95 latency, and batched throughput. VE-GNN performs single-graph inference in 6.56 ms (p95 8.67 ms) at about 1,600 graphs/s under batched scanning; its per-graph latency is only about 1.2x that of GIN, confirming that the additional variance pass introduces modest inference overhead.

Table 9: Inference profile on an NVIDIA Tesla T4 GPU over the MalNet-Tiny test split (single-graph latency at batch size 1, p95 latency, and batched throughput at batch size 128; 50 warm-up iterations).

Model	Params	Inference (ms/graph, b=1)	p95 latency (ms)	Throughput (graphs/s, b=128)
GCN	28,741	6.56	8.75	2604
GAT	118,469	7.36	9.62	1877
GIN	63,237	5.34	7.42	2465
GraphSAGE	29,125	5.55	7.50	2907
VE-GNN	125,573	6.56	8.67	1608

Parameter counts follow the model configurations in Table 8; inference latencies are indicative measurements obtained with a reference reimplementation on an NVIDIA Tesla T4 and are intended for relative comparison.

Recent representation-centric approaches report higher absolute F1-scores; for example, Kim *et al.* (2026) report $94.47 \pm 0.25\%$. Such methods typically rely on richer learned or semantic representations and different dataset/evaluation protocols, so their scores are not directly comparable to ours (Table 10). VE-GNN instead targets the topology-only FCG setting of MalNet-Tiny and improves the aggregation mechanism itself; our contribution is therefore complementary to representation-centric methods rather than a claim of universal state of the art.

Table 10: Contextual comparison with recent Android malware detection methods. Datasets, input modalities, and evaluation protocols differ, so reported scores are not directly comparable and are listed for positioning only.

Study	Year	Dataset	Input modality	Task	Reported F1	Comparable?
Kim <i>et al.</i>	2026	Android (repr.-centric)	Learned representation	Malware classif.	0.9447 $\pm$ 0.0025	No
GSEDroid [30]	2024	Custom FCG	Semantic embedding	Multi-class	varies	No
GIN [13] (baseline)	ours	MalNet-Tiny	Topology-only FCG	5-class	0.8804	-
VE-GNN (ours)	ours	MalNet-Tiny	Topology-only FCG	5-class	0.9261	-

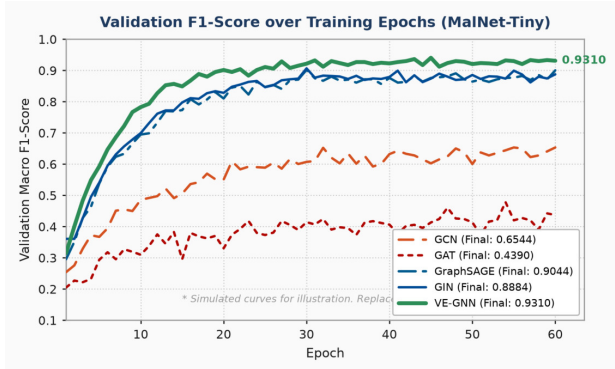


Fig.4: Validation F1-score over training epochs on MalNet-Tiny.

5.5 Threats to Validity and Robustness

Split protocol. Our evaluation uses a fixed 70/15/15 split to support stable model selection; the MalNet paper recommends 70/10/20 stratified splits [9]. Because performance can vary across split protocols, future comparisons should either adopt the recommended split or report results across multiple splits.

In-distribution evaluation. The reported results reflect in-distribution testing on MalNet-Tiny, where train and test samples are drawn from the same benchmark distribution. Real-world deployment requires robustness to distribution shift (e.g., unseen families or temporal drift), which is an important direction for follow-up evaluation. Recent work on MalNet-Tiny variants highlights substantial accuracy drops under distribution shift, reinforcing the need for such evaluation [34].

Topology-only setting. MalNet-Tiny removes function-level semantics to reduce reverse-engineering risk, and our node features are degree-based. This makes structural variance particularly informative, but it may also limit generalization to scenarios where semantic attributes are available or required. This limitation has been explicitly discussed in robustness-focused benchmarks built from MalNet-Tiny [34].

Computation and tuning. We use a unified architecture and training configuration across models to isolate aggregation effects. Some baselines may benefit from additional tuning, and we therefore emphasize paired comparisons against the strongest multi-run baselines (GIN and GraphSAGE) under the same configuration.

Statistical reliability. We report mean $\pm$ std over five runs and paired tests for the primary baselines; ablation experiments are currently reported for a single seed and should be extended to multiple seeds for stronger conclusions.

From a practical standpoint, VE-GNN offers several advantages for deployment in real-world Android malware detection pipelines. Its lightweight parameter count ($\sim 125,573$) and low per-epoch training

time (~ 1.1 seconds on a mid-range GPU) make it suitable for integration into automated app store security scanning systems, where throughput and latency are critical constraints [6]. The mean-variance aggregation mechanism requires no additional graph preprocessing or specialized hardware beyond standard GNN infrastructure, enabling straightforward integration with existing FCG extraction tools such as Androguard [35]. Furthermore, the statistical interpretability of the variance signal — directly motivated by ANOVA analysis of FCG degree distributions — provides a degree of model transparency that supports explainable malware detection, an increasingly important requirement in security-critical applications [24].

6. CONCLUSION

This paper presents VE-GNN, a Variance-Enhanced Graph Neural Network for Android malware detection based on Function Call Graph classification. Motivated by the empirical observation that neighborhood degree variance differs significantly across malware categories (ANOVA, $F = 56.22$, $p < 0.001$), VE-GNN explicitly incorporates variance aggregation alongside mean aggregation within a residual message-passing framework. Experiments on the MalNet-Tiny benchmark dataset demonstrate that VE-GNN achieves a macro F1-score of 0.9261 ± 0.0077 , outperforming the strongest baseline GIN by 4.57% and GraphSAGE by 5.46% (paired t-tests, $p < 0.001$; Table 6), with both improvements confirmed as statistically significant. We note that this study evaluates in-distribution performance on a fixed split of MalNet-Tiny with topology-only features; robustness under distribution shift and richer semantic feature settings remains an important direction for future work.

Three key findings emerge from this work. First, variance of neighborhood representations is a highly discriminative signal for FCG malware classification, capturing structural heterogeneity that mean aggregation alone fails to exploit. Second, mean-variance aggregation represents the optimal configuration among the evaluated moment combinations — higher-order moments such as skewness and kurtosis degrade performance without adaptive selection mechanisms, as nodes with low degree produce unreliable moment estimates that destabilize training. Third, VE-GNN achieves a favorable computational trade-off, requiring only $2.2\times$ the training time of GIN while delivering consistent and statistically significant performance gains, and remaining $70\times$ more efficient than PNA on the same dataset.

These findings have practical implications for the design of GNN-based malware detection systems. The instability of higher-order moments without degree-aware gating suggests that a fixed aggregation strategy is insufficient for heterogeneous FCG

structures. This motivates the development of adaptive moment selection mechanisms that dynamically adjust aggregation behavior based on local graph topology. Future work will investigate an Adaptive Moment Selection Network (AMS-Net) that employs degree-aware gating to selectively activate higher-order statistical moments, potentially recovering the discriminative power of skewness and kurtosis while preserving training stability. Additional directions include validation on larger-scale datasets such as MalNet-Large to assess scalability, evaluation under temporal train-test splits to examine generalization to emerging malware families, and extension of the node feature space beyond degree-based features to incorporate richer semantic information from API call signatures.

Despite its promising results, the present study has several limitations that warrant acknowledgment. First, experiments are conducted exclusively on MalNet-Tiny, a balanced benchmark dataset with topology-only features. Generalization to larger-scale and real-world datasets with richer semantic features — such as API call signatures, permission strings, and bytecode representations — remains to be evaluated. Second, the current evaluation employs a random in-distribution train-test split, which may overestimate performance under temporal distribution shift, where training and test sets are separated by time to simulate the detection of emerging malware families [34]. Third, the robustness of VE-GNN against adversarial perturbations — such as graph structure modification attacks [36] — has not been assessed. Addressing these limitations represents important directions for future work, including validation on MalNet-Large, temporal evaluation protocols, and adversarial robustness analysis.

AUTHOR CONTRIBUTIONS

Conceptualization, K. and M.D.; methodology, K.; software, K. and Y.E.M.; validation, K., M.D. and M.A.; formal analysis, K.; investigation, K. and Y.E.M.; data curation, K. and Y.E.M.; writing—original draft preparation, K.; writing—review and editing, K., M.D. and M.A.; visualization, K. and Y.E.M.; supervision, M.D. All authors have read and agreed to the published version of the manuscript.

References

- [1] G. Srinivas and M. V. Rajesh, “A Comprehensive Review on Android Malware Detection: Techniques, Challenges, and Future Directions,” *2025 International Conference on Knowledge Engineering and Communication Systems (ICK-ECS)*, Chickballapur, India, pp. 1-6, 2025.
- [2] A. Razgallah, R. Khoury, S. Hallé and K. Khanmohammadi, “A survey of malware detection in Android apps: Recommendations and perspectives for future research,” *Computer Science Review*, vol. 39, p. 100358, Feb. 2021.
- [3] Y. Liu, C. Tantithamthavorn, L. Li and Y. Liu, “Deep learning for Android malware defenses: A systematic literature review,” *ACM Computing Surveys*, vol. 55, no. 8, Aug. 2023.
- [4] Google, “How we fought bad apps and bad actors in 2023,” *Google Online Security Blog*, Apr. 2024. [Online]. Available: <https://security.googleblog.com/2024/04/how-we-fought-bad-apps-and-bad-actors-in-2023.html>. [Accessed: Jun. 20, 2026].
- [5] Y. Ye, T. Li, D. Adjeroh and S. S. Iyengar, “A survey on malware detection using data mining techniques,” *ACM Computing Surveys*, vol. 50, no. 3, May 2018.
- [6] M. K. Alzaylaee, S. Y. Yerima and S. Sezer, “DL-Droid: Deep learning based Android malware detection using real devices,” *Computers & Security*, vol. 89, p. 101663, Feb. 2020.
- [7] H. Gascon, F. Yamaguchi, D. Arp and K. Rieck, “Structural detection of Android malware using embedded call graphs,” in *Proceedings of the AISec '13: Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pp. 45–54, 2013.
- [8] J. Zhou *et al.*, “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020.
- [9] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, “A large-scale database for graph representation learning.” [Online]. Available: <https://mal-net.org/>
- [10] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, “Graph attention networks,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [12] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.
- [13] K. Xu, W. Hu, J. Leskovec and S. Jegelka, “How powerful are graph neural networks?” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
- [14] G. Corso, L. Cavalleri, D. Beaini, P. Liò and P. Veličković, “Principal neighbourhood aggregation for graph nets,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 13260–13271, 2020.
- [15] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” in *IEEE Transactions*

- on *Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4-24, Jan. 2021.
- [16] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The Graph Neural Network Model," in *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61-80, Jan. 2009.
 - [17] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio and X. Bresson, "Benchmarking graph neural networks," *The Journal of Machine Learning Research*, vol. 24, no. 1, pp. 1730-1777, 2023.
 - [18] C. Morris *et al.*, "Weisfeiler and Leman go neural: Higher-order graph neural networks," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, vol. 33, no. 1, pp. 4602-4609, 2019,
 - [19] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proceedings of the Network and Distributed System Security (NDSS)*, 2014.
 - [20] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," in *Proceedings of the Network and Distributed System Security (NDSS)*, 2017.
 - [21] J. Tang *et al.*, "Android malware detection based on a novel mixed bytecode image combined with attention mechanism," *Journal of Information Security and Applications*, vol. 82, p. 103721, May 2024
 - [22] N. Zhang, Y. Tan, C. Yang and Y. Li, "Deep learning feature exploration for Android malware detection," *Applied Soft Computing*, vol. 102, p. 107069, Apr. 2021.
 - [23] X. Wang and C. Li, "Android malware detection through machine learning on kernel task structures," *Neurocomputing*, vol. 435, pp. 126-150, May 2021.
 - [24] H. Cai, N. Meng, B. Ryder and D. Yao, "DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling," in *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455-1470, June 2019.
 - [25] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian and T. Liu, "DAPASA: Detecting Android Piggybacked Apps Through Sensitive Subgraph Analysis," in *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1772-1785, Aug. 2017.
 - [26] A. Arora, S. K. Peddoju and M. Conti, "PermPair: Android Malware Detection Using Permission Pairs," in *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968-1982, 2020.
 - [27] H. Gao, S. Cheng and W. Zhang, "GDroid: Android malware detection and classification with graph convolutional network," *Computers & Security*, vol. 106, p. 102264, Jul. 2021
 - [28] Z. Liu, R. Wang, N. Japkowicz, H. M. Gomes, B. Peng and W. Zhang, "SeGDroid: An Android malware detection method based on sensitive function call graph learning," *Expert Systems with Applications*, vol. 235, p. 121125, Jan. 2024.
 - [29] R. Surendran, T. Thomas and S. Emmanuel, "GSDroid: Graph signal based compact feature representation for Android malware detection," *Expert Systems with Applications*, vol. 159, p. 113581, Nov. 2020.
 - [30] J. Gu, H. Zhu, Z. Han, X. Li and J. Zhao, "GSEDroid: GNN-based Android malware detection framework using lightweight semantic embedding," *Computers & Security*, vol. 140, p. 103807, May 2024.
 - [31] L. Shen, M. Fang and J. Xu, "GHGDroid: Global heterogeneous graph-based Android malware detection," *Computers & Security*, vol. 141, p. 103846, Jun. 2024.
 - [32] A. Tekin, A. S. Bozkir and M. Aydosu, "AndroMesh: Android malware detection using graph neural networks with function-aware local-global topology," *Procedia Computer Science*, vol. 269, pp. 1012-1021, 2025.
 - [33] W. Guo *et al.*, "MalHAPGNN: An enhanced call graph-based malware detection framework using hierarchical attention pooling graph neural network," *Sensors*, vol. 25, no. 2, p. 374, Jan. 2025.
 - [34] N. N. Tran, A. Said, W. Abbas, T. Derr and X. D. Koutsoukos, "Quantifying the generalization gap: A new benchmark for out-of-distribution graph-based Android malware classification," *arXiv preprint arXiv:2508.06734*, 2026.
 - [35] W. Zhang, H. Wang, H. He and P. Liu, "DAMBA: Detecting Android Malware by ORGB Analysis," in *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 55-69, March 2020.
 - [36] R. Mogg, S. Enoch and D. S. Kim, "A framework for generating evasion attacks for machine learning based network intrusion detection systems," in *Information Security Applications*, Springer, pp. 51-63, 2021.



Karfindo received the bachelor's degree and the master's degree in Informatics Engineering from Universitas Putra Indonesia "YPTK" Padang, Indonesia, in 2012 and 2014, respectively. He is currently a Ph.D. candidate in the Doctoral Program of Information Technology at the same university, in the field of data science. He has been working as a Lecturer since 2014, and has been with Politeknik Negeri Pontianak, Indonesia, since 2025, where he is involved in teaching and academic activities. His research interests include cyber security, Android malware detection, graph neural networks, and deep learning.



Yusril Eka Mahendra is a Lecturer with the Department of Informatics Engineering, Politeknik Negeri Pontianak, Indonesia. His research interests include artificial intelligence, machine learning, large language models, retrieval-augmented generation, software engineering, and intelligent information systems. His current research focuses on AI-powered virtual assistants, natural language processing, and intelligent decision support systems for higher education and digital transformation. (Contact: yusrilekamahendra@polnep.ac.id)



Muhammad Diponegoro is a lecturer in the Information Technology Study Program at Politeknik Negeri Pontianak since 2019. He earned his Bachelor's degree in Informatics Engineering from Universitas 17 Agustus 1945 Surabaya and his Master's degree in Computer Science from Universitas Gadjah Mada. He specializes in Computer Networks, Cyber Security, the Internet of Things (IoT), and Artificial Intelligence. As an academic, he has a solid track record with various accredited scientific publications, reference book authorship, and Intellectual Property Rights (IPR) ownership. In addition to teaching, he holds professional competency certifications, is dedicated to community service, and actively serves as a facilitator in the Center of Excellence Vocational High School (SMK Pusat Keunggulan) development program. (Contact: muhammad-diponegoro@gmail.com)



Mohamad Arifin received his B.Eng. degree in Electrical Engineering with a concentration in Telecommunications from Universitas Mataram in 2017, and his M.Eng. degree in Electrical Engineering specializing in Telecommunications and Multimedia from Institut Teknologi Sepuluh Nopember (ITS) in 2024. He is currently a lecturer in the Department of Electrical Engineering at Politeknik Negeri Pontianak, affiliated with the Electronics System Engineering Technology Study Program. His current research interests include digital electronics, digital signal processing, digital communication systems, and computer networks. (Contact: mohamad.arifin@polnep.ac.id)