Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Classifier-Guided Discrete Diffusion for Malware Family-Conditional Graph Generation

Chandler Lu
San Jose State University
Younghee Park
San Jose State University
Fabio Di Troia
fabio.ditroia@sjsu.edu

San Jose State University

Research Article

Keywords: Discrete graph diffusion, Malware API call graphs, Directed weighted graphs, Synthetic graph generation

Posted Date: April 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9226468/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Journal of Computer Virology and Hacking Techniques on October 24th, 2026. See the published version at https://doi.org/10.1007/s11416-026-00680-7.

Classifier-Guided Discrete Diffusion for Malware Family-Conditional Graph Generation

Abstract

We study whether discrete graph diffusion can be adapted to generate directed, weighted malware API call transition graphs in the low-data regime typical of dynamic malware analysis. Each execution trace is converted into a behavior graph whose nodes represent API types and whose directed edges represent transition count frequencies. We adapt a graph generative model, DiGress, to this setting by removing undirected symmetry constraints, modeling dense directed adjacency tensors with masking for variable graph sizes, and introducing an ordinal, marginal-aware diffusion kernel for edge-weight categories to respect the numeric structure of counts. We further enable malware-family-conditional generation using classifier guidance applied during sampling. Empirically, generated graphs match the marginal distribution of graph sizes well, but diverge strongly from real samples in edge-weight structure and multiple directed-graph diagnostics, yielding class-consistent but unrealistic samples under our data-scarce setting. In downstream tests, synthetic graphs are highly separable by malware family, yet a real-vs-synthetic detector achieves perfect accuracy, and classifiers trained on synthetic graphs transfer poorly to real data, indicating substantial distributional mismatch. These results highlight a key failure mode of discrete diffusion over high-dimensional directed, weighted adjacency representations and motivate stronger priors or lower-dimensional representations for realistic malware synthesis in limited-data settings.

Keywords: Discrete graph diffusion, Malware API call graphs, Directed weighted graphs, Synthetic graph generation

1 Introduction

Malicious software remains a persistent security threat, and modern defenses increasingly rely on behavioral signals obtained by executing programs in controlled environments [1]. Dynamic malware analysis systems observe runtime interactions with the operating system—such as system calls, Windows API calls, file/registry operations, and network activity—to produce execution traces that support tasks such as detection and classification. Surveys of automated dynamic analysis emphasize its

practical value for exposing behaviors that may be obscured by packing or static obfuscation, while also documenting common limitations such as environment sensitivity and evasion [2].

API call traces are especially popular among behavioral modalities: they are commonly collected in many sandbox pipelines and provide a compact view of how a program interacts with OS services. Recent work continues to demonstrate that malware detection and classification can be performed effectively using features derived from API call sequences and related behavioral encodings [3]. However, raw sequences can be long, noisy, and difficult to compare directly across samples, motivating structured representations that summarize higher-order dependencies in behavior.

A natural representation is a graph of behavioral transitions or dependencies, where nodes denote event types (e.g., API calls) and edges encode relations such as temporal succession, dependency, or co-occurrence. Graph-based behavioral models have been explored for malware detection and family classification [4], showing that explicitly modeling relations can improve robustness and interpretability compared to purely sequential features. This perspective also suggests a generative goal: if we can learn to generate realistic behavior graphs, we may be able to (i) augment scarce datasets, (ii) enable easier sharing of representative examples without distributing the original execution traces, and (iii) probe which structural regularities are learnable from a limited set of sequences.

Diffusion probabilistic models offer a compelling framework for generative modeling. Foundational diffusion work [5] formalized the idea of progressively destroying structure and learning a reverse process to recover samples, and subsequent developments demonstrated high-fidelity synthesis. Yet, standard diffusion models are most natural in continuous spaces, whereas behavior graphs are discrete, combinatorial objects with categorical node/edge attributes and strong structural constraints. DiGress [6] addresses this mismatch by defining a discrete denoising diffusion process over graphs paired with a permutation-equivariant denoiser, achieving strong results across multiple graph-generation benchmarks.

Motivated by these works, we investigate whether DiGress-style discrete diffusion can be adapted to generate directed, weighted malware API call transition graphs directly from data, and what failure modes emerge in the low-data regime typical of dynamic malware datasets. Our setting differs materially from many canonical graph-generation benchmarks: the graphs are directed; self-loops are meaningful; edges carry discretized weights; and the number of observed nodes per sample can vary depending on which API types appear. In response, we adapt the diffusion state space and transitions to handle directed edges and explicitly incorporate an ordinally structured edge-weight diffusion mechanism to better reflect the semantics of count-valued edge categories.

We also study malware-family–conditional generation using classifier guidance, a general technique for steering diffusion sampling using gradients from an auxiliary classifier. While guidance can improve class-conditional fidelity in continuous diffusion, its interaction with discrete, structured graph diffusion under limited data is not well understood. Our experiments highlight a central tension: guidance can produce samples that are highly class-consistent under a downstream classifier, yet remain

detectably non-realistic under structural statistics and real-vs-synthetic discrimination testing—revealing a "class-consistent but distribution-mismatched" regime that is particularly relevant for security applications where realism and transfer matter.

The remainder of this paper is structured as follows. Section 2 reviews diffusion models, discrete diffusion for graphs, and classifier guidance, and situates our approach within related work on behavioral malware modeling. Section 3 describes the dataset, the global API vocabulary, and our transformation from execution traces into directed, weighted transition graphs, including discretization and clipping of edge weights. Section 4 presents our DiGress adaptation for directed, weighted graphs: the diffusion state space, marginal-preserving transitions, the ordinal edge weight diffusion design, denoiser architecture, training objective, and classifier-guided conditional sampling procedure. Section 5 reports quantitative evaluations of structural realism and conditional behavior, including graph statistics, per-family summaries, and downstream classification tests. We conclude with Section 6, discussing why class-consistency can coexist with poor realism in this setting, and outlining implications for low-data malware graph generation and future work.

2 Background and Related Work

This section provides the methodological context for our experiment. We first review diffusion probabilistic models in the standard continuous setting, emphasizing the forward–reverse Markov structure and the denoising objective. We then summarize DiGress, a discrete denoising diffusion framework for graph generation, which motivates our use of categorical diffusion over node and edge types. Next, we describe classifier guidance for conditional generation, the mechanism we employ to steer sampling toward specific malware family labels. We conclude with related work spanning latent-space representation diffusion for malware synthesis, graph-based malware modeling, and labeled graph generation under data scarcity.

2.1 Standard diffusion models

The problem of generative modeling may be stated as follows. Let $X = \{x_i\}_{i=1}^n$ denote a dataset of n observations, where each $x_i \in \mathbb{R}^D$ is assumed to be drawn from an unknown distribution $p_{\text{data}}(x)$. The objective is to learn a parameterized distribution $p_\theta(x)$ that approximates $p_{\text{data}}(x)$ and enables sampling of novel observations representative of the data. Diffusion probabilistic models address this objective by defining a fixed forward corruption process that progressively maps data to noise, and learning a reverse process that stochastically inverts the corruption [5].

Let $x_0 \sim p_{\text{data}}(x)$. A diffusion model specifies a Markov chain $\{x_t\}_{t=0}^T$ with forward transitions that gradually destroy information in x_0. In the standard continuous-valued formulation, the forward process is Gaussian with a prescribed variance schedule $\{\beta_t\}_{t=1}^T$, typically with $\beta_t \in (0, 1)$:

$$q(x_t \mid x_{t-1}) \;=\; \mathcal{N}\Big(x_t;\; \sqrt{1-\beta_t}\, x_{t-1},\; \beta_t I\Big), \quad t = 1, \ldots, T. \tag{1}$$

Defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, the forward process admits a closed-form expression for the marginal distribution of x_t given x_0:

$$q(x_t \mid x_0) \;=\; \mathcal{N}\big(x_t;\; \sqrt{\bar{\alpha}_t}\, x_0,\; (1 - \bar{\alpha}_t) I\big)\,. \tag{2}$$

Equivalently, x_t may be sampled directly via the reparameterization

$$x_t \;=\; \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I), \tag{3}$$

which enables drawing corrupted samples at arbitrary noise levels t without explicit calculation of all intermediate steps.

The generative model is defined by a learned reverse-time Markov chain that starts from a tractable prior distribution, commonly $p(x_T) = \mathcal{N}(0, I)$, and applies learned reverse transitions:

$$p_\theta(x_{0:T}) \;=\; p(x_T) \prod_{t=T}^{1} p_\theta(x_{t-1} \mid x_t). \tag{4}$$

In the DDPM parameterization, each reverse transition is modeled as a Gaussian whose parameters are predicted by a neural network [7]:

$$p_\theta(x_{t-1} \mid x_t) \;=\; \mathcal{N}(x_{t-1};\; \mu_\theta(x_t, t),\; \Sigma_\theta(x_t, t))\,. \tag{5}$$

Training can be derived from a variational bound on the negative log-likelihood, yielding a sum of per-timestep divergence terms between true reverse conditionals induced by q and the learned reverse model under p_θ. A reparameterization trains the model to predict the injected noise ε at timestep t, resulting in a denoising regression objective of the form

$$\mathbb{E}_{x_0 \sim p_{\text{data}},\, \varepsilon \sim \mathcal{N}(0,I),\, t \sim \mathcal{U}(\{1,\ldots,T\})} \left[\|\varepsilon - \varepsilon_\theta(x_t, t)\|_2^2 \right], \quad \text{where } x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon. \tag{6}$$

A related formulation is provided by score-based generative modeling, which expresses diffusion in continuous time as a stochastic differential equation (SDE) that transforms p_{data} into a simple prior, and performs sampling by integrating a reverse-time SDE whose drift depends on the time-dependent score $\nabla_x \log p_t(x)$ of the perturbed distribution [8].

Although the above formulation is most naturally expressed for data in $\mathbb{R}^D$ with Gaussian corruption, many problems of interest are inherently discrete and/or reside in non-Euclidean domains such as graphs and manifolds. In such settings, naive application of continuous diffusion can violate intrinsic constraints (discreteness, sparsity, permutation symmetries, etc.) and may yield states that are poorly aligned with the underlying data geometry. These considerations motivate the broader paradigm of geometric deep learning [9].

2.2 Discrete denoising diffusion for graph generation (DiGress)

The considerations above suggest that, for graphs, a structured non-Euclidean object, it is desirable to define diffusion processes that respect (i) the discrete nature of

node/edge attributes, (ii) the sparsity typical of real-world graphs, and (iii) the permutation symmetries implied by graph representations [10].

DiGress is a discrete denoising diffusion model designed for learning distributions over attributed graphs [6]. The method targets graphs with categorical node attributes and categorical edge attributes. For a graph with n nodes, node attributes are represented by a matrix $X \in \mathbb{R}^{n \times a}$, where each row $x_i \in \{0,1\}^a$ is a one-hot encoding of the categorical type of node i (with a possible categories). Edge attributes are represented by a tensor $E \in \mathbb{R}^{n \times n \times b}$, where each $e_{ij} \in \{0,1\}^b$ is a one-hot encoding of the categorical type of the directed edge (i,j) (with b categories), and where the absence of an edge is treated as an edge type. Denoting $G = (X, E)$, DiGress defines a diffusion process over the discrete state space of node and edge types rather than directly over the combinatorial space of graphs.

Let $G^0 = (X^0, E^0)$ denote a clean graph sample. DiGress defines a Markov chain $\{G^t\}_{t=0}^T$ by independently corrupting each node type and each edge type with categorical transition matrices. Specifically, for node attributes, a transition matrix $Q_X^t \in \mathbb{R}^{a \times a}$ defines

$$q(x_i^t \mid x_i^{t-1}) \;=\; x_i^{t-1} Q_X^t, \qquad i = 1, \ldots, n, \tag{7}$$

and similarly for edge attributes, a transition matrix $Q_E^t \in \mathbb{R}^{b \times b}$ defines

$$q(e_{ij}^t \mid e_{ij}^{t-1}) \;=\; e_{ij}^{t-1} Q_E^t, \qquad i, j = 1, \ldots, n. \tag{8}$$

Let $\bar{Q}_X^t = \prod_{s=1}^t Q_X^s$ and $\bar{Q}_E^t = \prod_{s=1}^t Q_E^s$ denote the cumulative transition matrices. The resulting marginals admit the closed form

$$q(x_i^t \mid x_i^0) \;=\; x_i^0 \bar{Q}_X^t, \qquad q(e_{ij}^t \mid e_{ij}^0) \;=\; e_{ij}^0 \bar{Q}_E^t, \tag{9}$$

which enables sampling of noisy states at arbitrary timesteps without explicit calculation of all intermediate steps. For categorical diffusion with transition matrices, the posterior distribution also has a tractable expression. Writing z for a generic categorical one-hot variable (either a node type or edge type) with transitions Q^t and cumulative $\bar{Q}^{t-1}$, DiGress uses the closed-form posterior

$$q(z^{t-1} \mid z^t, z^0) \;\propto\; z^t (Q^t)^\top \odot z^0 \bar{Q}^{t-1}, \tag{10}$$

where $\odot$ denotes the element-wise product. Applying (10) independently to each x_i and e_{ij} yields $q(G^{t-1} \mid G^t, G^0)$ in factorized form.

A commonly used categorical transition is the uniform mixture

$$Q^{(t)} \;=\; \alpha_t I_d \;+\; (1 - \alpha_t)\, \frac{1}{d}\, \mathbf{1}_d \mathbf{1}_d^\top, \tag{11}$$

(where $d = a$ for nodes or $d = b$ for edges), which drives the chain toward a uniform limiting distribution. The authors of [6] observe that, for graphs, a uniform limiting distribution can be mismatched to data (notably due to sparse edge marginals), and

proposes marginal-preserving transitions that converge to empirical marginals. Let $m_X \in \mathbb{R}^a$ and $m_E \in \mathbb{R}^b$ denote empirical marginal distributions of node and edge types in the training set. DiGress uses transition matrices of the form

$$Q_X^t \;=\; \alpha_t I + \beta_t \mathbf{1} m_X^\top, \qquad Q_E^t \;=\; \alpha_t I + \beta_t \mathbf{1} m_E^\top, \tag{12}$$

so that the diffusion chain approaches m_X and m_E as its limiting marginals under suitable schedules. In addition, because $(\mathbf{1}m^\top)^2 = \mathbf{1}m^\top$, the cumulative transition admits the closed form $\bar{Q}^t = \bar{\alpha}_t I + \bar{\beta}_t \mathbf{1} m^\top$, which preserves tractability.

The reverse-time generative process in DiGress is a Markov chain $p_\theta(G^{t-1} \mid G^t)$ defined by combining the tractable posterior (10) with a learned prediction of the clean graph. A denoising network ϕ_θ takes G^t as input and outputs distributions over clean node and edge types,

$$\hat{p}_\theta(X^0 \mid G^t) \in [0,1]^{n \times a}, \qquad \hat{p}_\theta(E^0 \mid G^t) \in [0,1]^{n \times n \times b}, \tag{13}$$

interpreted as factorized categorical probabilities for each node and edge. The reverse transition for each node and edge is obtained by marginalizing the posterior over the network prediction:

$$p_\theta(x_i^{t-1} \mid G^t) \;=\; \sum_{x \in \mathcal{X}} q(x_i^{t-1} \mid x_i^t, x_i^0 = x)\, \hat{p}_{\theta,i}(x \mid G^t), \tag{14}$$

with an analogous expression for $p_\theta(e_{ij}^{t-1} \mid G^t)$. Under the conditional-independence assumption of the forward process, the reverse distribution factorizes over nodes and edges:

$$p_\theta(G^{t-1} \mid G^t) \;=\; \prod_{i=1}^n p_\theta(x_i^{t-1} \mid G^t) \prod_{i,j=1}^n p_\theta(e_{ij}^{t-1} \mid G^t). \tag{15}$$

DiGress parameterizes ϕ_θ using a graph transformer architecture, which produces both node- and edge-level predictions while maintaining permutation equivariance; the loss is chosen to be permutation invariant by construction via summation over node and edge terms.

Training is derived from a variational bound and yields timestep-wise classification losses. In the commonly used parameterization, ϕ_θ is trained to predict the clean node and edge types from noisy inputs. Denoting by $\mathrm{CE}(\cdot,\cdot)$ the cross-entropy between a one-hot target and a predicted categorical distribution, a typical objective takes the form

$$\mathcal{L}(\theta) \;=\; \mathbb{E}_{t \sim \mathcal{U}(\{1,\dots,T\}),\, G^0 \sim p_{\mathrm{data}},\, G^t \sim q(\cdot \mid G^0)} \left[\sum_{i=1}^n \mathrm{CE}\big(x_i^0, \hat{p}_{\theta,i}(\cdot \mid G^t)\big) + \sum_{i,j=1}^n \mathrm{CE}\big(e_{ij}^0, \hat{p}_{\theta,ij}(\cdot \mid G^t)\big) \right], \tag{16}$$

with optional weighting between node and edge terms and optional likelihood evaluation via the evidence lower bound [6].

2.3 Classifier-guided diffusion for conditional generation

Conditional generation seeks to sample from $p(x \mid y)$, where y denotes an external condition such as a class label. Classifier guidance provides a mechanism for conditional sampling in which an unconditional diffusion model is combined with an auxiliary classifier (or regressor) trained to predict the condition from noisy diffusion states. This approach was introduced in large-scale diffusion for images by the authors in [11], who train a time-dependent classifier $p_\phi(y \mid x_t)$ on corrupted inputs x_t and use it to bias the reverse diffusion trajectory toward samples consistent with y.

There are two closely related motivations for classifier guidance.

(i) Score-decomposition view (continuous-time / score-based perspective). In the score-based formulation, one considers the perturbed data density $p_t(x)$ at diffusion time t. By Bayes' rule,

$$\log p_t(x \mid y) \;=\; \log p_t(x) \;+\; \log p_t(y \mid x) \;+\; \text{const}(y), \tag{17}$$

and taking gradients with respect to x yields the additive decomposition

$$\nabla_x \log p_t(x \mid y) \;=\; \nabla_x \log p_t(x) \;+\; \nabla_x \log p_t(y \mid x). \tag{18}$$

Thus, a conditional sampler may be obtained by augmenting the unconditional score $\nabla_x \log p_t(x)$ with the classifier score $\nabla_x \log p_t(y \mid x)$. This identity is commonly used to justify guidance in reverse-time SDE samplers [8], and [11] shows how to implement an analogous correction within discrete-time diffusion samplers.

(ii) Transition reweighting view (discrete-time / posterior shaping). An alternative explanation is to view guidance as reweighting the reverse transition at each timestep to favor intermediate states that the classifier deems likely under the condition. Let $p_\theta(x_{t-1} \mid x_t)$ denote the unconditional reverse transition of a diffusion model. The corresponding conditional transition can be written as

$$p(x_{t-1} \mid x_t, y) \;\propto\; p_\theta(x_{t-1} \mid x_t)\, p_\phi(y \mid x_{t-1}). \tag{19}$$

This expression is conceptually useful—conditioning increases probability mass on predecessors x_{t-1} that are both plausible under the generative model and predictive of the target label under the classifier—but it is not directly tractable because $p_\phi(y \mid x_{t-1})$ is represented by a neural network. The authors of [11] derive an efficient guided sampler by approximating this reweighting through a local first-order correction, resulting in a reverse update that steers samples in the direction that increases $\log p_\phi(y \mid x)$.

These views lead to the same operational procedure: during sampling, one evaluates $\nabla_{x_t} \log p_\phi(y \mid x_t)$ and uses it to guide the reverse process, with an explicit guidance strength λ that controls the tradeoff between unconditional realism and conditional adherence [11]. Under a DDPM parameterization where the reverse transition is Gaussian, this steering is often written as an approximate mean shift of the reverse

step,

$$\mu_{\theta,\phi}(x_t, t; y) \;=\; \mu_\theta(x_t, t) \;+\; \lambda\, \Sigma_t\, \nabla_{x_t} \log p_\phi(y \mid x_t), \tag{20}$$

where Σ_t denotes the reverse-step covariance. This expression is a convenient approximation that yields an efficient guided sampler.

DiGress makes use of this guidance principle to enable conditioning in discrete diffusion over graphs [6]. Specifically, DiGress proposes a discrete regressor (or classifier) guidance mechanism for graph-level properties, in which a predictor is trained on noisy graph states G^t and its gradient is used during sampling to bias the reverse diffusion toward graphs consistent with a target property. Because the diffusion variables in DiGress are discrete (categorical node and edge types), the method employs a continuous relaxation of the graph representation for the purpose of computing gradients, while preserving the factorized discrete reverse process used for sampling.

2.4 Related Work

The authors of [12] introduce Cluster-Tangent Diffusion (CT-Diff) as a diffusion-based approach for synthetic malware generation that operates in a learned representation space rather than directly over structured behavioral objects. Their pipeline constructs malware feature embeddings using pretrained NLP embedding models and refines them using autoencoders to obtain a continuous latent representation. CT-Diff then generates synthetic samples by diffusing within this embedding space, with an explicit emphasis on preserving local geometric structure through cluster- and manifold-inspired sampling mechanisms. This work provides a strong reference point for embedding-space diffusion on malware data under limited sample sizes.

In [13], the authors study synthetic malware generation primarily as a data augmentation mechanism, constructing sequential representations by decomposing binaries into mnemonic opcode sequences and training multiple generative models, including diffusion-based generators, alongside GAN variants. Their evaluation focuses on the utility of generated samples for downstream classification. While the representation differs from API-call traces, the methodological pattern is similar to CT-Diff, where diffusion is applied after a representation step that yields a continuous or sequence-like object, rather than directly generating structured relational objects such as behavior graphs.

The authors of [14] propose API2Vec, arguing that modeling raw API call sequences alone can be unreliable in realistic execution environments. API2Vec constructs graph-structured views of execution behavior and learns representations via graph-guided path extraction and representation learning. Although not a generative model, this line of work provides the motivation to treat API behavior as a relational structure and provides precedent for graph constructions intended to preserve behavioral patterns that may be obscured under sequence-based modeling.

In [15], a malware detection pipeline is proposed that converts API call sequences into directed graph structures and applies graph learning for classification. The approach combines sequence-derived graph construction with additional processing inspired by Markovian transition statistics before applying a graph convolutional classifier. This work supports the broader view that graph structure extracted from execution traces can encode discriminative information that is not captured by flat

feature vectors, and it provides a concrete example of constructing directed graphs from API execution order.

The work [16] presents a malware classification method that explicitly emphasizes directed API call relationships and proposes direction-aware graph convolution variants intended to better reflect sequential and asymmetric dependencies in execution behavior. The results reinforce that directionality is not merely a representational detail but an essential aspect of behavior structure, motivating generative modeling approaches that preserve directed statistics (e.g., in/out-degree patterns, connected component structure) rather than relying on undirected representations.

The authors of [17] study labeled graph generation under data scarcity and propose a meta-learning framework that transfers generative knowledge from auxiliary graph datasets to enable adaptation when only a small number of graphs are available. Their experiments demonstrate improved preservation of structural graph properties in low-data regimes relative to prior baselines. Importantly, they also note that improved structural fidelity does not necessarily imply capturing domain-specific properties (discussed in the context of molecular graphs), highlighting the need to distinguish between generic structural realism and application-specific constraints when evaluating graph generators under data scarcity.

Taken together, prior diffusion-based malware synthesis work largely operates in continuous representation spaces (embeddings or sequence-derived feature spaces) [12, 13], whereas graph-based malware research has primarily focused on discriminative modeling rather than generative modeling of behavior graphs [14–16]. At the same time, scarcity-focused graph generation results suggest that learning graph distributions from few samples may require strong inductive bias or transfer mechanisms, and that structural similarity alone may be insufficient to capture domain-specific properties [17]. These gaps motivate the present study: we investigate whether a discrete graph diffusion approach can be adapted to generate directed, weighted malware behavior graphs on the same dataset used in [12], and we characterize the resulting fidelity–utility trade-offs and failure modes under limited data.

3 Dataset and Graph Construction

This section describes the dataset used in our experiments and the procedure by which we convert raw malware execution traces into the directed, weighted graphs modeled by our diffusion generator. We begin by summarizing the dataset and split protocol, then define a global API vocabulary to obtain a consistent categorical node space across samples. We next formalize the sequence-to-graph transformation that maps each API call trace to a directed transition graph with integer-valued edge weights, and we introduce the clipping and discretization scheme that yields a tractable categorical edge state space for discrete diffusion. Finally, we describe the resulting PyTorch Geometric representation used throughout training, sampling, and evaluation.

3.1 Dataset and splits

We use the same malware API call dataset as in [12], consisting of 500 malware execution traces from seven families (Adload, Bancos, OnlineGames, Vbinject, Vundo,

WinWebSec, and Zwangi), originally sourced in [18] by dynamic execution of Windows files. We perform a stratified split at the graph level into 80% training, 10% validation, and 10% test data, ensuring that each split preserves the per-family proportions.

3.2 Global API vocabulary and node encoding

Each execution trace is represented as a sequence of API call names. To obtain a consistent categorical representation across graphs, we construct a global API vocabulary from all unique API calls appearing in the dataset, yielding $|\mathcal{V}| = 79$ unique API types. The vocabulary is built by collecting all unique API names across all traces and sorting them alphabetically; we then assign global API indices $0, \ldots, 78$ according to this alphabetical ordering.

For graph learning and generative modeling, node identities are encoded using one-hot vectors. For a graph with n nodes, node features are stored in a matrix $X \in \mathbb{R}^{n \times |\mathcal{V}|}$, where each row X_i is a one-hot vector indicating the global API type associated with node i. Specifically, if node i corresponds to the API call with global index $g(i) \in \{0, \ldots, 78\}$, then $X_{i,g(i)} = 1$ and all other entries are 0.

3.3 Sequence-to-graph transformation

Each API call sequence is converted into a directed, weighted transition graph. For a given trace, we define a graph $G = (V, E)$ where nodes correspond to unique API calls observed in the trace and directed edges represent observed transitions. For each adjacent pair (a_t, a_{t+1}) in the sequence, we add a directed edge $a_t \to a_{t+1}$. Self-loops are permitted and are retained when $a_t = a_{t+1}$, since such self-transitions occur in the dataset.

Because transitions can occur multiple times within a trace, we aggregate repeated transitions and define raw transition counts as edge weights. Concretely, for a directed pair of nodes (u, v), the edge weight $w(u, v)$ is defined as the number of times the transition $u \to v$ appears in the sequence. If a transition occurs multiple times, the corresponding edge is represented once with weight equal to the sum of occurrences.

Each graph uses local node indices $0, \ldots, n-1$. Local node indices are also assigned deterministically: for each trace, the set of unique API calls appearing in that trace is collected and sorted alphabetically by API name, and indices are assigned according to this alphabetical ordering.

3.4 Edge weight discretization and clipping

Raw transition counts are nonnegative integers and can be large for long traces. Directly modeling large integer weights as categorical edge types is computationally prohibitive because the size of edge embeddings and transition matrices scales with the number of categories. We therefore clip transition counts to a maximum value `max_edge_weight` $= 10$. Formally, for an observed count $c(u, v)$, the clipped weight is

$$\tilde{w}(u, v) \;=\; \min(c(u, v), 10). \tag{21}$$

Thus, observed edges take categorical weight values in $\{1, \ldots, 10\}$, while a missing edge corresponds to a separate "no-edge" category (weight 0) when constructing dense edge-type tensors for diffusion.

The clipping threshold is chosen to maintain a tractable state space. With `max_edge_weight` $= 10$, edges fall into 11 categories (0–10), yielding an 11×11 categorical diffusion transition matrix. In contrast, using the maximum observed weight as the number of categories would induce transition matrices with on the order of 10^8 parameters and substantially increased memory requirements for edge embeddings. Empirically, `max_edge_weight` $= 10$ captures the majority of observed transitions (96.25% of edges have weight ≤ 10), providing a practical compromise between representational capacity and computational feasibility.

3.5 PyG graph representation

We represent each graph using PyTorch Geometric (PyG) `Data` objects [19]. Each graph includes:

- `x`: node features of shape $(\texttt{num_nodes}, |\mathcal{V}|)$, stored as `torch.float`, containing one-hot encodings of global API types. Each row corresponds to a local node index i, and the one-hot position indicates that node's global API identifier.
- `edge_index`: directed edge connectivity of shape $(2, \texttt{num_edges})$, stored as `torch.long`, where each column specifies a directed source–target pair using local node indices $0, \ldots, n-1$.
- `edge_attr`: edge attributes of shape $(\texttt{num_edges}, \texttt{max_edge_weight} + 1)$, stored as `torch.float`, containing one-hot encodings of the clipped edge-weight category. In the sparse PyG representation, `edge_index` enumerates only present edges, so observed edges take weight categories in $\{1, \ldots, 10\}$. The "no-edge" category (0) is used when forming the dense $n \times n$ edge-type tensor required by the diffusion model.
- `y`: graph label indicating malware family, stored as `torch.long` with class indices $\{0, \ldots, 6\}$.
- `num_nodes`: the number of nodes in the graph.

This representation provides a consistent interface for both discriminative evaluation (e.g., GNN classifiers) and the graph diffusion pipeline used in subsequent sections.

4 Model: DiGress Adaptation for Directed, Weighted API-Call Graphs

We model each malware execution trace as a directed, weighted API call transition graph (Section 3) and learn a discrete denoising diffusion model in the style of DiGress [6]. The diffusion state is defined over categorical node types (API identities) and categorical edge types (discretized transition-count weights). We adapt the core DiGress framework to suit our API call graph setting. This requires several domain-driven modifications beyond the original molecular-graph setting in [6]: (i) removal of undirected symmetry constraints to support directed edges, (ii) representation of edges as discretized integer weights (including a no-edge category), (iii) explicit modeling of self-loops as valid diagonal edge entries, and (iv) an ordinal transition mechanism

for the edge diffusion kernel that respects the numeric structure of weight categories. Finally, consistent with DiGress's approach to conditioning on graph-level properties, we enable malware-family–conditional generation via classifier guidance during sampling, where we use the malware family label as the conditioned property.

4.1 Overview

We provide Figure 1, an end-to-end view of our approach, from sequence-to-graph construction through diffusion training and guided sampling, to the evaluation suite used later in Section 5.

4.2 Diffusion state space and dense tensorization

Let $G = (X, E)$ denote a graph with n nodes. Node attributes are represented as a one-hot matrix

$$X \in \{0,1\}^{n \times a}, \qquad a = 79, \tag{22}$$

where each row x_i is a one-hot encoding of the API type of node i from the global vocabulary.

Edge attributes are represented as a one-hot tensor

$$E \in \{0,1\}^{n \times n \times b}, \qquad b = 11, \tag{23}$$

where each $e_{ij} \in \{0,1\}^b$ encodes the directed edge category for (i, j). Edge categories are

$$\mathcal{C}_E = \{0, 1, \ldots, 10\}, \tag{24}$$

where category 0 denotes *no edge* and categories 1–10 denote discretized transition-count weights (Section 3.4). Because graphs are directed, e_{ij} and e_{ji} are treated as independent random variables; no symmetry constraint is imposed.

Self-loops.

In contrast to molecular graphs, API call transitions can contain self-loops when identical API calls occur consecutively [20]. We therefore treat diagonal entries e_{ii} as valid edge variables and include them in preprocessing, diffusion corruption, denoising prediction, and sampling.

Mini-batching via padding and masking.

Given a mini-batch of graphs with sizes $\{n_k\}_{k=1}^B$, we pad each graph to $n_{\max} = \max_k n_k$ to form dense tensors

$$X \in \mathbb{R}^{B \times n_{\max} \times a}, \qquad E \in \mathbb{R}^{B \times n_{\max} \times n_{\max} \times b}. \tag{25}$$

A boolean node mask $M \in \{0,1\}^{B \times n_{\max}}$ indicates valid (unpadded) nodes; all losses and model computations are masked accordingly so that padded entries do not contribute. Since self-loops are allowed in the directed setting, diagonal entries (i, i) are considered valid whenever $M_i = 1$.

4.3 Edge-weight construction and discretization

Edge categories determine the diffusion state space for E. Let us now summarize the edge construction process. Let an execution trace be a sequence of API calls $S = [s_1, \ldots, s_N]$. For any ordered pair of API types (u, v), we define the raw transition count

$$w(u \to v) \;=\; \sum_{i=1}^{N-1} \mathbb{I}\big[s_i = u \;\wedge\; s_{i+1} = v\big], \tag{26}$$

which includes self-loops when $u = v$. To keep the edge state space tractable, counts are clipped at $w_{\max} = 10$:

$$\tilde{w}(u \to v) \;=\; \min\big(w(u \to v), w_{\max}\big), \qquad w_{\max} = 10. \tag{27}$$

Edge categories are then defined as $\tilde{w} \in \{0, 1, \ldots, 10\}$, where $\tilde{w} = 0$ indicates no observed transition and $\tilde{w} \geq 1$ indicates an observed transition with clipped weight. Each edge is encoded as an 11-dimensional one-hot vector e_{uv}.

4.4 Discrete diffusion over node and edge categories

Let $G^0 = (X^0, E^0)$ denote a clean graph. We define a forward Markov chain $\{G^t\}_{t=0}^{T}$ with $T = 500$ timesteps that independently corrupts each node type and each directed edge type via categorical transitions. For nodes,

$$q(x_i^t \mid x_i^{t-1}) \;=\; x_i^{t-1} Q_X^t, \qquad i = 1, \ldots, n, \tag{28}$$

and for edges,

$$q(e_{ij}^t \mid e_{ij}^{t-1}) \;=\; e_{ij}^{t-1} Q_E^t, \qquad i, j = 1, \ldots, n, \tag{29}$$

where $Q_X^t \in \mathbb{R}^{a \times a}$ and $Q_E^t \in \mathbb{R}^{b \times b}$ are timestep-dependent categorical transition matrices. Then cumulative transitions are defined by

$$\bar{Q}_X^t = \prod_{s=1}^{t} Q_X^s, \qquad \bar{Q}_E^t = \prod_{s=1}^{t} Q_E^s, \tag{30}$$

so that noisy states can be sampled directly from $q(\cdot \mid G^0)$ at any timestep t. We use a discrete cosine noise schedule to set per-step noise levels $\{\beta_t\}_{t=1}^{T}$.

4.5 Marginal-preserving transitions for nodes and edges

As in DiGress [6], we employ marginal-preserving transitions so that the diffusion chain approaches empirical marginals rather than a uniform limiting distribution. Let $m_X \in \mathbb{R}^a$ denote the empirical marginal over node types (API categories) and $m_E \in \mathbb{R}^b$ denote the empirical marginal over edge types (weight categories). Marginals are computed with the global vocabulary construction, using all graphs in the dataset.

For node attributes, we use the standard marginal-mixture transition:

$$Q_X^t \;=\; (1 - \beta_t) I_a \;+\; \beta_t \mathbf{1} m_X^\top, \tag{31}$$

where I_a is the $a \times a$ identity matrix and $\mathbf{1}m_X^\top$ denotes the rank-one matrix with each row equal to m_X.

For edges, we again use a mixture with probability β_t, but replace the limiting matrix by an ordinal-aware transition matrix (Section 4.6) that respects the numeric ordering of edge weights.

4.6 Ordinal edge-weight diffusion transitions

Edge categories correspond to discretized integer weights, so treating them as unordered labels is mismatched to the structure of frequency counts. To incorporate ordering information, we define an ordinal transition kernel over edge categories.

Let edge categories be indexed by $i, j \in \{0, 1, \ldots, w_{\max}\}$ with $w_{\max} = 10$ (thus $b = w_{\max} + 1 = 11$). Then the category distance is defined by

$$d(i, j) = |i - j|. \tag{32}$$

We define a distance-decay weight scheme

$$\tilde{P}(i \to j) \;=\; \exp\!\left(-\frac{d(i,j)^2}{2\sigma^2}\right), \tag{33}$$

where $\sigma > 0$ (ordinal_sigma) controls the decay rate; smaller σ concentrates mass on nearby weights, while larger σ makes transitions more diffuse.

To preserve empirical edge statistics while maintaining ordinal locality, we use a marginal-weighted variant:

$$\tilde{P}(i \to j) \;=\; m_E(j)\, \exp\!\left(-\frac{|i-j|^2}{2\sigma^2}\right). \tag{34}$$

Normalizing row-wise yields the ordinal transition matrix $U_E \in \mathbb{R}^{b \times b}$:

$$U_E[i, j] \;=\; \frac{\tilde{P}(i \to j)}{\sum_{k=0}^{w_{\max}} \tilde{P}(i \to k)}. \tag{35}$$

Given timestep noise level β_t, we then define the one-step edge transition as the mixture

$$Q_E^t \;=\; (1 - \beta_t) I_b \;+\; \beta_t U_E. \tag{36}$$

The cumulative transition is computed by the ordered product

$$\bar{Q}_E^t \;=\; \prod_{s=1}^{t} Q_E^s, \tag{37}$$

which is precomputed before training by iterative 11×11 matrix multiplication. As $t \to T$, the chain approaches U_E as its limiting distribution, preserving the intended ordinal and marginal structure in the noise prior. Importantly, (36)–(37) are applied

to every directed pair (i, j), including diagonal entries (i, i), because self-loops are part of the modeled edge state space.

4.7 Denoising network architecture

We parameterize the denoiser using a transformer-based graph network [21] consistent with DiGress [6]. The network takes as input the noisy graph state $G^t = (X^t, E^t)$ together with (i) structural extra features and (ii) a timestep encoding, and outputs logits over clean node and edge categories:

$$\hat{p}_\theta(X^0 \mid G^t) \in [0,1]^{n \times a}, \qquad \hat{p}_\theta(E^0 \mid G^t) \in [0,1]^{n \times n \times b}. \tag{38}$$

Input embeddings and hidden dimensions.

Node and edge one-hot vectors are mapped to hidden representations using MLPs (linear layers with ReLU), producing node embeddings in $\mathbb{R}^{d_x}$ and edge embeddings in $\mathbb{R}^{d_e}$. A global feature vector y is used to hold extra graph-level features; in the unconditional diffusion model, no explicit label conditioning is injected during training and no global prediction head is used (d_y is present only as an internal hidden dimension).

Transformer layers.

The model applies n_{layers} graph transformer layers with multi-head attention (n_{head} heads), updating node, edge, and global features while preserving permutation equivariance. Masking ensures padded nodes and edges do not affect computation. For directed graphs, the model does not symmetrize edge predictions, and diagonal edge entries are not masked, allowing self-loops to be predicted.

Extra features and timestep.

We enable extra_features=`all`, including cycle counts and Laplacian spectral features (eigenvalues/eigenvectors). Because Laplacian eigendecomposition requires a symmetric operator with real spectrum, the Laplacian used for spectral features is symmetrized when the underlying adjacency is directed; this affects only the feature computation, while the core edge tensor E remains directed. The timestep t is normalized to $[0, 1]$ and concatenated as a scalar to the global feature vector.

4.8 Training objective and optimization

We train the diffusion model unconditionally using cross-entropy losses to predict the clean node and edge categories from noisy inputs, masking padded entries. We denote the cross-entropy between a one-hot target and a predicted categorical distribution by $\mathrm{CE}(\cdot, \cdot)$. The training loss is

$$\mathcal{L}(\theta) = \mathbb{E}_{\substack{t\sim\mathcal{U}(\{1,\dots,T\}) \\ G^0\sim p_{\text{data}},\, G^t\sim q(\cdot|G^0)}} \Bigg[\sum_{i=1}^{n} \mathrm{CE}\big(x_i^0, \hat{p}_{\theta,i}(\cdot \mid G^t)\big) + \lambda_E \sum_{i,j=1}^{n} \mathrm{CE}\big(e_{ij}^0, \hat{p}_{\theta,ij}(\cdot \mid G^t)\big) \Bigg], \tag{39}$$

where λ_E is a hyperparameter that controls the relative importance of edge loss compared to node loss. In the directed API-call setting, the edge sum includes all directed pairs (i, j), including diagonal terms (i, i), since self-loops are valid.

Optimization uses AdamW with AMSGrad, gradient clipping, and an exponential moving average (EMA) of parameters for improved stability during validation and sampling.

We show the training and validation loss curves for the unconditional diffuson model in Figure 2.

4.9 Classifier guidance for malware-family–conditional generation

To condition generation on malware family, we employ classifier guidance as proposed in DiGress [6] and more generally in classifier-guided diffusion [11]. We train a separate classifier $p_\phi(y \mid G^t)$ to predict the malware class $y \in \{1, \dots, C\}$ (with $C = 7$ families) from noisy diffusion states G^t. The classifier shares the same GraphTransformer backbone and feature construction as the diffusion model, ensuring gradients are well-defined over the same representation space.

During sampling, the unconditional reverse step produces base categorical distributions for nodes and edges at step $t \to s$, which we denote by

$$p_{\text{base}}(X^s \mid G^t), \qquad p_{\text{base}}(E^s \mid G^t), \tag{40}$$

computed via the DiGress posterior construction combined with the denoiser's predicted clean distributions. Guidance then reweights these distributions using classifier gradients evaluated on a continuous relaxation of the one-hot tensors. Let $y^\star$ be the target malware family and define the classifier loss

$$\mathcal{L}_{\text{cls}}(G^t; y^\star) \;=\; -\log p_\phi(y^\star \mid G^t). \tag{41}$$

We compute gradients with respect to the relaxed node and edge tensors:

$$g_X^t = \nabla_{X^t} \mathcal{L}_{\text{cls}}(G^t; y^\star), \qquad g_E^t = \nabla_{E^t} \mathcal{L}_{\text{cls}}(G^t; y^\star), \tag{42}$$

and mask out padded entries. Because guidance is applied at the $t \to s$ transition, gradients are projected through the corresponding transition matrices (denoted generically by $Q_{s\to t}$) to match the categorical space at timestep s:

$$\tilde{g}_X = g_X^t Q_{s\to t}^{(X)}, \qquad \tilde{g}_E = g_E^t Q_{s\to t}^{(E)}. \tag{43}$$

Parameter	Value	Description
Dataset & preprocessing		
Node categories a	79	Number of unique API calls in global vocabulary
Edge categories b	11	Categories $\{0, \ldots, 10\}$ (no edge + weights 1–10)
Max edge weight $w_{\max}$	10	Clipping threshold for transition counts
Directed edges	true	Model edges as directional ($i \to j \neq j \to i$)
Integer edge weights	true	Edge types represent discretized integer weights
Self-loops allowed	true	Preserve and model diagonal edges (i, i)
Split strategy	Stratified	Graph-level stratified split by class (malware family)
$N_{\max}$	Dynamic	Maximum nodes per batch (computed per batch)
Diffusion process		
Diffusion steps T	500	Number of diffusion timesteps
Noise schedule	Cosine (discrete)	Discrete cosine schedule for $\{\beta_t\}$
Cosine offset s	0.008	Cosine schedule offset parameter
Transition type	marginal	Marginal-preserving transitions
Ordinal sigma σ	0.5	Decay width for ordinal edge transitions

Table 1 Hyperparameters for dataset/preprocessing and diffusion configuration.

We then define guidance distributions via a softmax with guidance scale $\lambda > 0$:

$$p_\eta(X^s \mid G^t, y^\star) \;=\; \mathrm{softmax}\big(-\lambda\, \tilde{g}_X\big), \qquad p_\eta(E^s \mid G^t, y^\star) \;=\; \mathrm{softmax}\big(-\lambda\, \tilde{g}_E\big), \tag{44}$$

applied pointwise over nodes and edges. Finally, we combine guidance with the base reverse distributions by element-wise multiplication and renormalization:

$$p_{\text{guided}}(X^s \mid G^t, y^\star) \;\propto\; p_\eta(X^s \mid G^t, y^\star) \;\odot\; p_{\text{base}}(X^s \mid G^t), \tag{45}$$

$$p_{\text{guided}}(E^s \mid G^t, y^\star) \;\propto\; p_\eta(E^s \mid G^t, y^\star) \;\odot\; p_{\text{base}}(E^s \mid G^t). \tag{46}$$

We then sample X^s and E^s from these guided categorical distributions. λ is constant over all timesteps and guidance is applied to both node and edge distributions.

We show the training and validation loss curves for the auxiliary classifier used for guidance in Figure 3.

4.10 Comprehensive hyperparameter summary

Tables 1–4 list the hyperparameters used in our implementation, grouped by process component.

Parameter	Value	Description
Denoising network (GraphTransformer)		
Number of layers n_{layers}	1	Transformer layers
Node hidden dim d_x	128	Hidden dimension for node features
Edge hidden dim d_e	32	Hidden dimension for edge features
Global hidden dim d_y	32	Hidden dimension for global features
Attention heads n_{head}	8	Multi-head attention heads
Node FFN dim	128	Feed-forward dimension for node updates
Edge FFN dim	64	Feed-forward dimension for edge updates
Global FFN dim	64	Feed-forward dimension for global updates
Input MLP hidden (X)	128	Hidden dimension of node input MLP
Input MLP hidden (E)	64	Hidden dimension of edge input MLP
Input MLP hidden (y)	64	Hidden dimension of global input MLP
Dropout	0.1	Dropout rate in transformer layers
Activation	ReLU	Activation for input/output MLPs
Node output dim	79	Logits over API categories
Edge output dim	11	Logits over edge-weight categories
Global output dim	0	Unconditional diffusion model
Extra features & timestep		
Extra features	all	Cycle + spectral features enabled
Node cycles	$k3, k4, k5$	Cycle counts per node
Graph cycles	$k3$–$k6$	Cycle counts per graph
Eigenvalues	5	Smallest non-zero Laplacian eigenvalues
Eigenvectors	2	Lowest Laplacian eigenvectors
LCC indicator	1	Largest connected component indicator
Timestep embedding	concat	Concatenate scalar $t \in [0, 1]$ to global features

Table 2 Hyperparameters for the denoising network and extra features.

5 Results

We evaluate the fidelity of graphs generated by our DiGress-style discrete diffusion model (Section 2.2) on the directed, weighted API call graphs defined in Section 3. We compare real graphs from the dataset to synthetic graphs sampled from the trained model under malware-family conditioning via classifier guidance (Section 2.3). The real set contains 492 graphs; we generate 70 graphs for each of the seven malware families, totaling 490 graphs.

Post-generation pruning and connectedness.

Because each graph is derived from an execution sequence, the underlying transition structure is expected to be connected. We therefore implement a post-processing step that retains only the largest connected component (LCC) of each generated graph to

Parameter	Value	Description
Diffusion-model training		
Optimizer	AdamW	Adam with weight decay
Learning rate	2×10^{-4}	Constant learning rate
Weight decay	10^{-12}	ℓ_2 regularization coefficient
AMSGrad	true	AMSGrad variant enabled
Batch size	8	Graphs per batch
Grad accumulation	16	Accumulation steps
Effective batch size	128	8×16
Max epochs	1000	Maximum epochs (with early stopping)
Gradient clipping	1.0	Maximum gradient norm
EMA decay	0.999	Exponential moving average decay
Loss type	Cross-entropy	Categorical CE training loss
Edge loss weight λ_E	5.0	Edge loss multiplier
Global loss weight	0.0	No global prediction loss
Early stopping	true	Enabled
Monitor metric	val/epoch_NLL	Validation NLL for early stopping
Patience	10	Early stopping patience
Min delta	0.001	Minimum improvement threshold
Sampling		
Graph size sampling	Empirical	Sample n from empirical node-count distribution
Initialization at $t = T$	Limit dist.	Initialize from diffusion limit distribution
Reverse sampler	Ancestral	Discrete ancestral sampling
Temperature	1.0	No temperature scaling
Top-k	None	No top-k filtering
Dense→sparse	Argmax	One-hot to discrete via arg max

Table 3 Hyperparameters for diffusion model training and sampling.

enforce this sequential structure. In the present experiment, however, all generated graphs are already connected, so pruning has no effect.

Evaluation criteria.

We report (i) distributional similarity of graph statistics via two-sample tests (Kolmogorov–Smirnov (KS) [22] and Mann–Whitney [23]) and categorical chi-square tests [24] where appropriate, and (ii) downstream discriminative behavior, measuring both malware-family predictability of synthetic samples and real-vs-synthetic detectability.

Parameter	Value	Description
Classifier (for guidance)		
Core architecture	GraphTransformer	Same architecture family as denoiser
Number of layers n_{layers}	1	Matched to diffusion model
Hidden dims	d_x=128, d_e=32, d_y=32	Matched to diffusion model
Num. classes C	7	Malware families
Optimizer	AdamW	AdamW (AMSGrad enabled)
Learning rate	10^{-3}	Classifier LR
Batch size	32	Classifier batch size
Max epochs	1500	Maximum epochs (with early stopping)
LR scheduler	ExponentialLR	Exponential decay
Gamma	0.99	Scheduler decay factor
Loss type	Cross-entropy	Classification CE loss
Early stopping	true	Enabled
Monitor metric	val/epoch_accuracy	Validation accuracy
Patience	10	Early stopping patience
Min delta	0.0005	Minimum improvement threshold
Classifier guidance		
Guidance mode	classification	Condition on malware family label
Guidance scale λ	500	Guidance strength
Time schedule	Constant	λ constant across timesteps
Applied to nodes	true	Guidance reweights node categories
Applied to edges	true	Guidance reweights edge categories
Combination rule	product	$p_{\text{guided}} \propto p_\eta \odot p_{\text{base}}$

Table 4 Hyperparameters for classifier training and classifier-guided conditional generation.

5.1 Graph size statistics

The node-count distributions of real and synthetic graphs are closely aligned. We show this in Figure 4. Real graphs have mean node count 21.93±6.51, while synthetic graphs have mean 22.25 ± 6.95. A two-sample KS test does not reject equality of distributions ($D = 0.0455$, $p = 0.6648$), and a Mann–Whitney test likewise indicates no significant shift ($U = 117,737.50, p = 0.5273$).

5.2 Node-type coverage and node-type frequency

Although graph sizes match well, synthetic graphs exhibit reduced node-type coverage. Real graphs span the full global vocabulary of 79 API node types, whereas the sampled synthetic set realizes fewer unique node types, spanning only 63 unique nodes. Moreover, synthetic node-type frequencies are noticeably more concentrated among a small subset of high-frequency API calls, indicating partial mode collapse in node-type

usage. We show this in Figure 5 , with collapse to the $QueryProcessInformation$ node being the most evident.

5.3 Edge-weight distribution

Edge weights, discretized into categories $\{1, \ldots, 10\}$, with 0 representing "no edge", show the largest mismatch between real and synthetic graphs. In the real dataset, observed edge weights are concentrated on low counts: the mean observed edge weight is 2.10, with weight-1 edges forming the majority (63.53%) of observed transitions. In contrast, synthetic graphs shift mass toward intermediate and high categories, yielding a substantially larger mean edge weight of 5.42. We visualize this mismatch in Figure 6.

This discrepancy is supported by decisive statistical evidence: a KS test strongly rejects equality of edge-weight distributions ($D = 0.80$ with $p < 0.001$), and a chi-square test on categorical frequencies likewise rejects equality ($x^2 = 237{,}218$, $p < 0.001$).

5.4 Degree structure and edge volume

To quantify overall connectivity, we report the directed edge density as the fraction of ordered node pairs assigned a nonzero edge category, including self-loops. Let $E_{ij} \in \{0, \ldots, 10\}$ denote the discretized edge type with $E_{ij} = 0$ indicating "no edge." We define the loop-inclusive density

$$\delta_{\circlearrowleft}(G) \;=\; \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{I}[E_{ij} > 0]\,, \tag{47}$$

which is guaranteed to lie in $[0, 1]$. Under this definition, real graphs have mean density $\delta_{\circlearrowleft} = 0.135$, whereas synthetic graphs have mean density $\delta_{\circlearrowleft} = 0.998$, indicating that the generator produces graphs that are nearly complete directed graphs.

For completeness, we also report the classic simple digraph density that excludes self-loops,

$$\delta_{\times}(G) \;=\; \frac{1}{n(n-1)} \sum_{\substack{i,j=1 \\ i \neq j}}^{n} \mathbb{I}[E_{ij} > 0]\,, \tag{48}$$

for which the means are 0.125 (real) versus 0.999 (synthetic).

We additionally report the edge-to-node ratio

$$\rho(G) \;=\; \frac{|E^{+}(G)|}{n}, \qquad |E^{+}(G)| = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{I}[E_{ij} > 0]\,, \tag{49}$$

which increases from 2.83 (real) to 22.22 (synthetic). We visualize this imbalance in Figure 7. Taken together, these statistics show substantial over-production of edges in the generated graphs.

5.5 Shortest-path and connectivity statistics

We further quantify connectivity via shortest-path distances. For a directed graph $G = (V, E)$ with $|V| = n$, let $d(i, j)$ denote the length, based on number of edges, of the shortest directed path from node i to node j, with $d(i, j) = \infty$ if j is not reachable from i. Let $R = \{(i, j) \in V \times V : i \neq j,\ d(i, j) < \infty\}$ denote the set of reachable ordered node pairs. We define the average shortest path length as

$$\ell(G) \ = \ \frac{1}{|R|} \sum_{(i,j)\in R} d(i, j), \tag{50}$$

and the diameter as

$$\mathrm{diam}(G) \ = \ \max_{(i,j)\in R} d(i, j). \tag{51}$$

Under these definitions, real graphs have mean average shortest path length 2.46 ± 0.31 and diameter 5.06 ± 1.08 (maximum diameter 9). In contrast, synthetic graphs collapse to $\ell(G) = 1.0$ and $\mathrm{diam}(G) = 1.0$ across the generated set, meaning that nearly all reachable node pairs are connected by a direct edge (i.e., reachability in a single hop), consistent with the extreme edge density reported above. We visualize the distribution of these connectivity statistics in Figure 8 and Figure 9.

5.6 Directed-graph structure: self-loops, reciprocity, SCC structure, and transitivity

We additionally evaluate directed-structure summaries that are particularly relevant for API call transition graphs. Several statistics differ systematically between real and synthetic graphs:

- **Self-loop mass.** Although self-loops are permitted and modeled, the fraction of total edge weight contributed by self-loops is substantially larger in real graphs (0.288) than in synthetic graphs (0.050).
- **Reciprocity.** Synthetic graphs exhibit much higher reciprocity (mutual edge pairs $u \to v$ and $v \to u$) (0.949) than real graphs (0.342), indicating that the generator tends to produce bidirectional connectivity patterns that are atypical of real data.
- **Strongly connected components (SCCs).** A strongly connected component (SCC) of a directed graph is a maximal set of nodes such that every pair of nodes in the set is mutually reachable via directed paths. In our generated graphs, the mean number of SCCs is 1.00 and the mean largest-SCC fraction is 1.00, indicating that synthetic samples are, on average, entirely strongly connected. In contrast, real graphs have 2.25 SCCs on average and a mean largest-SCC fraction of 0.919, implying that real API-call graphs typically contain multiple SCCs and are not fully strongly connected.
- **Reciprocal and transitive closure motifs.** We report two simple directed motif summaries computed on the binarized adjacency $A_{ij} = \mathbb{I}[E_{ij} > 0]$. First, the number

of mutual dyads counts unordered node pairs that are connected in both directions,

$$M(G) \;=\; \sum_{1\le i<j\le n} A_{ij}A_{ji}. \tag{52}$$

Synthetic graphs exhibit substantially more mutual dyads (mean 260.17) than real graphs (mean 12.43), indicating that the generator tends to produce near-bidirectional connectivity. Second, we report the transitive ratio, which we define as a directed two-step closure rate: letting $B = \mathbb{I}[A^2 > 0]$ indicate ordered pairs (i, k) that are reachable by at least one directed length-2 path, we compute

$$\tau(G) \;=\; \frac{\sum_{i,k} B_{ik}\, A_{ik}}{\sum_{i,k} B_{ik}}. \tag{53}$$

Synthetic graphs have $\tau = 0.948$ versus $\tau = 0.146$ for real graphs, showing that whenever a two-step reachability relation exists, it is almost always "closed" by a direct edge in synthetic samples, creating over-connected structure.

- **Normalized outgoing transition entropy.** To quantify how concentrated each node's outgoing transition mass is, we compute the Shannon entropy [25] of the outgoing edge-weight distribution and normalize by its maximum possible value. Concretely, for a node u with outgoing weights $\{w_{uv}\}_{v:\, w_{uv}>0}$, define

$$p_{u\to v} \;=\; \frac{w_{uv}}{\sum_{v':\, w_{uv'}>0} w_{uv'}}, \qquad H_{\text{out}}(u) \;=\; -\sum_{v:\, w_{uv}>0} p_{u\to v}\log p_{u\to v},$$

and the normalized entropy $\widetilde{H}_{\text{out}}(u) = H_{\text{out}}(u)/\log d_u^+$ for $d_u^+ > 1$, where $d_u^+ = |\{v : w_{uv} > 0\}|$ is the out-degree (we set $\widetilde{H}_{\text{out}}(u) = 0$ when $d_u^+ \le 1$). Averaging $\widetilde{H}_{\text{out}}(u)$ over nodes with nonzero out-degree yields a graph-level summary. Under this metric, synthetic graphs exhibit near-maximal normalized entropy (0.986) compared to real graphs (0.479), indicating that generated nodes distribute outgoing transition mass almost uniformly across many successors rather than concentrating probability on a small set of plausible next calls.

5.7 Per-family structural summaries under conditional generation

Because generation is class-conditional via classifier guidance, we also report per-family averages of size and weight statistics in Figure 10 and Figure 11. Across families, synthetic graphs generally match node counts well, but consistently exhibit substantially inflated edge totals relative to real graphs. This indicates that the principal mismatch is not confined to a particular family, but is systematic across conditioning values.

Table 5 Per-family accuracy for synthetic-only evaluation.

Family	Accuracy	Support
Adload	0.7857	14
Bancos	1.0000	14
OnlineGames	0.8571	14
Vbinject	0.8571	14
Vundo	0.8571	14
WinWebSec	1.0000	14
Zwangi	1.0000	14
Overall	0.9082	98

5.8 Downstream classification: class-consistency vs. detectability

We assess four complementary discriminative properties.

(i) Train-on-syntheetic, test-on-synthetic accuracy.

A 7-way malware-family classifier evaluated on synthetic graphs achieves strong accuracy (overall ≈ 0.91), indicating that generated graphs encode family-discriminative signal that is detectable by a downstream classifier. We show these results in Table 5. This supports the conclusion that classifier guidance successfully steers generation toward class-specific samples.

(ii) Train-on-real, test-on-synthetic accuracy.

To test how well our synthetic graphs preserve the same class-conditional structure as the real distribution, we evaluate cross-domain generalization: we train a 7-way malware-family classifier on real graphs and test it directly on the full synthetic set. Accuracy drops to 0.382 on 490 synthetic graphs, substantially below the real-data baseline of 0.869 (training and testing on real graphs), as seen in Table 7 and only modestly above the $1/7 \approx 0.143$ chance level. Moreover, performance is highly family dependent: the real-trained classifier transfers well to some synthetic families (e.g., OnlineGames and Vbinject), while collapsing to near-zero accuracy on others (e.g., Adload and Bancos). We display these results in Table 6. So although guided generation yields samples that are discriminative within the synthetic distribution, the discriminative cues do not consistently align with those that define the real class-conditional distributions.

(iii) Train-on-synthetic, test-on-real accuracy.

We also evaluate the reverse transfer direction by training the 7-way malware-family classifier on synthetic graphs and testing it on real graphs. This probes whether the synthetic distribution contains class-discriminative features that generalize back to the real data manifold. Overall accuracy is 0.187 on 492 real graphs, only slightly above the $1/7 \approx 0.143$ chance level. The per-family breakdown is highly sparse: effective transfer

Table 6 Per-family accuracy for train-on-real, test-on-synthetic evaluation.

Family	Accuracy	Support
Adload	0.0571	70
Bancos	0.0571	70
OnlineGames	0.8286	70
Vbinject	0.7857	70
Vundo	0.5857	70
WinWebSec	0.1714	70
Zwangi	0.1857	70
Overall	0.3816	490

Table 7 Baseline real-only per-family accuracy.

Family	Accuracy	Test n
Adload	0.9286	14
Bancos	0.7857	14
OnlineGames	0.7857	14
Vbinject	0.9286	14
Vundo	1.0000	15
WinWebSec	0.7857	14
Zwangi	0.8571	14
Overall	0.8687	99

is observed primarily for Vbinject (0.757) and WinWebSec (0.557), while accuracy collapses to 0.0 for the remaining families (Adload, Bancos, OnlineGames, Vundo, Zwangi). We show these findings in Table 8. This asymmetric pattern indicates that, for most families, the synthetic graphs encode class information in a way that does not align with real-data decision boundaries, reinforcing the conclusion that guided generation yields class-separable structure within the synthetic domain without consistently preserving the real class-conditional structure.

(iv) Real-vs-synthetic detectability.

Finally, to concretely assess the distributional mismatch, we train a binary classifier to distinguish real graphs from synthetic graphs [26]. The detector achieves perfect test accuracy (1.00), indicating that synthetic samples occupy a region of feature space that is trivially separable from real data. This result reinforces the cross-domain generalization failure in (ii) and (iii): the mismatch is sufficiently large that, even when synthetic graphs carry class-discriminative signal, they do so in a way that remains strongly non-realistic.

Table 8 Per-family accuracy for train-on-synthetic, test-on-real evaluation.

Family	Accuracy	Support
Adload	0.0000	70
Bancos	0.0000	71
OnlineGames	0.0000	70
Vbinject	0.7571	70
Vundo	0.0000	71
WinWebSec	0.5571	70
Zwangi	0.0000	70
Overall	0.1870	492

5.9 Summary of results

Overall, synthetic graphs match the marginal distribution of graph sizes (node counts) reasonably well, but diverge sharply from real graphs in edge-weight distribution, degree/edge volume, shortest-path geometry, and multiple directed-structure metrics. At the same time, classifier-guided sampling yields synthetic graphs that remain strongly discriminative by malware family, even while a real-vs-synthetic classifier can perfectly separate the two domains. These findings motivate the analysis in Section 6, where we discuss mechanistic reasons for "class-consistent but unrealistic" generation under limited-data discrete diffusion on directed, weighted graphs.

6 Discussion

Our results support a consistent pattern: the adapted discrete graph diffusion model can generate samples that are class-discriminative, but structurally unrealistic [27–30]. This combination is not contradictory. In modern generative modeling, it is common to evaluate distributional mismatch using classifier-based two-sample tests and related discriminative diagnostics; strong detectability can coexist with preservation of some label-relevant structure. In our setting, this tension is plausibly amplified by the limited dataset size (492 graphs) relative to the effective dimensionality of the discrete, directed, weighted adjacency representation, with $O(n^2)$ categorical edge variables per graph.

6.1 Why classifier guidance yields class-consistent samples

Classifier guidance provides a direct mechanism for steering sampling toward a target label by modifying the reverse diffusion transition using gradients of a classifier evaluated on noisy states. In our setting, the condition is the malware family $y \in \{1, \ldots, 7\}$, and the auxiliary classifier is trained to predict y from noisy graphs (X^t, E^t). During sampling, guidance increases the probability of reverse-steps that increase the classifier likelihood of the target family. Consequently, even if the unconditional generator is imperfect, the guided sampler can amplify features that the classifier deems predictive of the target family, producing samples that are aligned with y.

A critical implication is that guidance optimizes for conditional predictability rather than full distributional realism. In particular, if the classifier relies on a subset of highly predictive but non-representative cues, then guidance can over-emphasize those cues. This provides a mechanistic explanation for why we observe strong family-consistency under downstream classification while simultaneously observing large structural deviations from real graphs.

6.2 Why realism fails under limited-data discrete diffusion on directed, weighted graphs

The central negative result is not that the model fails to encode label information, but that it fails to match the unconditional graph distribution. We view this primarily as a sample complexity and representation issue: the discrete diffusion model is attempting to learn a very high-dimensional categorical distribution over directed, weighted adjacency tensors from a dataset that is small in the relevant degrees of freedom.

(i) Combinatorial output space: $O(n^2)$ categorical edge variables.

For a graph with n nodes, the diffusion state includes n node-type variables and n^2 directed edge-type variables. In our dataset, typical graphs have $n \approx 22$, so each sample contains on the order of $n^2 \approx 484$ edge positions, each taking values in $\{0, \ldots, 10\}$. Even if the mean number of observed edges is small in real data, the model must still learn to place nearly all mass on the no-edge category for the vast majority of node pairs, while simultaneously learning class-discriminative deviations for just a few number of edges. With only 492 graphs, the empirical coverage of the joint distribution over directed edges and weights is extremely sparse relative to the dimensionality of the state space, making it easy for the learned reverse process to drift toward incorrect solutions. In our case, this takes the form of overly dense connectivity.

(ii) Discrete diffusion magnifies modeling errors in the no-edge mass.

In categorical diffusion, realism depends strongly on accurately modeling the probability of the dominant categories (in our case, the no-edge category). Small bias away from the no-edge class can lead to dramatic qualitative failures because errors compound over $O(n^2)$ edge variables and over many reverse steps. This is consistent with our most salient discrepancy, where synthetic graphs massively over-produce edges and collapse shortest-path statistics toward complete-digraphs. That is, even if the model is moderately accurate on present edge weights, it can still be globally unrealistic if it misallocates probability mass between "edge" and "no edge."

(iii) Guidance can preserve label signal while worsening off-manifold artifacts.

As discussed in Section 6.1, guidance introduces an explicit tradeoff: increasing the guidance strength improves adherence to the target label but can push samples toward regions that are high under the classifier yet low under the data distribution. This effect is well-known in guided diffusion more broadly and is especially acute when the base generative model is imperfect. In our setting, the base model already exhibits

structural bias toward over-connectivity, and guidance can further exacerbate such effects if they correlate with classifier confidence for particular classes.

(iv) Continuous/latent diffusion as an empirical counterpoint.

The contrast with [12] is informative because it holds the dataset fixed while changing the modeling domain. The authors of [12] perform diffusion in a continuous representation space (learned embeddings) rather than directly in the discrete graph state space. This changes the effective geometry of the learning problem: continuous diffusion operates on smooth perturbations in $\mathbb{R}^D$, while discrete diffusion must learn many coupled categorical decision boundaries. More importantly, continuous or latent diffusion can leverage representation learning to compress and denoise important factors of variation, reducing the effective dimensionality that must be learned from limited samples. This provides a plausible explanation for why embedding-space diffusion can appear substantially less "data-hungry" on the same dataset than discrete diffusion over directed, weighted adjacency tensors.

(v) Directed, weighted behavior graphs impose additional realism constraints.

Finally, API transition graphs are not generic directed graphs: they inherit constraints from an underlying sequence-generating process (e.g., sparsity, asymmetry, nontrivial SCC structure, etc.). Learning these constraints from few samples is difficult, and our directed-structure diagnostics indicate that the generator does not recover them (e.g., inflated reciprocity, inflated connectivity, and near-maximal entropy of the succeeding node). These are precisely the kinds of constraints that are easy to violate in graph tensor generation, because they arise from global organization (paths, reachability, flow concentration) rather than from local marginal statistics.

6.3 Implications

The main implication is that, on datasets of the scale considered in our experiment ($\sim$ 500 graphs), discrete diffusion directly over directed, weighted adjacency tensors can succeed at conditional discrimination, via guidance, while failing at distributional fidelity. This suggests that, in low-data regimes, evaluation should explicitly separate (i) label-consistency or class-conditional separability from (ii) realism under two-sample tests and structural statistics. It also suggests that, for malware synthesis under limited samples, continuous/latent diffusion approaches may be better matched to the available data than direct discrete diffusion over full graph state spaces, even when the downstream object of interest is graph-structured.

Overall, our findings indicate that the primary bottleneck is not the ability to encode family information, but the fundamental difficulty of learning a high-dimensional discrete graph distribution from a small corpus. This perspective helps contextualize why embedding- or latent-space diffusion methods can appear less data-hungry on the same dataset: they shift modeling burden from a combinatorial adjacency tensor to a lower-dimensional continuous space. This motivates future work

on representations or priors that reduce the effective complexity of the graph generation problem under data scarcity, rather than further increasing guidance strength or model capacity alone.

Declarations

- The work in this article was supported by the National Science Foundation (NSF) under Award #2244597.
- The dataset used in this article can be downloaded from the following address: https://drive.google.com/drive/folders/1eDv1gSO-RUjILiZobprOyoflNMy65Oe4 (accessed March 2026)

Editorial Policies for:

Springer journals and proceedings: https://www.springer.com/gp/editorial-policies

Nature Portfolio journals: https://www.nature.com/nature-research/editorial-policies

Scientific Reports: https://www.nature.com/srep/journal-policies/editorial-policies

BMC journals: https://www.biomedcentral.com/getpublished/editorial-policies

References

[1] Perera, W.S., Jiang, H.: BEACON: Behavioral Malware Classification with Large Language Model Embeddings and Deep Learning (2025). https://arxiv.org/abs/2509.14519

[2] Egele, M., Scholte, T., Kirda, E., Krügel, C.: A survey on automated dynamic malware-analysis techniques and tools (2012). https://api.semanticscholar.org/CorpusID:1863333

[3] Maniriho, P., Mahmood, A.N., Chowdhury, M.J.M.: API-MalDetect: Automated malware detection framework for windows based on API calls and deep learning techniques (2023). https://doi.org/10.1016/j.jnca.2023.103704 . https://www.sciencedirect.com/science/article/pii/S1084804523001236

[4] Nikolopoulos, S.D., Polenakis, I.: A graph-based model for malware detection and classification using system-call groups (2016). https://api.semanticscholar.org/CorpusID:255167673

[5] Sohl-Dickstein, J., Weiss, E.A., Maheswaranathan, N., Ganguli, S.: Deep Unsupervised Learning using Nonequilibrium Thermodynamics (2015). https://arxiv.org/abs/1503.03585

[6] Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., Frossard, P.: DiGress: Discrete Denoising diffusion for graph generation (2023). https://arxiv.org/abs/2209.14734

[7] Ho, J., Jain, A., Abbeel, P.: Denoising Diffusion Probabilistic Models (2020). https://arxiv.org/abs/2006.11239

[8] Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-Based Generative Modeling through Stochastic Differential Equations (2021). https://arxiv.org/abs/2011.13456

[9] Bronstein, M.M., Bruna, J., Cohen, T., Veličković, P.: Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges (2021). https://arxiv.org/abs/2104.13478

[10] Qin, Y., Vignac, C., Frossard, P.: Sparse Training of Discrete Diffusion Models for Graph Generation (2024). https://arxiv.org/abs/2311.02142

[11] Dhariwal, P., Nichol, A.: Diffusion Models Beat GANs on Image Synthesis (2021). https://arxiv.org/abs/2105.05233

[12] Kapoor, G., Nadipalli, S., Di Troia, F.: Embedding-Driven Synthetic Malware Generation with Autoencoders and Cluster-Tangent Diffusion (2025). https://doi.org/10.3390/app152111791 . https://www.mdpi.com/2076-3417/15/21/11791

[13] Bao, T., Trousil, K., Duy Tran, Q., Di Troia, F., Park, Y.: Generating Synthetic Malware Samples Using Generative AI (2025). https://doi.org/10.1109/ACCESS.2025.3556704

[14] Cui, L., Cui, J., Ji, Y., Hao, Z., Li, L., Ding, Z.: API2Vec: Learning Representations of API Sequences for Malware Detection. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3597926.3598054 . https://doi.org/10.1145/3597926.3598054

[15] Li, S., Zhou, Q., Zhou, R., Lv, Q.: Intelligent malware detection based on graph convolutional network (2022). https://doi.org/10.1007/S11227-021-04020-Y . https://doi.org/10.1007/s11227-021-04020-y

[16] Ma, C., Li, Z., Long, H., Bilal, A., Liu, X.: A malware classification method based on directed API call relationships. Public Library of Science (2025). https://doi.org/10.1371/journal.pone.0299706 . https://doi.org/10.1371/journal.pone.0299706

[17] Manchanda, S., Gupta, S., Ranu, S., Bedathur, S.J.: Generative Modeling of Labeled Graphs Under Data Scarcity. PMLR (2024). https://proceedings.mlr.press/v231/manchanda24a.html

[18] Aggarwal, S., Di Troia, F.: Malware Classification Using Dynamically Extracted API Call Embeddings (2024). https://doi.org/10.3390/app14135731 . https://www.mdpi.com/2076-3417/14/13/5731

[19] Fey, M., Lenssen, J.E.: Fast Graph Representation Learning with PyTorch Geometric (2019). https://arxiv.org/abs/1903.02428

[20] Fellicious, C., Bischof, M., Mayer, K., Eikenberg, D., Hausotte, S., Reiser, H.P., Granitzer, M.: Malware Detection based on API calls (2025). https://arxiv.org/abs/2502.12863

[21] Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.J.: Graph Transformer Networks (2020). https://arxiv.org/abs/1911.06455

[22] Smirnov, N.: Table for Estimating the Goodness of Fit of Empirical Distributions. Institute of Mathematical Statistics (1948). https://doi.org/10.1214/aoms/1177730256 . https://doi.org/10.1214/aoms/1177730256

[23] Mann, H.B., Whitney, D.R.: On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. Institute of Mathematical Statistics (1947). https://doi.org/10.1214/aoms/1177730491 . https://doi.org/10.1214/aoms/1177730491

[24] Pearson, K.: On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to Have Arisen from Random Sampling. Taylor & Francis (1900). https://doi.org/10.1080/14786440009463897

[25] Shannon, C.E.: A mathematical theory of communication (1948). https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[26] Lopez-Paz, D., Oquab, M.: Revisiting Classifier Two-Sample Tests (2018). https://arxiv.org/abs/1610.06545

[27] Du, Y., Wu, G., Tang, G., Wang, W., Fan, Q.: Characterizing Model Behavior Under Synthetic Data Training: An Empirical Study Across Scales and Mixing Ratios (2025). https://arxiv.org/abs/2510.05133

[28] Breugel, B., Qian, Z., Schaar, M.: Synthetic data, real errors: how (not) to publish and use synthetic data (2023). https://arxiv.org/abs/2305.09235

[29] Hennicke, L., Adriano, C.M., Giese, H., Koehler, J.M., Schott, L.: Mind the Gap Between Synthetic and Real: Utilizing Transfer Learning to Probe the Boundaries of Stable Diffusion Generated Data (2024). https://arxiv.org/abs/2405.03243

[30] Koca, G.: Synthetic to Real Domain Adaptation for Weather Classification (2024). https://hdl.handle.net/20.500.12608/83214



Fig. 1 End-to-end workflow of our experiment.

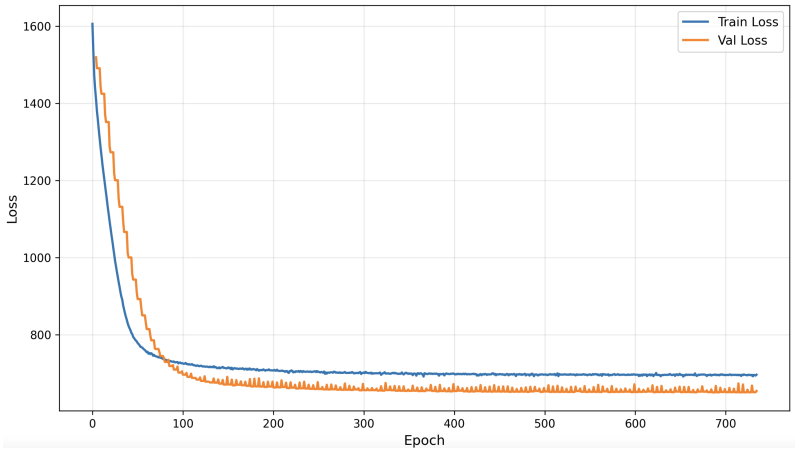


Fig. 2 Training and validation loss curves for the unconditional diffusion model.

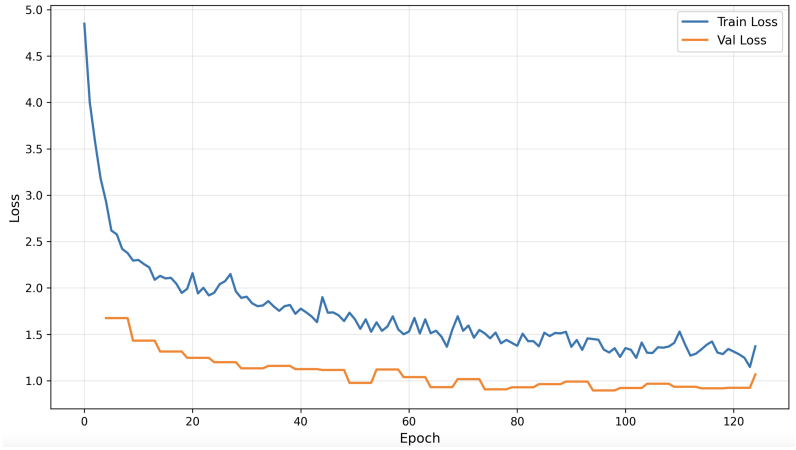


Fig. 3 Training and validation loss curves for the guidance classifier.

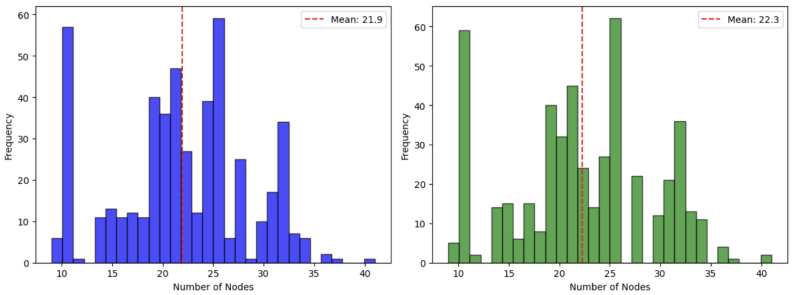


Fig. 4 Number of nodes distribution in the real dataset (blue) and synthetic dataset (green).

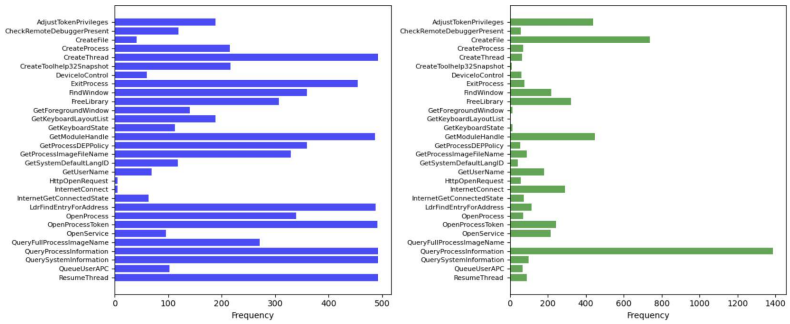


Fig. 5 Frequency of the top 30 nodes in the real dataset (blue) and the corresponding frequency of each node in the synthetic dataset (green).

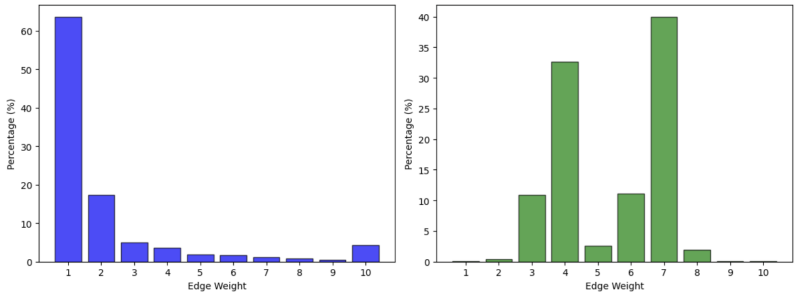


Fig. 6 Frequency of the top 30 nodes in the real dataset (blue) and the corresponding frequency of each node in the synthetic dataset (green).

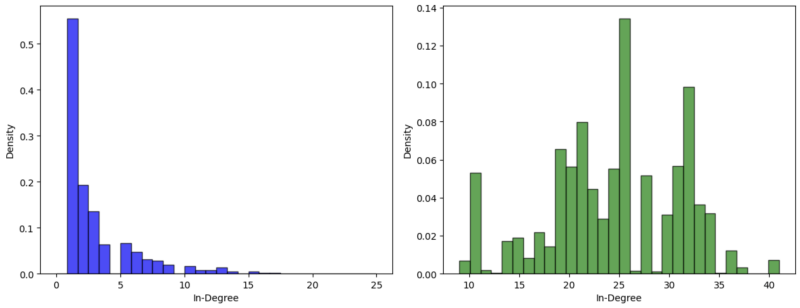


Fig. 7 Distribution of in-degree of nodes in the real dataset (blue) and synthetic dataset (green).

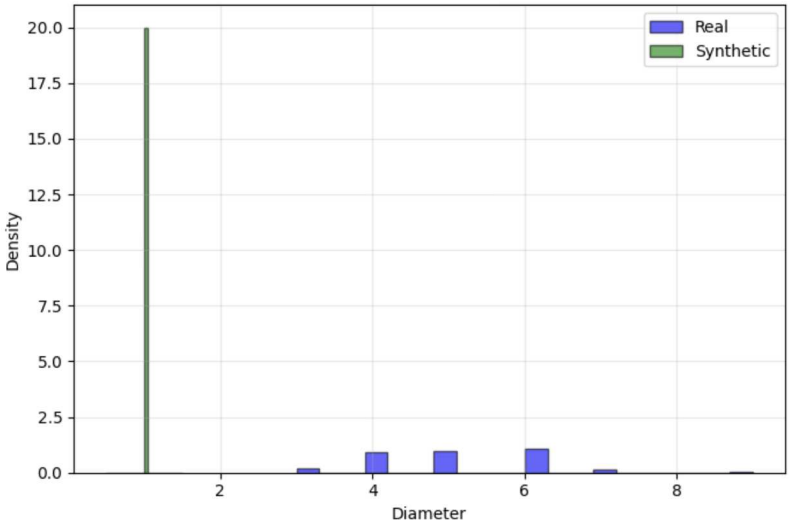


Fig. 8 Graph diameter distribution.

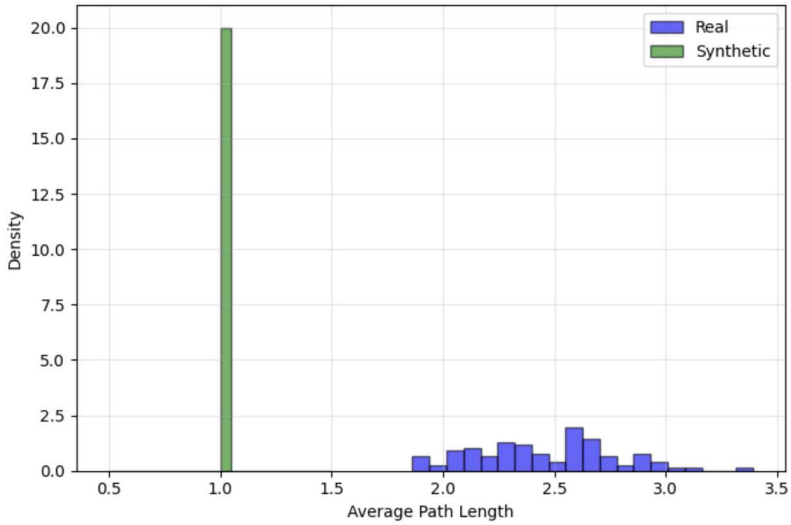


Fig. 9 Distribution of average shortest path length.

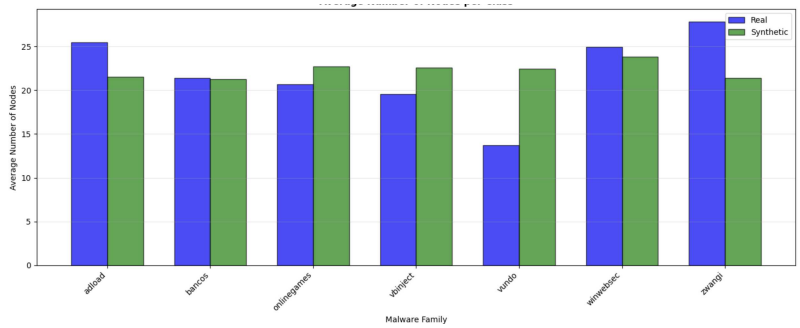


Fig. 10 Average number of nodes per graph compared across all malware families.

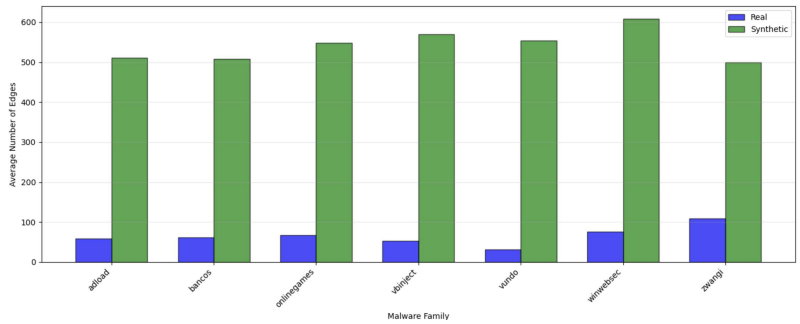


Fig. 11 Average number of edges per graph compared across all malware families.