Modeling Realistic Malware Behaviors: A Two-Stage Pipeline for Synthetic Malware Log Generation using Deep Learning Methods

by

Sharaf Rashid
S.B. Computer Science and Engineering, Massachusetts Institute of Technology, 2025

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2026

Authored by:	Sharaf Rashid Department of Electrical Engineering and Computer Science January 12, 2026
Certified by:	Amar Gupta Research Scientist, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair Master of Engineering Thesis Committee

Modeling Realistic Malware Behaviors: A Two-Stage Pipeline for Synthetic Malware Log Generation using Deep Learning Methods

by

Sharaf Rashid

Submitted to the Department of Electrical Engineering and Computer Science in on January 12, 2026 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Synthetic malware log generation is a useful method for augmenting behavior-based malware detection methods, since real malware logs are scarce, costly to obtain, and often unavailable for new malware threats that emerge. Existing generative approaches typically focus on either synthesizing static tabular features or dynamic behavioral sequences, but not both. This thesis proposes a two-stage generative pipeline that first synthesizes realistic static malware profiles and then conditions the generation of dynamic event sequences on these profiles. The first stage evaluates GAN, VAE, and diffusion-based models in generating static profiles of malware. The second stage evaluates a transformer-based autoregressive model conditioned on static profiles to produce realistic and semantically correct system-call traces. Comprehensive evaluation across multiple distributional, semantic, and embedding-based metrics shows that both stages produce highly realistic synthetic data. Importantly, static profile conditioning enables end-to-end malware log generation where the static attributes are consistent with the resulting dynamic behavior.

Thesis supervisor: Amar Gupta
Title: Research Scientist

Acknowledgements

I want to begin by thanking my thesis supervisor, Dr. Amar Gupta, for his continuous support and guidance as I conducted research and wrote this thesis. Working in his lab has been an enjoyable experience for me since I started as a UROP in the fall semester of my senior year at MIT. The experience of leading one of his research groups has allowed me to grow immensely both technically and non-technically, whether it be approaching abstract and complex technical problems, managing students under me, or preparing content for academic writing and presentations.

I want to also thank the other members who worked on my project either currently or in the past: Dr. Rafael Palacios, Fabian Yañez-Laguna, Aiden Foucault Etheridge, Maureen Zhang, Ellington Hemphill, Jacob Mercado, and Sicheng Zhou. I especially appreciate the guidance of Dr. Palacios in academic writing and project direction, Fabian's efforts in assisting me in the early conceptions of the project, Aiden for continuing to lead the project research while I was away for the summer, and Ellington for helping me explore many of the evaluation methods used in this thesis.

Next, I want to thank the team at Itaú Unibanco, Lucas Orosco, Edson Bollis, Darian Rabbani, Agatha Noguiera, Felipe Alemida, and Jair Junio for their constant feedback and support during this project. Their input was immensely useful in deciding the direction of this thesis and iterating on ideas.

Last, I would like to thank my family for their constant support in my life, and my friends for making my experience at MIT unforgettable. Our shared experiences and memories are some of the best times of my life.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Motivations

In recent years, malware has become both more frequent and more complex with one report indicating that over 560,000 new malware instances are detected daily, which contributes to an already large ecosystem of a billion known variants of malware (Ferdous et al., 2025). This increase in threat volume and evolution in malware has left many traditional detection methods insufficient, meaning new detection methods must be developed to address the issue. In the past, malware detection systems relied on a signature-based approach, utilizing known matching patterns in executable files to identify threats (Kothamali & Banik, 2022). This method is a reactive approach, however, and fails in cases where the attack is a completely novel pattern. Additionally, attackers are beginning to use obfuscation techniques and machine learning to craft malware that is designed to evade standard detection methods, making legacy detection systems obsolete (Ferdous et al., 2025; Kothamali & Banik, 2022).

Due to the shortcomings of prior methods, researchers have begun exploring behavior-based detection methods which analyze how programs interact with a system during their execution (Huang et al., 2024). These techniques utilize structured logs such as API or system call traces, permission usage, file accesses and other system events to look for malicious patterns. Ferdous et al. (2025) support this approach as well, stating that behavior-based and ML-driven approaches are now an essential part of modern cybersecurity defense.

In practice, training behavior-based detection models requires a large amount of high-quality malware logs to exist. However, such datasets are scarce, with only a few publicly available options (Huang et al., 2024). This scarcity is compounded by privacy and proprietary concerns, which discourage organizations from sharing malware logs. Moreover, manually generating logs requires executing malware in sandboxes or testbeds, a potentially dangerous and expensive process, further limiting the availability of malware logs (Huang et al., 2024). All these challenges highlight the appeal of using synthetic logs in training malware detection methods to both increase the number of available training samples and potentially discover new attack patterns before they arrive.

With the significance of log-based synthetic malware generation established, we now arrive at the central motivation of this thesis: being able to control and steer dynamic log generation using static traits. In real world contexts, security researchers may often know how specific malware strains target a particular API, system permission, or hardware (e.g. location services, contact access, messaging APIs), but they might not have access to the dynamic traces of these known malwares. In these cases, it becomes useful to have samples of what this malware may look like, which requires a plausible static profile to steer generation of the dynamic log. This thesis examines this problem in detail by developing a full end-to-end architecture for addressing it. We propose a design that produces synthetic static malware profiles which then are used to condition the synthetic generation of dynamic malware event logs.

1.2 Problem Statement and Contributions

Although synthetic malware logs offer a promising solution to the scarcity of real logs, existing approaches (as discussed in the next section) focus on just one aspect of malware: its static, tabular-like characteristics or its dynamic, sequential behaviors. However, in practice, both components are linked with each other since a malware sample's declared permissions and flags should, by logic, influence the dynamic behavior that occurs during execution. Moreover, in practical threat-analysis scenarios, researchers often know which static characteristics a new malware strain is likely to exploit (e.g., a certain OS level permission). This creates the need to predict what such an attack might look like both statically and dynamically using the subset of known static information.

The central problem addressed in this thesis is the absence of a synthetic data generation framework that produces malware logs that contain both static and dynamic information which are logically consistent with one another (described in Section 3.2). Addressing this problem requires solving two interconnected problems. First, we must be able to generate realistic synthetic static profiles. Second, given such profiles, we must generate dynamic event sequences which are both realistic and consistent with the provided static profile. These two problems motivate the design used in this thesis that first synthesizes static information about malware and then uses it to condition the generation of malware event sequences.

This thesis makes the following key contributions:

1. **Static malware profile generation.**
We train and evaluate GAN-based (Goodfellow et al., 2020), VAE-based (Kingma & Welling, 2022), and diffusion-based generative models (Ho et al., 2020) to identify the most effective approach for generating high-dimensional static malware profiles. We specifically assess whether these models preserve the distributional characteristics of real malware data.

2. **Conditional dynamic event sequence generation.**
We design and evaluate a transformer-based (GPT-2) autoregressive decoder (Radford et al., 2019) for generating dynamic malware event sequences conditioned on static profiles. This allows the model to preserve static-dynamic relationships present in real malware logs. We assess both general realism and conditional consistency between synthetic and dynamic parts.

3. **A unified two-stage pipeline for synthetic malware log generation.**
We design and implement an end-to-end system combining the above methods that use real malware logs to produce consistent, realistic synthetic malware logs.

Chapter 2: Related Work

This chapter explores past approaches in both general and malware-specific literature in the areas of synthetic generation of static, tabular data and synthetic generation of dynamic, sequential data. A primary goal of this thesis is to bridge the static section of the malware log to its dynamic section, so this chapter will also explore past approaches in conditional sequence generation from "static" profiles. Lastly, this chapter will discuss gaps in the current literature and how this thesis will close these gaps, with its proposed two-stage pipeline for synthetic malware log generation.

2.1 Synthetic Generation of Tabular Data

Before the rise of deep learning techniques, data scarcity and class imbalance was addressed using oversampling techniques. The most prominent method was SMOTE, which created new instances of minority samples by interpolating between existing ones (Chawla et al., 2002). Chawla et al. showed in their work how by oversampling harder-to-learn examples, they could improve classifier performance on imbalanced data. Variants of SMOTE subsequently arose that improved upon Chawla et al.'s results such as ADASYN (He et al., 2008). However, while these techniques were effective with simple feature spaces, they fell apart when introduced to complex feature dependencies in high-dimensional data (Blagus & Lusa, 2012). Deep learning methods have been able to overcome the shortcomings of prior approaches by learning richer representations of data distributions, with generative adversarial networks (GANs) and variational autoencoders (VAEs) being some of the first effective approaches for synthetic tabular data generation (Kingma & Welling, 2022; Xu et al., 2019).

GANs are a type of generative model with two components, a generator and a discriminator, designed to synthetically produce complex, high dimensional data (Goodfellow et al., 2020). The generator's role is to produce synthetic data from the original samples' latent space and the discriminator's role is to then distinguish real data from the generated samples, giving the generator feedback to update on. One weakness with GANs, however, is that they are known to suffer from mode collapse, where generators produce repetitive outputs or have training instability (Saxena & Cao, 2021). The Wasserstein GAN (WGAN) is designed to address these issues, modifying the GAN loss to use the Earth Mover (Wasserstein-1) distance instead of the Jensen-Shannon Divergence (JSD) from the original GAN model (Arjovsky et al., 2017). WGAN-GP builds on the WGAN by further stabilizing training through replacing weight clipping with a gradient penalty term that enforces the Lipschitz constraint (Gulrajani et al., 2017). In the field of synthetic tabular data generation, specialized GANs have also been developed to better fit the characteristics of tabular data. For instance, Table-GAN introduced convolution networks and added information and classification losses to the design to better preserve tabular statistics (Park et al., 2018). CTGAN was developed shortly after and applied conditional GANs to model data with both continuous and categorical distributions (Xu et al., 2019).

VAEs are a type of generative model that—similar to GANs—represent data through a low-dimensional latent space (Kingma & Welling, 2022). Unlike GANs which struggle sometimes with mode collapse, VAEs use a training objective that optimizes an evidence lower bound (ELBO), resulting in often more stable training (Kingma & Welling, 2022). A downside to this approach is that, compared to GANs, the synthetic samples are sometimes less precise (Doersch, 2021). Because standard VAEs tend to struggle with the heterogeneity of tabular data, coming from a mix of continuous, categorical and binary variables, Variational Autoencoder with Arbitrary Conditioning (VAEAC) was designed to address these issues and allow for better generation of tabular data (Nazábal et al., 2018).

Beyond GANs and VAEs, there exist Copula-based methods which do not rely on deep learning (Kamthe et al., 2021). Copula methods, however, tend to not scale well on very high-dimensional data (Nagler et al., 2019). Last, the most recent approach to tabular data generation is through diffusion models, which are a type of probabilistic generative model that learns to generate data by reversing a gradual noising process (Ho et al., 2020). TabDDPM, a recent diffusion model approach for tabular data generation, has been shown to outperform GAN and VAE approaches across a wideset of benchmarks including statistical fidelity, machine learning utility, and sampling diversity (Kotelnikov et al., 2023).

In the realm of malware, researchers have explored methods for applying generative techniques to the static features of malware. One early work using GANs is MalGAN, which generates malware feature vectors (based on API-call frequencies or permissions) designed to evade detection (W. Hu & Tan, 2017). Another work called GANG-MAM aims to generate new Android malware variants by altering the feature vectors of known malware samples (G et al., 2021). The idea behind GANG-MAM's approach is to apply the generated feature vector to existing malware samples to create new synthetic malware samples that are harder to detect. GANG-MAM, however, works directly on APKs of malware which is different from the focus of this thesis on creating synthetic malware logs. Both works still show that GANs, and in general, generative techniques, can be effectively applied to create new synthetic static malware samples.

2.2 Synthetic Generation of Sequential Data

Historically, sequential or time-series data has been modeled using recurrent neural networks such as LSTMs (long short-term memory) (Hochreiter & Schmidhuber, 1997), and more recently, using transformer-based models such as GPT-2 (Radford et al., 2019). LSTM-based models have shown strong performance in producing coherent sequences in areas ranging from text to handwriting generation (Yu et al., 2017). They have also been used directly in GAN architectures as the generator to further improve the task of realistic generation, as seen in designs such as SeqGAN, the first GAN-based sequence generation model (Yu et al., 2017). These LSTM-based GAN models have been used in domain-specific tasks to help in training detectors to identify anomalies in sequences as seen with LogGAN which generates sequences of log events to mimic normal operational logs (Xia et al., 2021).

LSTM-based models struggle in capturing long-term dependencies across large gaps in sequences and are not highly scalable due to the use of sequential computation algorithms; but both of these issues have been solved by the introduction of the transformer-based LLM architectures (Vaswani et al., 2017). In the field of network traffic, researchers designed PAC-GPT, a fine-tuned GPT-3 model that generates synthetic network traffic flows and patterns (Kholgh & Kostakos, 2023). The work of Kholgh & Kostakos (2023) demonstrates that finetuned LLMs provide a strong approach to reliably capture and reproduce patterns in complex and long sequential data.

In the area of malware, most works that focus on sequence generation deal primarily with software-level reconstructions such as at the opcode level. For instance, MalGPT fine-tunes GPT-2 to produce adversarial byte sequences of malware designed to evade malware detectors (J. L. Hu et al., 2021). This approach follows the auto-regressive structure used in the synthetic generation of other sequential or time-series data described above. However, more recently, an alternative line of work has emerged in malware sequence generation focusing on non-auto-regressive generation (Bao et al., 2025). With this approach, the entire malware sequence is encoded into a fixed-length embedding using transformer-based models like BERT or Word2Vec (Devlin et al., 2019; Mikolov et al., 2013). The embeddings are then fed into generative models such as GANs and diffusion models to produce new synthetic sequences, like the approach used in the generation of synthetic tabular data. Bao et al. (2025) explores this approach in detail finding that this embedding-based technique paired with diffusion-based generation produces highly realistic opcode sequences that improve malware classifiers.

2.3 Conditional Sequence Generation from Static Profiles

In each malware sample, the static information generated should influence the dynamic sequence that ends up being created. This essentially means that sequence generation is conditioned upon a specific "static profile". Although in the field of synthetic malware generation, there have been no prior works focusing on this topic, past work across numerous fields have explored the idea of generating sequences that are conditional on some static information. In healthcare, generating patient time-series data conditioned on static patient profiles has been a major area of research. RCGAN is an early framework that augments a recurrent (LSTM) GAN with static feature conditioning, by feeding the generator the patient's static vector at each time step of training (Esteban et al., 2017). Clinical-GAN builds upon the goals of RCGAN but uses the more recent transformers-based models instead of the LSTM-type ones from RCGAN (Shankar et al., 2023). In finance, conditional sequence generation has also been applied to generate financial event sequences given either customer profiles or market conditions. For instance, PersonaLedger produces streams of financial transactions conditioned on user personas relying on transformers-based LLMs for generation (Anonymous, 2025).

2.4 Gaps in Literature

Despite the growing use of generative models in both static and sequential contexts of malware, there remains a significant gap in methods that combines these two areas. As mentioned in the previous section, conditional sequence generation has been explored in fields like healthcare and finance but remains underdeveloped in malware. Specifically, there is an absence of architectures that generate realistic dynamic sequences conditioned on a malware's static attributes such as permissions and API usages. This thesis aims to close that gap by proposing a two-stage pipeline that can generate synthetic static attributes of malware that then condition the generation of dynamic sequences.

Additionally, most existing malware sequence generation approaches work directly on the software-artifacts of malware (such as bytecode and opcode) as seen with MalGPT and the work of Bao et al. In contrast, this work aims to focus on generating log level representations, which are more interpretable and easier to integrate into downstream detection tasks. This direction is in line with the approaches of LogGAN and PacGPT. Table 1 compares all works described in this chapter in relation to the work contributed to the thesis. In the next chapter, we present the proposed architecture that builds upon the foundations discussed in this section.

Table 1. Summary of related work in synthetic data generation and its relation to this thesis.

Category	Representative Work(s)	Data type	Generative Approach	Key Contribution	Limitations / Gaps
Oversampling (pre-deep learning)	SMOTE, ADASYN	Tabular	Interpolation-based	Provides solution to data scarcity and class-imbalance	Fails with high-dimensional data
Tabular GANs	Table-GAN, CTGAN	Tabular	GAN	Models complex feature distributions; handles mixed data	Training instability
Tabular VAEs	TVAE, VAEAC	Tabular	VAE	Stable training; smooth latent representations	Over-smoothing
Tabular Diffusion Models	TabDDPM	Tabular	Diffusion model	Statistical fidelity and diversity	Computationally expensive
Static Malware Feature Generation	MalGAN	Malware static features	GAN	Generates evasive malware feature vectors	Operates on APKs
Sequence Generation (LSTM)	LSTM, SeqGAN, LogGAN	Sequences / Logs	LSTM or LSTM-based GAN	Generates coherent event sequences	Poor long-range dependency modeling

Sequence Generation (Transformer)	PAC-GPT, MalGPT	Network / Malware sequences	Autoregressive Transformer	More efficient / scalable, captures long-range sequential dependencies	N/A
Embedding-based Sequence Generation	Bao et al. (2025)	Opcode sequences	BERT plus Diffusion model	High-realism for opcode-level sequences	Loses token-level temporal structure, not log-based
Conditional Sequence Generation	RCGAN, ClinicalGAN, PersonaLedger	Time-series data	LSTM or Transformer	Demonstrates conditioned sequence generation	Not applied to malware system logs
This Thesis	*This work*	Malware logs with static and dynamic information	For static: GAN, VAE, Diffusion model For dynamic: Transformer	First end-to-end pipeline generating consistent static and dynamic malware logs using conditioning	Data reductions made for computational efficiency

Chapter 3: Design and Methodology

This chapter discusses the selected dataset, preprocessing methodology of malware logs, chosen model architectures for static and dynamic generation, and the chosen evaluation metrics for assessing the quality of the synthetic malware logs. The design choices discussed in this section each contribute to the main goal of building a generative framework that can produce realistic, logically consistent synthetic malware logs containing both static profiles and dynamic event sequences.

3.1 Dataset and Preprocessing

This section describes the dataset used in this thesis and the preprocessing pipeline designed to transform raw malware logs into data representations suitable for input into generative models. We first introduce the CIC-Maldroid 2020 dataset, explaining its suitability for the primary tasks of this thesis (Mahdavifar et al., 2020). We then explain the design of the custom preprocessing framework, MalwareLogCodec, which extracts and encodes static profiles and dynamic execution traces from raw log JSON files into model-compatible data formats. Finally, we discuss feature and token selection strategies used to reduce dimensionality, as well as dataset balancing techniques to mitigate class imbalance during training. The diagram in Figure 1 below summarizes the preprocessing steps described in this section, showing how a raw JSON is converted into preprocessed representations ready to be inputted into generative models.

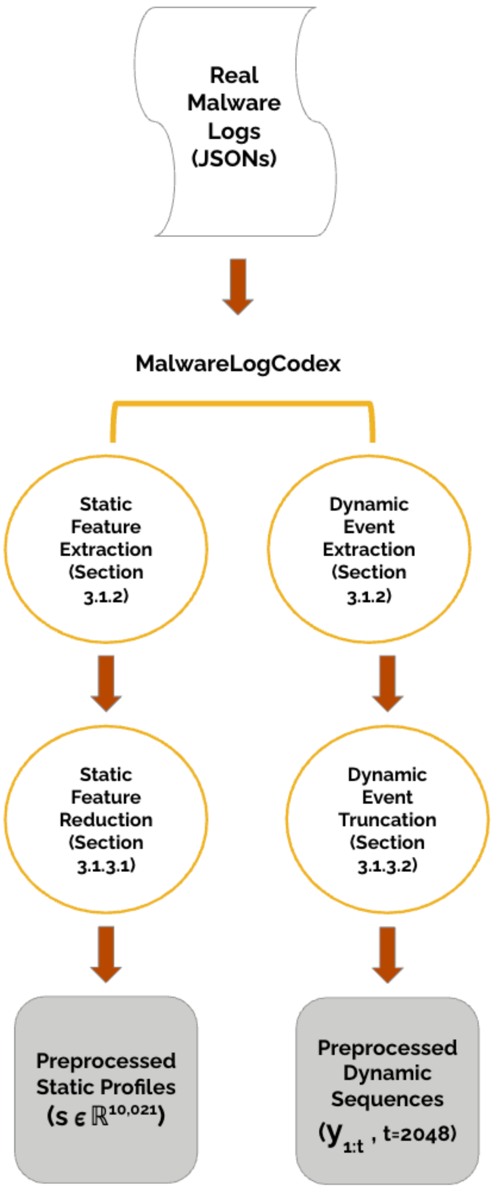

Real Malware Logs (JSONs)
MalwareLogCodex
Static Feature Extraction (Section 3.1.2)
Dynamic Event Extraction (Section 3.1.2)
Static Feature Reduction (Section 3.1.3.1)
Dynamic Event Truncation (Section 3.1.3.2)
Preprocessed Static Profiles ($s \in \mathbb{R}^{10,021}$)
Preprocessed Dynamic Sequences ($y_{1:t}$, t=2048)

Figure 1. Preprocessing pipeline to create model-ready representations.

3.1.1 Dataset Overview

This thesis uses Android malware logs from the CIC-Maldroid 2020 dataset (Mahdavifar et al., 2020). We choose this dataset because it is one of the few publicly available datasets that provides both static and dynamic behavioral data for each malware sample at an operating system level, unlike datasets which focus on either APK features or bytecode/opcodes. This allows us to complete our goal of building a generative framework that lets us model static-dynamic dependencies. Moreover, the dataset is large and diverse, containing 9,803 malware samples from 4 separate classes of malware: banking, SMS, adware, and ransomware, allowing our models to train on a rich set of patterns. Figure 1 highlights what the static and dynamic components of each malware log sample contain:

Table 2. Android malware log static and dynamic breakdown.

Component	Description
Static Profiles	Application permissions, application manifest flags, component existence, metadata, resource counts, configuration fields, etc.
Dynamic Sequences	Ordered sequences of system calls, file operation events, network operations, etc.

3.1.2 Preliminary Preprocessing

Because the malware logs are contained in JSON objects, preprocessing must be done to parse the logs and convert them into suitable forms to input into our generative models for both static and dynamic generation. For accurate modeling and later reconstruction, preprocessing must maintain the following rules: represent logs in a format that keeps the behavioral meaning of the malware; produce separate static vectors and dynamic sequences for each sample of malware; be compatible with the generative models (e.g., GANs, diffusion models, transformers), and last be reversible, meaning the components can be turned back into a JSON log format. These principles inspire the design of the MalwareLogCodec, which is a custom encoding/decoding system developed for this thesis to aid the process described above. The MalwareLogCodec performs two main preprocessing tasks: static feature extraction and dynamic event extraction.

For static feature extraction, the codec first identifies four types of static fields which account for all static features found within the malware logs:

numerical fields, Boolean fields, categorical fields, and map-like fields. This allows us to organize the static information in a tabular-like structure suitable for synthetic generation using generative models. In addition to purely static information, the static profile also appends a small set of dynamic summary information such as sequence length and event frequencies, which is used later in conditioning of our sequence generation models. For input into the generative models, each profile is then vectorized, where each static profile becomes:

$$s \in \mathbb{R}^D, \tag{1}$$

a sparse, interpretable vector and D is the total number of static features. To ensure compatibility with generative models that operate on continuous data, prior to training, static feature vectors are transformed into a normalized representation, following standard practices in synthetic tabular data generation (Kotelnikov et al., 2023; Park et al., 2018; Xu et al., 2019). Binary features are preserved as indicators while count-based and continuous features are scaled to bounded ranges using feature-specific caps from the training data. Such bounded normalizations improve training stability and prevent out-of-range artifacts from appearing during generation (Kotelnikov et al., 2023). This transformation is reversible allowing synthetic samples to be mapped back to the original feature space.

For dynamic event extraction, the events are converted into unique tokens that make up a symbolic vocabulary representing every distinct dynamic event found in our dataset. Each sequence is represented as:

$$y_1, y_2, \ldots\ y_t = y_{1:t}\ \in \{1, 2, \ldots, V\} \tag{2}$$

where $y_{1:t}$ is a sequence of t tokens where each token belongs to the vocabulary of tokens $\{1, 2, \ldots, V\}$. This abstraction of dynamic events allows us to compress dynamic event traces into sequential structures that share a common vocabulary, allowing us to directly train autoregressive decoder models.

3.1.3 Feature/Token Selection for Static and Dynamic Representations

Although the MalwareLogCodec from section 3.1.2 creates powerful, high-dimensional representations of malware logs, training deep generative models on the full static feature size and full dynamic event vocabulary can result in several issues. We find through analysis of the dataset that the full static feature space contains over 1 million unique features, and the full dynamic event vocabulary is over 98,000 tokens. Such high dimensionality often results in high memory requirements, high training times, risk of overfitting, and difficulty reaching training stability amongst generative models (Goodfellow et al., 2016). For these reasons, it is necessary to reduce the sizes while retaining the most important features and tokens.

3.1.3.1 Static Feature Selection

Let the static feature matrix be:

$$X \in \mathbb{R}^{N \times D}, \tag{3}$$

where N is the number of malware samples. For each static feature j, we compute the empirical variance and then pick the top 10,000 features by variance:

$$Var(X_j) = \mathbb{E}[X_j^2] - (\mathbb{E}[X_j])^2 \tag{4}$$

By selecting with variance, we are able to eliminate features with near-zero variance since they contribute vary little to the learning of models, because they are either always mostly present or mostly absent (Pudjihartono et al., 2022). Empirically, we find that the top 10,000 features capture over 99.99% of total variance, allowing us to stay within computational limits and keep features that are most important to the structure of static malware logs.

Selecting based purely on variance, however, can sometimes disregard semantically important but infrequent feature groups. To address this issue, we also ensure that the final reduced static vector includes important categories such as permissions, intent actions, sensitive APIs, and main activity. Similarly, for later construction of conditioned dynamic sequences, we also preserve top-level dynamic summary features such as sequence length and system call counts. This hybrid approach allows for both structural accuracy and computational efficiency in the models we train. After applying variance-based selection and preservation rules to the static vectors, the final reduced static profile dimensionality is 10,021 features.

3.1.3.2 Dynamic Token Selection and Static-Dynamic Interface

Dynamic execution traces consist of ordered sequences of system-level events that compose a large and highly sparse vocabulary. These dynamic events provide two roles in the proposed framework of the thesis. First, they provide a summary-level conditioning signal contained within static profiles, and second, they make up the full event vocabulary used in sequence generation. For the first role, by encoding what type of behaviors occur and at what frequency, it allows the conditional sequence generator to generate a sequence that is connected to the profile it is conditioned on, while also learning how those behaviors unfold over time.

We mentioned earlier that the total static profile dimensionality is 10,021 features and contained dynamic summary features. It would be infeasible for the static profile to contain summary features of all 98,152 tokens, so of the 10,021 retained features, the top 128 most frequent dynamic event count features are retained. We choose the number 128 as the cutoff value because we find that empirically, these 128 event tokens cover 99.9995% of all event occurrences in the dataset.

It is important to clarify that this reduction done on the static profiles does not apply to the dynamic sequence generator itself. Using transformer-based architectures, it is computationally feasible to train dynamic event sequences on the full dynamic vocabulary. This also results in a richer model. To maintain computational feasibility of evaluating our sequences, we, however, truncate sequences to be a maximum length of 2,048 events.

3.1.4 Dataset Balancing

The CIC-Maldroid 2020 is imbalanced in the number of malware samples for each class. The following table shows counts of each malware class in our dataset:

Table 3. Frequency of malware samples by class.

Malware Class	**Count**
Adware	1253
Banking	2100
SMS malware	3904
Riskware	2546

To prevent our generative models from overfitting to any single malware class, we balance the dataset so that each class appears with equal frequency: 1,251 samples. (The slight deviation from 1253 is due to 2 corrupted adware logs that were removed.) The balanced malware-only dataset is then used for static profile generation and conditional sequence generation, ensuring no single malware family disproportionately influences the learned behavior of models.

The dataset also contains benign log samples. These are excluded from model training since the focus is on malware generation and are instead used for downstream evaluation tasks such as assessing the utility of synthetic malware data in supervised classification tasks (described in Section 3.3.1).

3.2 Experimental Setup and Methodology

The primary goal of this thesis is to model the joint generative process of malware logs, specifically the relationship between a malware sample's static attributes and its dynamic behavior during execution. More formally, our objective is to learn the joint generative model:

$$p(s, y_{1:t}) = p(s)\, p(y_{1:t} \mid s) \tag{5}$$

where $s \in \mathbb{R}^D$ is a static malware profile, and $y_{1:t}$ is a variable-length dynamic event sequence. The static profile s consists of both features from the static portion of a malware log as well as dynamic summary statistics like sequence length and dynamic event frequencies.

This naturally calls for a two-stage generative pipeline. The first stage involves applying generative models to sample plausible static profiles from the learned distribution $p(s)$. The second stage involves applying generative models to produce event sequences that are conditioned on static profiles (i.e. modeling $p(y_{1:t} \mid s)$). Importantly, the conditional sequence generator is only conditioned on the dynamic summary subset of s. This design choice avoids the computational and optimization challenges of conditioning on the full 10,021 length vector s while retaining enough information to generate a synthetic sequence related to the full static profile.

This section begins by outlining the primary research questions that motivate the experiments that have been performed at each stage of the pipeline. For each stage, we describe the different generative architectures that we test in our experiments and the reasons we choose to evaluate them.

3.2.1 Research Questions

These are the two central research questions guiding our experimental methodology at each stage of the pipeline:

1. Which class of generative models, GANs, VAEs, or diffusion models, is most effective at learning the complex, high-dimensional distribution of malware static profiles?

2. How effective is a transformer-based autoregressive decoder at generating dynamic malware event sequences that are both realistic and consistent with a given static malware profile?

These questions reflect the two objectives of the thesis mentioned in Chapter 1: accurately modeling the structural dependencies present in static malware features and generating dynamic event sequences that are semantically consistent with the conditioned static profile.

3.2.2 Overview of the Two-Stage Generative Pipeline

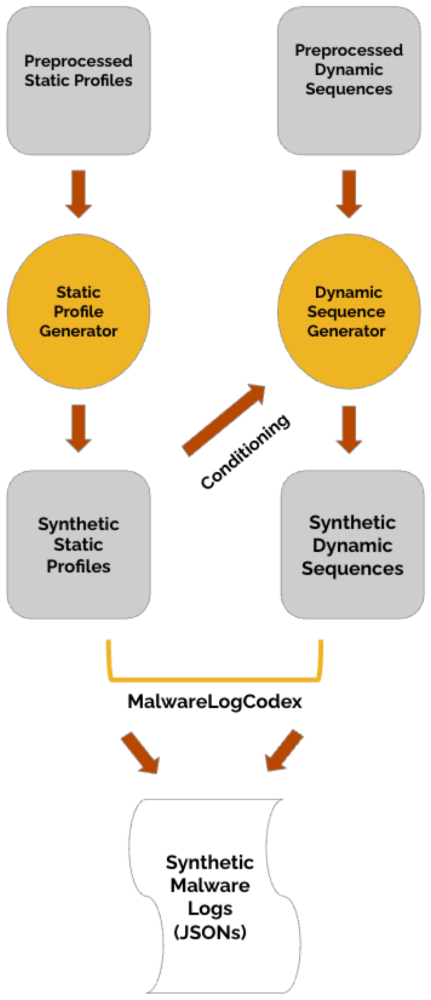

Preprocessed Static Profiles
Preprocessed Dynamic Sequences
Static Profile Generator
Dynamic Sequence Generator
Conditioning
Synthetic Static Profiles
Synthetic Dynamic Sequences
MalwareLogCodex
Synthetic Malware Logs (JSONs)

Figure 2. High level architecture of the two-stage generative pipeline.

Figure 2 provides a high-level overview of the proposed two-stage generative pipeline for producing synthetic malware logs. The remainder of this section provides at each stage, theoretical and practical motivations for each of the chosen model architectures.

3.2.3 Static Malware Profile Generation

The first stage of the pipeline seeks to model a distribution over static profiles, $p(s)$, where s is a vector representing a static profile. As a reminder, static malware profiles contain a variety of features including permissions, manifest attributes, resource counts, etc., and these set of features are strongly interdependent of each other. For instance, Android malwares that request high-privilege permissions (e.g., READ_SMS, RECEIVE_BOOT_COMPLETED) frequently coincide with components like background services or broadcast receivers. The resulting distribution is sparse and multimodal (including counts, length, binary values) making it a strong candidate for generation methods used for generating tabular style data (discussed in section 2.2).

To find the best architecture for this task, we evaluate three model architectures, GANs, VAEs, and diffusion models, each offering their own strengths in capturing the high-dimensional distributions of static profiles. In this stage, the static generators are trained on the full preprocessed static profiles, each of which includes both static features and dynamic summary information as described in Section 3.1.3. All architecture details and hyperparameters for each of the models in this subsection can be found in Appendix A.

3.2.3.1 Generative Adversarial Networks (GANs)

GANs approach generation through the lens of a minimax problem between a generator and a discriminator can be represented as (Goodfellow et al., 2020):

$$\min_G \max_D (E_{s\sim p_{data}}[D(s)] - E_{z\sim p(z)}[D(G(z))]) \tag{6}$$

We test the WGAN-GP variant since it is known that this variant helps stabilize training and prevent mode collapse. In general, GANs are known for being able to learn highly complex, high-dimensional data distributions and can generate sharp, realistic samples, while also avoiding mode averaging which typically occurs in models trained with pointwise losses like mean squared error (Goodfellow et al., 2020).

3.2.3.2 Variation Autoencoders (VAEs)

VAEs assume a latent structure where the static profile s is generated from a latent vector z (Kingma & Welling, 2022):

$$z \sim \mathcal{N}(0, \mathrm{I}), \qquad s \sim p_\theta(s \mid z) \tag{7}$$

Training the model maximizes the Evidence Lower Bound (ELBO) (Kingma & Welling, 2022):

$$\log p(s) \geq \mathbb{E}_{q_{\phi(z|s)}}[\log p_\theta(s \mid z)] - KL\big(q_\phi(z \mid s) || \, p(z)\big) \tag{8}$$

VAEs are known to provide smooth latent spaces that allow for sampling and interpolation. Their inductive bias results in a broad coverage of the data distribution, so it is hypothesized that static profiles generated with VAEs may not be as precise in reconstruction as GANs but may be able to generate a wider range of static profile types.

3.2.3.3 Diffusion Models

Diffusion models constitute a more recent class of generative models that work by iteratively denoising corrupted samples. The model learns reverse-time transitions $p_\theta(s_{t-1} \mid s_t)$ that ultimately recover samples from $p(s)$ using a forward diffusion process (Ho et al., 2020):

$$s_t = \alpha_t s_0 + (1 - \alpha_t)\epsilon, \qquad \epsilon \sim \mathcal{N}(0, \mathrm{I}) \tag{9}$$

In prior works (see section 2.1), diffusion models have demonstrated state-of-the-art performance in generating tabular and structured data because of their ability to capture complex distributions. Additionally, diffusion models have been shown to be more stable in training compared to GANs and less likely to result in mode collapse.

3.2.3.4 Post-processing of Synthetic Samples

The output of our static profile generators are continuous-valued vectors in a normalized feature space. To ensure that synthetic samples are comparable to real data, we apply a post-processing procedure before we evaluate the synthetic samples from all static generators (GAN, VAE, and Diffusion model).

First, all values in the synthetic sample vector are clamped to the valid normalized range of [0,1], to prevent out-of-range artifacts which occasionally occur from generator outputs. Next, we apply marginal-matching for all features that are binary. Since binary features are either 0 or 1, they cannot be compared to real data when they have a decimal value in between that range. Therefore, each binary feature value is converted to either 0 or 1, such that the activation rate in the synthetic data matches the activation rate that occurs in real samples, using feature-specific thresholds. Continuous features are left unmodified since they can be directly compared already to the real vectors.

3.2.4 Conditional Dynamic Event Sequence Generation

Having generated a synthetic static profile s, the second stage of the pipeline models the conditional distribution of dynamic events. Formally, this is represented as learning a model of the form:

$$p(y_{1:t} \mid s_{dyn}), \tag{10}$$

where $y_{1:t}$ is a variable-length sequence of dynamic events and $s_{dyn} \subset s$ is the dynamic summary information from the static profile including sequence length and event frequencies. As mentioned at the beginning section 3.2, conditioning on the subset s_{dyn} is both computationally efficient and has enough information for dynamic sequence generation. With an effective static generator, the generator should already be able to effectively predict the dynamic events that will appear corresponding to specific static profiles, so it remains the task of the conditional sequence generator to predict how a specific dynamic summary profile will look over time as a sequence of events.

In this thesis, we use a decoder-only autoregressive architecture for this sequence generation task. Although recent work (e.g. Bao et al., 2025) has shown the success of non-autoregressive embedding-based approaches in opcode sequence generation, the characteristics of Android dynamic event logs differ substantially from opcode streams and are more like logs used in works such as Kholgh & Kostakos, 2023, as event ordering is highly important. For instance, *open* precedes *read* and *connect* precedes *sendto* which preceds *recvfrom.* These event ordering patterns often occur over long ranges in sequence length and autoregressive models are better equipped to capture these natural orderings since they explicitly model:

$$p(y_{1:t} \mid s) = \prod_{t=1}^{T} p(y_t \mid y_{<t}, s) \tag{11}$$

allowing for a direct dependency between the current event and its preceding context. Conditioning of static profiles also benefits from models that allow for token-level generation, as it allows for the process to happen token-by-token, whereas with embedding-based models, the process would only happen at the global latent level. Moreover, decoder-only models allow for more fine-grained control of the sequences that are outputted by letting you adjust sequence length, so all of these factors combined lead us to choose decoder-only autoregressive models.

3.2.4.1 LSTM-based Conditional Decoders (Background)

LSTMs generate events sequentially by evolving a hidden state that summarizes past context (Hochreiter & Schmidhuber, 1997). In conditional sequence generation, static information can be incorporated into LSTMs by conditioning the recurrence on an embedding of the static profile. Formally, the hidden state is updated as following:

$$h_t = LSTM(x_t, h_{t-1}, e_s), \tag{12}$$

where e_s is the learned embedding of the static profile, x_t is the input token at time t, $h_{t-1}$is the previous hidden state.

While LSTM-based conditional decoders have been successfully applied in healthcare as seen with RCGAN and ClinicalGAN (as described in Section 2.3), they are infeasible to use in the case of malware logs. Specifically, malware dynamic logs often contain thousands of events and have long-range dependencies, leading to high memory usage, slow training, and worse dependency strength when using LSTMs (Tay et al., 2020). As a result, all evaluations in this thesis are conducted using a transformer-based decoder architecture.

3.2.4.2 Transformer-based Conditional Generators

Transformers model the conditional sequence distribution:

$$p(y_t \mid y_{<t}, s_{dyn}) \tag{13}$$

using multi-head self-attention over previously generated events, $y_{<t}$, allowing for long-range context of events. In this thesis, we implement a conditional sequence generator using a GPT-2 architecture. All architecture, hyperparameters, and decoding details for the GPT-2 model in this subsection can be found in Appendix B. We choose GPT-2 because it has a well-established autoregressive design, stable training behavior and is relatively small compared to more modern

transformer-based LLM architectures, with 1.5 billion parameters (Radford et al., 2019).

Conditioning is implemented using a prefix tuning process, where the dynamic summary information is linearly projected into a sequence of learned prefix embeddings (Li & Liang, 2021). These embeddings are then appended to the start of token embeddings of the dynamic event sequence, so they act as global conditioning tokens that are visible to the transformer at every layer through self-attention. This design allows the model to factor in information from s_{dyn} like sequence length and event frequency patterns while also capturing the semantic patterns of malware dynamic event sequences. Figure 3 below visualizes how prefix tuned embeddings enter the sequence generation model.

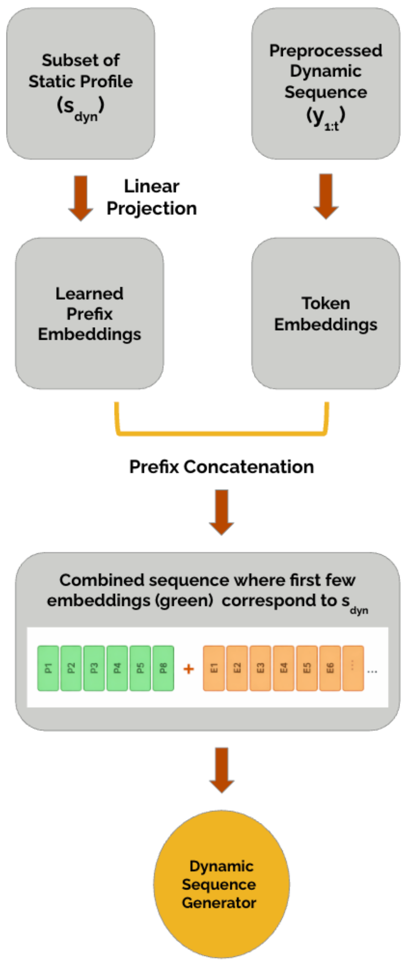

Subset of Static Profile (s_{dyn})
Preprocessed Dynamic Sequence ($y_{1:t}$)
Linear Projection
Learned Prefix Embeddings
Token Embeddings
Prefix Concatenation
Combined sequence where first few embeddings (green) correspond to s_{dyn}
P1 P2 P3 P4 P5 P6 + E1 E2 E3 E4 E5 E6 ...
Dynamic Sequence Generator

Figure 3. Prefix-based static conditioning for dynamic malware event sequence generation.

3.2.4.3 Sequence Generation Procedure

After the training of the conditional sequence generation model, dynamic event sequences are generated autoregressively by sampling the next event token from the previous tokens and conditioning information in an in iterative manner. Generation always begins with a designated start-of-sequence token which is the first shared system call present in all real malware dynamic logs.

To ensure that we have both coherent sequences and diversity in generation, we choose a hybrid decoding approach which combines greedy decoding and nucleus (top-p) sampling. Using greedy decoding in the first part of our sequence generation procedure allows the model to establish a sequence prefix that will be semantically correct and stable, and afterward, nucleus sampling will allow the model to have variability in events that occur, allowing for the generation of more novel sequence structures. To further improve diversity and discourage overly repetitive sequence patterns, we also add a repetition penalty which prevents the model from using recently generated tokens too often.

3.2.4.4 Unconditional Baseline Model

To understand the effects of conditioning, we train an unconditional sequence generation model to serve as a control. This baseline uses the same transformer architecture, vocabulary, training sequences, and decoding method as the conditional model, but without any conditional inputs. Formally, the unconditional model learns the following distribution:

$$p(y_{1:t}) = \prod_{t=1}^{T} p(y_t \mid y_{<t}) \qquad (14)$$

The purpose of the unconditional model is to set a lower bound on sequence quality when there is no static-dynamic conditioning modeled. By holding all other factors constant, one can credit any improvements in realism, diversity or behavioral validity to the performed conditioning instead of the transformer architecture itself. This baseline is, thus, a reference point for evaluating any improvements in metrics such as correct sequence logic, sequence diversity, and static-dynamic consistency. Any improvements over the unconditional model would then imply that incorporating dynamic summary information can meaningfully steer the generation process towards behaviors that are more consistent with real malware patterns.

3.3 Evaluation Metrics

Evaluating a system that outputs full malware logs consisting of both a static profile and dynamic event sequences requires evaluating both the individual stages of the pipeline and the combined synthetic logs. Accordingly, the evaluation framework used in this thesis contains: (i) static profile evaluation, which measures how well synthetic profiles preserve global structure, feature-wise distributions, inter-feature dependencies, and downstream utility, and (ii) dynamic event sequence evaluation, which assess how well generated sequences across multiple evaluation categories, measuring sequence realism, diversity, semantic validity, and consistency between static and dynamic components. This framework allows for a comprehensive assessment of the two-stage generation pipeline proposed in this thesis. Figure 4 summarizes the evaluation framework described in this section.

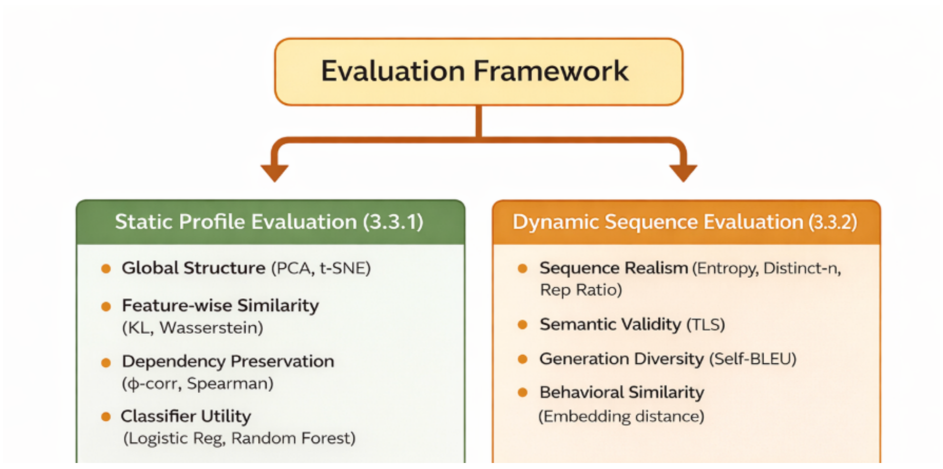

Evaluation Framework
Static Profile Evaluation (3.3.1)
Global Structure (PCA, t-SNE)
Feature-wise Similarity (KL, Wasserstein)
Dependency Preservation (ϕ-corr, Spearman)
Classifier Utility (Logistic Reg, Random Forest)
Dynamic Sequence Evaluation (3.3.2)
Sequence Realism (Entropy, Distinct-n, Rep Ratio)
Semantic Validity (TLS)
Generation Diversity (Self-BLEU)
Behavioral Similarity (Embedding distance)

Figure 4. Evaluation framework for generated static profile and dynamic sequence evaluation.

3.3.1 Static Profile Evaluation

This subsection describes the metrics used to determine how synthetic static profiles accurately replicate the underlying distribution of real malware and add value. We evaluate global structure visualization, feature-wise distributional similarity, feature dependency preservation, and downstream classifier utility. For consistency, all evaluations in this subsection are performed in the normalized static feature space that is used during model training.

3.3.1.1 Global Structure Visualization

Before looking at quantitative metrics, we first examine whether real or synthetic static profiles are visually similar in a lower-dimensional embedding space (i.e. have similar geometric structure). To do so, we use two visualization techniques:

1. Principal Component Analysis (Gewers et al., 2021): This technique performs a linear projection onto directions of maximal variance. An overlap between real and synthetic data points in PCA space means that the global variance structure is preserved.

2. T-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten & Hinton, 2008): This technique shows local neighborhood relationships across embeddings. An overlap between real and synthetic data points in t-SNE means that the synthetic distribution can capture the local co-occurrence patterns that occur in real malware samples.

These visualizations give high-level insights into the general performance of the generators that we have trained, allowing us to easily tell if there was possibly mode collapse during training.

3.3.1.2 Feature-Wise Distribution Similarity

The next set of metrics we use in our evaluation measure the similarity of individual feature distributions by computing the divergence metrics between the real distribution P_j and synthetic distribution Q_j for each static feature j. These metrics are important to evaluate since each feature in our overall distribution has its own distribution that the generator should retain. Prior works like CTGAN, TabDDPM, and TVAE, all use these types of metrics in their evaluation as well:

1. KL Divergence (both directions) is defined as the following:

$$KL\left(P_j \parallel Q_j\right) = \sum_x P_j(x) \log \frac{P_j(x)}{Q_j(x)}, \qquad KL\left(Q_j \parallel P_j\right), \tag{15}$$

   where x is all possible values for feature j. KL divergence is a standard measure for comparing probability distributions and is widely used in the evaluation of synthetic tabular data generators (Choi et al., 2025). $KL\left(P_j \parallel Q_j\right)$ measures if the synthetic distribution fails to have probability mass where the real data does (i.e. if the generator does not cover everything the real data does) and $KL\left(Q_j \parallel P_j\right)$ measures if the generator produces unrealistic values that are not present in the real distribution.

2. Wasserstein-1 Distance is defined as the following:

$$W_1\left(P_j, Q_j\right) = \inf_{\gamma \in \Pi(P_j, Q_j)} \mathbb{E}_{(x,y)\sim\gamma}\left[|x - y|\right], \tag{16}$$

   where x is a value drawn from the real feature distribution P_j and y is a value drawn from the synthetic feature distribution Q_j. The Wasserstein-1 distance provides a useful notion of distributional similarity for continuous features and has been adopted as an evaluation metric in synthetic tabular data works such as TabDDPM (Arjovsky et al., 2017; Kotelnikov et al., 2023).

These distribution metrics are computed independently for each feature. In our results, we report an aggregate score by averaging the per-feature divergence across all features.

3.3.1.3 Feature Dependency Preservation

Static malware features often have strong dependency patterns across features such as permissions and services. It is important to preserve these dependencies in generation; otherwise, the synthetic samples lose meaning. Instead of computing full correlation matrices over all feature pairs, we use a pairwise dependency evaluation since full correlation matrices would be dominated by statistically uninformative feature combinations (Serra et al., 2018). This approach performs evaluation on a large random subset of feature pairs.

For binary feature pairs, the phi correlation coefficient, which measures the association between two Bernoulli variables, is computed (Pearson, 1900). Given two binary features x_i, x_j, the phi coefficient is defined as:

$$\phi(x_i, x_j) = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{(n_{11} + n_{10})(n_{01} + n_{00})(n_{11} + n_{01})(n_{10} + n_{00})}} \tag{17}$$

where n_{ab} represents the number of samples with $x_i = a$ and $x_j = b$.

For continuous feature pairs, the Spearman rank correlation, which captures monotonic dependencies, is computed (Spearman, 1904). Mixed binary-continuous feature pairs are not evaluated since these dependencies are often ambiguous and not commonly evaluated in prior work. Prior work in synthetic tabular data generation commonly assesses structural fidelity by comparing pairwise feature correlations between real and synthetic data as we do below (Kotelnikov et al., 2023; Park et al., 2018; Xu et al., 2019). For each generator, feature pairs are then randomly sampled and then we compute the mean absolute discrepancy between real and synthetic correlations for both continuous and binary feature pairs:

$$\Delta_{dep} = \mathbb{E}_{i,j}[|\rho_{i,j}^{real} - \rho_{i,j}^{syn}|], \tag{18}$$

where ρ is either the phi coefficient for binary feature pairs or Spearman correlation for continuous feature pairs. A lower correlation discrepancy means a better preservation of feature dependencies.

3.3.1.4 Classifier-Based Utility

Lastly, to assess the value of synthetic static profiles, we evaluate their utility in the downstream task of malware versus benign classification, an evaluation technique performed in several past data generation works (Kamthe et al., 2021; Kotelnikov et al., 2023; Park et al., 2018; Xu et al., 2019). Performance is measured by classifier accuracy. We follow a procedure that ensures no data leakage, ensuring that performance differences in classifiers are due to the addition of synthetic data. To prevent data leakage, for this specific evaluation, we train new versions of our generators on a train set containing 80% of the malware samples, leaving a 20% test set of unseen samples to be used for classifier evaluation. An equal number of benign samples are also added to this test set since the task is malware-benign classification.

For robustness and accurately measure the effect of adding synthetic data, we test both logistic regression (Hastie et al., 2009) and random forest classifiers (Breiman, 2001), and train classifiers on the following dataset variants:

1. Real-only: Only real samples are included.
2. Replace-half: Half of the real malware samples are replaced with synthetic malware samples.
3. Add-synthetic: An equal number of synthetic malware samples are added to the real training set.

3.3.2 Conditional Dynamic Event Sequence Evaluation

Dynamic event sequences represent the operating system-level behavior of malware during its execution in an ordered manner. In static profile evaluation, we look mostly at distributional similarities between real and synthetic samples. But in dynamic sequences, we must also assess factors such as sequence realism and semantic constraints that must be upheld when generating diverse, meaningful dynamic event sequences. Accordingly, we measure sequence realism, semantic validity, and generation diversity.

Moreover, to directly measure the semantic similarity between sequences, which is necessary when comparing whether conditioning is successful, we learn a behavioral embedding representation. This embedding-based comparison serves as the main quantitative measure of sequence level similarity which is discussed further in section 3.3.2.2.

We do not measure the downstream utility of generated dynamic event sequences because unlike static malware profiles, there are no standardized or widely adopted frameworks for detecting malware on long, malware event traces. Developing and validating such a detection pipeline would involve additional research and experimentation and is outside the scope of this thesis.

3.3.2.1 Evaluation Groups and Pairing Strategies

To rigorously evaluate the effect of conditioning on dynamic sequence generation, we define distribution-level evaluation groups and sequence-sequence pairing strategies to allow for controls and baselines in our experiments. Distribution-level metrics such as entropy, Distinct-n, repetition ratio, self-BLEU and temporal logic score (TLS), are used to assess the set-level properties of generated sequences. The primary evaluation groups used in this thesis are listed below:

1. Real sequences (full set): The complete set of real malware event sequences available in our balanced dataset (n=5004). This set serves as a reference distribution for all distribution-level metrics.

2. Real sequences (n=100): A random subset of 100 real sequences, included to compare for sample-size effects since generated sequence sets are of size 100, allowing for a fair comparison.

3. Unconditional synthetic sequences: Sequences that are generated by a model that has no conditioning. This set serves as a baseline for evaluating the effects of conditioning on sequence generation.

4. Real static-conditioned synthetic sequences: Sequences generated while conditioning on real static profiles. This group evaluates whether conditioning on static information has meaningful effects on the behavior of generated sequences.

5. Synthetic static-conditioned synthetic sequences: Sequences generated while conditioning on synthetic static profiles. This group evaluates whether static information generated at the first stage of the pipeline remains useful in downstream conditional sequence generation. All synthetic static profiles are generated from the WGAN-GP model. This choice is motivated by Section 4.1, where we see WGAN-GP achieved the closest alignment to real data both in feature-wise distribution similarity and visual structure similarity.

In embedding-based behavioral similarity, comparisons happen at a sequence-to-sequence level so instead of evaluation groups (detailed further in section 3.3.2.3), pairing strategies are defined to keep experiments controlled. The pairing strategies used in this thesis are listed below:

1. Real static-conditioned synthetic sequences paired with real sequences sharing the same static profile
2. Synthetic static-conditioned synthetic sequences paired with real sequences corresponding to their nearest-neighbor static matches
3. For both (1) and (2), we also compare the synthetic sequences to randomly sampled real sequences to ensure observed similarity is due to conditioning and not general similarity amongst malware sequences.

3.3.2.2 Sequence Realism

We evaluate sequence realism by measuring whether generated sequences have similar token usage patterns, token variability and repetition behaviors as real malware sequences. We outline below each of the metrics used to measure these qualities.

Token Distribution Entropy:

Let p(y_v) represent the empirical probability of token y_v in generated sequences. The entropy is defined as:

$$H(y_{1:t}) = -\sum_{y_v} p(y_v) \log p(y_v) \tag{19}$$

Entropy essentially measures, at a unigram-level, the global token-level diversity of our malware sequences (Tae et al., 2025). Comparing the entropies between synthetic and real samples allows us to identify whether we have under-generation (mode collapse) or over-generation (overly random behavior). Ideally, the entropies between the two sets of samples should be close to one another.

Distinct-n Diversity:

For n ∈ {2,3}, the distinct-n diversity score is:

$$\text{Distinct-n} = \frac{\#\{unique\ n\text{−}grams\}}{\#\{total\ n\text{−}grams\}} \tag{20}$$

Although both entropy and Distinct-n measure diversity, they assess different properties. Entropy looks at token variability at a global level whereas Distinct-n measures the structural sequence diversity via unique n-grams (Tae et al., 2025; Zhu et al., 2018). It is possible for a model to exhibit high entropy yet generate repetitive sequences, or vice versa, which is why we use both metrics. This metric also helps us identify mode collapse or excess repetitiveness at a sequence level. A high Distinct-n score signifies that the sequence has high token diversity and low repetition.

Repetition Ratio:

To further evaluate the realism of generated sequences, we compute repetition ratio, which measures the rate of local repetition within sequences. In autoregressive models, often models can degenerate and repeat the same token too many times. In the context of real malware dynamic event sequences, a token being repeated multiple times in a row is common, so a sequence generation model must be able to maintain a similar level of repetition without collapsing into constant repetition. Formally, for a sequence $y_{1:t}$, we measure the repetition ratio as the fraction of adjacent token pairs that are identical to each other, a lightweight variant of repetition metrics used in prior sequence generation works (Zhao et al., 2022):

$$Rep(y_{1:t}) = \frac{1}{t}\sum_{i=2}^{t} 1[y_i = y_{i-1}] \tag{21}$$

3.3.2.3 Semantic Validity

Dynamic event sequences must obey semantic constraints that come from real operating system behavior. System calls typically have multi-sequence patterns to them (e.g., files are opened before being read or written and then eventually closed). To evaluate whether generated sequences follow these constraints and exhibit realism, we compute a temporal logic score (TLS). This metric is original and inspired by prior uses of temporal-logic satisfaction or coverage rates for behavioral validation (Lin et al., 2025). Let R represent a set of temporal-logic rules for system calls (ex. open → read → close). We measure a normalized rule adherence, which represents how frequently valid patterns are completed when a corresponding operation begins (such as open in the pattern open → read → close).

For a generated sequence $\tilde{y}$, we define TLS as:

$$TLS(\tilde{y}) = \frac{1}{|R|}\sum_{r\in R} score_r(\tilde{y}), \tag{22}$$

where $score_r(\tilde{y})$ represents the fraction of valid completions for a rule r relative to the number of opportunities that rule has an opportunity to appear in a sequence. TLS values similar to those in real sequences indicate that a generated sequence is semantically valid.

3.3.2.4 Generation Diversity

To measure the diversity of sequences across the entire set of generated sequences, we compute self-BLEU scores (Zhu et al., 2018). This metric is important because it quantifies the degree that generated sequences are similar to each other, independent of their similarity to real malware sequences. It is a metric commonly used in prior works (Perez et al., 2022). Diversity is critical in the context of synthetic log generation because there is less value in synthetic logs which are extremely similar to each other. A high self-BLEU score means all generated sequences are highly similar to one another on a structural level. It is calculated as follows (Zhu et al., 2018):

$$\text{Self-BLEU} = \frac{1}{M}\sum_{i=1}^{M} BLEU(y_i, Y\setminus i), \tag{23}$$

Where M is the number of generated sequences, y_i is the i-th generated sequence, and $Y\setminus i$ is the set of all generated sequences except y_i, and BLEU is the trigram-level BLEU score of two sequences.

3.3.2.5 Behavioral Embedding Similarity

Directly comparing two sequences, beyond token-level metrics and semantic rule checks, is necessary when evaluating whether conditioning on generation is successful. Thus, inspired by Bao et al., we learn a behavioral embedding space that allows us to make these comparisons at a whole sequence level (Bao et al., 2025). To convert sequences into embeddings in a shared space, we train a lightweight sequence behavior encoder. The encoder maps variable-length event sequences into a shared embedding space using chunked encoding and pooling. The architecture and hyperparameters of this encoder can be found in appendix C of the thesis. For each evaluation pairing, embeddings are computed for both real and generated sequences using this encoder. Once our embeddings are computed, we need to measure behavioral similarity at the individual sequence level. This is done by measuring cosine distances between embeddings of paired real and generated sequences that share similar or the same static profiles.

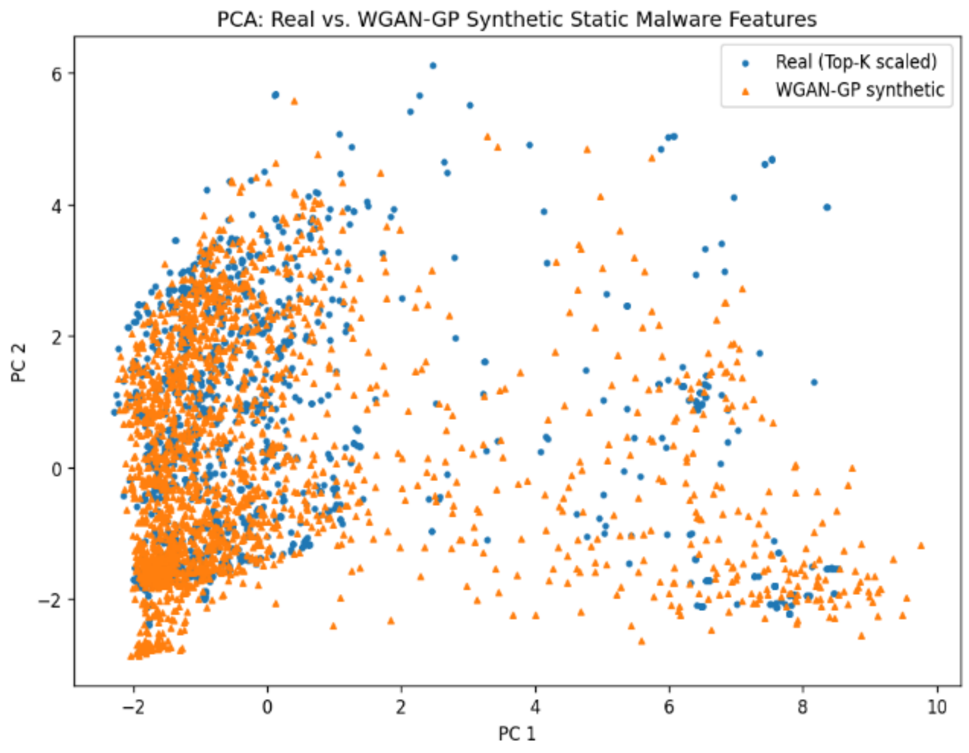
Chapter 4: Results

This chapter presents the empirical results of the proposed two-stage synthetic malware log generation framework. We first present results on static malware profile generation using GANs, VAEs, and diffusion models for each of the evaluation metrics discussed in Section 3.3.1. We then report the results for conditional dynamic sequence generation using a GPT-2 model for each of the evaluation metrics discussed in Section 3.3.2. All discussion and analysis of these results is found in Chapter 5.

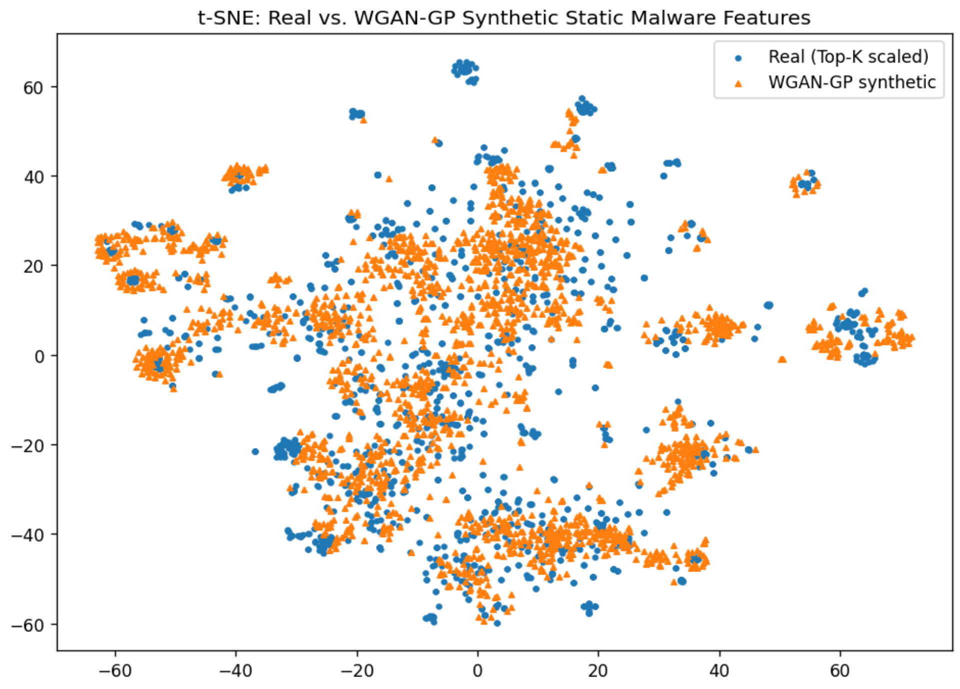
4.1 Static Profile Generation Results

This section presents both quantitative and qualitative results using the evaluation metrics discussed in section 3.3.1 across three model architectures: WGAN-GP, VAE, and diffusion models. We evaluate synthetic static profiles on global structure visualization, feature-wise distribution similarity, feature dependency preservation, and downstream classifier utility.

Our evaluation begins with a global structure visualization of synthetic and real data. Figures 2-4 show PCA and t-SNE projections that compare real and synthetic static profiles for each model. We see visible overlap between real and synthetic data points across all models, suggesting each generator can model aspects of the real static data. Qualitatively, WGAN-GP synthetic samples overlap most closely with the set of real samples while diffusion synthetic samples overlap much less with the set of real samples.

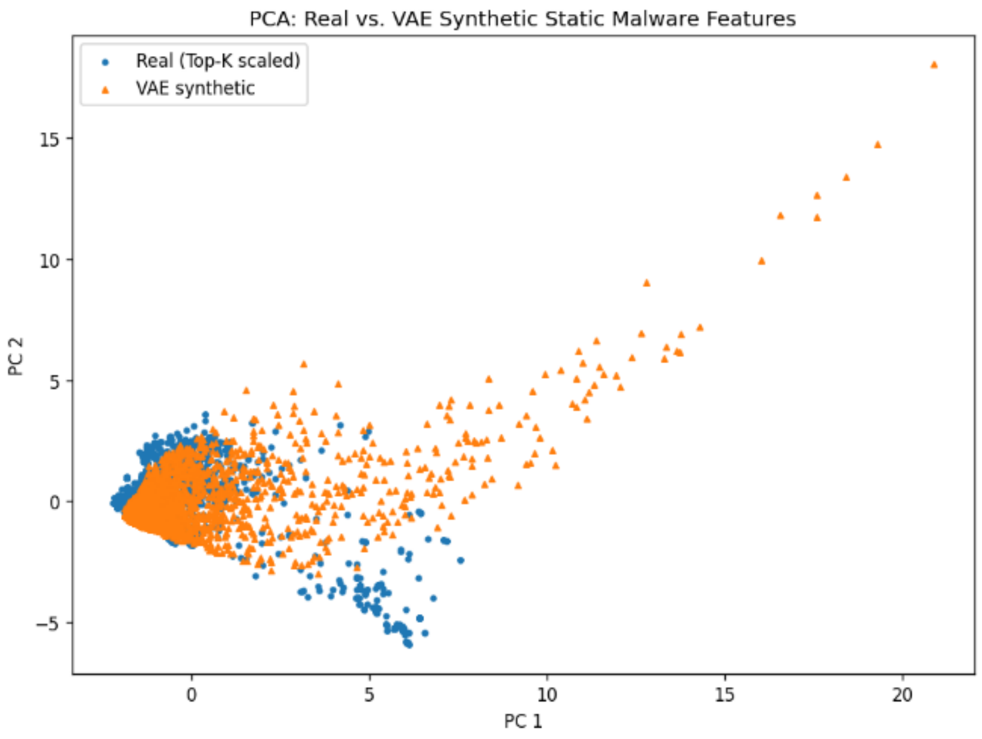


(a)

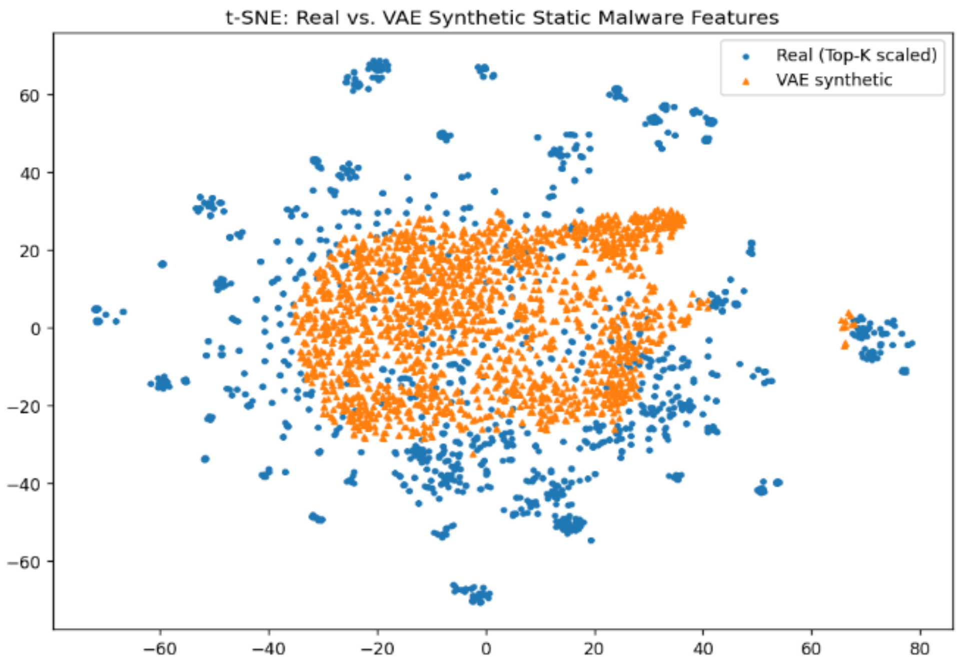


(b)

Figure 5. Comparison of real and synthetic static profiles generated using WGAN-GP.
(a) PCA projection showing alignment of global variance structure.
(b) t-SNE project showing local neighborhood overlaps and coverage of data.

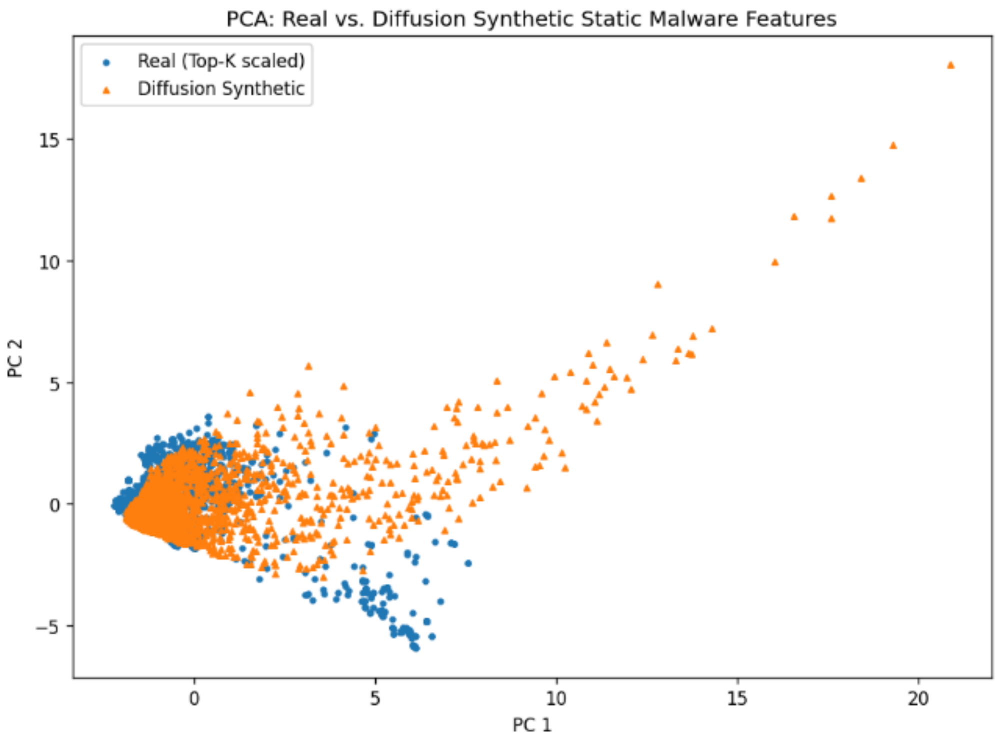


(a)

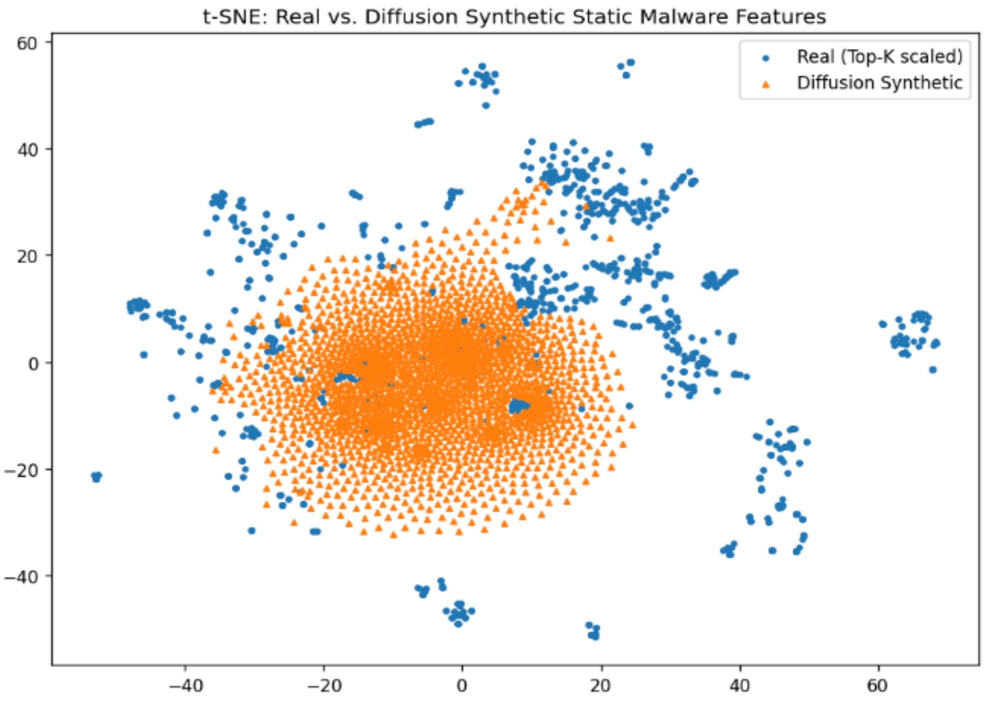


(b)

Figure 6. Comparison of real and synthetic static profiles generated using VAE.
(a) PCA projection showing alignment of global variance structure.
(b) t-SNE project showing local neighborhood overlaps and coverage of data.



(a)



(b)

Figure 7. Comparison of real and synthetic static profiles generated using diffusion model.
(a) PCA projection showing alignment of global variance structure.
(b) t-SNE project showing local neighborhood overlaps and coverage of data.

Following the analysis of visual global structure, we present feature-level distributional alignment results using quantitative divergence metrics (KL and Wasserstein-1). Feature-wise distribution similarity metrics are reported in Table 4. WGAN-GP achieves the lowest KL divergence in both P to Q and Q to P directions in addition to the smallest Wasserstein-1 distance, indicating synthetic data has closer alignment to real feature distributions. The diffusion model exhibits the highest divergence value in the Q to P direction whereas the VAE model exhibits the highest diverge value in the P to Q distance. The KL divergence values in our findings fall within the same order of magnitude as those reported for TVAE, CTGAN, and TabDDPM in a prior work, where KL divergence values ranged between 0.2389 to 1.6833 depending on model class and experiment (Choi et al., 2025). Prior work on tabular generative modeling found that diffusion-based models achieved the lowest Wasserstein distances (Kotelnikov et al., 2023), but we observe a different ordering where WGAN-GP and VAE achieve significantly lower Wasserstein-1 distances in comparison to the diffusion model.

Table 4. Averaged feature-wise distribution similarities across trained models.

Model	**KL(P||Q)**	**KL(Q||P)**	**Wasserstein-1**
WGAN-GP	0.0189	0.1310	0.0030
VAE	0.2267	0.1665	0.0034
Diffusion	0.1750	2.4199	0.0334

In addition to distributional alignment, we evaluate how well each model preserves the correlation between different features. Dependency structure preservation metrics are reported in Table 5. The diffusion model achieves the lowest correlation discrepancies in both phi-correlation and Spearman whereas the VAE model exhibits the highest correlation discrepancies in both phi-correlation and Spearman. While prior works also evaluate dependency preservation via correlation-based measures, our study uses more fine-grained metrics tailored to feature type, so a direct numerical comparison cannot be made to results such as L2 distances between correlation matrices found in TabDDPM's analysis (e.g., Kotelnikov et al (2023).

Table 5. Dependency structure preservation across trained models.

Model	**φ-corr. discrepancy**	**Spearman discrepancy**
WGAN-GP	0.1800	0.0531
VAE	0.2803	0.9375
Diffusion	0.0114	0.0314

Our last evaluation metric for generated static profiles is through downstream evaluation of classifiers. Tables 6 and 7 show downstream malware-versus-benign classifier accuracy scores between classifiers trained on just real data, half real data and half synthetic data, and full real data plus added synthetic data. In the logistic regression classifier, across all models we see slight performance decreases when adding synthetic samples. In the random forest classifiers, we see that replacing half the real data results in slight performance decreases whereas adding synthetic samples results in slight performance increases. The overall performance results match the slight performance gains or losses seen in a prior work as well that measures downstream classifier performance of TVAE, CTGAN, and TabDDPM (Choi et al., 2025).

Table 6. Logistic regression malware classifier accuracies across model and training styles.

Training Style	**WGAN-GP**	**VAE**	**Diffusion**
Real-only	0.9905	0.9905	0.9905
Replace-half	0.9895	0.9895	0.9870
Add-synthetic	0.9900	0.9870	0.9895

Table 7. Random forest malware classifier accuracies across model and training styles.

Training Style	**WGAN-GP**	**VAE**	**Diffusion**
Real-only	0.9860	0.9860	0.9860
Replace-half	0.9805	0.9790	0.9815
Add-synthetic	0.9870	0.9865	0.9880

4.2 Conditional Dynamic Sequence Generation Results

This section presents results for conditional dynamic sequence generation using a GPT-2-based architecture. We evaluate synthetic sequences using metrics discussed in Section 3.3.2: distribution-level realism and diversity, temporal logic satisfaction, and embedding-space similarity. Unlike Section 4.1, where trend-level comparisons to prior tabular generation work is possible, the evaluation results in this section are specific to malware system-call sequence generation and cannot be directly compared to metrics reported in prior work, since no existing studies evaluate generative models on the same data modality.

Table 8 displays the results for set-level realism and diversity metrics (entropy, repetition ratio, Distinct-n, Self-BLEU) on each type of sequence set (real, unconditioned synthetic, real static conditioned synthetic, and synthetic static conditioned synthetic). Due to computational constraints, each synthetic set contains N=100 samples. To ensure fair comparison and mitigate sample size effects, for real data, we report metrics for both the full real set and on a random subset of 100 real sequences. This allows differences between real and synthetic distributions to be interpreted independent of their dataset size.

Table 8. Distribution-level metrics across evaluation groups.

Evaluation Group	**Entropy**	**Distinct-2**	**Distinct-3**	**Repetition Ratio**	**Self-BLEU**
Real (full set)	3.6925	0.0128	0.0246	0.4006	0.7379
Real (n=100)	3.5815	0.0258	0.0559	0.4072	0.7357
Unconditioned synthetic	3.5272	0.0257	0.0539	0.3235	0.8413
Real static conditioned synthetic	3.6413	0.0284	0.0614	0.3553	0.7562
Synthetic static conditioned synthetic	3.7084	0.0345	0.0698	0.3103	0.7541

We next present our results on temporal logic satisfaction, which measures the semantic validity of generated sequences. Table 9 showcases the temporal logic scores (TLS) across evaluation groups for different sets of system-call rules. These results evaluate whether generated sequences can follow the semantic constraints present in real operating-system level event behavior.

Table 9. Distribution of TLS for various rules across evaluation groups.

Evaluation Group	**Open→ Read→ Close**	**Open→ Write→ Close**	**Open→ Chmod→ Close**	**Fork→ Execve→ Exit**	**Connect→ Sendto→ Recvfrom**
Real (full set)	0.3263	0.2414	0.0452	0.2544	0.3761
Real (n=100)	0.2969	0.2772	0.0500	0.2667	0.3317
Unconditioned synthetic	0.3676	0.2897	0.0499	0.0000	0.05
Real static conditioned synthetic	0.3090	0.2932	0.0589	0.0000	0.5821
Synthetic static conditioned synthetic	0.3106	0.2648	0.0599	0.0000	0.5034

Our final evaluation tests the effects of conditioning on sequence generation by measuring distance between real and synthetic sequences. Table 10 displays the aggregate embedding-space (from our learned encoder) cosine distances between evaluation pairs. All pairs are evaluated at sample sizes of n=100 due to computation constraints in generating synthetic sequences. Lower cosine distances indicate that there is greater behavioral similarity between sequences of that pair type.

Table 10. Embedding space cosine distance across evaluation pairs.

Evaluation Pair	**Mean**	**Median**	**p90**	**P95**
Real ↔ real (random pairs)	0.3894	0.3751	0.6905	0.7652
Unconditioned synthetic ↔ random real	0.3512	0.3552	0.4824	0.5233
Real static-conditioned synthetic ↔ matched real	0.2707	0.2546	0.4420	0.4840
Real static-conditioned synthetic ↔ random real	0.3309	0.2996	0.5418	0.5951
Synthetic static-conditioned synthetic ↔ matched real	0.2595	0.2353	0.4367	0.5045
Synthetic static-conditioned synthetic ↔ random real	0.3502	0.3377	0.4999	0.6421

Chapter 5: Discussion

This chapter first interprets the results presented in Chapter 4 for both static malware profile generation and dynamic malware event sequence generation in the context of the thesis' broader goals and objectives. It then explains the significance of the results in relation to the proposed two-stage generative pipeline. The chapter concludes by discussing limitations of the thesis and suggests specific directions for future research in this field.

5.1 Static Profile Generation: Interpretation

The results in section 4.1 reveal that there is no single best model architecture for static profile generation and each model has its own unique tradeoffs. For instance, in terms of global structure and feature-wise distribution similarity, WGAN-GP performs the best, while for feature dependency preservation and downstream classifier utility, the diffusion model performs the best. We discuss this more in depth in the following subsections.

Even though all three evaluated models were able to learn the high-dimensional structure of static malware profiles, the synthetic samples that they produce possess different strengths depending on the evaluation metric. These results suggest that the choice of a static generator depends on the intended downstream use of the synthetic data. For instance, some applications may emphasize samples that are similar to known malware distributions, while others may prioritize the exploration of novel malware structures that still maintain real malware feature dependencies.

5.1.1 Global Structure Visualization

From the PCA and t-SNE visualizations in Figures 2-4, we see that the WGAN-GP possesses the strongest overlap visually with real static profiles, where for most clusters of real data, there also exist clusters of synthetic data. This behavior is consistent with prior works that suggest WGAN-GP models can capture both global variance and local neighborhood relationships (Arjovsky et al., 2017). The diffusion-based model, on the other hand, shows weaker overlap in both PCA and t-SNE projection. It is worth noting though that there is not mode collapse in the static profiles produced by diffusion models as synthetic samples still occupy a wide region of the embedding space. Overall, diffusion model-generated synthetic samples could be used to predict more novel malware that is different from existing known ones. The VAE model lies in between the visualizations of WGAN-GP and diffusion models where synthetic samples have overlap with the largest cluster of real samples but fail to occupy smaller clusters of real samples.

5.1.2 Feature-wise Distribution Similarity

Table 4 shows that WGAN-GP generated samples most closely matching marginal feature distributions, since they achieve the lowest KL divergences (in both directions) in addition to Wasserstein-1 distance. The VAE follows closely to the results of WGAN-GP, meaning samples generated by this model are also capable of matching marginal feature distribution. Moreover, for KL divergence in the P to Q direction and Wasserstein-1 distance, the diffusion model is close to the values of the first two models. However, the diffusion model possesses high KL divergence in the Q to P direction, indicating that the model generates feature values that are rarely present in real samples. This result is consistent with the PCA and t-SNE plots, where the diffusion model had many samples that occupied regions of the embedding space that real samples did not.

5.1.3 Feature Dependency Preservation

Despite performing the worst in feature-wise distributional similarity, the diffusion model outperforms both WGAN-GP and VAE models in preserving feature dependencies for both binary and continuous features, as seen in Table 5. This means that the diffusion model can effectively retain inter-feature relationships even when their marginal feature distributions differ significantly. Interestingly, the results from the diffusion model also show that no single static generator performs the best across every metric. Next, similar to diffusion models, WGAN-GP models also perform strongly in preserving continuous feature dependencies (Spearman correlation discrepancy) but exhibit moderate dependency discrepancy for binary features (φ-correlation discrepancy). VAEs perform the worst in preserving feature dependencies with moderate levels of discrepancy for binary features and high discrepancy for continuous features.

5.1.4 Classifier-based Utility

In the downstream malware-benign classifier experiments, for all three model architectures, we see that replacing half of the real samples with synthetic samples does not significantly hurt classifier accuracy, implying that all three models are able to produce useful synthetic representations for this downstream classification task. Moreover, for random forest classifiers, adding synthetic malware samples (add-synthetic scenario) improves classification accuracy across all model architectures, with the diffusion model seeing the largest gain (+0.4 percentage points) in classifier accuracy. This result could be attributed to the fact diffusion model generated synthetic samples can strongly preserve inter-feature dependencies, which makes them useful at enriching decision boundaries for nonlinear classifiers like random forest.

Logistic regression classifiers, in the add-synthetic scenario, worsens performance slightly (accuracy for all three models is reduced by less than 1 percentage point). One possible cause for this inferior performance could be that the logistic regression classifier trained on just real samples already performs extremely well with 99% accuracy so adding synthetic samples does not immediately help. Our results, overall, for this evaluation metric imply that synthetic static samples are most beneficial when trained on classifiers that can understand non-linear patterns.

5.2 Dynamic Event Sequence Generation: Interpretation

The results from section 4.2 overall indicate that using static information to condition dynamic event sequence generation provides meaningful improvements to both realism and behavioral consistency compared to the unconditioned baseline. Across most evaluated metrics (distribution-level realism, semantic validity, and embedding-based similarity), static-conditioned generators outperform unconditioned generation. Importantly, these improvements persist between both real and synthetic static profiles, revealing that the two-stage pipeline is a robust end-to-end approach for generating synthetic malware logs that contain consistent static and dynamic portions.

5.2.1 Distribution-level Realism and Diversity

Table 8 reveals that static-conditioned synthetic sequences more closely match real malware behavior across all distribution-level metrics that we measure. Entropy and distinct-n scores are closer to those observed in real sequences, meaning that the conditioning helps the sequence generator capture realistic token patterns while maintaining structural variability. As seen from the lower entropy and higher self-BLEU scores of the unconditioned sequences, without conditioning, there is a tendency for the generator to produce sequences that are both extremely similar to each other and more repetitive in global token usage.

Looking at repetition ratio, we see again that unconditioned synthetic sequences exhibit lower ratios than real sequences, meaning that the pattern of legitimate repetition (e.g. repeated reads/writes) that occurs in real malware sequences is not completely captured. Conditioning on real static profiles corrects this issue more, bringing the repetition ratio closer to its value in real sequences.

However, sequences conditioned on synthetic static profiles possess a repetition ratio that is farthest from the real sequences (still it is within 0.10 and not drastically far). This likely means that while synthetic static profiles can push sequence generation to match distributional patterns in real malware sequences, exact token to token patterns are more difficult to match. These results imply that future work is needed to improve synthetic static profile generation, even though it is already quite close to real sequences. With that said, across every other metric, even sequences generated with synthetic static profiles outperform unconditioned sequences.

5.2.2 Semantic Validity

Table 9 evaluates semantic validity using temporal logic scores (TLS), explained in section 3.3.2.3. These scores measure whether sequences complete common multi-event patterns such as file (open → read → write) or network (connect → sendto → recvfrom) lifecycles. Across almost all evaluated rules, we see that static-conditioned sequences exhibit TLS values that are either closer to those observed in real malware or comparable to the unconditioned baseline when that baseline already closely matches real behavior. There are certain rules though that are not matched correctly in TLS between real and synthetic sequences.

One issue we see with the connect → sendto → recvfrom rule is that after conditioning, the TLS is moderately higher (about 0.10 to 0.20 in difference) than in real sequences. A worse issue is we see that the fork → execve → exit pattern fails to appear in any generated sequence conditioned or not. Both issues likely arise because the tokens in these specific rules occur much less often in the real data, meaning the sequence generation model does not have sufficient data to learn the pattern well. Given that the TLS of other rules in synthetic sequences are matched well to the TLS in real sequences, the overall assessment is that the TLS results also show that static conditioning meaningfully improves semantic validity.

5.2.3 Behavioral Embedding Similarity

Unlike distributional and semantic metrics which aggregate the properties of generated sequences, embedding-based similarity allows us to directly compare how close synthetic sequences are to real sequences with either the same or similar static profiles. This provides the strongest evidence that conditioning on static information is an effective strategy for linking static and dynamic portions of malware logs.

Table 10 reveals that sequences generated with static conditioning achieve substantially lower mean and tail cosine distances to their corresponding real sequences compared to all other control / baseline evaluation pairs. Synthetic static-conditioned sequences show an even stronger effect compared to real static-conditioned sequences, indicating that conditioning on synthetic static profiles does not hurt behavioral alignment. This result is important because it validates the end-to-end pipeline and confirms that the output of stage 1 of the pipeline (Section 3.2.3) can effectively be used to produce the output of the pipeline's stage 2 (Section 3.2.4).

5.3 Limitations and Future Work

Although this thesis demonstrates promising results in generating realistic and conditionally consistent malware logs, there are still several limitations that need to be addressed and considered when interpreting the findings discussed above. These limitations motivate a few research areas that could be explored further to extend and strengthen the two-stage pipeline proposed in this thesis.

First, due to computational constraints, static profile generation relied on a reduced feature representation of around 10,000 features. Although we found that this reduction retained over 99.99% of empirical variance, it is likely that the reduction also discarded rare but potentially important malware features. For instance, features that future malware attacks may possess could have been removed, since malware usually targets high-impact permissions or configuration flags that may not appear frequently enough to be retained. Similarly, when training our dynamic event sequence generator, we conditioned on a subset of the static profile, specifically the dynamic summary statistics containing sequence length and event frequencies. While this reduction in data was necessary to overcome computational constraints, it could potentially introduce further information bottlenecks. Certain static features may influence dynamic behavior in subtle ways which our current model may not accurately capture. Moreover, we trained our sequence generator on truncated sequences of length 2048 tokens due to computational constraints. In our actual dataset, real malware samples had an average length of over 10,000 tokens, so there could be a significant number of malware behavior patterns that were not captured because of our truncation.

The underlying effect in all the reductions we made in both static and dynamic modeling portions of our pipeline is that they may remove consideration of rarer malware patterns that might still go undetected even when using our framework to enhance malware detection systems. A valuable future research continuation would be to explore the methodology of the proposed two-stage pipeline under full data availability without reductions in data made because of computational constraints.

Additionally, a key post-processing step that was applied to synthetic static profiles was marginally matching binary features to their real distribution to improve comparability. This means that some aspects of our synthetic generator are enforced heuristically instead of being learned by the generative model. While this is a common approach in the field of data synthesis for tabular-like data, another interesting future research direction would be enhancing the synthetic profile generators used in this thesis to learn binary features directly without needing to apply any marginal matching.

Lastly, as mentioned earlier, we do not focus on evaluating the downstream use cases of our full synthetic malware sequence generator. This is because training this type of detection system involves its own detailed research and there is no current definitive method that is widely used for such a task. This means that although we have sufficiently backed our full end-to-end two stage generation pipeline with substantive results, there remains the need to test these fully synthetic logs in useful scenarios for malware prevention research. An excellent follow-up to this thesis would be to design a detection system based on dynamic malware event logs to measure the full utility of our synthetic data.

5.4 Conclusions

The main goal in this thesis was to address the central gap in prior malware log generation work: the absence of a unified generative pipeline that can produce logically consistent malware logs with both static and dynamic section. Due to the growing prominence of behavior-based malware detection and the scarcity of malware data containing both static and dynamic information, this work proposed and evaluated a two-stage generative pipeline that modeled malware logs as a joint distribution of static profiles and dynamic event sequences.

The first part of the research explored which class of generative models (GANs, VAEs, or diffusion models) is most effective at learning the complex, high-dimensional distribution of malware static profiles. Through quantitative and qualitative evaluation, across metrics including global structure, feature-wise distribution similarity, feature dependency preservation, and classifier utility, we found that no single model performed the best across all tracked metrics. The WGAN-GP model performed well at matching real malware distributions in terms of global structure and feature-wise distribution similarity whereas diffusion models performed best at feature dependency preservation and downstream classifier utility. Together, these results suggest that the choice of static malware generator depends on the downstream use case, and which evaluation metrics matter more for that use case.

The second part of the research examined how effective transformer-based autoregressive decoders were at generating dynamic malware events sequences that are both realistic and consistent with a given static malware profile. The results demonstrate that static conditioning of our sequence generation model, across distribution-level metrics, semantic validity, and embedding-based behavioral similarity, strongly outperformed the unconditioned baseline. We found that conditioning on static profiles improved token diversity, repetition behavior, semantic validity, resulting in dynamic malware sequences that more closely resembled real malware sequences with similar static profiles. These improvements show that transformer-based autoregressive decoders proved effective at generating dynamic malware events sequences that were realistic and consistent with a given static malware profile.

Beyond just the research questions answered in this thesis, the most significant contribution of this thesis is in terms of providing a full end-to-end pipeline of synthetic malware log generation that demonstrates static and dynamic malware components can be generated jointly in a coherent and controllable way. We bridge a gap between tabular static synthesis and sequential behavior generation, providing a strong framework that can be adapted to any malware log data which is in the format of static and dynamic components. This thesis advances the current state-of-the-art by: showing synthetic malware log generation does not need to operate solely on one data format (either static or dynamic); and establishing that one format can steer the behavior generation of the other, leading to richer and more controllable synthetic datasets. As malware becomes more potent and new forms of malware attacks emerge, frameworks such as the proposed one offer a promising tool for strengthening malware detection systems through high quality synthetic data.

Appendix A: Static Profile Generation Models

A.1 Wasserstein GAN (WGAN-GP)

Table 11. WGAN-GP architecture.

Component	Description
Generator input	Gaussian noise $z \sim \mathcal{N}(0, \mathrm{I})$
Generator layers	Dense(512) → LayerNorm → LeakyReLU (x2), residual block
Generator output	Dense(d)
Discriminator layers	Dense(512) → LeakyReLU → Dense(256) →LeakyReLU
Discriminator Output	Scalar critic score

Table 12. WGAN-GP hyperparameters.

Hyperparameter	Value
Noise dimension	100
Batch size	128
Optimizer	Adam
Learning rate	5×10^{-5}
Adam β_1, β_2	(0.0, 0.9)
Critic updates per step	5
Feature match weight	5.0
Covariance match weight	1.25
Training epochs	200

A.2 Variational Autoencoder (VAE)

Table 13. VAE architecture.

Component	**Description**
Encoder input	Static profile vector z
Encoder layers	Dense(512) $\rightarrow$ ReLU $\rightarrow$ Dense(256) $\rightarrow$ ReLU
Latent space	Gaussian, dimension 100
Decoder layers	Dense(256) $\rightarrow$ ReLU $\rightarrow$ Dense(512) $\rightarrow$ReLU
Decoder output	Dense(d) $\rightarrow$ Sigmoid

Table 14. VAE hyperparameters.

Hyperparameter	**Value**
Latent dimension	100
Optimizer	Adam
Learning rate	1×10^{-3}
Training epochs	50

A.3 Diffusion Model

Table 15. Diffusion model architecture.

Component	**Description**
Input	Noisy static profile $\mathbf{x}_t$
Time embedding	Sinusoidal embedding (128-dim)
Backbone	Residual MLP with FiLM conditioning
Hidden width	2048
Residual blocks	4
Output	Noise prediction $\hat{\epsilon}$

Table 16. Diffusion model hyperparameters.

Hyperparameter	**Value**
Diffusion steps (T)	200
Sampling steps	50 (DDIM)
Optimizer	Adam
Learning rate	1×10^{-3}
Training epochs	50
Noise schedule	Cosine
Time sampling	Importance sampling

Appendix B: Dynamic Event Sequence Generation Models

Table 17. Sequence generation model architecture.

Component	**Static-conditioned model**	**Unconditioned model**
Backbone	GPT-2	GPT-2
Transformer layers	12	12
Attention heads	12	12
Hidden size	768	768
Maximum sequence length	2048	2048
Vocabulary size	98,152	98,152
Static input	Yes	No
Static profile dimension	129	---
Conditioning mechanism	Learned prefix embeddings	---
Prefix length	8 tokens	---
Token embedding type	Learned	Learned
Language modeling head	Linear	Linear

Table 18. Sequence generation conditioning mechanism.

Component	**Value**
Conditioning method	Continuous prefix concatenation
Static $\rightarrow$ prefix mapping	Linear projection
Prefix embedding shape	8×768
Attention behavior	Prefix fully attends and is attended to
Loss contribution	Prefix positions masked

Table 19. Sequence generation model training hyperparameters.

Hyperparameter	**Value**
Training objective	Next-token prediction
Loss function	Cross-entropy
Optimizer	AdamW
Learning rate	3×10^{-5}
Training epochs	3
Precision	Mixed precision
Padding token	Ignored in loss
EOS token	Optional

Table 20. Sequence generation decoding strategy.

Parameter	**Value**
Decoding method	Hybrid greedy + sampling
Greedy steps	512 tokens
Sampling method	Nucleus (top-p)
Top-p	0.9
Temperature	1.0
Repetition penalty	1.10
Max generation length	2048
Early stopping	EOS token (option)

Appendix C: Behavioral Sequence Embedding Encoder

Table 21. Behavioral sequence embedding encoder architecture.

Component	**Value**
Model type	Transformer encoder
Input representation	Token IDs
Vocabulary size	98,152
Chunk length	256 tokens
Token embedding dimension	192
Positional embedding	Learned
Transformer layers	2
Attention heads	6
Feedforward width	4 x hidden dim
Activation function	GELU
Padding handling	Key padding mask
Padding strategy	Masked mean pooling
Output embedding dimension	128
Projection head	2-layer MLP (Linear → GELU → Linear)

Table 22. Behavioral sequence embedding encoder training hyperparameters.

Hyperparameter	**Value**
Learning paradigm	Contrastive learning
Loss function	Symmetric InfoNCE
Temperature	0.10
Positive pairs	Two augmented views of same sequence
Optimizer	AdamW
Learning rate	2×10^{-4}
Weight decay	0.01
Batch size	128
Precision	Mixed precision
Training epochs	25

References

Anonymous. (2025). PersonaLedger: Generating Realistic Financial Transactions with Persona Conditioned LLMs and Rule Grounded Feedback. *Submitted to The Fourteenth International Conference on Learning Representations.* https://openreview.net/forum?id=YPfSfqVedI

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 214–223.

Bao, T., Trousil, K., Duy Tran, Q., Di Troia, F., & Park, Y. (2025). Generating Synthetic Malware Samples Using Generative AI. *IEEE Access*, *13*, 59725–59736. https://doi.org/10.1109/ACCESS.2025.3556704

Blagus, R., & Lusa, L. (2012). Evaluation of SMOTE for High-Dimensional Class-Imbalanced Microarray Data. *2012 11th International Conference on Machine Learning and Applications*, *2*, 89–94. https://doi.org/10.1109/ICMLA.2012.183

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *J. Artif. Int. Res.*, *16*(1), 321–357.

Choi, W. C., Lam, C. T., & Mendes, A. J. (2025). Comparison of Data Imputation Performance in Deep Generative Models for Educational Tabular Missing Data. In C. Mills, G. Alexandron, D. Taibi, G. L. Bosco, & L. Paquette (Eds.), *Proceedings of the 18th International Conference on Educational Data Mining* (pp. 133–142). International Educational Data Mining Society. https://doi.org/10.5281/zenodo.15870169

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

Doersch, C. (2021). *Tutorial on Variational Autoencoders.* https://arxiv.org/abs/1606.05908

Esteban, C., Hyland, S. L., & Rätsch, G. (2017). *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs.* https://arxiv.org/abs/1706.02633

Ferdous, J., Islam, M. R., Mahboubi, A., & Islam, M. Z. (2025). A Survey on ML Techniques for Multi-Platform Malware Detection: Securing PC, Mobile

Devices, IoT, and Cloud Environments. *Sensors*, *25*, 1153. https://doi.org/10.3390/s25041153

G, R., Laudanna, S., S, A., Visaggio, C. A., & P, V. (2021). GANG-MAM: GAN based enGine for Modifying Android Malware. *CoRR*, *abs/2109.13297*. https://arxiv.org/abs/2109.13297

Gewers, F. L., Ferreira, G. R., Arruda, H. F. D., Silva, F. N., Comin, C. H., Amancio, D. R., & Costa, L. D. F. (2021). Principal Component Analysis: A Natural Approach to Data Exploration. *ACM Comput. Surv.*, *54*(4). https://doi.org/10.1145/3447755

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Commun. ACM*, *63*(11), 139–144. https://doi.org/10.1145/3422622

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). *Improved Training of Wasserstein GANs*. https://arxiv.org/abs/1704.00028

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 1322–1328. https://doi.org/10.1109/IJCNN.2008.4633969

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. *CoRR*, *abs/2006.11239*. https://arxiv.org/abs/2006.11239

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hu, J. L., Ebrahimi, M., & Chen, H. (2021). Single-Shot Black-Box Adversarial Attacks Against Malware Detectors: A Causal Language Model Approach. *CoRR*, *abs/2112.01724*. https://arxiv.org/abs/2112.01724

Hu, W., & Tan, Y. (2017). Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR*, *abs/1702.05983*. http://arxiv.org/abs/1702.05983

Huang, Y.-T., Guo, Y.-R., Yang, Y.-S., Wong, G.-W., Jheng, Y.-Z., Sun, Y., Modini, J., Lynar, T., & Chen, M. C. (2024). *SAGA: Synthetic Audit Log Generation for APT Campaigns*. https://arxiv.org/abs/2411.13138

Kamthe, S., Assefa, S., & Deisenroth, M. (2021). *Copula Flows for Synthetic Data Generation*. https://arxiv.org/abs/2101.00598

Kholgh, D. K., & Kostakos, P. (2023). PAC-GPT: A Novel Approach to Generating Synthetic Network Traffic With GPT-3. *IEEE Access*, *11*, 114936–114951. https://doi.org/10.1109/ACCESS.2023.3325727

Kingma, D. P., & Welling, M. (2022). *Auto-Encoding Variational Bayes.* https://arxiv.org/abs/1312.6114

Kotelnikov, A., Baranchuk, D., Rubachev, I., & Babenko, A. (2023). TabDDPM: modelling tabular data with diffusion models. *Proceedings of the 40th International Conference on Machine Learning.*

Kothamali, P. R., & Banik, S. (2022). *Limitations of Signature-Based Threat Detection.*

Li, X. L., & Liang, P. (2021). Prefix-Tuning: Optimizing Continuous Prompts for Generation. In C. Zong, F. Xia, W. Li, & R. Navigli (Eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (pp. 4582–4597). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-long.353

Lin, V., Kaur, R., Yang, Y., Dutta, S., Kantaros, Y., Roy, A., Jha, S., Sokolsky, O., & Lee, I. (2025). Safety Monitoring for Learning-Enabled Cyber-Physical Systems in Out-of-Distribution Scenarios. *Proceedings of the*

ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025). https://doi.org/10.1145/3716550.3722022

Maaten, L. van der, & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*(86), 2579–2605.

Mahdavifar, S., Abdul Kadir, A. F., Fatemi, R., Alhadidi, D., & Ghorbani, A. A. (2020). Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 515–522. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space.* https://arxiv.org/abs/1301.3781

Nagler, T., Bumann, C., & Czado, C. (2019). Model selection in sparse high-dimensional vine copula models with an application to portfolio risk. *Journal of Multivariate Analysis*, *172*, 180–192. https://doi.org/10.1016/j.jmva.2019.03.004

Nazábal, A., Olmos, P. M., Ghahramani, Z., & Valera, I. (2018). Handling Incomplete Heterogeneous Data using VAEs. *CoRR*, *abs/1807.03653*. http://arxiv.org/abs/1807.03653

Park, N., Mohammadi, M., Gorde, K., Jajodia, S., Park, H., & Kim, Y. (2018). Data Synthesis based on Generative Adversarial Networks. *CoRR*, *abs/1806.03384*. http://arxiv.org/abs/1806.03384

Pearson, K. (1900). 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. *Philosophical Magazine*, *50*(302), 157–175.

Perez, E., Huang, S., Song, F., Cai, T., Ring, R., Aslanides, J., Glaese, A., McAleese, N., & Irving, G. (2022). *Red Teaming Language Models with Language Models*. https://arxiv.org/abs/2202.03286

Pudjihartono, N., Fadason, T., Kempa-Liehr, A. W., & O'Sullivan, J. M. (2022). A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction. *Frontiers in Bioinformatics*, *2*, 927312. https://doi.org/10.3389/fbinf.2022.927312

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. *OpenAI Blog*, *1*(8), 9.

Saxena, D., & Cao, J. (2021). Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions. *ACM Comput. Surv.*, *54*(3). https://doi.org/10.1145/3446374

Serra, A., Coretto, P., Fratello, M., & Tagliaferri, R. (2018). Robust and Sparse Correlation Matrix Estimation for the Analysis of High-Dimensional Genomics Data. *Bioinformatics*, *34*(4), 625–634.

Shankar, V., Yousefi, E., Manashty, A., Blair, D., & Teegapuram, D. (2023). Clinical-GAN: Trajectory Forecasting of Clinical Events using Transformer and Generative Adversarial Networks. *Artificial Intelligence in Medicine*, *138*, 102507. https://doi.org/10.1016/j.artmed.2023.102507

Spearman, C. (1904). The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, *15*(1), 72–101. https://doi.org/10.2307/1412159

Tae, J., Ivison, H., Kumar, S., & Cohan, A. (2025). TESS 2: A Large-Scale Generalist Diffusion Language Model. In W. Che, J. Nabende, E. Shutova, & M. T. Pilehvar (Eds.), *Proceedings of the 63rd Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers) (pp. 21171–21188). Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.acl-long.1029

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., & Metzler, D. (2020). *Long Range Arena: A Benchmark for Efficient Transformers*. https://arxiv.org/abs/2011.04006

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Xia, B., Bai, Y., Yin, J., Li, Y., & Xu, J. (2021). LogGAN: a Log-level Generative Adversarial Network for Anomaly Detection using Permutation Event Modeling. *Information Systems Frontiers*, *23*(2), 285–298. https://doi.org/10.1007/s10796-020-10026-3

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional GAN. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc.

Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017). SeqGAN: sequence generative adversarial nets with policy gradient. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2852–2858.

Zhao, Y., Khalman, M., Joshi, R., Narayan, S., Saleh, M., & Liu, P. J. (2022). *Calibrating Sequence likelihood Improves Conditional Language Generation*. https://arxiv.org/abs/2210.00045

Zhu, Y., Lu, S., Zheng, L., Guo, J., Zhang, W., Wang, J., & Yu, Y. (2018). Texygen: A Benchmarking Platform for Text Generation Models. *CoRR*, *abs/1802.01886*. http://arxiv.org/abs/1802.01886