THE UNIVERSITY OF CHICAGO

# GENERATIVE MODELING FOR PACKET-LEVEL NETWORK TRAFFIC SYNTHESIS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY

XI JIANG

CHICAGO, ILLINOIS

AUGUST 2026

To my parents and grandparents.

*"My brain is only a receiver, in the Universe there is a core from which we obtain knowledge, strength and inspiration. I have not penetrated into the secrets of this core, but I know that it exists."*

— Nikola Tesla

**CONTENTS**

## LIST OF FIGURES

## LIST OF TABLES

## ACKNOWLEDGMENTS

I would like to sincerely thank the following people for their support throughout my PhD journey.

I am deeply grateful to my advisor, Nick Feamster, for his consistent support throughout every stage of this journey. Through every project that worked and every one that did not, his guidance was steady, generous, and unwavering. He gave me the freedom to explore ideas on my own terms while always being there with sharp feedback and patient encouragement whenever I needed it. I am a far better researcher, writer, and thinker for having been his student, and I am thankful for the countless hours he invested in helping me grow.

I would also like to thank my committee members, Francesco Bronzino, Sanjay Krishnan, and Paul Schmitt, for their thoughtful feedback and the time they generously gave to help shape this dissertation.

To my labmates and collaborators, Noah Apthorpe, Arjun Nitin Bhagoji, Kevin Bock, Jacob Brown, Andrew Chu, Tajveer Singh Dhesi, Ryan E. Dougherty, Aaron Gember-Jacobson, Dave Levin, Shinan Liu, Anna Lorimer, Tarun Mangla, Jonatas Marques, Kyle MacMillan, Prateek Mittal, Saloua Naama, Sadia Nourin, Nguyen Phong Hoang, Siddhant Ray, Ranya Sharma, Taveesh Sharma, Eddie Shi, Anagha Tiwari, Van Tran, Synthia Wang, and Vinod Yegneswaran, thank you for the collaborations, discussions, and camaraderie that made this work possible and made the day-to-day of a PhD so much more enjoyable.

Finally, to my friends and family, Bobby Chen, Jean David-Zhu, Lin Gui, Amy Huang, Xavier Hu, Thomas Ieema, Zishi Han, Han Jiang, Yibo Jiang, Irene Jin, Bryan Liu, Ziyi Lin, Bobby Lv, Leon, Robert Mao, Hakan Madenci, Holden Meng, Maurice, Merlijn, Leslie Montagne, Jamie Montagne, Tom Montagne, Sheng Qian, Jin Qiu, Amy Song, Jack Tregidga, Fei Tu, Qiong Tu, Yue Tu, Harry Wang, Wency Wang, Chris Xu, Ken Xie, Jingwen Yang, Kefan Yao, Alex Zhang, and George Zhu, thank you for your patience, humor, and encouragement,

and for standing by me through the ups and downs of this journey. This dissertation would not have been possible without you.

## ABSTRACT

Synthetic network traffic plays a critical role in enabling research and development in networking, security, and machine learning. Many modern network analysis systems rely on large-scale datasets of packet traces to train models, evaluate protocols, and reproduce realistic workloads. However, obtaining representative real-world network traces is challenging in practice. Data collection requires specialized infrastructure, operational coordination, and significant storage resources, while privacy concerns and data governance policies often restrict the sharing of captured traffic. As a result, publicly available datasets are limited in both scale and diversity, and frequently become outdated as network applications and protocols evolve.

This dissertation investigates generative modeling approaches for synthesizing realistic packet-level network traffic. The goal is to develop methods capable of producing synthetic traces that faithfully replicate the statistical, structural, and behavioral characteristics of real network communication, while remaining compatible with existing analysis tools and machine learning workflows.

First, this dissertation introduces *NetDiffusion*, a diffusion-based framework for packet-level traffic generation. NetDiffusion transforms packet captures into structured image representations and leverages controlled text-to-image diffusion models to synthesize new traffic traces. The results reveal that *NetDiffusion* is capable of producing high-fidelity packet traces that closely match the statistical properties of real traffic while enabling precise control over traffic classes and characteristics. This approach demonstrates that diffusion models can effectively generate short network traces with strong distributional similarity and protocol compliance.

Second, this dissertation presents *NetSSM*, a state-space-model-based generator that directly models packet sequences using the Mamba architecture. Unlike diffusion-based approaches, which operate over fixed-size representations, NetSSM models network traffic as a sequential process and learns long-range dependencies between packets. This formulation

enables the generation of substantially longer traces and allows the model to capture stateful protocol behaviors and interactions between multiple interleaved flows within a network session.

Together, these approaches highlight complementary strengths of diffusion models and state-space models for synthetic network traffic generation. Diffusion models provide strong control and fidelity for generating short traces with specific characteristics, while state-space models excel at modeling long sequential dependencies and complex multi-flow interactions. By exploring these generative architectures and their trade-offs, this dissertation contributes new methods and evaluation perspectives for synthesizing realistic network traffic, helping to address the persistent scarcity of high-quality network datasets.

# CHAPTER 1

# INTRODUCTION

Modern computer networks are increasingly reliant on machine learning (ML) techniques to support a wide range of operational tasks, including network security, intrusion detection, traffic classification, and performance optimization [140, 100, 114, 12]. These systems depend heavily on large collections of labeled network traffic data in order to train, validate, and evaluate models that can accurately characterize real-world communication patterns. However, the availability of such datasets remains a persistent challenge. Collecting representative network traces requires specialized infrastructure, significant operational effort, and long-term data management. Moreover, privacy concerns and data governance policies often restrict the sharing of captured traffic, particularly when traces contain potentially sensitive information about users or applications [9, 45, 101].

As a consequence, publicly available network datasets are both limited in scale and frequently outdated. Many widely used datasets were collected years or even decades ago, and therefore fail to capture the evolving behavior of modern applications, protocols, and traffic workloads [83, 121, 87]. The resulting scarcity of realistic training data significantly hinders the development of robust machine learning systems for networking. Models trained on small or outdated datasets often fail to generalize to contemporary network environments, limiting their practical utility.

One promising approach to addressing this challenge is the generation of *synthetic* network traffic. Synthetic traffic generation aims to produce new traces that preserve the essential statistical and structural properties of real communication while introducing controlled variation. Such traces can be used to augment existing datasets, expand training corpora for ML models, and enable controlled experimentation without exposing sensitive information contained in real network captures [131, 163, 165, 142, 158]. In principle, syn-

thetic data generation offers a scalable and privacy-preserving mechanism for addressing the scarcity of network datasets.

Despite its promise, designing a high-quality synthetic network traffic generator is far from straightforward. Unlike many other forms of structured data, network traffic exhibits complex dependencies across multiple dimensions, including packet structure, protocol semantics, temporal ordering, and interactions between concurrent flows. Generating realistic packet traces therefore requires capturing both the fine-grained details of individual packets and the higher-level dynamics of network sessions. In particular, a practical traffic generator must balance four key requirements.

1. **Fine-grained protocol fidelity.** Synthetic traces must adhere to the strict structural rules defined by network protocols. For example, TCP communication requires consistent sequence and acknowledgment numbers, valid checksum fields, and correct handshake ordering. Synthetic traffic must also reproduce the statistical distributions of packet header fields observed in real traces. Without such fidelity, generated traffic cannot be processed by standard networking tools or meaningfully improve downstream ML tasks.

2. **Generation control.** A practical generator must do more than passively imitate a fixed training distribution: it must support directed generation of specific traffic classes, application types, or scenarios on demand. Controllability is what allows synthetic traffic to be used for targeted tasks such as rebalancing underrepresented classes or producing traces that exercise a particular network behavior, rather than merely reproducing whatever distribution happens to be most common in the training data.

3. **Long-range sequential reasoning.** Many meaningful network behaviors only emerge over long sequences of packets. Examples include video segment downloads, congestion control dynamics, retransmission events, and connection teardown procedures. Gener-

ators must therefore be capable of learning from and producing traces long enough to capture these flow-state-dependent behaviors.

4. **Multi-flow interaction modeling.** Real-world network workloads frequently involve multiple concurrent and interleaved flows. Video streaming applications interact with content delivery networks, IoT devices communicate with cloud services, and distributed systems coordinate across multiple endpoints. Generating realistic sessions therefore requires modeling the joint dynamics of multiple interacting flows rather than isolated connections.

Existing approaches to synthetic traffic generation typically satisfy only a subset of these four requirements, and the gaps are not incidental implementation shortcomings but consequences of the generative architecture each system is built on. Flow- and attribute-based generators such as DoppelGANger [93] and NetShare [165] model aggregated traffic statistics or a limited set of packet-level attributes rather than complete packet traces; they offer some control over the attributes they model, but sacrifice packet-level fidelity entirely and cannot represent the full structure of real network communication. Transformer-based raw packet generators such as TrafficGPT [115] close part of this gap by generating actual packet bytes, but the quadratic scaling of self-attention with sequence length restricts them to relatively short, single-flow sequences and offers little explicit control over the resulting traffic beyond the training distribution. No existing system simultaneously achieves fine-grained protocol fidelity, directed control over generated traffic, long-range sequential reasoning, and multi-flow interaction modeling.

The central thesis of this dissertation is that *the choice of generative architecture fundamentally determines which of these requirements a synthetic traffic generator can satisfy.* The limitations of prior work described above are not failures of engineering effort but architectural consequences: the representation and inductive biases of a generative model determine, in advance, which properties of network traffic it is capable of reproducing. Diffusion models and state-space models, the two architectures studied in this dissertation,

occupy complementary positions in this design space, and no single architecture satisfies all four requirements at once.

This dissertation explores the design space of generative architectures for synthetic network traffic generation. Specifically, it investigates how different classes of generative models can be adapted to produce realistic packet-level traffic traces and examines the trade-offs they introduce across the four requirements described above. The work is structured around two complementary systems, each leveraging a different generative modeling paradigm.

## 1.1 Dissertation Overview

**Chapter 1: Introduction.** This chapter motivates the need for synthetic packet-level network traffic, arguing that the persistent scarcity and rapid obsolescence of publicly available traffic datasets limit the development of machine learning systems for networking. It identifies four requirements that a practical traffic generator must satisfy—fine-grained protocol fidelity, generation control, long-range sequential reasoning, and multi-flow interaction modeling—and argues that no existing generator satisfies all four simultaneously because these gaps stem from the generative architecture underlying each system rather than from implementation effort. This observation motivates the dissertation's central thesis: that the choice of generative architecture fundamentally determines which of these requirements a synthetic traffic generator can satisfy. The chapter then previews the two systems developed in this dissertation, NetDiffusion and NetSSM, and summarizes the dissertation's key contributions.

**Chapter 2: Background and Related Work.** This chapter surveys prior work in synthetic network traffic generation and situates the dissertation's contributions within that broader design space. It organizes existing approaches along three dimensions: whether traffic is produced through explicit simulation or data-driven learning; whether data-driven methods represent traffic as aggregated attributes or raw packet captures; and how methods

evaluate the realism and utility of generated data. The chapter reviews the major generative model architectures used in this space, including generative adversarial networks, diffusion models, transformers, and state-space models, and argues that diffusion-based and state-space-based approaches occupy complementary positions in this design space, motivating the two systems developed in the remainder of the dissertation.

**Chapter 3: NetDiffusion.** The first contribution of this dissertation addresses the challenge of achieving high protocol fidelity and fine-grained generation control in packet-level synthetic packet traces. Prior GAN-based traffic generators [93, 165] primarily produce aggregated flow statistics or limited sets of packet attributes, resulting in synthetic data that lacks the richness and realism of raw packet captures. These limitations often lead to poor statistical similarity to real traffic and reduced effectiveness when used for downstream machine learning tasks.

To address these limitations, this dissertation introduces *NetDiffusion*, a framework that applies controlled diffusion models to the generation of packet-level network traffic. The key idea underlying NetDiffusion is a representation bridge between network traffic and the image domain: packet captures are converted into structured image representations, allowing powerful text-to-image diffusion models to be applied to network trace generation. Diffusion models have demonstrated remarkable ability to capture intricate spatial patterns in high-dimensional data, and NetDiffusion leverages this capability to reproduce complex packet-level distributions. Generation is conditioned on text prompts describing the target traffic class or application, giving NetDiffusion direct, prompt-driven control over the composition of the generated dataset. Post-generation heuristics enforce protocol constraints across packets, ensuring the resulting traces remain compliant with TCP/IP semantics.

NetDiffusion demonstrates that *diffusion models are well suited for generating short, high-fidelity, and controllable single-flow network traces.* The approach achieves strong statistical similarity to real traffic and improves downstream machine learning performance when synthetic data is used for training. However, the image-based representation introduces a

fundamental limitation: traces are restricted to fixed-size representations, typically on the order of $10^3$ packets per flow, preventing the capture of longer session dynamics.

**Chapter 4: NetSSM.** The second contribution of this dissertation addresses the limitations identified in NetDiffusion by adopting a fundamentally different generative modeling paradigm. Instead of representing network traffic as images, *NetSSM* directly models packet byte sequences using the Mamba structured selective state-space model (SSM) architecture. Packet generation is treated as a sequential prediction problem, allowing the model to explicitly capture temporal dependencies between packets.

State-space models provide a natural framework for modeling network traffic, as their recurrent hidden state mirrors the stateful behavior of many network protocols. By leveraging the Mamba architecture, NetSSM scales efficiently to very long sequences while preserving the ability to model complex packet-level dependencies.

NetSSM trains with context windows more than 8× longer than prior transformer-based packet generators, enabling it to capture flow-state events that occur only after substantial session setup. Importantly, NetSSM is also the first generator capable of producing realistic *multi-flow network sessions*, where multiple connections interact within a single trace. This capability enables the modeling of real-world workloads such as video streaming sessions involving CDN communication and adaptive segment downloads.

Overall, NetSSM demonstrates that *state-space models are well suited for generating long, stateful, multi-flow network traces.* The system achieves better performance across statistical similarity metrics, downstream ML tasks, and newly introduced semantic similarity evaluations compare to existing methods. This capability comes at a cost along the control dimension, however: because generation proceeds autoregressively from a seed of label and packet tokens rather than through explicit conditioning, NetSSM offers less fine-grained control over the resulting traffic than NetDiffusion's prompt-driven generation.

## 1.2 Key Contributions

This dissertation makes the following contributions:

- **NetDiffusion**: the first diffusion-based framework for generating raw packet captures, demonstrating that image representations of network traffic enable controlled synthesis of short, high-fidelity single-flow packet traces (Chapter 3).
- **NetSSM**: the first state-space-model-based network traffic generator capable of producing long packet traces and realistic multi-flow sessions, substantially extending the capabilities of prior approaches (Chapter 4).
- **A comparative understanding of generative architectures** for synthetic network traffic, characterizing the trade-offs between diffusion models and state-space models along the dimensions of protocol fidelity, generation control, trace length, and session complexity. More importantly, we explore how these unique characteristics influence their suitable use cases.

Together, these contributions advance the state of the art in synthetic network traffic generation and provide new insights into how generative model architectures influence the realism and utility of synthetic network datasets. The results suggest promising directions for future hybrid architectures that combine the controllability of diffusion models with the long-context modeling capabilities of state-space approaches.

# CHAPTER 2

# BACKGROUND AND RELATED WORK

Synthetic network traffic generation has long been studied as a means of supporting controlled experimentation, benchmarking network systems, and augmenting datasets for machine learning. Broadly, the goal is to generate traffic that preserves important properties of real network communication while avoiding the operational overhead, privacy risks, and limited accessibility associated with collecting large-scale real-world traces.

Prior work in this area can be understood along three main dimensions. First, methods differ in whether they generate traffic through *explicit simulation* or *data-driven learning*. Second, among data-driven approaches, methods vary in the *representation* of generated traffic, ranging from aggregated traffic attributes to raw packet captures. Third, methods differ in the *evaluation methodologies* used to assess the realism and utility of generated data. This chapter surveys the most relevant literature along these dimensions and situates the contributions of this dissertation within that broader design space, providing a detailed technical treatment of the systems introduced at a high level in Chapter 1.

## 2.1 Simulation-Based Traffic Generators

Simulation frameworks represent one of the earliest and most widely used approaches to synthetic network traffic generation. Tools such as NS-3 [64], TRex [8], and related systems [21, 25, 88] generate traffic by simulating network topologies, protocol stacks, and application workloads. Users specify network structure, routing policies, link characteristics, and traffic sources, and the simulator produces synthetic communication events that approximate real network activity.

A key strength of simulation-based systems is that they can generate packet-level traffic, including long traces and complex communication scenarios, while offering substan-

tial configurability and control. As a result, they remain valuable for protocol evaluation, stress testing, and benchmarking networked systems. However, because their behavior is defined by explicit models of network structure, protocols, and workloads rather than learned directly from measured traffic, they often struggle to capture the variability, complexity, and emergent dynamics of real-world network environments, especially those shaped by modern applications and user behavior [26, 149]. Accurately modeling contemporary workloads therefore often requires substantial manual effort and domain expertise.

## 2.2 Data-Driven Traffic Generators

In contrast to simulation-based approaches, a growing body of work seeks to learn traffic generation directly from measured network traces. These data-driven systems attempt to model the statistical structure of observed traffic and synthesize new samples that preserve important properties of real communication patterns. Existing data-driven generators can be broadly divided into two categories based on the representation they model: generators that operate over traffic attributes and generators that attempt to synthesize raw packet traces.

### 2.2.1 Traffic Attribute Generators

A large body of data-driven work focuses on generating high-level traffic descriptions rather than raw packet traces. These systems operate over traffic attributes such as flow duration, average packet size, protocol identifiers, header field values, or event metadata. Because such representations are more compact than full packet captures, they are often easier to model and can be useful for reproducing aggregate traffic behavior.

More recent work in this space applies machine learning to traffic generation by treating it as a time-series modeling problem over traffic attributes. These methods learn statistical dependencies among attributes and generate synthetic samples that match the marginal and temporal distributions observed in real data.

DoppelGANger [93], for example, applies a generative adversarial network (GAN) to sequences of traffic attributes. NetShare [165] extends this idea to richer traffic representations, generating both aggregate flow statistics and selected packet-level header field values. NetDiff [171] combines diffusion models with transformer components to better encode and generate traffic attribute sequences, with a particular focus on mobile network data.

Although these systems improve the statistical realism of synthetic traffic, they remain limited by their reliance on aggregated or partial representations. In particular, they do not model packet traces directly and therefore cannot capture packet-level protocol semantics in full detail. This limitation is especially pronounced for stateful transport protocols such as TCP, where realistic behavior depends on detailed interactions among header fields, packet ordering, and session state. Consequently, attribute-based generators are useful for some forms of traffic modeling, but they generally lack compatibility with packet-level analysis tools and are less suitable for applications that require realistic end-to-end traces.

### 2.2.2 Raw Packet Generators

To address the limitations of attribute-based generation, a growing body of data-driven work has turned to generating synthetic traffic directly at the packet level. Raw packet generators produce complete packet captures (PCAPs), allowing the resulting traces to be inspected with commodity analysis tools such as Wireshark [17] or replayed with frameworks such as tcpreplay [46]. This packet-level representation is substantially more expressive than aggregate flow statistics, as it preserves fine-grained protocol structure and supports downstream tasks that rely on detailed traffic semantics.

Recent work has explored several architectural directions for packet-level traffic generation, each offering different trade-offs in terms of fidelity, scalability, and controllability.

**Transformer-based packet generators.** One recent direction applies large sequence models to packet bytes. TrafficGPT [115] treats packet generation as an autoregressive token prediction problem, applying a transformer decoder to raw PCAP byte sequences.

This formulation draws a natural analogy between packet generation and language modeling. However, because transformer attention scales quadratically with sequence length, such models are fundamentally limited in the length of packet traces they can generate efficiently. As a result, capturing long-range dependencies across extended network sessions remains challenging for transformer-based approaches.

**Diffusion-based packet generators.** Another direction leverages diffusion models for packet-level generation. NetDiffusion, presented later in Chapter 3, maps packet captures into structured image representations and uses a fine-tuned diffusion model to synthesize new packet traces. Diffusion-based generation offers strong distributional fidelity and fine-grained control over generated structure, enabling the synthesis of packet traces that closely match the statistical and protocol characteristics of measured traffic. However, diffusion models operate over fixed-size representations and therefore naturally target shorter traces. In practice, they are best suited for generating high-fidelity traffic segments on the order of $10^3$ packets where precise control over packet structure is important.

**State-space approaches to packet generation.** A preliminary feasibility study by Chu and Jiang *et al.* explored whether state-space models based on Mamba-1 could be used for packet-trace generation [35]. That study focused on single-flow traffic and demonstrated the basic viability of using state-space sequence models for packet generation. Building on that foundation, NetSSM, presented in Chapter 4, develops a more complete packet generator based on Mamba-2 and extends the setting from single-flow traces to long, multi-flow network sessions. State-space models provide linear-time sequence processing and therefore scale naturally to very long packet sequences, allowing them to capture long-range temporal dependencies across complex network workloads.

Taken together, these results suggest that no single generative architecture is universally optimal for packet-level traffic synthesis. Instead, different architectures excel in different regimes. Diffusion-based generators provide exceptional fidelity and controllability for relatively short single-flow traces where fine-grained protocol structure must be pre-

served. In contrast, state-space sequence models offer strong scalability for modeling long packet streams and complex multi-flow interactions, albeit with less direct structural control during autoregressive generation.

This observation motivates the central design philosophy of this dissertation. Rather than pursuing a single monolithic generator, this work develops two complementary systems optimized for different operating regimes. NetDiffusion targets short, highly controlled packet synthesis with extremely high distributional fidelity, while NetSSM focuses on scalable generation of long, multi-flow traffic sequences. Together, these systems demonstrate how different generative architectures can be selected based on the requirements of the target traffic synthesis task.

## 2.3 Generative Model Architectures

The methods described above also differ fundamentally in the generative architectures they employ. Because the strengths and weaknesses of synthetic traffic generators are closely tied to the properties of their underlying models, it is useful to review the major architectural families used in this space.

**Generative Adversarial Networks (GANs).** GANs [39] train a generator and discriminator in an adversarial loop, with the generator attempting to produce samples that are indistinguishable from real data. GAN-based methods have been widely used for synthetic traffic generation, particularly in attribute-level settings such as DoppelGANger and NetShare. While GANs can generate realistic samples, they are well known to suffer from training instability, sensitivity to hyperparameters, and mode collapse, all of which may limit the diversity and robustness of generated network traffic.

**Diffusion Models.** Diffusion models [138, 40] generate data by learning to reverse a gradual noising process. They have become a dominant approach in image generation [123, 117], and have also been extended to video and structured data generation. Compared to GANs,

diffusion models generally offer more stable training and higher fidelity outputs [81, 79]. Stable Diffusion [123], in particular, combines latent diffusion with powerful pretrained image representations, making it attractive for domains that can be mapped into structured image-like forms. ControlNet [170] further extends this paradigm by allowing the use of conditioning inputs such as edge maps or structural cues, enabling tighter control over generated samples. These properties make diffusion models especially appealing for network traffic generation when fidelity and structural control are primary objectives.

**Transformers.** Transformers [156] model sequential dependencies through self-attention and have achieved remarkable success in natural language and code generation [116]. Their main advantage is expressive sequence modeling over flexible tokenized inputs. Their main limitation, in the context of network traffic generation, is the quadratic scaling of self-attention with sequence length. This makes very long packet traces computationally expensive to model and generate.

**State-Space Models (SSMs).** Structured state-space models [55] provide an alternative approach to sequence modeling that scales linearly with sequence length. The Mamba architecture [53] augments SSMs with selective state transitions, allowing the model to focus adaptively on relevant parts of an input sequence. Mamba-2 [44] improves further on Mamba-1 through larger modeled state and more efficient training kernels. Because network communication is inherently sequential and often stateful, SSMs offer a natural inductive bias for modeling packet streams, session evolution, and dependencies that unfold over long time horizons.

## 2.4 Evaluation of Synthetic Network Data

Evaluating the quality of synthetic network traffic remains an open and multifaceted challenge. The most common evaluation methodology focuses on *statistical similarity*, measuring how closely distributions extracted from synthetic traces match those observed in real traf-

fic. Representative metrics include Jensen–Shannon divergence, Wasserstein distance, and related distributional measures over traffic attributes.

A second widely used criterion is *downstream machine learning utility*, which asks whether synthetic traffic improves the performance of classifiers or other models trained for networking tasks such as traffic classification, anomaly detection, or intrusion detection. This evaluation is especially important when the goal of generation is dataset augmentation.

However, these criteria alone are insufficient for evaluating packet-level synthetic traffic. A packet trace can match marginal distributions reasonably well and still fail to reflect the structure and semantics of real network communication. In particular, high-quality synthetic packet traces should also satisfy several additional properties.

- **Protocol compliance.** Generated packets should obey the structural and semantic rules of network protocols, including valid header formats, checksums, and TCP state transitions.

- **Tool compatibility.** Generated traces should be analyzable using standard networking tools such as Wireshark and replayable using systems such as tcpreplay.

- **Application-level realism.** Generated traces should reflect higher-level workload behavior, such as segment download patterns in video streaming, CDN interactions, or session-level communication structure.

- **Session realism.** For generators that target long traces or multiple flows, synthetic traffic should preserve long-range dependencies and inter-flow dynamics rather than merely reproducing short local packet statistics.

To capture these broader notions of realism, this dissertation evaluates synthetic traffic using a combination of statistical similarity, downstream ML performance, protocol compliance, tool compatibility, and semantic similarity. In particular, the NetSSM work introduces semantic similarity as an explicit evaluation dimension for assessing whether synthetic traces reproduce application-level communication patterns observed in real traffic.

# CHAPTER 3

# NETDIFFUSION: PROTOCOL-CONSTRAINED TRAFFIC GENERATION VIA DIFFUSION MODELS

As established in Chapter 2, existing synthetic traffic generators based on GANs and flow-level statistics fail to produce raw packet captures with sufficient protocol fidelity or statistical resemblance to real traffic. This chapter presents *NetDiffusion*, our first approach to this problem, which explores a novel representation: encoding network packet captures as images and applying powerful text-to-image diffusion models to generate new synthetic traces.

The central insight behind NetDiffusion is that diffusion models, which have demonstrated superior quality and training stability over GANs in the image domain, can be harnessed for network traffic generation by treating traffic as a structured spatial artifact. By converting raw packet headers into a pixel-level image representation, we enable fine-tuned stable diffusion models to learn the intricate distributional patterns of real traffic across all packet header fields simultaneously. Conditional generation via ControlNet provides coarse-grained protocol distribution control, and post-generation heuristics enforce TCP/IP protocol rule compliance at the field level.

NetDiffusion establishes that *diffusion models with image representations achieve strong protocol fidelity and statistical similarity for short network traces*, substantially outperforming prior GAN-based methods. Its outputs are compatible with standard networking tools (Wireshark, tcpreplay) and improve downstream ML classifier performance. The fundamental limitation—a cap of 1,024 packets per trace due to fixed image resolution—motivates the sequential approach taken in Chapter 4.

The remainder of this chapter is organized as follows. Section 3.1 introduces the problem setting and provides an overview of the NetDiffusion framework. Section 3.2 analyzes the limitations of prior work. Section 3.3 describes the NetDiffusion pipeline in detail. Section 3.4 presents our evaluation results. Section 3.5 discusses directions for future work.

## 3.1 Introduction

Modern networks are increasingly reliant on machine learning (ML) techniques for a wide range of management tasks, ranging from security to performance optimization. A central impediment when training network-focused ML models is the scarcity of labeled network datasets, as their collection and sharing are often associated with high costs and privacy concerns, particularly when data is collected from real-world networks [140, 100, 101, 9, 45]. Unfortunately, existing public datasets rarely receive updates, making them static and unable to reflect evolving network behaviors [83, 121, 87]. These limitations hinder the ability to train robust ML models that accurately reflect evolving real-world network conditions.

These challenges can be addressed through the creation of new synthetic network traces based on existing datasets. This approach aims to preserve the inherent characteristics of network traffic while introducing variations, thereby enhancing dataset size and diversity [131, 163, 111, 165, 142, 158, 93, 166]. Unfortunately, current state-of-the-art synthetic trace generation methods, particularly those based on Generative Adversarial Networks (GANs)-based methods [120, 165, 93, 163, 162], are not always sufficient for producing high-quality synthetic network traffic. Specifically, these approaches tend to focus on a limited set of attributes or statistics, as early machine learning for network tasks often relied on basic flow statistics for classification [113, 47, 36, 89, 90, 18, 80]. With recent ML advancements utilizing detailed raw network traffic to achieve enhanced classification accuracy [119, 174, 96, 10, 164, 160, 130, 41, 24, 97, 148], there is a clear need for synthetic traffic generation that includes the intricate, potentially unforeseen patterns present in full network traces. Existing traffic generation methods face two main issues: (1) a lack of statistical similarity with real data due to the limited attributes in existing methods, making the synthetic data highly sensitive to variations, and (2) unsatisfactory classification accuracy when synthetic statistical attributes are used to augment existing datasets. Moreover, their simplistic attribute focus and disregard for transport and network layer protocol behaviors prevent their use with traditional networking tools such as tcpreplay [46] or Wireshark [17].

Fortunately, the general increase in available computational power and the breakthroughs in high-resolution image generation techniques, particularly diffusion models [138, 123, 117], offer a promising avenue to overcome these challenges. Specifically, we harness the capabilities of text-to-image diffusion models, which execute conditioned generation based on descriptive text prompts. These models are adept at creating detailed, accurate visual representations from textual descriptions. By translating the intricate characteristics of network traffic into an appropriate image format, we can tap into the unique advantages offered by these models. In contrast to GANs, diffusion models are able to capture both broad patterns and detailed dependencies. This inherent generative quality makes them an ideal choice for producing network traces with high statistical resemblance to real traffic and full packet header values. By incorporating conditioning techniques, diffusion models can generate structured data that conforms to specific network properties, which ensures the desired sequential inter-packet characteristics and rough protocol dependencies. Moreover, the gradient dynamics of the training process in diffusion models is much more stable than GANs. We discuss the technical details and benefits of diffusion models in depth in Section 3.3. These attributes collectively position diffusion models as a compelling choice for advancing the state-of-the-art for synthetic network trace generation, addressing the extant limitations of current methodologies.

In this paper, we introduce NetDiffusion, an approach to synthetic raw network traffic generation for producing packet headers leveraging fine-tuned, controlled stable diffusion models. Our contributions are as follows:

1. **Generation of synthetic network traces with high resemblance to real traffic:** Using stable-diffusion techniques, we propose a two-fold strategy: (1) a conversion process for transforming raw packet captures to image representations (and vice versa), and (2) fine-tuning a text-to-image diffusion model based on packet capture-converted images for generating synthetic packet captures. To improve resemblance to real network traffic, we employ controlled generation techniques to maintain fidelity to the

protocol and header field value distributions observed in real data and, post generation, use domain knowledge-based heuristics to finely check and adjust the generated fields, ensuring their semantic correctness in terms of compliance with transport and network layer protocol rules.

2. **Improved classification accuracy in ML scenarios with synthetic network traffic augmentation:** We conduct a case study evaluation on a curated traffic classification dataset. By integrating NetDiffusion-generated network traffic into the real dataset at varying proportions during training and testing, we observe a general increase in accuracy compared to the state-of-the-art generation method [165]. This improvement is attributed to our synthetic data's significantly high statistical resemblance to the real dataset. Additionally, our method shows promise in addressing class imbalance issues, enhancing the accuracy of ML models in such cases.

3. **Extended applicability of synthetic network traffic for network analysis and testing beyond ML tasks:** NetDiffusion-generated network traffic can be converted into raw packet captures suitable for traditional network analysis and testing tasks. We validate this compatibility through tests with tools such as Wireshark and Scapy [122], as well as tcpreplay for retransmission. More importantly, we show that critical statistical features for various network operations can be effectively extracted from the generated network traffic.

## 3.2 Motivation

The use of publicly available network datasets has significantly aided advancements in applying ML to networks, as well as network analysis and testing methodologies. For example, models trained on network datasets have been helpful in tackling challenges like anomaly detection, traffic classification, and network optimization, which in turn enhances network security and performance [22, 73, 94, 96, 75, 77, 9, 107, 82, 152, 33]. Additionally, these

datasets are valuable for network analysts, aiding in understanding network behaviors, identifying performance issues, and evaluating the performance of network security tools like firewalls and intrusion detection systems [155, 109, 118].

### 3.2.1 Network Data Scarcity

Well-known network datasets such as CAIDA [7], MAWI [32], UNSW-NB15 [108], and KDD [6, 151] have been essential for numerous research projects in network science. However, the lack of updated datasets often hinders further progress. Those with the means to capture large-scale traffic, typically network operators and organizations with specialized hardware and network, are often hesitant to share their data due to the risk of exposing sensitive or personally identifiable information (PII). Even when entities are amenable to sharing, the task of providing consistent updates and ensuring reliable labels for sanitized data is daunting. Labeling network data is inherently challenging because of its dynamic nature, such the continuous evolution of network behaviors and threats. Notably, the CAIDA, UNSW-NB15, KDD Cup 99 [6], and NSL-KDD [151] datasets were last updated in 2020, 2015, 1999, and 2009 respectively, revealing notable gaps in data recency which render them less reflective of evolving network dynamics. Even frequently updated datasets like MAWI [32] are not exempt from issues, with instances of missing data from hardware failures and substantial duplicate traces. While not an exhaustive list of datasets, the issues highlighted are common across the board, accentuating the need for newer data to fuel ongoing network research and analysis.

### 3.2.2 Data Augmentation Using Synthetic Data

Data augmentation through synthetic data has proven effective in many fields. In computer vision, for instance, synthetic images have improved model performance, especially when there's a shortage of labeled data [103, 31, 150, 134, 95]. The success of synthetic data augmentation is largely attributed to generative methods which have showcased remarkable

versatility in a variety of domains: In medical imaging, GANs have been harnessed to augment datasets, significantly enhancing the performance of diagnostic models [52, 57, 29]. In the domain of natural language processing, Variational Autoencoders (VAEs) have been utilized to create synthetic text data, aiding in tasks such as sentiment analysis and language translation [161, 153, 34]. In audio processing, the advent of WaveGAN has facilitated the expansion of sound datasets, proving indispensable for applications like speech recognition and sound event detection [48, 49, 14].

Translating these successes to the networking domain, certain endeavors have emerged, attempting to augment network datasets through the generation of synthetic network data [120, 165, 93, 163, 162]. The overarching goal of these generative methods is to produce synthetic traffic that exhibits high levels of statistical similarity to real-world network traffic. Distinguishing them from simulation-based approaches, these generative techniques introduce subtle variations in the synthetic data, diverging slightly from the real data. This aspect is crucial; it simulates potential, unseen variations and the dynamic nature of real-world network environments. By doing so, while the synthetic traffic largely mirrors the general patterns of the real data, it also aids in enhancing the generalizability of various applications, such as anomaly detection systems and ML models. For instance, in anomaly detection, these nuanced variations in synthetic traffic allow models to adapt better to unpredictable or novel network behaviors, such as small variations in the IP addresses or TCP flags, thereby improving their effectiveness and robustness in real-world scenarios. A notable state-of-the-art attempt in this regard is NetShare [165] which utilizes GANs to produce IPFIX [38]/NetFlow [37]-style statistics on network traffic. For simplicity, we refer to this general type of aggregated statistucal attributes as NetFlow for the rest of the paper. Yet, its focus on statistical properties might miss important network patterns essential for high ML accuracy. At the same time, the limited number of attributes that it focuses prohibits the generation of comprehensive, raw network traffic such as packet captures, which are essential for additional non-ML tasks such as network analysis and testing. In this paper, we do not consider other

non-generative methods [88, 64, 25, 21] like TRex [8] and NS-3 [64] because, while useful for specific tasks, they lack the flexibility needed for broader dataset augmentation. They often rely on predefined templates or rules, which may not capture the evolving nature of network traffic or the complex interactions between various network protocols and applications.

### 3.2.3 Inadequate Performance from Existing Methods

We provide a short case study on NetShare, which is the current state-of-the-art network data generation method that produces NetFlow attributes, *i.e.,* , derived statistics from raw network traffic flows. Following the method in the original paper, we test the accuracy of a variety of ML models–Random Forest (RF), Decision Tree (DT), and Support Vector Machine (SVM)–under three scenarios: (1) training and testing on real NetFlow data, (2) training on NetShare synthetic data and testing on real data, and (3) vice versa. In this paper, we focus on a traffic classification task using a curated dataset, detailed in Section 3.4. The classification task is divided into *micro* and *macro* levels. On the micro level, the goal is to classify network traffic flows into specific applications, encompassing 10 distinct classes such as YouTube and Amazon. On the macro level, the aim is to classify network traffic flows into broader service categories, spanning 3 classes, like streaming and web browsing.

**Unsatisafactory ML Accuracy.** (1) NetShare Limitations: The results from Table 3.1 showcases a clear drop in accuracy when models are trained or tested on synthetic NetFlow data generated by NetShare compared to when only real NetFlow data is used, irrespective of the classification level. This points to a likely shortfall of NetShare in preserving critical distinguishing feature values inherent in the real dataset, which adversely affects the models' ability to accurately classify network traffic. (2) NetFlow vs. Raw Traffic: Following the NetShare evaluation, we compared the accuracy achieved with real NetFlow data to that when models train and test on raw network traffic in pcap format. This comparison underscores the information loss encountered when using NetFlow data and the potential classification performance gain from leveraging raw network traffic: The highest accuracy is achieved with

raw network traffic, with SVM arriving at near-perfect accuracy. Conversely, a noticeable decrease in accuracy is observed with real NetFlow data, suggesting that its limited feature set adversely affects classification accuracy.

These observations lead to two main insights: First, the need for synthetic data generation methods to effectively preserve critical distinguishing feature values to maintain classification accuracy; Second, the advantage of using raw network traffic over NetFlow data due to its richer information content. The push towards generating raw network traffic to retain the fine-grained details and statistical properties of real network traffic, appears to be a crucial step to overcome the limitations observed with NetShare generated data and NetFlow data.

| **Training/Testing** | **Data Format (Generation Method)** | **Highest Accuracy (Model)** | |
|---|---|---|---|
| | | **Macro-level** | **Micro-level** |
| Real/Real | Network traffic (pcap) (N/A) | 1.00 (SVM) | 0.978 (SVM) |
| | NetFlow (N/A) | 0.86 (RF) | 0.648 (DT) |
| Synthetic/Real | NetFlow (NetShare) | 0.396 (RF) | 0.140 (SVM) |
| Real/Synthetic | NetFlow (NetShare) | 0.503(SVM) | 0.102 (RF) |

**Table 3.1:** Comparison of model accuracy using real, synthetic NetFlow data, and raw network traffic. Results highlight a decrease in accuracy with NetShare's synthetic data and a boost when using raw traffic. Only the top-performing model is displayed.

**Limited Applicability to Non-ML tasks.** Besides the performance of synthetic network data in ML tasks, another important metric to validate its usefulness is its applicability to traditional network analysis and testing tasks, such as packet-wise analysis and replay. This is important because, unlike other forms of data such as images where the quality of the data can be inferred relatively easily by visual resemblance to real images, it is hard for network experts to manually examine raw network traffic to verify its quality. NetFlow data, encapsulating aggregated or derived statistics from raw network traffic flows, lacks the detailed information crucial for these tasks. For instance, network analysts often use tools like Wireshark to investigate network traffic details like packet headers and sequences to diagnose issues or assess performance, such as tracing latency causes or detecting unauthorized access. However, the high-level statistical nature of NetFlow data omits such fine-grained

details, rendering it inadequate for such in-depth analysis. Furthermore, synthetic NetFlow data cannot be retransmitted or replayed over network interfaces using tools like tcpreplay. Retransmitting network traffic is pivotal for various network testing and validation scenarios, such as evaluating the performance of network security tools under realistic traffic conditions or stress-testing network infrastructure. The absence of packet-level details in NetFlow data precludes its use for retransmission tasks. Moreover, certain network analysis tasks require deriving additional attributes directly from raw network traffic. For example, estimating the window size or investigating the distribution of packet sizes across a network necessitates access to raw traffic data. These computations are crucial for understanding network behavior and optimizing network configurations.

Converting high-level statistical NetFlow data back to raw network traffic, such as packet captures, is inherently challenging due to the loss of detailed attributes like header values and flags, thereby limiting the utility of synthetic NetFlow data for a myriad of non-ML network tasks. This challenge underscores the necessity of generating raw network traffic. At the same time, existing network data generation methods often neglect to ensure that synthetic data adhere to critical transport and network layer protocol rules found in real network traffic, which are crucial for traditional network analysis and testing tools. For example, a protocol constraint is that packet length frames must conform to specific sizes as per protocol standards. Generating synthetic data with incorrect frame sizes could lead to misinterpretations in data analysis or malfunctions in network testing scenarios. Other protocol constraints, like correct sequence numbers in TCP transmissions, valid checksums, and appropriate flag settings, are also crucial as they affect how network devices and analysis tools interpret and process the data. Hence, adhering to protocol rules is essential for the accuracy and reliability of network analysis and testing tasks, and serves as a gauge for the quality of synthetic network data generated.

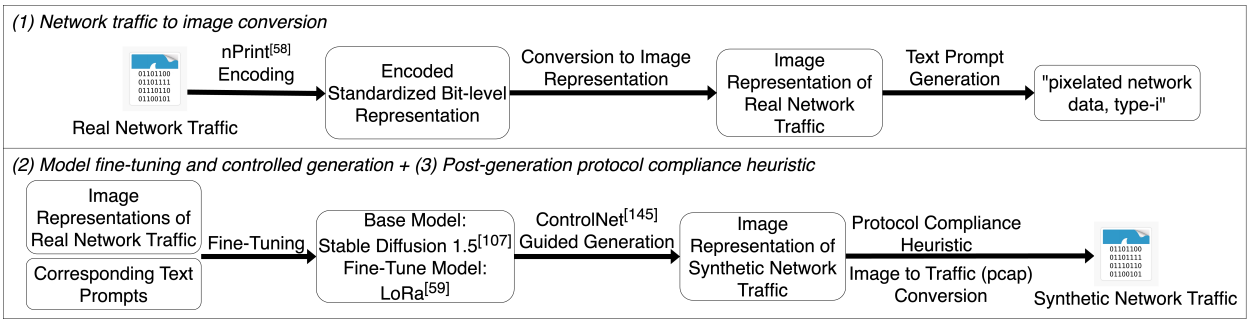



**Figure 3.1:** Generation Framework Overview.

## 3.3 Method

In this section, we introduce *NetDiffusion*, a framework that harnesses controlled text-to-image diffusion models [123] to generate synthetic raw network traffic that complies with transport and network layer protocol rules. We find this approach not only elevates classification accuracy when utilized for data augmentation in ML scenarios but also facilitates a broad range of network analysis and testing tasks (see Section 3.4), overcoming the limitations described in Section 3.2.3. We first provide an overview of our method which has the 3 components shown in Figure 3.1, before providing details of each component.

**How do diffusion models work?** Diffusion models synthesize data by modeling data generation as the process of noise removal from noisy data (referred to as the reverse process) [145, 65]. The noise removal is performed by a complex ML model, usually a neural network, that has been trained to predict the noise that was sequentially added to real data (the forward process). Running the forward and reverse processes in the latent space of a model has been found to generate better quality data [123]. Mathematically, consider an initial noise vector ($\mathbf{z}$) in the latent space. The goal of diffusion models is to transform $\mathbf{z}$ into an data point ($\mathbf{x}$) drawn from the desired distribution. The idea is to define a differential equation that controls the transformation from $\mathbf{z}$ to $\mathbf{x}$ over a series of discrete time steps. The rationale behind this is that by breaking down the generation process into a series of incremental diffusion steps, the model can capture intricate dependencies and details in the data manifold. An essential component of this approach is score-based generative modeling,

where the gradient of the data log-likelihood with respect to the data (often termed the "score" function) is estimated. Modeling the score function is preferable because directly modeling the probability distribution poses challenges, especially in obtaining the correct normalization constant [144, 145]. Diffusion models have been adopted most effectively in the context of text-to-image synthesis, where images matching a given text prompt are to be generated. The text prompt serves as a conditioning variable to guide the reverse process towards generating an image that semantically aligns with the text prompt. By iteratively applying the score function, conditioned on the text prompt, the model steers the data from a simple prior distribution (like Gaussian noise) to the desired complex image distribution. In our case, the text prompts characterize the *specific classes/types of network traffic* that we aim to generate.

We extend these principles on the operation of diffusion models to network traffic data via our NetDiffusion framework. Our approach is structured around three primary components:

1. Converting raw network traffic in packet capture (pcap) format into image-based representations (§3.3.1).

2. Fine-tuning a Stable Diffusion model to enable controlled text-to-traffic generation with high distributional similarity across header fields to real-world network traffic (§3.3.2).

3. Domain knowledge-based post-processing heuristics for detailed modification of generated network traffic to ensure high level of protocol rule compliance (§3.3.3).

### 3.3.1 Network Traffic to Image Conversion

In this subsection, we explain the motivation and the process of representing network traffic as images, an important step in our method.

**Advantages of Using Image Representation of Network Traffic.** Network traffic data, with intricate inter-packet dependencies and vast range of attributes, presents a complex landscape that introduces specific challenges when it comes to accurate representation and efficient learning. Network traffic data exhibits *high dimensionality*, particularly when using standardized representations such as nPrint [68]. For instance, between the IP and TCP headers alone, there is an abundance of fields (*e.g.,* , IP addresses, ports, sequence and acknowledgment numbers, flags, etc.). nPrint uses a bit-level and standardized representation to have a consistent format for each packet by accounting for all potential header fields (even if not present in the original packet). For instance, while a TCP packet won't have UDP header bits, the nPrint still includes placeholders for these bits. While this ensures a uniform input structure for ML models, the attribute count per packet often exceeds a thousand. This high dimensionality introduces computational bottlenecks for generative models.

Additionally, each network traffic trace, considered as a single network flow/session, inherently contains *sequential dependencies* between packets. For example, in TCP, packets need to follow a particular sequence to ensure the integrity and reliability of data transmission. The order of packets, dictated by sequence and acknowledgment numbers, is crucial to reconstruct the transmitted data accurately at the receiver's end. These dependencies are also beneficial to improve ML classification accuracy as they may be unique to different classes of network traffic. Traditional tabular formats fall short in preserving these sequential relations due to their static nature, which could lead to the misrepresentation of the underlying network behavior [176, 175, 42].

Since recent strides in synthetic data generation have centered around image generation [167, 27, 123, 117, 40], we seek to leverage these methods to generate high-fidelity synthetic network data with low computational complexity. Our reasons for adapting these models are: (1) *Maturity of Image Generative Models*: The advancements in the domain of image generative models, such as diffusion models, offer a robust foundation to produce de-

tailed synthetic network traffic. These models have been optimized over years to understand and reproduce intricate patterns in high-resolution images [123, 117, 40]. (2) *Spatial Hierarchies and Connectivity*: Images inherently capture spatial hierarchies, which is crucial for representing intricate inter-packet and intra-packet dependencies in network traffic. Pixels in images naturally form patterns and structures. Deep learning models, especially convolutional neural networks (CNNs), are adept at exploiting these structures to capture both local and global dependencies. Unlike traditional tabular formats where data points might be perceived as independent entities, images inherently emphasize the significance of a packet concerning its neighboring packets, preserving crucial contextual information [133, 30, 86, 98]. A straightforward example demonstrating the effectiveness of appropriate image representations in capturing network dependencies is the use of edge detection in discerning packet protocol distributions within a flow, detailed later in Section 3.3.2. (3) *Visualization and Interpretability*: Image representations offer a intuitive way to discern packet flows, anomalies, and patterns in network traffic. (4) *Research and Tools Availability*: The extensive research and tools available in computer vision mean that scalability and optimization are already mature, providing a significant advantage when handling high-dimensional data like network traffic [177, 123, 173].

**Conversion Process.** To arrive at image representations of network traffic, we first encode packet captures (pcaps) using nPrint [68], which converts network traffic into standardized bits where each bit corresponds to a packet header field bit as shown in Figure 3.1. This binary representation is simple yet effective, where the presence or absence of a bit in the packet header is denoted as 1 or 0 respectively, and a missing header bit is represented as -1. This encoding scheme ensures a standardized representation irrespective of the protocol in use. The payload content is not encoded since it is often encrypted. However, the size of the packet payloads can be inferred from other encoded header fields such as the IP Total Length fields.

Following this encoding, a sequence of packets in a pcap is converted into a matrix,

which is then interpeted as an image. The colors green, red, and gray represent a set bit (1), an unset bit (0), and a vacant bit (-1), respectively. This color coding provides a visually intuitive representation of the network traffic. We then group the packets in groups of 1024, representing the packet headers of the first 1024 packets in a flow. Through this process, any network traffic in pcap format is transformed into an image with dimensions of 1088 pixels in width and 1024 pixels in height, with each row of pixels representing a packet in the network traffic flow as shown in Figure 3.2. Any image in this format can be converted back to pcaps in a straightforward manner. This representation not only preserves the complexity of the data but also retains the essential sequential relationships among packets, laying a robust foundation for the ensuing steps in the NetDiffusion pipeline. We opt for a grouping of 1024 packets, aligning with conventional practices in training and fine-tuning text-to-image diffusion models, where image resolutions are commonly capped at 1024 by 1024 pixels. This choice is primarily driven by computational constraints encountered at higher data resolutions. Nonetheless, this aspect presents an opportunity for future exploration, as we discuss in Section 3.5, where the potential to handle larger packet groups could be considered.

### 3.3.2 Fine-Tuning Diffusion Model and Controlled Generation

Given a real, labeled network traffic dataset, we first transform the network traffic flows into their corresponding image representations as previously described. Leveraging these image-based representations, we then fine-tune a generative model, specifically a diffusion model, to produce synthetic network traffic.

**Advantages of Text-to-Image Diffusion Models for Network Traffic Generation.** The decision to employ diffusion models for image-based network traffic generation over other generative approaches, such as GANs, is anchored on several compelling advantages:

1. *High-Fidelity Generation:* Diffusion models excel in capturing and replicating intricate

data distributions with remarkable fidelity [66, 128, 112]. This attribute is pivotal, given the complex and nuanced patterns inherent in real network traffic. The ability of diffusion models to closely mimic these patterns ensures that the synthetic traces they produce have high resemblance to real traces.

2. *High-Resolution Image Handling:* Diffusion models, through techniques like latent diffusion, are adept at generating and managing high-resolution images [117, 123, 110]. This capability is pivotal for our framework, where the image representation of network traffic demands high resolution for accuracy and detail retention. While diffusion models can be tailed to handle tabular data directly, this may forgo the distinct benefits of image representations, such as capturing spatial and sequential intricacies, as previously discussed.

3. *Conditional Generation:* By allowing for conditional generation based on textual prompts, text-to-image diffusion models can be instructed to generate network traffic that mirrors specific classes or types, offering an unparalleled fusion of precision and versatility. The model learns the relationship between text and image during its training phase by adjusting its reverse diffusion trajectory based on the given textual prompt [125, 56, 168]. This ensures that the final generated image aligns with desired image distribution. This becomes invaluable when the need arises to produce specific classes of network traffic or to conform to protocol rules and other essential network characteristics.

4. *Transparency and Training Stability:* The nature of diffusion models ensures a transparent generation process, leading to reproducible results. Such transparency are critical for producing network traces that meet specific patterns or constraints, as it shows good interpretability. Moreover, unlike the often unpredictable training dynamics of GANs due to their adversarial nature [39, 11, 105], diffusion models exhibit stable training behavior and well-behaved gradient dynamics [145, 65, 143]. This stability

not only ensures consistent and anomaly-free output but also streamlines the optimization process.

In totality, these advantages make diffusion models a robust and versatile choice for the generation of synthetic network traces, effectively addressing the challenges and constraints observed with other generative techniques. At the same time, a key goal of our study is to promote the generation of highly granular network traffic, with our preference for diffusion models partly influenced by their swift recent advancements. Nevertheless, investigating alternative methods suited for generating long sequential time-series data, such as transformer-based architectures, could be a promising approach. While this is not the focus of this paper, we discuss potential future directions in Section 3.5.

**Fine-Tuning a Stable Diffusion Base Model Using LoRa.** Training generative models, especially those as sophisticated as diffusion models, from scratch can be resource-intensive and time-consuming. This is particularly true when considering existing base models like Stable Diffusion 1.5 [123] which have been pre-trained on the LAION-5B dataset [126] containing over 5.85B CLIP-filtered image-text pairs. While the out-of-the-box Stable Diffusion model is undeniably potent, we cannot directly use it to synthesize network traffic because it's designed to cover a broad spectrum of patterns and intricacies, causing it to lack the depth needed in specific generation tasks. For instance, generating images based on task-specific prompts might yield results that, although thematically aligned, lack the precision and high-fidelity one might expect. In our case, a 'Netflix Network Traffic' prompt might yield a generic image like a highway scene within a Netflix player. This is particularly evident when the textual description provided has multiple potential visual interpretations, causing the model to produce images that may be blurry or off-target. By fine-tuning Stable Diffusion on specific network datasets, we can address these limitations. Fine-tuning augments the model's expressiveness, enabling it to better associate with specific patterns or embeddings of the network traffic domain.

As a result, in this framework, we build upon Stable Diffusion 1.5 and fine-tune this model on our specific network datasets as shown in Figure 3.1, making it aptly suited for generating synthetic network traffic that mirrors the complexities and nuances of real-world network traffic. To facilitate this fine-tuning, we employ Low-Rank Adaptation (LoRa) [70], which is a training technique tailored to swiftly fine-tune diffusion models, particularly in text-to-image diffusion models. Its crux lies in enabling the diffusion model to learn new concepts or styles effectively, while maintaining a manageable model file size. This is beneficial given the traditionally large sizes of models like Stable Diffusion, which can be cumbersome for storage and deployment. With LoRa, the resultant models are compact, striking a balance between file size and training capability. This compactness doesn't sacrifice the model's ability but rather applies minute yet effective changes to the base/foundational model, ensuring that the core knowledge remains intact while adapting to new data.

In our fine-tuning process, we start by sampling classes of real network traffic from our dataset that we aim to generate synthetically. These traffic samples are then transformed into their image representations. For each of these images, we craft an unique encoded text prompt (*e.g.,* , 'pixelated network data, type-0' for Netflix traffic) that succinctly describes its class type. Consequently, this results in a number of text prompt categories corresponding to the variety of network traffic types within the dataset. For example, in the subsequent evaluation task (Section 3.4), we utilize a dataset comprising flows from 10 distinct applications to fine-tune our NetDiffusion pipeline. This results in 10 unique categories of text prompts, each corresponding to different flow applications, e.g., 'pixelated network data, type-5' for Zoom flows and 'pixelated network data, type-6' for Google Meet flows. The choice of our encoded prompt, though seemingly simplistic, achieves two main objectives. It offers a specific vocabulary that reduces ambiguity and ensures the model hones in on the network traffic's nuances. Additionally, it minimizes interference from the base model's original word embeddings, optimizing the generative process. Experimentally, we found that this specific prompt structure provides a balance between specificity and simplicity to prevent

overfitting and misinterpretations, leading to better results. Subsequently, these image-text pairs are fed into the fine-tuning process, where the base Stable Diffusion model, augmented with LoRa, learns to generate network traffic images conditioned on our prompts. By merging the power of Stable Diffusion models with the adaptability of LoRa, we create a potent mechanism to generate high-fidelity, synthetic network traffic images, tailored to our specific requirements.

**Controlled Prompt-based Generation Via ControlNet.** Upon fine-tuning the generation model, the next phase involves generating the desired class of synthetic network traffic. This is achieved by supplying the appropriate text prompts to the diffusion models to produce the image representations of the traffic. Diffusion models operate by simulating a reverse process from a simple noise distribution to the data distribution, which enables them to capture and replicate the intricate patterns inherent in real-world data. The noise is progressively reduced over several steps, allowing the model to gradually refine the generated image until it closely resembles genuine network traffic patterns. We show an example synthetic network traffic in image representation for Amazon traffic in Figure 3.2. This prompt-based generation process facilitates the creation of a synthetic nPrint-encoded network traffic dataset tailored to specific class distribution requirements. For instance, to curate a dataset with a certain class distribution and size, one would provide the corresponding quantity of text prompts per class and activate the generation process accordingly.

However, a challenge arises from the inherent flexibility of general diffusion models. While they are designed to foster creativity in the generated output, it can lead to anomalies in the context of network traffic generation. For example, generated traffic might incorrectly populate packet header fields, leading to protocol distribution discrepancies between synthetic and real traffic. Such deviations can compromise ML accuracy and make it arduous to ensure strict adherence to critical protocol rules as we describe later. To ensure that the generated traffic closely aligns with the prevalent protocol and header field value distributions observed in real traffic, certain constraints are introduced during the generation process. If,

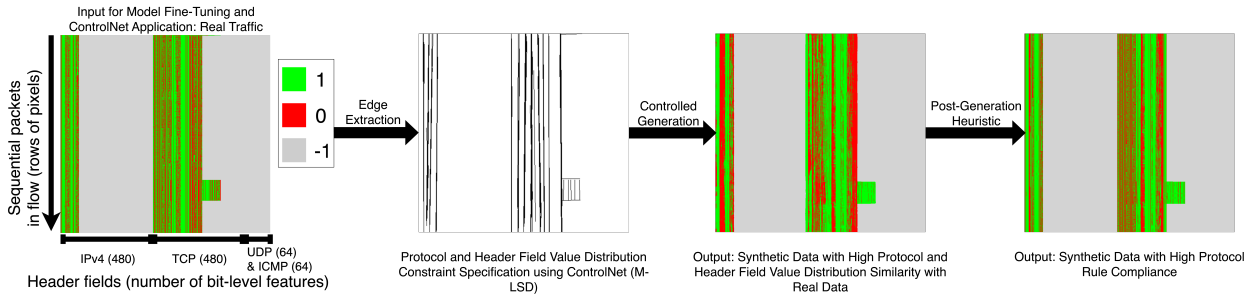



**Figure 3.2:** Synthetic Amazon network traffic outputs: (1) Using ControlNet, we detect regions present in the original traffic and ensure protocol and header field value distribution conformance by generating within specified regions. (2) We apply post-generation heuristic to refine field details for protocol conformance.

for instance, the actual Amazon network traffic primarily consists of TCP packets, the generation process should prioritize populating header fields associated with TCP packets. This approach ensures that the generated traffic image predominantly features green (set bit) or red (unset bit) pixels corresponding to TCP packet headers, while other pixels remain gray (vacant bit). Moreover, consistent bit characteristics within headers, such as consistently unset bits, should be mirrored in the synthetic output.

Leveraging the controllable nature of diffusion models, we incorporate ControlNet [170] into the generation process. ControlNet is a commonly used neural network architecture designed to add spatial conditioning controls to large, pre-trained text-to-image diffusion models. It capitalizes on the robust encoding layers of these models, which are pre-trained with vast datasets, to learn a diverse set of conditional controls. With "zero convolutions", the architecture gradually grows its parameters from an initialized state, ensuring no adverse noise affects the fine-tuning. ControlNet can work with a range of conditioning controls, from edges and depth to segmentation and human poses. It offers flexibility in training, demonstrating robustness with both small and extensive datasets. In our specific use case of ControlNet, we leverage M-LSD straight line detection for detection the boundaries between fields that are suppose to be populated and those that are not, as shown in Figure 3.2. Other edge detection methods such as Canny edge detection produce similar results. Such line or edge detection methods are effective because they align with the inherent columnar

consistency present in packet traces.

Incorporating ControlNet allows the synthetic generation process to more closely emulate the protocol and header field value distributions observed in real network traffic. This minimizes deviations and ensures that the generated packets largely adhere to the expected protocol type and header field values, enhancing the quality and reliability of the synthetic network traffic dataset. And while ControlNet offers coarse-grained control, determining which image regions to populate, the diffusion model provides fine-grained control, specifying individual pixel values which further contributes to high resemblance to real traffic. In Appendix A.1, we carry out detailed ablation studies showcasing the necessity of applying ControlNet during the generation process, as well as the generation improvement from fine-tuning a base text-to-image diffusion model. Despite the integration of ControlNet, the inherent variability and generative nature of diffusion models can lead to differences in generated instances. For a detailed analysis of these differences, refer to Appendix A.2. On the other hand, while our current approach utilizes pretrained ControlNet models as an initial step for automating constraint enforcement, they fall short in supporting more specific constraints, such enforcing and confining highly detailed constraints within smaller header field columns. Future efforts will focus on training ControlNet models dedicated to network traffic synthesis from scratch. This direction, as described later in Section 3.5, aims to achieve finer-grained control over the synthetic traffic generation process.

### 3.3.3 Improving Transport and Network Layer Protocol Compliance

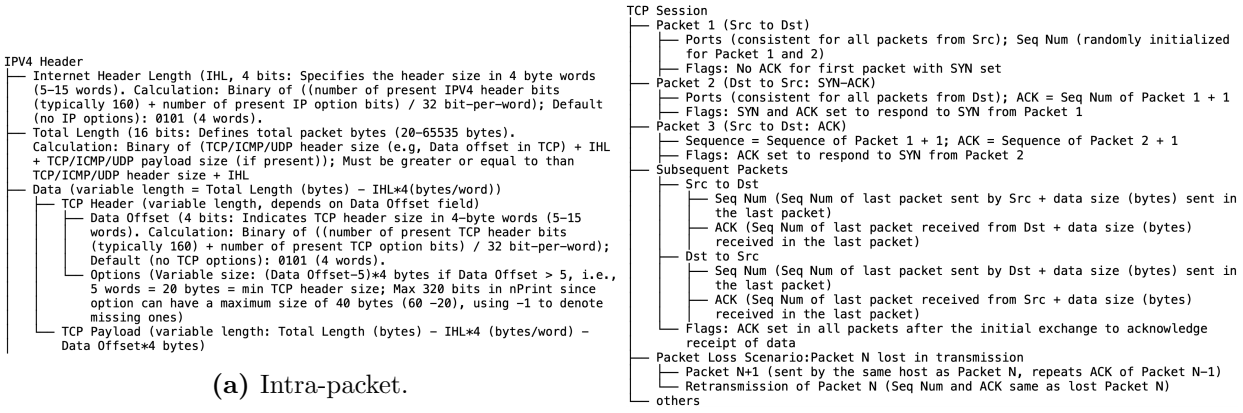
Utilizing the controlled diffusion model, we generate encoded network traffic that adeptly resembles the protocol and header field value distributions inherent in real-world data. Our encoded format not only captures every feature observed in real network traffic but also minimizes the statistical disparity between real and synthetic feature values, ensuring models can recognize and act on underlying patterns. Yet, the domain of network dataset augmentation presents unique challenges. While the synthetic data's quality in ML applications is

crucial, its utility extends beyond. The data's relevance in traditional network analysis and testing tasks – often requiring raw network traffic – becomes equally significant. Despite the guidance provided by ControlNet during the generation process, converting our synthetic encoded traffic back to raw formats, like pcaps, isn't straightforward. This complexity arises from the multitude of detailed transport and network layer protocol rules at both inter and intra-packet levels. Properly formatted traffic must strictly adhere to these rules. We now explore these complexities and their implications.

**Inter and Intra Packet Transport and Network Layer Protocol Rules.** At its core, both transport and network layer protocol rules define the conventions and constraints that ensure seamless communication between devices in a network. These rules are crucial as they dictate the structure, formatting, and sequencing of packets, ensuring that data transmission occurs efficiently and reliably. While transport layer rules emphasize end-to-end communication and reliability, network layer protocols focus on packet routing and address assignment. Combined, these rules can be broadly divided into two categories:

1. *Inter-Packet Rules:* These rules dictate the relationships and sequencing between header fields within multiple packets in a network flow. For instance, in a typical TCP connection, packets need to be sequenced properly, starting with the handshake process involving SYN and SYN-ACK flags. The integrity of data transfer is ensured by aligning sequence numbers and acknowledgment numbers. Misalignment or incorrect sequencing can disrupt the connection or data transfer process.

2. *Intra-Packet Rules:* These pertain to the structure and contents within individual packets. For example, many protocol headers have a checksum field computed based on the packet's contents to detect errors during transmission. It's crucial that the checksum is consistent with the packet's payload. Additionally, certain fields within a packet, such as port numbers in TCP and UDP headers, must adhere to specific formatting and value constraints to ensure the packet's validity and proper routing.




**(a)** Intra-packet.

**(b)** Inter-packet.

**Figure 3.3:** Example TCP protocol rules/dependencies.

Ensuring compliance with these rules is vital. Properly formatted traffic is not only more efficient but also crucial for network applications devices, such as analysis tools, routers, and firewalls, which rely on well-structured packets to function correctly. The challenge with synthetic data generation, especially when optimized for ML accuracy, is that ML models primarily focus on patterns within features that contribute to classification or prediction accuracy. These models might overlook intricate protocol rules in favor of patterns that enhance classification performance. For example, a ML model might deem certain bit patterns as significant for classifying a particular type of traffic, even if those patterns violate protocol rules. While our use of ControlNet aids in approximating the general protocol and header field value distributions by ensuring correct field population, it does not fully capture the nuances of specific bit-level values. This disparity between ML optimization and protocol rule compliance accentuates the necessity for post-generation adjustments. Instead of entirely overhauling the ML generation process, which would require embedding a vast amount of rule-bound constraints derived from domain knowledge, post-generation adjustments offer a more manageable approach to refine the generated data for protocol compliance. Implementing such detailed control during generation remains a challenging endeavor, and we envision addressing these complexities in future work.

**Post-Processing Heuristic for Protocol Rule Compliance.** To maximize the encoded synthetic network traffic's compliance with transport and network layer protocol rules, we first discern a subset of critical header fields that mandate strict adherence to their formatting rules, *e.g.,* , sequence and acknowledgement numbers. In contrast, some fields can accommodate a degree of flexibility without compromising the integrity of the network traffic, such as TCP window size or TTL. The objective here is to limit the scope of fields requiring modification during post-processing, ensuring we retain as much of the original generative model's output as possible. By doing so, we minimize potential impacts on ML-driven tasks, while still ensuring the synthetic traffic's compatibility with network analysis tools. With the critical fields identified, we develop a systematic way to calculate their correct values based on other generated fields. This is achieved by constructing two dependency trees—one for intra-packet header field dependencies and another for inter-packet dependencies. These trees are built upon domain knowledge and are sourced from standard network protocol documentation [1, 3, 2, 4, 5, 17]. Although constructing them requires significant manual effort to extract critical protocol rules from these standards, this process is a one-time endeavor. Future applications and iterations of our tool will benefit from this foundational work without the need for repeated effort. We present example protocol rules and the associated dependency trees for TCP protocol in Figure 3.3. More comprehensive and detailed dependency trees can be found in the open-sourced repository[1].

Given a generated encoded network traffic, we begin the correction process by traversing the trees in an automated, bottom-up fashion. Initially, we satisfy intra-packet dependencies, ensuring that individual packets are internally consistent. Subsequently, we address inter-packet dependencies, guaranteeing that the packets in a flow relate correctly to one another. Certain fields necessitate uniformity across packets within the same network traffic trace—like IP addresses and ports. Others require specific initialization values, such as the IP identification number and TCP acknowledgment number. To determine the most

[1] https://anonymous.4open.science/r/packet-capture-dependency-DB0C/README.md

**Algorithm 1** Post-Processing Heuristic Example: Intra-packet Dependency Adjustments for IP headers

**Input:** Diffusion model generated synthetic traffic
Analyze and sort source IP address distribution ($P_{src}$) from a radomly sampled real flow ▷ Ensure that packet directions in the synthetic flow closely align with those in real flows.
Identify the top two IPs ($IP_1$, $IP_2$) with the highest occurrence from $P_{src}$ ▷ Select the most frequent source IP addresses as primary nodes for transition probability modeling.
Construct synthetic IP address sequence ($IP_{sequence}$) using transition probabilities $T_{IP_1 \to IP_2}$ and $T_{IP_2 \to IP_1}$ ▷ Use transition probabilities to model the likelihood of switching between $IP_1$ and $IP_2$ in the synthetic sequence, reflecting packet flow dynamics in real traffic.
Correct packet directions within synthetic flow using $IP_{sequence}$
**if** version_field $\neq$ 'IPv4 (0100)' **then** set to 'IPv4 (0100)' ▷ Ensure Version field adheres to IPv4 standards for compatibility.
**for** bit in IPv4 headers: **if** bit = -1 **then** set bit to random (0,1) ▷ Replace undefined bits in IPv4 headers with random binary values to maintain structural integrity.
Identify IPv4 protocol $\mathcal{P}$ in the synthetic flow by maximizing $\frac{\sum \text{non-negative bits in } \mathcal{P}}{\text{total bits in } \mathcal{P}}$ ▷ Optimize protocol identification by maximizing the ratio of valid to total bits in IPv4 protocol.
**for** bit in non-IPv4 headers: **if** *bit* $\notin \mathcal{P}$ **then** set bit to -1 ▷ Discard irrelevant bits in non-IPv4 headers to focus on IPv4 protocol consistency.
**for** bit in IPv4 options: **if** bit $\neq$ -1 **then** set bit to -1 ▷ Render IPv4 options bits obsolete to reflect modern Internet protocols.
**if** IPv4 TTL = 0 **then** set TTL using majority voting across all packets ▷ Minimum IPv4 TTL guarantee to prevent packet expiration.
Set HL field to number of active bits in the IPv4 fields ▷ Adjust Header Length (HL) field to accurately represent the active bits in IPv4 headers.
**Output:** Post-processed synthetic traffic with IP header intra-packet dependency compliance

appropriate values for these fields, we employ a majority voting system by selecting the most frequently appearing value within the generated traffic. Another notable challenge is timestamp assignment for individual packets, given its intricate time-series dependencies. Diffusion models, while adept at spatial dependencies, might struggle with long-range temporal patterns inherent in time-series. As a result, our current generation process isn't fully optimized for this. As an interim solution, we sample original time distributions from real traffic to produce similar timestamp distribution in the post-generated synthetic data. An example post-processing algorithm for realizing the intra-packet dependency adjustments for IP headers is in Algorithm 1. We recognize the potential for improvement and plan to address this in future research. Post these steps, the synthetic traffic should be in a state where it closely adheres to the essential protocol rules we've identified. This post-processing ensures that the encoded synthetic traffic can be seamlessly converted into raw network traffic formats (like pcap) and subsequently be utilized for a range of non-ML tasks.

## 3.4 Evaluation

To assess the effectiveness of our generative framework, we applied it to an exemplary real network traffic dataset, generating its synthetic counterpart as a case study. Our ML-oriented evaluation comprises two main analyses: a statistical comparison to gauge the fidelity of the synthetic data and a model accuracy assessment to determine its utility in enhancing ML outcomes. In the non-ML scenarios, we explore the broader implications of our synthetic data by applying it in diverse network analysis and testing scenarios. The choice of our baseline dataset, which we detail next, serves as a demonstration of our method's validity.

### 3.4.1 Dataset Overview and Synthetic Traffic Generation

Our dataset for real network traffic, summarized in Table 3.2, comprises pcap files that capture traffic from ten prominent applications in areas such as video streaming, video conferencing, and social media. Although we focus our study on a single dataset, it comprises varied network data collected from multiple scenarios, locations, and times from three different data sources [23, 99, 75], underpinning our methodology's applicability and generalizability. During preprocessing, we analyze DNS queries to identify relevant IP addresses for the specified services and applications, keep only packets associated with these IPs, and split the traffic into individual flows. We retain the application and service label for each processed flow, which is later used for generating text prompts and assessing classification accuracy[2]. The comprehensive dataset contains nearly 20,000 flows. For feasibility and consistency in our evaluations, we randomly sampled 10% of this collection We

| **Macro Services** | **Total Flows** | **Application Labels** | **Collection Date** |
|---|---|---|---|
| Video Streaming [23] | 9465 | Netflix<br>YouTube<br>Amazon<br>Twitch | 2018-06-01 |
| Video Conferencing [99] | 6511 | MS Teams<br>Google Meet<br>Zoom | 2020-05-05 |
| Social Media [75] | 3610 | Facebook<br>Twitter<br>Instagram | 2022-02-08 |

**Table 3.2:** Summary of the real network traffic dataset: 10 applications across these three macro service types.

[2]All traces used in this paper are sanitized and contain no personally identifiable information (PII).

also carried out evaluations on both larger and smaller subsets of the dataset and obtained comparable results. We adapted the fined-tuned diffusion model to this dataset, resulting in the generation of a synthetic dataset as outlined in the previous section. The volume of this synthetic dataset adjusts based on specific evaluation requirements, as we will detail further. The prompt-driven nature of diffusion model allows for the generation of synthetic network traffic in any desired quantity, providing flexibility for diverse analytical needs.

### 3.4.2 Statistical and ML Performance Analysis

**Statistical Similarity Results.** A primary measure of synthetic data quality is its statistical resemblance to the original data. This comparison is critical as the essence of synthetic data lies in its ability to represent the statistical properties of the real data without mirroring it exactly. Ensuring statistical similarity ensures that models trained on synthetic data generalize well to real-world scenarios. In our evaluation, we benchmarked our synthetic data against two baselines: the NetShare method, which produces synthetic NetFlow attributes and outperforms most of the othe GANs-based methods [165], and a naive random generation approach. The latter, by generating purely random values, acts as a worst-case scenario, illustrating the lower bounds of similarity and underscoring the value added by more sophisticated methods. While our diffusion model inherently captures a broader set of network attributes, for fairness in comparison, we examine similarity both at an aggregated level, encompassing all features, and at a more focused level, targeting only the common features between NetDiffusion and NetShare – using the Protocol attribute as an example.

We employ three distinct metrics to quantify statistical similarity: Jensen-Shannon Divergence (JSD), Total Variation Distance (TVD), and Hellinger Distance (HD). JSD gauges informational overlap between distributions, offering insights into shared patterns. TVD captures the maximum difference between two distributions, highlighting worst-case discrepancies. Meanwhile, HD, rooted in Euclidean distance, is especially sensitive to differences in the tails of distributions, shedding light on distinctions in rare events or outliers.

| Data Format | Generation Method | All Generated Features | | | Ex. Common Feature: Protocol | | |
|---|---|---|---|---|---|---|---|
| | | Avg. JSD | Avg. TVD | Avg. HD | Avg. JSD | Avg. TVD | Avg. HD |
| NetFlow | Random Generation | 0.67 | 0.80 | 0.76 | 0.82 | 0.99 | 0.95 |
| | NetShare | 0.16 | 0.16 | 0.18 | 0.04 | 0.03 | 0.04 |
| Network Traffic (pcap) | Random Generation | 0.82 | 0.99 | 0.95 | 0.83 | 1.00 | 1.00 |
| | **NetDiffusion** | **0.04** | **0.04** | **0.05** | **0.02** | **0.03** | **0.02** |

**Table 3.3:** Average normalized statistical differences between real and synthetic network data across (1) all generated fields and (2) example commonly generated field – IPv4 protocol: Jensen-Shannon Divergence (JSD), Total Variation Distance (TVD), and Hellinger Distance (HD).

Collectively, these metrics provide a holistic view of the statistical overlap between the real and synthetic datasets. Values for all three metrics range between 0 and 1, with values closer to 0 indicating superior statistical similarity and thus a closer resemblance to the original dataset.

The results in Table 3.3 offer a nuanced view into the challenges and successes of synthetic data generation for network traffic. At a foundational level, raw network traffic in pcap format is inherently more intricate than the NetFlow format. This complexity is evident in the stark contrast in statistical distances when generating synthetic data for these two formats using random methods. The higher statistical distance observed for the randomly generated raw network traffic underscores the inherent challenges in replicating its multifaceted nature. Yet, it's this very complexity that highlights the prowess of NetDiffusion. Despite pcap being a more challenging format, NetDiffusion—specialized in generating raw network traffic—demonstrates a remarkable capability. Its statistical distances relative to real network traffic are notably lower than even NetShare's distances when NetShare is generating for the simpler NetFlow attributes. This finding underscores the efficacy of NetDiffusion in producing high-fidelity synthetic data for intricate formats like pcap.

In summary, while the inherent challenges in replicating the complex pcap format are evident, NetDiffusion's capability to produce synthetic data with high statistical similarity, even surpassing methods for simpler formats, validates its potential as a robust tool for data augmentation in the realm of raw network traffic.

**ML Classification Results.** To gauge the efficacy of our synthetic network traffic in ML-based data augmentation, we employ two classification tasks. The first task aims to categorize network flows at a granular level, aligning them with their corresponding applications (micro-level). The second task operates at a broader scale, classifying flows into their overarching services (macro-level). We conduct evaluation using three prominent models, including random forest (RF), decision tree (DT), and support vector machine (SVM). We adopt accuracy as the performance metric for these ML-driven augmentation evaluations due to its intuitive interpretability, allowing for a straightforward comparison between different augmentation techniques. Specifically, in scenarios involving classification tasks where classes are (or are made to be) approximately balanced, accuracy provides a clear picture of how well the model performs across all classes. Three distinct augmentation scenarios, utilizing synthetic data, are evaluated:

- *Complete Synthetic Data Usage:* Here, either the training or testing set is entirely composed of synthetic data, *e.g.,* , training exclusively on synthetic data and testing on real data, and vice versa. This approach tests the robustness of synthetic data and its capacity to emulate real-world data intricacies. Using synthetic data in isolation ensures that models are not biased by any inherent patterns of the real dataset during training, allowing for an assessment of the synthetic data's standalone quality.

- *Mixed Data Proportions:* Synthetic data is interspersed with real data at varying proportions, *e.g.,* , a 50-50 split between synthetic and real data during training. This strategy evaluates the synergy between real and synthetic data. Mixing allows models to benefit from the diversity of synthetic data while still grounding the learning process in real-world patterns, potentially improving generalization.

- *Class Imbalance Rectification:* Synthetic data is employed specifically to address and rectify class imbalances in the training set. For instance, underrepresented classes in the real dataset are augmented using synthetic data until a balance is achieved. This

| Training/Testing | Data Format (Generation Method) | Post-Gen. Heuristic | Highest Accuracy (Model) Macro-level | Micro-level |
|---|---|---|---|---|
| Real/Real | Network traffic (N/A) | N/A | 1.00 (SVM) | 0.978 (SVM) |
| | Netflow (N/A) | N/A | 0.86 (RF) | 0.648 (DT) |
| Synthetic/Real | **Network traffic (NetDiffusion)** | **Not Used** | **0.738 (DT)** | **0.262 (DT)** |
| | | **Used** | **0.676(DT)** | **0.222 (DT)** |
| | Netflow (NetShare) | N/A | 0.396 (RF) | 0.140 (SVM) |
| Real/Synthetic | **Network traffic (NetDiffusion)** | **Not Used** | **0.542 (SVM)** | **0.249 (SVM)** |
| | | **Used** | **0.529 (SVM)** | **0.182 (SVM)** |
| | Netflow (NetShare) | N/A | 0.503(SVM) | 0.102 (RF) |

**Table 3.4:** Across all complete synthetic data usage scenarios, Net-Diffusion augmented dataset yields to higher classification accuracy. Only the top-performing model is displayed.

| Rank | Real/Real | Synthetic/Real |
|---|---|---|
| 1 | 3_tcp_wsize_13: 0.0115 | 1_tcp_opt_92: 0.0125 |
| 2 | 3_tcp_wsize_15: 0.0107 | 2_udp_cksum_14: 0.0098 |
| 3 | 1_udp_len_5: 0.0100 | 8_tcp_opt_78: 0.0093 |
| 4 | 1_tcp_opt_24: 0.0099 | 4_tcp_opt_93: 0.0085 |
| 5 | 1_ipv4_tl_5: 0.0090 | 9_udp_cksum_14: 0.0085 |
| 6 | 0_tcp_opt_6: 0.0088 | 5_tcp_opt_93: 0.0084 |
| 7 | 0_ipv4_dfbit_0: 0.0088 | 5_tcp_urp_8: 0.0082 |
| 8 | 1_tcp_opt_6: 0.0086 | 9_tcp_cksum_1: 0.0081 |
| 9 | 9_ipv4_proto_3: 0.0085 | 6_tcp_opt_78: 0.0079 |
| 10 | 1_tcp_opt_23: 0.0076 | 2_tcp_opt_93: 0.0079 |

**Table 3.5:** Macro-level RF feature importance for complete NetDiffusion synthetic data usage; Green highlights denote shared header fields with the real/real scenario. Feature structure: packet_protocol_header_bit.

targeted augmentation ensures that the model is exposed to a balanced representation of all classes, mitigating biases and improving performance on minority classes. Addressing class imbalance is crucial as it prevents models from becoming skewed towards overrepresented classes, thereby enhancing their predictive accuracy across all classes.

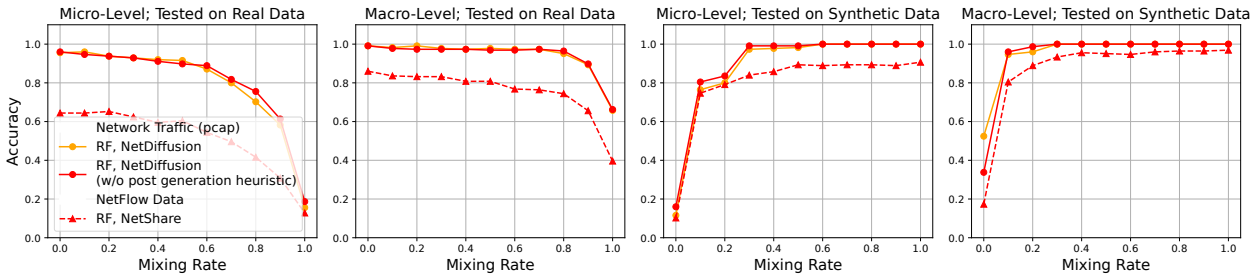
**Result on Complete Synthetic Data Usage.** The results presented in Table 3.4 reveal that our method, which specializes in generating raw network traffic data in pcap format, consistently outperforms the NetShare approach, which is tailored for the simpler NetFlow data format. Note that the classification accuracy tends to be higher for the macro-level task in all train/test scenarios. This is anticipated because the macro-level task categorizes broad traffic service types (like video streaming vs. video conferencing), while the micro-level task involves finer distinctions between specific applications (e.g., YouTube vs. Twitch). The increased specificity of the micro-level task generally yields lower accuracy compared to the broader macro-level classification.

The rich feature space inherent in raw network traffic offers a plethora of learnable patterns that can bolster model accuracy. With a broader and more intricate feature set, there's more room for the model to identify and leverage intricate patterns, nuances, and correlations within the synthetic network traffic data to enhance its predictive prowess. At the same time, the higher statistical similarity between real and our synthetic datasets (as

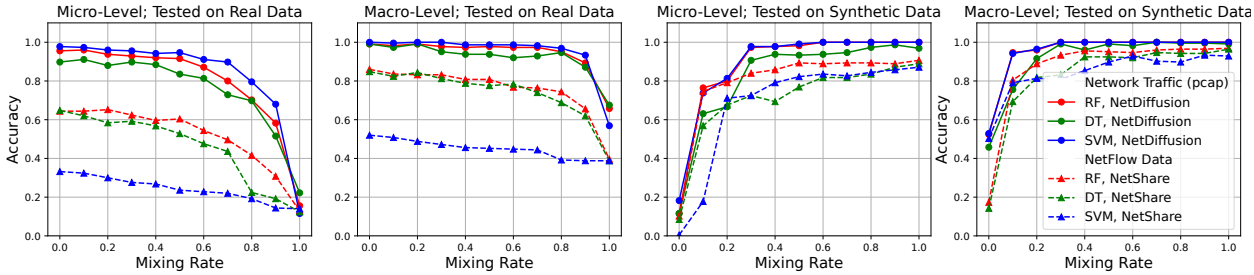
we observed previously) implies that the synthetic network traffic data mirrors real-world patterns more closely. This, in turn, means that features in the real dataset that are pivotal in distinguishing between network flows are likely to retain their discriminative power in the synthetic dataset. Such preservation of feature significance ensures that models trained on synthetic data can generalize more effectively to real-world scenarios. Supporting this is our feature importance analysis for the RF model in the macro-level classification task as shown in Table 3.5. When trained on NetDiffusion synthetic data and tested on real data, the RF model exhibited a propensity to prioritize features (on the header level) that are also critical when both training and testing are done on real data. This nuanced focus on specific feature subsets is indicative of the model's ability to discern and leverage patterns in the synthetic data that are reflective of real-world traffic. While we use the RF model for subsequent detailed analysis due to its relatively consistent performance and interpretability across scenarios, our later sections will delve into a broader spectrum of model performances.

An additional layer to our analysis pertains to the post-generation heuristic for enhancing the synthetic network traffic's adherence to protocol rules while minimally altering the diffusion-generated outputs. The heuristic affects only $\sim$8% of the synthetic traffic features which produces marginal influence on ML performance, with accuracy reductions ranging from 0.013 to 0.067 across all scenarios. This minor accuracy degradation aligns with expectations: Diffusion models are adept at achieving high statistical similarity to real data during fitting and generation, enhancing ML accuracy. However, the necessary post-generation heuristic adjustments for protocol compliance inevitably alter the output. Although we strive to minimize this interference, it does slightly impact accuracy, as it deviates from the model's original generation output.

**Result on Mixed Data Proportions.** We introduce the "mixing rate" to denote the percentage of real data replaced by synthetic data in the training set. This approach ensures the training set size remains constant across diverse mixing rates, enabling a clear evaluation of the interplay between the mixing rate and the resultant model accuracy. Introducing a




**(a)** Classification accuracy comparison using the RF model with mixed data proportions. Datasets augmented with NetDiffusion-generated traffic consistently outperform those using NetShare-produced NetFlow attributes.




**(b)** Comparative ML performance across different model choices using NetDiffusion-augmented datasets versus NetShare-augmented NetFlow datasets. NetDiffusion consistently yields superior results.

**Figure 3.4:** Evaluation result on mixed data proportions.

controlled blend of synthetic data into real datasets can often enhance model robustness by potentially introducing diverse patterns, a practice that is considered standard in data augmentation.

Using the RF model as an example, Figure 3.4a shows that models trained with dataset augmented with NetDiffusion-generated traffic consistently achieve higher classification accuracy than those with NetShare-produced NetFlow attributes. When testing on real data, models trained entirely on real network traffic demonstrate notably higher accuracy than those trained solely on real NetFlow (also highlighted in Table 3.4). This starting accuracy discrepancy is critical. When integrating synthetic network traffic data into training, any potential degradation in accuracy is offset by the inherently higher baseline accuracy of real network traffic. In simpler terms, with a more accurate starting point (real network traffic), there exists more "buffer" before accuracy noticeably degrades. Another pivotal factor is the higher statistical similarity of NetDiffusion-generated traffic to real data, compared to

the similarity of NetShare's NetFlow data to real NetFlow. With this closer resemblance, as we incorporate more synthetic data into the training, the gradual decline in accuracy is less pronounced than when using NetFlow data, especially in macro-level classification. This advantage is not confined to testing on real data. Even when evaluating on synthetic data, models trained with NetDiffusion's output generally outperform those trained with NetShare. Additionally, we carry out a similar analysis as in the case of complete synthetic data usage by examing the effects of applying our post-generation heuristic on the model accuracy, as depicted in Figure 3.4a. Consistent with the previous findings from Table 3.4, the example RF model experiences little to no accuracy degredation as a result of the post-generation modification.

A notable observation is the sharp decline in accuracy when the data composition of the training set diverges significantly from the test set. For instance, macro-level classification accuracy drops when the mixing rate exceeds 0.8. This is expected since adequate samples from the test data distribution are needed in the training set for effective cross-validation. As the mixing rate increases, models might overfit to the synthetic data, hindering their performance on real data. This behavior is evident in the changing feature importance with increasing mixing rates, as seen in Table 3.7. In practical scenarios, it's rare to rely heavily or solely on synthetic data for training. Our results suggest that, barring extremes, NetDiffusion-generated traffic can be effectively used for training.

Lastly, we show that across different model choices, as shown in Figure 3.4b, NetDiffusion-augmented datasets generally lead to better ML performance than NetShare-augmented NetFlow datasets. Notably, the SVM classifier demonstrates markedly superior performance when tasked with classifying raw network traffic as opposed to NetFlow traffic. SVMs are intrinsically adept at handling datasets with high dimensionality and complex relationships between features. The reason lies in SVM's ability to transform the original data into a higher-dimensional space and find optimal hyperplanes to segregate different classes. The richer and more intricate the feature space, the more advantageous this capability becomes.

| | Mean ($\mu$) | Med | Min | Max | Std Dev ($\sigma$) | Range | Var ($\sigma^2$) |
|---|---|---|---|---|---|---|---|
| **Before Balancing** | 10.00% | 8.89% | 4.44% | 17.78% | 5.13% | 13.34% | 26.37% |
| **After Balancing** | 10.00% | 10.00% | 10.00% | 10.00% | 0.00% | 0.00% | 0.00% |

**Table 3.6:** Synthetic balancing on under-represented classes to mitigate class imbalance.

This observation accentuates the importance of NetDiffusion in generating synthetic network traffic, which retains the intricacies of real traffic, allowing sophisticated classifiers like SVM to effectively discern patterns and relationships.

**Result on Class Imbalance Rectification.** Class imbalance is an ubiquitous challenge in many datasets used for training. For instance, in our collected network trace, the least represented application class constituted a mere 4.44% of the total flow samples, while the dominant class accounted for 17.78%, as shown in Table 3.6. Such imbalances can negatively skew model performance, as models trained on imbalanced data may struggle to correctly classify underrepresented classes, especially when real-world test data exhibits a more balanced distribution. To combat this, a plausible approach is to selectively augment the training set by appending synthetic data to minority classes, ensuring an even class distribution. Simultaneously, by limiting the addition of synthetic data to well-represented classes, we minimize the drawbacks associated with integrating synthetic data, such as the risk of accuracy degradation from overly introduced variations and insufficient amount of real data in the training set, as we observed in the case of mixed data proportions. Using synthetic data offers advantages over traditional methods like SMOTE [28], random oversampling [16], ADASYN [61], and boosting [51]. It produces diverse and novel examples, enriching the feature space and bolstering model generalization to unseen real-world scenarios. While techniques like SMOTE replicate close counterparts of real data, they might miss certain variations.

We apply synthetic balancing using NetDiffusion-generated network traffic by iteratively generating instances of the underrepresnted classes until all classes have equal representation, resulting in a balanced network traffic dataset across all applications. Similarly,

| Rank | Mixing Rate (Accuracy) | | |
|---|---|---|---|
| | **0.0 (0.960)** | **0.4 (0.928)** | **1.0 (0.187)** |
| 1 | 5_ipv4_ttl_0: 0.0081 | 1_ipv4_ttl_3: 0.0053 | 5_ipv4_ttl_3: 0.0055 |
| 2 | 4_tcp_wsize_10: 0.0061 | 4_ipv4_ttl_3: 0.0049 | 8_tcp_cksum_14: 0.0054 |
| 3 | 5_tcp_wsize_4: 0.0061 | 1_ipv4_ttl_0: 0.0047 | 7_ipv4_ttl_4: 0.0052 |
| 4 | 1_ipv4_ttl_0: 0.0061 | 5_ipv4_ttl_3: 0.0046 | 4_ipv4_ttl_5: 0.0052 |
| 5 | 1_ipv4_dfbit_0: 0.0059 | 4_ipv4_ttl_0: 0.0045 | 3_ipv4_tl_12: 0.0047 |
| 6 | 4_ipv4_ttl_0: 0.0059 | 7_ipv4_tl_14: 0.0044 | 1_udp_cksum_4: 0.0044 |
| 7 | 4_ipv4_ttl_3: 0.0055 | 1_ipv4_dfbit_0: 0.0040 | 6_ipv4_tos_4: 0.0042 |
| 8 | 4_ipv4_ttl_1: 0.0054 | 4_tcp_wsize_10: 0.0039 | 2_ipv4_ttl_3: 0.0041 |
| 9 | 5_ipv4_ttl_1: 0.0053 | 5_ipv4_ttl_1: 0.0038 | 6_ipv4_ttl_3: 0.0040 |
| 10 | 1_tcp_opt_54: 0.0053 | 5_ipv4_ttl_0: 0.0036 | 8_tcp_wsize_13: 0.0040 |

**Table 3.7:** Feature importance at varying mixing rates for micro-level classification on real network traffic data. Red highlights indicate header fields that are not in the top 10 most important features in the real/real scenario (mixing rate = 0).

| Training Data (Balancing Source) | Model | Test Accuracy Pre/Post Balancing | Δ Acc. |
|---|---|---|---|
| Network Traffic (NetDiffusion) | **RF** | **0.982 → 0.986** | **0.004 Facebook (0.955 → 1.00)** |
| | **DT** | **0.973 → 0.982** | **0.009 Meet (0.909 → 1.00) Zoom (0.955 → 1.00)** |
| | SVM | 0.991 → 0.991 | 0 |
| Netflow Data (NetShare) | RF | 0.645 → 0.628 | -0.017 |
| | DT | 0.603 → 0.600 | -0.003 |
| | SVM | 0.290 → 0.290 | 0 |

**Table 3.8:** Comparison of micro-level classification accuracy: Class balancing using NetDiffusion synthetic data contributes to accuracy improvement on minority classes.

the NetFlow dataset is balanced using synthetic attributes from NetShare. However, due to the limitation of GAN-based methods in prompt-invoked generation, we address underrepresentation by independently training a NetShare GAN for each specific class. Our evaluation reveals that models trained on the balanced NetDiffusion dataset either match or outperform those trained on the original imbalanced dataset as shown in Table 3.8. Notably, the accuracy gains were predominantly attributed to the improved performance on the previously underrepresented classes. For instance, with the DT model, a notable 0.09 increase in overall classification accuracy is observed using the NetDiffusion augmented dataset. A breakdown of this improvement pinpoints Meet and Zoom traffic, two underrepresented classes with sample counts roughly *half or less than* the most populated class in the original real dataset, as the primary beneficiaries. Their classification accuracies improved by 0.091 and 0.045, respectively. In contrast, classifiers trained using the NetShare augmented NetFlow dataset do not yield such gains and occasionally even faced accuracy degredations. This further underscores the higher fidelity of NetDiffusion-generated traffic, which not only mirrors real data more closely but also supports larger feature space to enhance model performance.

### 3.4.3 Extendability to Additional Network Anlysis and Testing Tasks

The efficacy of synthetic data augmentation extends beyond just ML performance, especially within the networking realm. While ML-centric tasks may primarily focus on ensuring that

generated, encoded network traffic produces a consistent feature set with high statistical similarity to real traffic, conventional network analysis and testing tasks demand the conversion of this generated data back into raw formats, such as packet captures. Moreover, these tasks require adherence to specific protocol rules, as elaborated in Section 3.3.3. By harnessing the capabilities of ControlNet and the post-generation heuristics we employ, NetDiffusion facilitates the generation of network traffic that can be seamlessly converted into raw packet captures while maintaining a robust adherence to protocol rules. To illustrate this, we use the synthetic Amazon network traffic generated by NetDiffusion as an example and show that (1) the generated traffic can be smoothly parsed and interpreted by Wireshark, a renowned network analysis tool, without encountering exceptions and (2) the synthetic traffic supports retransmission, as corroborated using the established packet retransmission tool, tcpreplay. Beyond these observations, we demonstrate that NetDiffusion-generated traffic can successfully support a broad spectrum of common network tasks, ranging from intricate traffic analyses to network behavior studies. Crucially, the derived features routinely employed in these tasks can be extracted from NetDiffusion's outputs using Scapy [122]. This underscores the versatility of our approach, suggesting that NetDiffusion's synthetic network traffic can integrate into a multitude of network analysis and testing tasks beyond the confines of ML-centric applications.

We abstain from juxtaposing NetDiffusion's capabilities with NetShare in this section. The rationale is simple: NetShare's design fundamentally restricts it to generating a limited set of statistical attributes, which inherently curtails its ability to be converted into raw packet captures, a prerequisite for the tasks discussed here.

**Wireshark Parsing Analysis.** Table 3.9 details the results from Wireshark's parsing of the NetDiffusion synthetic traffic, stored as capsinfo log [17]. Several observations can be drawn: (1) *Data Format and Integrity:* The generated traffic is stored in the standard pcap format with Raw IP encapsulation. This confirms the synthetic data's adherence to widely accepted network trace data formats, ensuring broad compatibility with networking tools.

(2) *Comprehensive Metrics:* All the essential metrics that Wireshark uses to describe and analyze traffic are present, ranging from packet count and data size to encapsulation and timing details. These observations underscore our design's success in producing protocol rule-compliant synthetic traffic, ensuring compatibility with analysis tools like Wireshark that demand structural and semantic correctness.

**Tcpreplay's Retransmission Analysis.** Table 3.10 shines light on the retransmission capabilities of the synthetic traffic via tcpreplay [46]. Notable results are: (1) *Successful Retransmission:* All 1,024 packets were successfully sent without any failures or truncations, indicating the traffic's high fidelity and adherence to transport layer protocol rules. (2) *Correct Packet Handling:* Metrics like retried packets standing at zero and the exact match of unique flow packets to successful packets further reiterate the synthetic traffic's quality. (3) *Metrics Completeness:* Key metrics like data bit rate and packet rate, essential for evaluating traffic characteristics, are present and well-defined in both synthetic and real traffic. However, there are noticeable discrepancies in metric values, such as total bytes sent and rates, between synthetic and real traffic. This variance is anticipated, given that NetDiffusion generates traffic flows at a bit level. Consequently, even minor deviations, such as a single-bit difference within an 8-bit header field, can lead to significantly altered field values. Addressing this issue could involve more precise controls during generation, as we discuss in Section 3.5, or implementing value rescaling in post-generation heuristic, which we leave to future work.

**Feature Extraction for Core Network Analysis Tasks.** One of the distinguishing attributes of NetDiffusion generated traffic, compared to earlier works like NetShare, is the ability to derive detailed metrics from the synthetic traffic using tools such as Scapy, making it exceptionally valuable for an expansive range of network analysis tasks. To demonstrate this, we evaluate the applicability of our synthetic traffic on representative tasks including traffic and protocol analysis [84, 15, 147, 172, 20, 43], network performance assessment [20, 43], de-

| Attribute | Value |
|---|---|
| File type | Wireshark/tcpdump/... - pcap |
| File encapsulation | Raw IP |
| File timestamp precision | microseconds (6) |
| Packet size limit (file hdr) | 65535 bytes |
| Number of packets | 1,024 |
| File/Data size | 1,335 kB/1,318 kB |
| Capture duration | 0.602296 seconds |
| First/last packet time (absolute) | 00:00:00.000000/00:00:00.602296 |
| Data byte/bit rate | 2,189 kBps/17 Mbps |
| Average packet size/rate | 1287.76 bytes/1,700 packets/s |
| Strict time order | True |
| Number of interfaces in file | 1 |
| Interface #0 info | Encapsulation = Raw IP (129 - rawip4)<br>Capture length = 65535<br>Time precision = microseconds (6)<br>Time ticks per second = 1000000<br>Number of stat entries = 0<br>Number of packets = 1024 |

**Table 3.9:** Wireshark capinfos validation log on parsing NetDiffusion generated Amazon network traffic.

| Metric | Value (NetDiffusion) | Value (Real) |
|---|---|---|
| Total Packets Sent | 1024 | 1024 |
| Total Bytes Sent | 1318664 bytes | 164189 bytes |
| Duration | 0.613297 seconds | 0.694616 seconds |
| Rate (Bps) | 2150123.0 Bps | 236454.7 Bps |
| Rate (Mbps) | 17.20 Mbps | 1.89 Mbps |
| Packets per Second (pps) | 1669.66 pps | 1474.70 pps |
| Total Flows | 2 | 2 |
| Flows per Second (fps) | 183.06 fps | 2.88 fps |
| Unique Flow Packets | 1024 | 1024 |
| Successful Packets | 1024 | 1024 |
| Failed Packets | 0 | 0 |
| Truncated Packets | 0 | 0 |
| Retried Packets (ENOBUFS) | 0 | 0 |
| Retried Packets (EAGAIN) | 0 | 0 |

**Table 3.10:** Tcpreplay results on retransmitting (1) NetDiffusion generated and (2) real Amazon network traffic.

vice identification [85, 102], routing and user behavior characterization [154, 124, 58, 13, 19], and error evaluation [129, 72, 146]. As shown in Table 3.11, using Amazon traffic as an illustrative example, our synthetic network traffic effectively delivers both fundamental metrics, such as packet and byte counts, as well as more nuanced measures like the average TTL used for routing behavior analysis. Furthermore, metrics linked to network performance, device identification, routing behavior, and error analysis further accentuate our synthetic traffic's authenticity and granularity. A noteworthy point is the zero count for error metrics like checksum errors and fragmented packets in both real and synthetic datasets, underscoring our synthetic traffic's semantic correctness. While there are variances between some of the metric values from our synthetic data and real traffic, especially in areas like TCP flag distribution, these differences are inherent to our generative approach. Instead of merely replicating real data values, our method generatively produces them, introducing variations to enhance data diversity. The delicate balance between maintaining the realism of these variations and the utility of the resultant metrics for downstream tasks necessitates a nuanced, task-specific assessment. In future iterations, we aim to focus on enhancing the congruence between synthetic and real data metrics, such as addressing the overrepresentation of non-ACK packets in the synthetic data – a reflection of the complexities tied to

| Network Task | Metric | Unit | Value (Real Network Traffic) | Value (NetDiffusion Generated Traffic) |
|---|---|---|---|---|
| Traffic Analysis | Packet Count | packets | 1024 | 1024 |
| | Byte Count | bytes | 1100406 | 1318664 |
| | Avg. TCP Window Size | bytes | 32739.95 | 13234.63 |
| Protocol Analysis | Protocol Distribution | packets | TCP: 1024, UDP: 0, ICMP: 0 | TCP: 1024, UDP: 0, ICMP: 0 |
| | Flags Distribution | flags | SYN: 0, ACK: 1023, FIN: 0, RST: 0, PSH: 0, URG: 16 | SYN: 68, ACK: 574, FIN: 556, RST: 1, PSH: 6, URG: 495 |
| | Src Port Distribution | port (packets) | 46508 (303), 443 (721) | 30202 (376), 14443 (648) |
| | Dest Port Distribution | port (packets) | 443 (303), 46508 (721) | 14443 (376), 30202 (648) |
| Network Performance | Packet Size Distribution | packets | 0-499: 306, 500-999: 6, 1000-1499: 6 1500-1999: 706, 2000+: 0 | 0-499: 35, 500-999: 56, 1000-1499: 257 1500-1999: 676, 2000+: 0 |
| Device Identification | Src IP Distribution | packets | 192.168.43.37 (303), 54.182.199.148 (721) | 156.76.135.124 (376), 132.81.26.166 (648) |
| | Dest IP Distribution | packets | 54.182.199.148 (303), 192.168.43.37 (721) | 132.81.26.166 (376), 156.76.135.124 (648) |
| Routing Behavior | Average TTL | seconds | 186.51 | 123.33 |
| User Behavior | Number of Sessions | sessions | 1 | 1 |
| Error Analysis | Checksum Errors | packets | 0 | 0 |
| | Fragmented Packets | packets | 0 | 0 |
| | Fragmented IP Datagrams | datagrams | 0 | 0 |

**Table 3.11:** Comparison of Real and Generated Amazon Network Traffic.

emulating the TCP state machine.

In sum, our NetDiffusion generated traffic stands out by allowing the extraction of vital metrics for additional network tasks, a capability previous works lacked due to their inability to produce protocol rule compliant, fine-grained network traffic.

## 3.5 Summary

This chapter presented *NetDiffusion*, a diffusion-based framework for generating synthetic packet-level network traffic. Unlike prior approaches that primarily synthesize aggregate flow statistics or limited packet attributes, NetDiffusion generates complete packet headers and produces traces that can be converted back into PCAP format. Through its image-based representation and controlled diffusion pipeline, NetDiffusion is able to generate short traces with strong statistical similarity to real traffic while preserving important protocol structure. The resulting synthetic traces improve downstream machine learning performance relative to prior methods and remain compatible with conventional analysis and replay tools such as Wireshark and tcpreplay. Together, these results demonstrate that diffusion models provide a viable and effective approach for high-fidelity packet-level traffic generation.

At the same time, the strengths of NetDiffusion are accompanied by important limitations. Most notably, the image-based formulation naturally operates over fixed-size representations, which constrains the length of traces that can be generated. In practice, NetDif-

fusion is best suited to settings in which the target traffic can be captured within a relatively short span of packets and in which fine-grained control over protocol structure and traffic characteristics is especially important. However, many network behaviors of practical interest extend well beyond this regime. Stateful communication patterns such as connection setup, retransmissions, adaptive data transfer, and teardown often unfold over long packet sequences, and real network sessions frequently involve multiple interacting flows rather than a single isolated connection. These characteristics are difficult to capture within a fixed-size diffusion-based representation.

These observations motivate the next stage of this dissertation. If diffusion models are particularly well suited for controlled generation of short, high-fidelity single-flow traces, then a natural question is what class of models is better suited for long sequential generation and multi-flow session modeling. The next chapter addresses this question by moving from image-based diffusion to direct sequence modeling. Specifically, Chapter 4 introduces *NetSSM*, a state-space-model-based generator that treats network traffic as a sequential process over packet bytes. By leveraging the long-context and recurrent properties of structured state-space models, NetSSM is designed to capture stateful protocol behavior over substantially longer horizons and to model interactions across multiple interleaved flows. In this sense, NetSSM is not simply an alternative to NetDiffusion, but a complementary response to the limitations revealed by it.

# CHAPTER 4

# NETSSM: MULTI-FLOW AND STATE-AWARE TRACE GENERATION VIA STATE-SPACE MODELS

NetDiffusion (Chapter 3) demonstrated that diffusion models can generate short, high-fidelity single-flow network traces with strong protocol compliance. However, the image representation imposes a hard ceiling on trace length, limiting generation to approximately 1,024 packets per flow. This constraint precludes the capture of flow-state-dependent events—video segment downloads, retransmission behavior, connection teardowns—that only manifest after substantial session setup. Furthermore, neither NetDiffusion nor any prior raw packet generator can reliably produce sessions comprised of multiple interleaved flows, a pervasive characteristic of real-world workloads.

This chapter presents *NetSSM*, which takes a fundamentally different approach: rather than converting traffic to images, it treats raw packet generation as a *sequence modeling* problem and trains a structured selective state-space model (Mamba-2) directly on packet byte sequences. The key motivation is architectural alignment: SSMs maintain a recurrent hidden state that evolves as input is processed, naturally mirroring how network connections accumulate and depend on prior state. This alignment allows NetSSM to train with context windows more than 8× longer than prior transformer-based generators and to produce traces up to 78× longer.

Beyond length scaling, NetSSM's recurrent, session-level modeling enables a capability absent from all prior work: generating synthetic sessions comprised of *multiple interleaved flows*. This makes it possible to model real-world workloads—such as Netflix streaming (which uses 3–5 parallel flows for video/audio download) or IoT device communication—with high semantic fidelity.

NetSSM outperforms all prior generators on established benchmarks of statistical similarity and downstream ML performance, and introduces *semantic similarity* as a new

evaluation dimension that better captures practical utility. These results confirm that *state-space models are well-suited for the long-range, stateful, multi-flow regime* of network traffic generation.

The remainder of this chapter is organized as follows. Section 4.1 introduces the problem and provides an overview of the NetSSM framework. Section 4.2 reviews the SSM and Mamba architecture. Section 4.3 describes the NetSSM design. Section 4.4 presents our evaluation. Section 4.5 discusses limitations and future directions.

## 4.1 Introduction

There is high demand for representative, scalable network data, driven by applications in security analysis, traffic modeling, and performance evaluation [136, 68, 141, 114, 12]. Unfortunately, acquiring large-scale, high-fidelity network data is difficult due to data governance policies, and high collection costs [9, 45]. In response, methods have been developed to generate synthetic network data that accurately replicates real networks, allowing researchers and practitioners to test, evaluate, and model scenarios while minimizing collection overhead and obstacles in accessibility.

Existing methods for generating synthetic network data output this data in two forms: (1) sequences of single or multiple derived network *traffic attributes*, such as flow statistics (*e.g.,* duration, average packet size), packet header fields (*e.g.,* IP flags, addresses), or metadata (*e.g.,* web page views, event types) and (2) raw packet capture (PCAP) traces. Generators producing traffic attributes can be used to replicate patterns in arbitrarily long network communications. Generators producing PCAP traces capture the verbose, detailed, communication exchanged between hosts, and commodity packet analyzers (*e.g.,* Wireshark) can analyze their resulting PCAPs.

Unfortunately, current methods for either output format have limitations that impact their practical use. Traffic attribute generators cannot reason about the raw contents of stateful protocols, such as TCP, and require retraining to learn the patterns of new targets

in a session. Raw packet generators are limited in the length of traces they can train on and produce and, thus, may not capture meaningful communication between nodes beyond initial connection setup. Further, neither generator type can reliably produce data for sessions comprised of more than a single flow, preventing them from being applied to various workloads in the real world, where interleaved, multi-flow communication is common (*e.g.,* distributed systems, IoT). Finally, current methods for evaluating the quality of synthetic network data (*i.e.,* statistical similarity to real-world traces and downstream performance of ML models trained on synthetic data) are insufficient. Synthetic data that perform well in, or towards, these evaluations can still fall short in scenarios that require analysis of multi-flow interactions or stateful behaviors in network traffic (*e.g.,* QoE estimation [132], application fingerprinting [91]). Thus, determining the criteria for what qualities or characteristics make synthetic network data "good" is an ongoing area of research.

In this paper, we present NetSSM, a raw packet generator for network traffic data built on the recently proposed structured selective state-space model (Mamba) architecture. NetSSM bridges the gap between traffic attribute and raw packet generators by combining the former's length-scaling capabilities with the latter's comprehensive packet-level detail. This enables NetSSM to capture a substantially wider range of target events while retaining the ability to capture inter- and intra-packet dependencies across any protocol and layer. Furthermore, the sequential, stateful nature of how NetSSM learns network data allows it to generate sessions comprised of multiple interleaved flows with high fidelity, addressing a limitation of existing methods.

We evaluate NetSSM on social media, video conferencing, and video streaming traffic. First, we assess its performance using established metrics of synthetic network data fidelity (statistical similarity and downstream ML performance). We then evaluate NetSSM's *semantic similarity*, testing how well its generated data aligns with the behavioral characteristics of real-world network communication. Finally, we verify that NetSSM's traces are both protocol compliant, and mimic, rather than memorize patterns in training data. This

analysis aims to offer a more functional and application-oriented perspective on the quality of synthetic data, emphasizing its practical utility beyond statistical resemblance. Our main contributions are:

- **Synthetic multi-flow sessions.** NetSSM's recurrent nature enables it to produce traces for sessions comprised of both single flow and multi-interleaved flows, with high fidelity. Multi-flow trace generation is a new contribution largely unexplored in prior generators.
- **Capturing flow-state-dependent session events.** NetSSM trains using a context window more than 8× longer, and produces traces up to 78× longer than existing transformer-based raw packet generators. This enables it to learn from and output traces that capture flow-state-dependent events occurring later in a session that rely on early connection setup, or multiple interactions between flows and/or packets.
- **Superior performance on existing benchmarks.** NetSSM outperforms current state-of-the-art network data generators in existing benchmarks. In statistical similarity, NetSSM achieves an average Jensen-Shannon Divergence across generated traffic attributes of 0.05, versus 0.18 and 0.06 for NetShare [165] and NetDiffusion [74], respectively. In performance of downstream ML models trained on synthetic data, a random forest classifier trained entirely on synthetic NetSSM data achieves accuracy of 0.97 on held-out ground truth data, compared to 0.13 and 0.16 for NetShare and NetDiffusion respectively.
- **Behaviorally accurate and protocol-adherent traffic.** NetSSM generates synthetic traffic with high semantic similarity to real traces. This traffic can (1) capture application-specific traffic patterns, and (2) show robust session-level compliance with standard TCP protocol requirements, capturing both correct stateful behavior and common real-world anomalies (*e.g.,* partial teardowns, conflicting flags). For (1), NetSSM can generate traces that capture the sequential communication and distributional patterns in traffic, even presented with complex, multi-flow traffic comprised

of multiple steps (*e.g.,* setup with CDN endpoints before video segment downloads for video streaming traffic).

NetSSM's code and training datasets are open sourced at https://github.com/noise-lab/netssm.

## 4.2 State Space Models for Network Traffic Generation

Much communication between networked devices is stateful, and these exchanges may span long sequences of packets for multiple steps (*e.g.,* setup, payload download, teardown). Our choice of Mamba [53, 44], a line of selective structured SSMs, accommodates these characteristics. In this section, we provide background on SSMs, specifically, the Mamba model (Section 4.2.1), and compare Mamba against the existing approaches in raw packet generators (Section 4.2.2).

### 4.2.1 Background: State Space Models and Mamba

SSMs are probabilistic graphical models built on the control engineering concept of a state space [78]. Similar to Hidden Markov Models, SSMs model discrete observations over time, but use continuous, as opposed to discrete, latent variables. SSMs encode a running hidden state representative of prior observed context of input using recurrent scans, and use this state to calculate an output for a given unobserved input. Specifically, SSMs use first-order ordinary linear differential equations to capture the relationship (output) between unobserved variables (state) and a series of continuous observations (input), irrespective of time (*i.e.,* is linear time-invariant [LTI]). Unfortunately, SSMs suffer from the same pitfall as other recurrently updating models, in that over time, information about data earlier in an input becomes increasingly compressed in the hidden state. This leads to the "vanishing gradient," where the model can no longer recall dependencies between inputs. Prior works by Gu *et al.* and Voelker *et al.* remedy this challenge by fixing the state matrix used in SSMs, resulting in improved model performance for recalling long-range dependencies [54,

159]. Follow-up works by Gu *et al.* provide additional improvements to the SSM, improving training efficiency for practical use via convolutional kernel (S4 [55]) and sequence modeling performance via a selection mechanism and a fixed state matrix (Mamba, Mamba-2 [53, 44]).

Specifically, Mamba builds on S4, and additionally implements two modifications to the general SSM that provide *structure* and *selection*. It implements structure by replacing the general SSM state matrix (typically randomly initialized) with a HiPPO matrix [54], which enforces a probability measure for dictating how the SSM state is compressed. This, in effect, remedies the vanishing gradient and improves the Mamba SSM's ability to model long-range dependencies in sequences. For selection, the general LTI SSM lacks expressiveness, *i.e.,* all discrete inputs compressed in the state affect the state with equal weighting. In language modeling, this prevents semantically important "keywords" from more heavily influencing the SSM state and developing a better understanding of input. Mamba improves expressiveness by removing the LTI quality of the general SSM and makes the model *time-variant*, in which the state is calculated using learned (rather than fixed) functions of the inputs. Mamba's structure and selection modifications to the general SSM architecture provide competitive performance against conventional transformer-based approaches for sequence modeling, with better scaling (linear versus quadratic).

### 4.2.2 Why Mamba?

We select the Mamba architecture because it is inherently suited to the nature of network data, specifically the stateful nature of a large portion of network traffic (*e.g.,* TCP flows). Communication between hosts often explicitly depends on the sequential exchange of packets to ensure correct data assembly and to maintain the connection. This can be mapped to the recurrent quality of the *state-space* architecture, where the model sequentially updates the hidden state on each new input. In our use of Mamba for synthetic trace generation, this enables NetSSM to effectively learn from and produce sessions composed of multiple flows. In contrast, prior traffic attribute and raw packet generators can only operate within

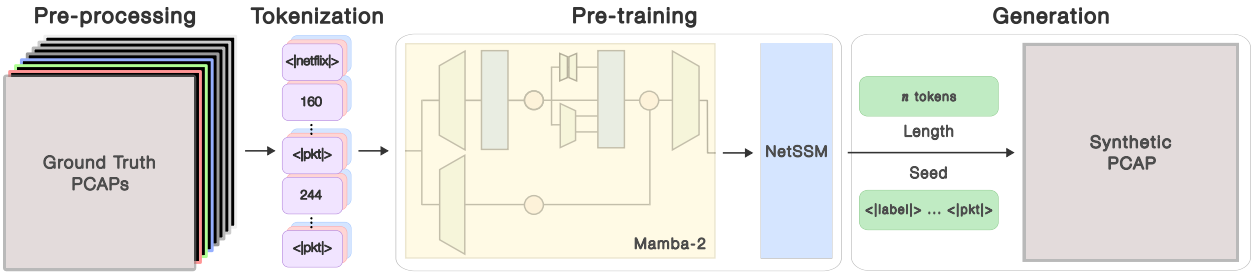



**Figure 4.1:** Overview of the NetSSM pipeline.

the scope of single-flow sessions.

The architecture's convolutional kernel further complements the network domain by enabling updates to be performed in parallel, allowing the model to train on substantially long communication while still implicitly capturing sequential dependencies. As such, Mamba is a much more "natural" fit for modeling network data compared to prior raw packet generators. Diffusion-based approaches require abstracting network data to a different domain (*i.e.,* images in NetDiffusion), and further generate traces based on signals from the entire trace, neglecting the sequential delivery of network traffic. Transformer-based models likewise learn input semantics in a completely parallel fashion, where attention is calculated per token of a sequence, against all other tokens in the sequence simultaneously, also not strictly sequentially. The completely parallel computation nature of either approach is also resource-intensive. NetSSM can generate traces roughly 10× longer than NetDiffusion and 78× longer than TrafficGPT. This is a key improvement, allowing NetSSM to capture flow-state-dependent sessions events that manifest only after substantial setup has occurred, and thus may not be captured with other models. For instance, in video streaming traffic (generated/evaluated in Section 4.4.3), a flow's representative state for downloading audio/video segments may not be reached until after a few hundred or thousand packets.

## 4.3 NetSSM

Motivated by shortcomings in existing synthetic network data generators, and strong alignment with the operation and capabilities of SSMs and the qualities of networked commu-

nications, we present NetSSM, a new raw packet generator. NetSSM uses the Mamba-2-backbone, and is trained the raw byte contents of packets, to synthesize packet traces. Figure 4.1 provides an overview of the NetSSM pipeline, and we provide details for each pipeline step below.

### 4.3.1 Pipeline Overview

**Pre-processing Networking Data.** Input to NetSSM are sequences of the raw bytes which comprise the packets in a session trace. Specifically, NetSSM parses the Packet Data field of each packet record in a PCAP file [60] to a representative format that aligns with the token-based, sequence generation objective of the Mamba SSM.

**Tokenization.** We define a custom tokenizer using Huggingface Tokenizers [106] that one-to-one maps the decimal values of the raw bytes comprising each packet to a corresponding token ID in range $[0, 255]$. In this way, NetSSM reasons about the raw contents of networking traffic close to its original form. This differs from prior work where network data is represented/tokenized at the flow level [92], as a mix of packet-level and flow attributes [115], or created using a tokenization algorithm that may map raw bytes to tokens using logic suited to a different domain (*i.e.,* WordPiece from NLP) [104]. Our tokenizer also defines label special tokens (*e.g.,* `<|facebook|>` `<|meet|>`, `<|netflix|>`) and a packet special token (`<|pkt|>`) to allow NetSSM to differentiate between traffic dynamics of different workloads, and packet boundaries in sessions.

**Creating training data.** We extract input to NetSSM from labeled (*i.e.,* the workload/service type of collected traffic is known) collections of PCAPs based on the desired modeling granularity, *i.e.,* single-flow or multi-flow sessions. For single-flow sessions, we use `pcap-splitter` [135] to first split the original PCAP into multiple PCAPs, each corresponding to a single comprising flow based on connection (*i.e., five-tuple*: source IP, source port, destination IP, destination port, IP protocol). No pre-processing is needed for multi-flow sessions (*i.e.,* captures containing multiple connections/five-tuples). We then use our

custom PCAP parser written in Go, which performs the following tasks: (1) converts the raw bytes comprising each packet in a PCAP to a sequence of 8-bit decimal values (*i.e.,* value $\in [0, 255]$) in string form, (2) delimits each string form packet with `<|pkt|>` special tokens, and (3) prepends the PCAP's corresponding label special token to the string. Finally, we use the custom NetSSM tokenizer to tokenize the parsed, string-based PCAP data to a format consumable by NetSSM, producing one input sample for each PCAP in a dataset. Our parser currently supports processing both IPv4 and IPv6 packets, the TCP and UDP transport protocols, and the DNS application protocol. The parser can easily be extended to additional protocols or workloads by simply adding a new corresponding processing function.

**Pre-training NetSSM.** Training data are fed into NetSSM to learn the semantics of packets, and correspondingly, flows and sessions. Specifically, NetSSM treats generating network traffic data as a self-supervised sequence generation problem. During training, NetSSM minimizes the cross-entropy loss function, which measures how well the predicted probabilities for a token at a specific index match the correct token. Because this learning objective is irrespective of the input used, NetSSM is easily extensible to learning the semantics of any protocol or workload. For our experiments with NetSSM, we train using a batch size of one, which allows each input/training sample to be 100,000 tokens in length (the maximum length supported for our experiment setup). This maximizes the length of packet sequences our model learns from (*i.e.,* context length), with 100,000 tokens corresponding to a context of at least 943 packets (when using different packet representations from our various case studies).

**Generating Synthetic Traces.** Trace generation requires two arguments: a generation seed and length. The generation seed matches the format of NetSSM's training samples – a label special token followed by a sequence of any number of full or partial packets represented by their raw-byte contents in decimal form (*e.g.,* `<|amazon|> 188 34 203. .. <|pkt|>`). The seed is used to "prompt" NetSSM for generation, equivalent to the

"start token" or string in NLP generative models. The generation length dictates the length of output (in tokens, or optionally, packets) that NetSSM should generate. NetSSM's packet model encodes the generation seed to its latent representation before passing it first through the actual SSM linear system, and second the softmax function. This results in a set of probabilities each token (*i.e.,* byte value) has of being selected as the generation candidate. On each generation step, the single highest probability token is both output by NetSSM and used to update the existing latent representation, becoming the input for the next round of generation. This procedure continues until the given generation length is satisfied. NetSSM then constructs the intermediate synthetic trace, concatenating the sequence of generated packets represented by their tokenized raw bytes in decimal format, and prepending the label special token. The pipeline concludes with a simple script which converts the token-based trace representation to a complete PCAP binary, with the option to assign packet inter-arrival times (IATs) to packets by sampling from the IAT distribution of a given ground truth capture (presumably from the same traffic class or workload).

### 4.3.2 What NetSSM Does and Does Not Do

NetSSM generates traces of raw packet communication in the form of PCAPs. These traces can be of arbitrary, user-specified length, and may be comprised of either a single flow (*i.e.,* two endpoints) or multiple flows (*i.e.,* more than two endpoints).

These traces capture the sequential characteristics of packet communication (and thus, may act as a weak proxy for time). Unfortunately, NetSSM does not extract or parse, and hence, does not learn from and autoregressively generate, the timestamp/IAT values for each packet. We provide additional discussion on this shortcoming and future directions for addressing it in Section 4.5.

## 4.4 Evaluation

We evaluate the quality of synthetic data produced by NetSSM through five analyzes: (1) *statistical similarity* between generated and real traffic, (2) *downstream utility* of generated data towards training and improving ML-for-networking models, (3) *semantic similarity* between generated and real traffic, (4) *protocol compliance* between generated and real traffic, and (4) *analysis of memorization* in synthetic NetSSM traffic. Previous traffic attribute and raw packet generators are measured using metrics of statistical similarity and downstream performance. We introduce semantic similarity and protocol compliance as additional aspects that should be considered when evaluating synthetic network data models or systems. Finally, we perform analysis on NetSSM's output and show that it is learning to mimic, not memorize, patterns in network data used during training.

### 4.4.1 Statistical Similarity

We first evaluate NetSSM with conventional metrics of statistical similarity used to evaluate prior traffic generators, which assess the byte-wise matching between generated synthetic traces and the ground truth training traces. Specifically, we train a NetSSM model on *single-flow* traces (*i.e.,* comprised of a single connection/five-tuple) collected from various types of multimedia traffic. We examine the single-flow granularity so that we can provide direct comparison against prior work, which all evaluate at this level. After training, we generate synthetic traces and compare them to their ground truth counterparts. We find that NetSSM's synthetic traces exhibit high statistical similarity to real data at the content level (byte-wise comparisons), outperforming previous synthetic network trace generation methods in various statistical metrics.

**Setup.** We evaluate the statistical similarity of traces produced by (1) a base NetSSM model that trains on and produces continuous sessions, and (2) a fine-tuned version of the base model that generates packets comprising distinct flow stages (*e.g.,* TCP teardown,

**Table 4.1:** Overview of datasets used to train and evaluate NetSSM.

| Dataset | Source | Evaluation | Content Type | | Size | |
|---|---|---|---|---|---|---|
| | | | Classification | # Sub-classes | Raw | # Captures |
| Multimedia Traffic | Bronzino *et al.* [23] | §4.4.1, 4.4.2, 4.4.4 | Video Streaming | 4 | 6.36 GiB | 273 |
| | MacMillan *et al.* [99] | §4.4.1, 4.4.2, 4.4.4 | Video Conferencing | 3 | 17.36 GiB | 339 |
| | Jiang *et al.* [76] | §4.4.1, 4.4.2, 4.4.4 | Social Media | 3 | 5.40 GiB | 151 |
| Netflix Streaming | Bronzino *et al.* [23] | §4.4.3, 4.4.4 | Video Streaming | 1 | 216.36 GiB | 5,882 |
| YouTube Streaming | Gutterman *et al.* [59] | §4.4.3 | Video Streaming | 1 | 2.06 GiB | 619 |

characterized by `ACK` and `FIN` packets, or data transmission, characterized by `PUSH` and `ACK` packets). Here, we wish to examine if additional fine-tuning can yield performance improvements, particularly in generating these distinct components of networked communication. Fine-tuned models could be especially useful for applications requiring only subsets of a trace to study key network behaviors (*e.g.,* session termination indicators). We detail the setup for either model below.

**Base model.** We train our NetSSM model for single-flow trace generation using the Multimedia Traffic dataset outlined in Table 4.1. We first pre-process the data using `pcap-splitter` [135], splitting PCAPs into their comprising single-flow PCAPs based on five-tuple, and parse them into the string representations of their raw bytes in decimal form, as described in Section 4.3.1. We fix each packet to be represented by 94 tokens, corresponding to the maximum practical lengths of the Ethernet (14 bytes), IPv4 (20 bytes excluding options), and TCP headers (60 bytes including extensions). We train the NetSSM model for this evaluation on TCP traffic only, and do not consider TCP payload, as this data is becoming increasingly encrypted [67, 71, 50] and would be noise our model would not learn from. Next, we create a custom tokenizer following the configuration described in Section 4.3.1, defining 10 label special tokens corresponding to the 10 distinct applications

in our dataset. We then tokenize all string representations resulting from splitting our data to their single-flows resulting in a final dataset of 27,839 samples. Finally, we pre-train the single-flow packet NetSSM model on the created dataset using a single NVIDIA A40 48GB GPU for 30 epochs with a gradient clip value of 1.0 and default AdamW optimizer with learning rate of $5 \times 10^{-4}$. We use the same configuration as the smallest publicly available 130 million parameter pre-trained Mamba-2 (dimension of 768, 24 layers), but instead use our custom tokenizer. We generate traces using the process detailed in Section 4.3.1, producing a corresponding synthetic trace for each real trace used in training. Specifically, we use the first packet from the real training trace in tokenized form, along with its corresponding label as the seed. We set the generation length as the number of tokens needed to represent the total packets in a corresponding real trace. This ensures the generated trace contains the same number of packets as the real trace, providing a consistent basis for evaluating the synthetic trace's statistical similarity to the real data.

**Fine-tuned model.** We train the fine-tuned NetSSM model by first creating sub-datasets from the original dataset described above that isolate the packets relevant to specific stages of a flow's lifetime. These sub-datasets focus on distinct phases of network communication (*e.g.,* session initiation, data exchange, session termination). We use these phase-specific data to fine-tune the base 30-epoch single-flow NetSSM, using the same next-token prediction objective as the original model but with phase-specific packets as input. This allows the model to capture the intra-packet and flow dynamics unique to each phase, leading to improvements in both the quality and flexibility of output. When generating data with the fine-tuned models, we chain outputs from one phase-specific model to the next. Specifically, the final packet produced by the handshake model serves as the seed for the subsequent data transmission model, while the final packet generated by the data transmission model acts as the seed for the subsequent session teardown model.

**Baselines.** We compare the two NetSSM variants against three prior works: NetDiffusion, TrafficGPT, and NetShare. We also evaluate against two random generations of flow statis-

**Table 4.2: Byte-wise statistical similarity.** Across generators and data granularities, NetSSM traces are most statistically similar to real traffic (divergence/distance metrics $\geq 2\times$ lower versus the next best method).

| Generation Method | Traffic Attribute-level (Avg. JSD) ↓ | | | | | | Header-level ↓ | | |
|---|---|---|---|---|---|---|---|---|---|
| | SA | DA | SP | DP | PR | Avg. | Avg. JSD | Avg. TVD | Avg. HD |
| Random Generation (flow statistics) | 0.71 | 0.71 | 0.63 | 0.63 | 0.47 | 0.63 | — | — | — |
| Random Generation (raw packets) | — | — | — | — | — | — | 0.82 | 0.99 | 0.95 |
| NetShare | 0.14 | 0.19 | 0.29 | 0.25 | 0.04 | 0.18 | — | — | — |
| TrafficGPT* | 0.13 | 0.16 | 0.17 | 0.23 | — | 0.17 | — | — | — |
| NetDiffusion† | 0.00 | 0.00 | 0.14 | 0.17 | 0.06 | 0.06 | 0.04 | 0.04 | 0.05 |
| NetSSM (base) | 0.12 | 0.11 | 0.10 | 0.11 | 0.00 | 0.09 | 0.02 | 0.02 | 0.02 |
| NetSSM (fine-tuned) | 0.06 | 0.05 | 0.05 | 0.06 | 0.01 | 0.05 | 0.02 | 0.01 | 0.02 |

*As reported in [115]. †Post-generation correction applied.

tics (uniformly sampled random values across valid attribute ranges, *e.g.,* IP addresses/ports, and protocols) and raw packets (random assignment of 1, 0, -1 to indices in the nPrint packet format [68]) to serve as benchmarks for poor fidelity. Specifically, we train new NetShare and NetDiffusion models on our Multimedia Traffic dataset (for TrafficGPT, we rely on the paper's reported results as it is closed source), and use these models to generate corresponding synthetic traffic attributes, or raw PCAPs, based on each capture in the ground truth dataset.

**Results.** We evaluate NetSSM's generation fidelity by analyzing the statistical similarity between its synthetic data, and each of these synthetic data's real, ground truth counterpart, consistent with prior evaluations [74, 93, 165, 115]. Specifically, we calculate three distributional distance metrics: Jensen-Shannon Divergence (JSD), Total Variation Distance (TVD), and Hellinger Distance (HD), where lower values indicate closer alignment to the ground truth. We calculate these metrics at two levels: (1) traffic attribute-level for direct comparison against all prior works, and (2) header-level for comparison against NetDiffusion. We compute traffic-attribute level metrics for the fields of source and destination IP addresses and ports (SA, DA, SP, DP) and IP protocol (PR). Table 4.2 presents the overview of statistical similarity for each generator and level.

**Traffic attribute-level similarity.** We extract the ground truth traffic attributes from the ground truth traces, and compute the distance metrics between these values and their

synthetic counterparts. Specifically, for NetSSM and NetDiffusion, we extract traffic attributes from each generated PCAP corresponding to a ground truth capture. NetShare directly generates traffic attributes, and thus does not require additional parsing. For brevity, Table 4.2 shows only the average JSD for each field, though the TVD and HD are highly similar ($\pm 0.02$). Between the base and fine-tuned variants, NetSSM achieves the best or second-best average JSD in all fields.

**Header-level similarity.** We compare statistical similarity at the header level by calculating the three distance metrics across each bit position in the nPrint [68] representation of a trace, for all packets' TCP headers. We exclude NetShare (only generates a subset of header fields) and TrafficGPT (closed source preventing further analysis) from this evaluation. NetSSM consistently outperforms NetDiffusion in all metrics with distances as low as 0.01 in the fine-tuned NetSSM variant. While the distance delta between NetDiffusion may appear only marginal, we re-emphasize that the comparison is performed *after* post-generation correction in NetDiffusion is applied. To illustrate, some NetDiffusion-generated traces in this experiment were unparseable by packet analysis tools prior to applying the heuristic-based fix. In contrast, NetSSM requires no post-generation correction and yields better statistical similarity.

### 4.4.2 Downstream Utility

We examine the performance of downstream ML models trained on synthetic network data to evaluate this data's quality in practical application. Specifically, we train two types of classifiers that focus on (1) application-level classification (*e.g.,* YouTube, Amazon) and (2) service-type-level classification (*e.g.,* video streaming, web browsing).

**Setup.** To test the utility of synthetic data in augmenting downstream model training, we create *downstream training datasets* comprised of packet header-level features determined via nPrintML [69]. We extract these features from both the ground truth Multimedia Traffic dataset, and three sets of synthetic data generated by NetSSM, NetDiffusion, and NetShare

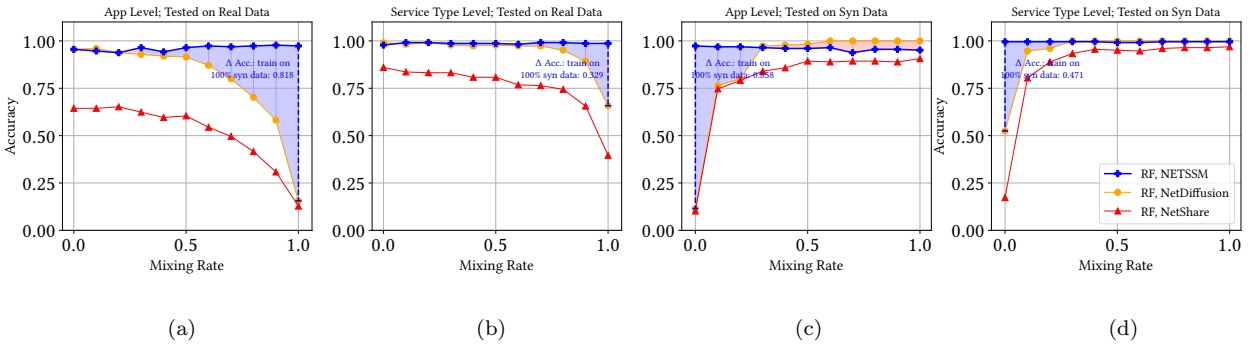



**Figure 4.2: Performance of random forest classifiers trained on mixed real/synthetic data.** Models trained on NetSSM data perform best across baselines. Shading denotes the delta between the next best baseline.

models trained on this dataset. Each downstream training dataset uses different *mixing rates* that represent the proportion of synthetic data used to replace original real data in the dataset. We create a new dataset at each 10% inclusive increments, resulting in 33 downstream training datasets (11 for each model). For example, a downstream training dataset with a 20% mixing rate contains 80% real data, and 20% synthetic data. We train three different types of ML classifiers (Decision Trees [DT], Random Forest [RF], and Support Vector Machines [SVM]) on these downstream datasets, resulting in a corresponding 33 models. Finally, we test each models' performance on held out samples of completely real, and completely synthetic data to assess their performance and generalization across different training and testing environments. In each scenario, we analyze if a generator's synthetic data can maintain and/or improve classification accuracy when mixed into the training data at various rates. Notably, because NetSSM never reproduces any trace identically (Section 4.4.5) even synthetic flows that are close to real flows in the downstream test set do not constitute train-test leakage, but instead act as supplemental samples for downstream models.

**Results.** Figure 4.2 shows the accuracy of RF models trained on the mixed downstream datasets for application and service-type-level traffic classification, and tested on both completely real and synthetic data. Comprehensive results for other model types, and models

trained on non-fine-tuned version NetSSM data are in Appendix B.1. We first examine our downstream models performance when tested on completely real data. Figures 4.2a and 4.2b visualize the results. Models trained on NetSSM data maintain consistently high classification accuracy in both the absolute and relative cases (as compared to those trained on NetShare and NetDiffusion data), across all mixing rates, even when the training set consists entirely of synthetic data. We observe substantial accuracy gains of ∼82% and ∼33% for application and service-type-level classification tasks respectively, when synthetic data constitutes 100% of the training set. Testing our downstream models on completely synthetic data yields similar results, as shown in Figures 4.2c and 4.2d. Models trained on NetSSM data consistently achieve high accuracy; at least 0.94 for application-level classification and 0.99 for service-type-level classification, regardless of mixing rate. This represents improvements of ∼86% and ∼47% in either task over the next best synthetic data generator.

### 4.4.3 Semantic Similarity

The existing measures of statistical similarity and downstream utility for network data generators are largely motivated by how well these synthetic data can improve downstream ML-for-networking model performance. However, raw packet generators introduce new challenges not captured by these metrics. While a synthetic trace may have both high traffic attribute and header-level similarity, these measures do not reflect the *longitudinal* quality of its contents. To illustrate, consider a synthetic trace that contains communication between two desired endpoints, but the contained setup and progression between flows is incorrect or out-of-order. Here, while the five attributes we evaluate in Section 4.4.1 (source/destination IP and port, protocol) would have high statistical similarity, this traffic may not be representative to replace real-world data.

To this end, we evaluate NetSSM's ability to produce *semantically similar* synthetic network traffic. Specifically, we generate both single-flow and *multi-flow* (*i.e.,* comprised of more than one connection/five-tuple) Netflix and YouTube video streaming traffic using new

NetSSM models further detailed in this section. We select the workload of video streaming as it contains well-defined patterns which can easily be deemed correctly/incorrectly modeled. We choose to inspect the multi-flow granularity to examine if the synthesized inter-flow interactions may positively influence the overall fidelity of the trace, and for novelty – no existing traffic generator can generate multi-flow traffic. We then evaluate these traces to examine whether they capture implicit characteristics for a given networked communication workload. To do so, we examine communication between end hosts and synthetic Netflix/YouTube video streaming servers in our generated traces, analyze the attributes of their downloaded video segments, and verify that they reflect the qualities found in real traffic (Section 4.4.3). In our generated multi-flow traces, we further sequentially visualize their segment sending patterns and find that NetSSM's synthetic data captures sending behavior that mimics progression of a real-world video streaming workload.

In all of these analyses, it is not our goal to declaratively state that multi-flow generation is superior to single-flow generation, or vice versa. This would require further work evaluating NetSSM on many additional workloads. Instead, we want to understand the characteristics of traffic that display higher fidelity (in regard to ground truth traffic) when synthesized in either generation granularity, to better inform how NetSSM can most effectively be used. We provide a recap and further discussion on this point and the results of our analysis in Section 4.5.

**Setup.** We train four NetSSM models, two for Netflix traffic and two for YouTube traffic, on single-flow and multi-flow traffic for Netflix and YouTube video streaming sessions, respectively. Specifically, we use the traces collected by Bronzino *et al.* [23] to train our Netflix NetSSM model and the traces from Gutterman *et al.* [59] to train our YouTube NetSSM model. Table 4.1 provides an overview of both datasets. We also note that the video streams contained in our Netflix traces exclusively use TCP-based Dynamic Adaptive Streaming over HTTP [137] (DASH), while the video streams in YouTube traces use both TCP and QUIC/UDP-based DASH.

**Training.** For the multi-flow models, we keep all captures in their original, multi-flow state (*i.e.,* do not split captures to their comprising single-flows, keep both UDP and TCP traffic), but perform all other pre-processing identically as described in Section 4.4.1. This results in training datasets of 5,882 and 619 multi-flow samples for Netflix and YouTube, respectively. For the single-flow models, we follow the procedure detailed in Section 4.4.1 (splitting the original, multi-flow traces into their numerous individual flows), but also filter out training samples comprising fewer than 10,000 packets (a value chosen for reasons described in the following paragraph). We do this to ensure our single-flow models only learn from the flows which likely correspond to video segment downloads, to provide a fairer comparison against the multi-flow models. This results in training datasets of 5,895 and 791 single-flow samples for Netflix and YouTube, respectively. We pre-train all models using the same training hardware, parameters and input size (samples of 100,000 tokens in length) as described in Section 4.4.1 for 30 epochs.

**Generation.** We then use the models to generate synthetic, single and multi-flow video streaming traffic for both Netflix and YouTube. We begin the process of creating prompts for generation by filtering the PCAPs in each of the four models' respective training sets (2 applications $\times$ 2 granularities: Netflix single-flow, Netflix multi-flow, YouTube single-flow, YouTube multi-flow) to find the captures with size $\geq$ 10MB, likely representative of downloading video streaming content. We empirically observe that in the multi-flow traces, the video stream segment download patterns described in the previous section for our training data become discernible after 2,250 packets for Netflix, and after 500 packets for YouTube. In the single-flow traces, we found these offsets to be 200 and 25 packets for Netflix and YouTube respectively. Accordingly, we use these lengths in our prompts to "bootstrap" generation, creating prompts comprised of the tokenized representations of the first corresponding $n \in \{2250, 500, 200, 25\}$ packets from the respective ground truth trace, for each of the four application/granularity pairs. We create 400 prompts total from 100 traces randomly selected from each of the four filtered trace sets. We then generate one synthetic trace

for each prompt, each of length 10,000 packets, for a total of 400 PCAPs (100 for each granularity/application pair). Appendix Section B.3 details the generation hyperparameters for each granularity/application pair. We choose to generate only 100 synthetic traces of 10,000 packets each to balance evaluating NetSSM's generation expressiveness over a sufficiently long context, and computational constraints – each trace takes approximately 20 minutes to generate, making full-dataset evaluation prohibitive (generating all synthetic traces for our 400 prompts took approximately five and a half days). In our below analyzes, we compare the generated traces against their ground truth counterparts, truncated to a matching length of 10,000 packets.

**Results.** We evaluate NetSSM's ability to generate traces that capture the semantics and session dynamics of application-level streaming traffic, for each of the four single/multi-flow and Netflix/YouTube models. Concretely, for each model's generated traces, we perform one-to-one comparison of a synthetic trace with the original trace whose first $n$ packets were used to prompt its generation, and compare the distributions of their quantities and sizes. Specifically, we infer the DASH *video content segments* found in both the ground truth Netflix/YouTube traces, and the synthetic traces generated by NetSSM. We use two different definitions of a segment for Netflix and YouTube, respectively, both matching the definition provided in the corresponding original work for either dataset. For Netflix, we initialize a segment for any uplink packet with non-zero payload, whose destination IP address corresponds to an address received in answers to DNS requests for Netflix domains (*i.e.,* `nflxvideo`, `netflix`, `nflxso`) sent at the beginning of a trace. Subsequent downstream traffic increments the size of the segment. For YouTube, we initialize a segment for any uplink packet with payload greater than 300 B, and further only consider it an audio/video segment if it has a final size of at least 80 KB. Notably, Bronzino *et al.* find that Netflix traffic "downloads, on average, four video segments and one audio segment" at a given time using many parallel flows, while Gutterman *et al.* report that "for most of [their] dataset, for a given session, audio and video chunks are transmitted from one server."

**Table 4.3: Statistical and distributional comparisons of video streaming segment downloads.** Comparison between ground truth and corresponding synthetic NetSSM traces.

| Comp. w/ GT | Evaluation | Statistical Measures | | | K–S Test | | Anderson-Darling Test | | KL Divergence | EMD |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Mean Δ | Median Δ | Std. Dev. Δ | Stat. ↓ | p-value ↑ | Stat. ↓ | p-value ↑ | Stat. (nats) ↓ | Dist. ↓ |
| Netflix | | | | | | | | | | |
| NetSSM SF | Raw Size | 1.87 | 1.68 | 104.39 | 0.31 | 0.04 | 3.40 | 0.03 | 2.06 | 95.04 |
| NetSSM MF | Raw Size | −1.09 | 0.00 | 193.91 | 0.21 | 0.03 | 6.86 | 0.01 | 1.14 | 72.94 |
| NetSSM SF | Avg. Size/Flow | −0.16 | −0.30 | 38.73 | 0.36 | 0.55 | 0.98 | 0.46 | 3.87 | 57.90 |
| NetSSM MF | Avg. Size/Flow | −2.27 | −0.66 | 71.73 | 0.22 | 0.82 | 0.29 | 0.73 | 2.67 | 30.75 |
| NetSSM SF | # Segments/Flow | 1.69 | 2.00 | 1.93 | 0.48 | 0.19 | 3.00 | 0.06 | 11.37 | 3.95 |
| NetSSM MF | # Segments/Flow | −42.24 | 0.00 | 6.22 | 0.17 | 0.95 | 0.41 | 0.76 | 2.97 | 8.04 |
| YouTube | | | | | | | | | | |
| NetSSM SF | Raw Size | 10.89 | 431.01 | 41.76 | 0.57 | 0.19 | 3.10 | 0.02 | 12.01 | 592.42 |
| NetSSM MF | Raw Size | 1.71 | 168.72 | 71.47 | 0.50 | 0.22 | 1.99 | 0.06 | 10.06 | 430.13 |
| NetSSM SF | Avg. Size/Flow | −78.43 | 483.07 | — | 1.00 | 1.00 | — | — | 19.56 | 666.35 |
| NetSSM MF | Avg. Size/Flow | −269.59 | −6.92 | — | 0.50 | 0.83 | — | — | 18.30 | 277.50 |
| NetSSM SF | # Segments/Flow | 6.01 | 7.00 | — | 1.00 | 0.67 | — | — | 24.23 | 7.30 |
| NetSSM MF | # Segments/Flow | 0.57 | 0.00 | — | 0.50 | 1.00 | — | — | 11.45 | 4.04 |

Values for the statistic, p-value, and distance are the median values. GT := ground truth, SF := single-flow, MF := multi-flow.

We extract these segments from the ground truth real, and synthetic data. To extract segments from synthetic Netflix traces, we use the IP subnets for Netflix domains found in the ground truth addresses to filter for video stream content, as our generated traces do not contain the DNS payload to perform the same procedure. All other extraction logic follows as previously described. We extract YouTube segments from our synthetic traces exactly as previously described. We then analyze the one-to-one differences in video segment download behavior between synthetic traces, and their corresponding real-world trace used to prompt generation.

**Segment Attributes.** We compare the distributions and summary statistics for segments in regard to 1) raw sizes of all downloaded segments 2) average segment size per flow, and 3) number of segments downloaded per flow, for each ground truth and NetSSM-generated trace pair, for each generation granularity and application. Table 4.3 provides the summary statistics and results of analysis using various standard statistical measures – the two-sample Kolmogorov-Smirnov (K-S) and Anderson-Darling tests, Kullback-Leibler (KL) divergence, and the earth mover's distance (EMD) – for each evaluation. In the Mean Δ and Median Δ summary statistics, $\Delta := median_{GT_i}(\texttt{eval}) - median_{Gen_i}(\texttt{eval})$, where

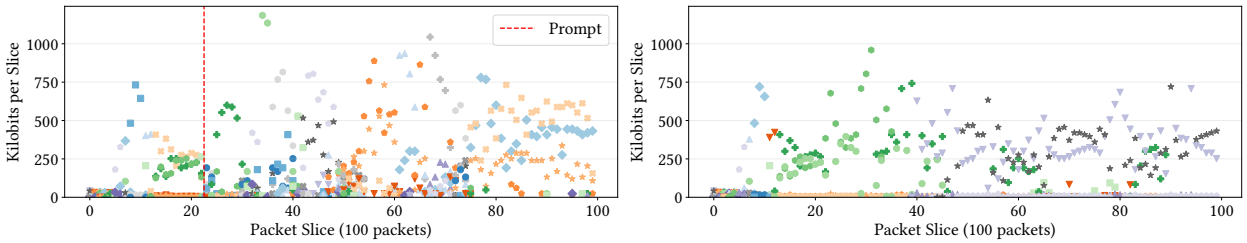
$i \in [1, 100]$, and $GT$ and $Gen$ denote ground truth data and generated data, respectively. To contrast, STD. DEV. $\Delta \coloneqq median\left(\sigma_{GT_i}(\texttt{eval}) - \sigma_{Gen_i}(\texttt{eval})\right)$ where $i \in [1, 100]$. Across statistical measures, smaller values for the statistic or distance and larger $p$-values suggest higher distributional similarity. We provide additional visualizations that compare other sampled synthetic/ground truth distribution examples in Appendix B.2.

We observe that as an aggregate across all Netflix traces, NETSSM's synthetic traces of both granularities contain segment downloads whose distribution patterns are similar to the ground truth. Comparing single and multi-flow traces, we observe that multi-flow traces are more similar to their ground truth counterparts. While the similar summary statistic values for either granularity are largely comparable (except for STD. DEV. $\Delta$ and number of segments downloaded per flow), the multi-flow traces have markedly more similar distributions to the ground truth than the single-flow traces. This is evidenced by the generally lower K-S and Anderson-Darling test statistic and EMD distance values, suggesting large overlap, with additionally larger p-values, in all evaluations except for raw segment size. In the raw segment size evaluation, NETSSM traces of either generation granularity have low median p-values, with the corresponding statistic for the K-S test, KL divergence, and Anderson-Darling test being low, low, and high, respectively. This suggests that while the general distribution for the traces may be similar, there exist differences in tail values for segment sizes that the traces do not reflect. This is likely explainable by the nature of the training data. Because downloading video segments is a largely stable workload across the network conditions in our dataset, our models learn to generate traces that predominantly reflect this norm, with tail segment download behavior less prevalent.

Examining NETSSM-generated YouTube traces, we observe less positively conclusive results. We omit calculating the standard deviation and two-sample Anderson-Darling test for the average segment size and number of segments per flow, as the ground truth behavior of YouTube traffic is downloading only from one server. We observe in single-flow generation that for both raw segment size and average segment size per flow, the mean $\Delta$ is close to, or

smaller than the ground truth respectively, while the median $\Delta$ is consistently larger. This suggests that in this granularity the majority of downloading flows NetSSM generates typically download segments whose sizes are larger than normally observed in the ground truth, but whose remaining downloaded segments are substantially smaller. Multi-flow generation displays similar behavior in size of raw segments, but appears to generally synthesize flows whose average segment size is very close to the ground truth. Unfortunately, it at times appears to "hallucinate" numerous outlier flows which download significantly smaller segments, straying from typical YouTube behavior (sequential segment downloads using only a single flow/from one server) and resulting in the substantially negative mean $\Delta$. The remaining statistical measures are similarly less conclusive, likely for the same reason. Though we can attempt to "guide" NetSSM towards generating traces with only a single dominant flow via generation hyperparameters (*i.e.,* influencing token selection), these mechanisms do not ensure that this is adhered to (in either generation granularity). We expect enforcing this constraint on NetSSM would significantly improve its performance for modeling YouTube streaming traffic, and other predominantly single-flow workloads.

**Sequential Sending Patterns.** We next evaluate if the multi-flow traces generated by NetSSM indeed capture the video segment send/download patterns found in real traffic. This allows us to determine if the events "behind" the previous distributional analysis are real-world consistent. To do so, we plot the throughput of traces based on their comprising flows, as a function of packet order. We do this to present a more direct evaluation of whether NetSSM meaningfully models networked communication over the span of its generation, as timestamps are not truly generated by NetSSM, but assigned post hoc (Section 4.3.1). For a given trace, we partition the trace into slices of 100 packets. We then assign the packets in each slice to their corresponding five-tuple flow, and sum the size of all packets in a flow to obtain the throughput in kilobits/slice for that flow. Figure 4.3 visualizes this throughput for both the ground truth (truncated to a matching 10,000 packets) and generated traces for a sample Netflix trace pair. Appendix Figure B.4 contains additional visualizations for either




**(a)** NetSSM-generated Netflix trace. **(b)** Ground truth Netflix trace.

**Figure 4.3: Comparison of throughput (synthetic vs. corresponding ground truth trace).** Each point's color/shape combination denotes a unique flow. Color/shape combinations are not shared between 4.3a/4.3b.

application. In both NetSSM-generated Netflix and YouTube traces, we see dominant flows that appear empirically similar to the behavior described in the original works (typically three to five active flows for Netflix, one for YouTube). Unfortunately as previously mentioned, a notable amount of small "hallucinated" flows for YouTube traffic are also present, resulting in these traces deviating from the ground truth behavior.

We quantify our analysis by comparing the aggregate throughputs of the generated and ground truth pairs. We compare aggregate throughput to allow communication between synthetic five-tuples (*i.e.,* not present in the ground truth) that reflect the correct sending/receiving behavior of video streaming traffic to count in analysis. We calculate aggregate throughput by summing the throughput for each flow in a slice, for all slices. We next calculate two metrics across all synthetic/ground truth pairs: (1) the median Pearson correlation coefficient (PCC) which measures overall alignment of generated and ground truth aggregate throughput, and (2) dynamic time warp (DTW) normalized by the length of the trace which quantifies magnitude-based error while allowing for minor shifts in alignment. We find NetSSM's Netflix traces have moderate positive correlation (PCC=0.52), while YouTube traces have weak positive correlation (PCC=0.31). Both results are statistically significant with $p = 0.00$. In magnitude, we find that Netflix and YouTube are 121.45 and 69.86 kilobits off from the ground truth at any given moment. Though not optimal, these metrics confirm that while the distribution of segment sizes and downloads synthesized by NetSSM can deviate from the ground truth, the traffic download/sending patterns of

the predominant flow(s) are captured. Thus, it appears that different workloads may likely require a specific generation hyperparameter configuration that best balances generation of realistic segment download distributions alongside the correct sequential communication patterns.

### 4.4.4 Protocol Compliance

We next evaluate if NetSSM's synthetic traces are "real-world" flow and session-compliant, assessing how well they approximate legitimate TCP operation and captures TCP anomalies observed in practice. Specifically, we compare the TCP state transitions of NetSSM-generated traces for the combined single-flow Multimedia traffic from Sections 4.4.1 and 4.4.2 and single and multi-flow Netflix streaming traffic from Section 4.4.3, against the behavior of their ground-truth traces. We also provide comparison against single-flow traces generated by NetDiffusion, without post-generation corrections applied. TCP is a stateful protocol that requires accurate ordering and flag usage, adherence to handshake procedures, and consistent usage of options. However, in real network traffic, these behaviors may deviate from RFC specifications for various reasons (*e.g.,* middlebox interventions, partial captures). We parse all traces generated in the previous evaluations using a custom TCP compliance checker that inspects flags, sequence numbers, acknowledgment numbers, and TCP options. Table 4.4 presents the results of this checker, showing the average percentage change in selected metrics as compared to the ground truth, for both single and multi-flow traces. We also note several prior and concurrent works that develop model-agnostic methods to "gate" synthetic output to be protocol compliant [62, 63]. While relevant, our objective in this analysis, measuring NetSSM's implicit ability to produce TCP-compliant behavior, differs.

We find that NetSSM can produce protocol compliant flows, with the rate of overall compliance being higher for single-flow, as compared to multi-flow NetSSM traces. Single-flow NetSSM traces largely follow the behavior of the ground truth, showing relatively low deltas in expected TCP behavior, and further contain similar rates of anomalies as found

**Table 4.4: TCP session compliance.** Average percentage change in selected metrics as compared to the ground truth, for multi-flow NetSSM and single-flow NetSSM and NetDiffusion traces, respectively.

| Metric | Model (Avg. %Δ from ground truth) | | |
|---|---|---|---|
| | NetSSM (multi-flow) | NetSSM (single-flow) | NetDiffusion (single-flow)† |
| *TCP session behavior* | | | |
| `FIN` seen | 50.0% | 9.4% | 45.7% |
| Correct handshakes found | 0.0% | −5.8% | −70.2% |
| `ACK` progress | −1.0% | −6.5% | −69.6% |
| `SEQ` progress | −1.0% | −8.3% | −68.9% |
| `FIN-ACK` observed | −4.0% | 2.6% | −0.5% |
| *Anomalies or deviations in TCP behavior* | | | |
| Conflicting flags | 57.0% | 0.5% | 34.4% |
| `SAck` used w/o `OK` | 44.0% | −1.8% | 15.8% |
| Unexpected `SYN` after estab. | 6.0% | 0.0% | 0.0% |
| MSS outside handshake | 5.0% | 1.4% | −2.0% |
| WScale outside handshake | 5.0% | 1.4% | −2.0% |
| `RST` in established state | −16.0% | 0.0% | −35.7% |

† No post-generation fixes applied.

in the real data. Multi-flow NetSSM traces are less consistent, showing increased rates of some behaviors (*e.g.,* sent `FIN` packets) and decreased rates of others (`RST` in established state). While these are indeed deviations from the training distribution, it is difficult to definitely label this behavior as desirable/undesirable as ground truth PCAPs are often truncated for various reasons unrelated to their comprising communication (*e.g.,* capture limits, packet loss, monitoring placement). Multi-flow traces also display higher rates of some anomalous behavior (*e.g.,* conflicting flags). We analyze the traces corresponding to this behavior and find that NetSSM appears to at times merge consecutive flag states. Without post-generation correction, single-flow NetDiffusion traces often are not protocol compliant.

### 4.4.5 Memorization Analysis

We verify that NetSSM learns from, rather than memorizes its training data by performing three analyzes on all combined single and multi-flow traces generated in our prior evaluations: (1) one-to-one byte-wise comparison of packets, (2) an approximate matching comparison of packets based on the normalized Hamming distance for each synthetic packet to its nearest neighbor (NN) in the ground truth trace, and (3) a diversity ratio we define as the mean

pairwise distance using the same normalized Hamming distance for all synthetic packets, divided by the mean pairwise distance found in the ground truth trace. Appendix Table B.2 provides detailed results of our analysis. For (1), we find that on average, only 2.35% of packets are identical per trace. Further, we verify that this percentage corresponds identically to the packets used for prompting NetSSM, accounting for our varying prompt lengths. In all other differing packets, the average percentage of differing bytes is 22.27%, with most differences largely manifesting in fields non-sessional to flow state or protocol compliance. For (2), we find only 3.83% of packets lie in a 5% distance of a ground truth packet, and that this value scales with the distance threshold. We also run this analysis across different sequential bins of indices (0–10, 10–50, and 50–100 packets) and find that the average NN distance ranges from 0.128 to 0.223, indicating that NetSSM learns deterministic setup phases from its prompt, but generates more varied content as the session progresses. Finally, we compute a diversity ratio of 0.53, suggesting that NetSSM produces more closely clustered samples than ground truth traffic. For applications requiring broader behavioral coverage, generation hyperparameters can be tuned to balance fidelity and diversity.

## 4.5 Discussion, Limitations, and Future Work

**Improving timestamp generation.** Currently in NetSSM, timestamps are not learned, but sampled from the distribution of a ground truth capture. This is not ideal for two reasons. First, it does not ensure sequential-temporal correlated key events are accurately represented in synthetic traces. Timestamps assigned to packet may fail to faithfully reflect the true dynamics of the key events which they correspond to. For instance, if NetSSM was trained on traces containing communication between three endpoints: two on the same local network and one geographically distant, higher latency timestamps of packets to/from the third endpoint could be assigned to communication between the local endpoints, and vice versa. Second, it requires a NetSSM user retain real data to sample from. This can be especially limiting in exporting trained NetSSM models in environments where the sharing

of any data (either raw or derived) is not allowed. In such cases, NetSSM can still generate PCAP files without sampling timestamps, but the utility of the resulting synthetic traces may substantially decrease, particularly and intuitively when modeling workloads and/or behaviors where time is a large, dependent variable (*e.g.,* buffer drain in video streaming).

Ideally, timestamps should be modelled in parallel with, and conditionally based on, flow or packet interactions that arise during generation. A number of challenges makes this particularly difficult. First, the modalities of packet contents (discrete values for bytes) and timestamps (continuous values for duration since epoch) are not the same, and thus cannot be simultaneously modeled using the same objective function. During experimentation, we confirmed this challenge when we attempted to modify NetSSM to contain an additional regression modelling component that took as input the most recent generated two consecutive packets, and output the packet IAT. Despite training this component under various custom loss functions, we were unable to obtain a model that accurately captured IATs. Instead, the generated IATs roughly regressed to the mean of the trace, despite loss functions placing emphasis on spikes, or other long-tail events. Second, there exist influences external to the data and communication contained in packets (*e.g.,* physical distance, link outages) that may have far greater impact on the timestamp value. While prior work in the ML-community [169, 139] has demonstrated simultaneous generation of both discrete and continuous-typed tabular data, the causal dependencies for these mixed types are wholly contained in each independent sample. This contrasts with the scope of our modelling, where timestamps have causal dependencies on the external influences mentioned above, not captured in the PCAP data NetSSM learns from. As such, it seems necessary to in tandem, consider a *third* modality that captures a network's topological characteristics to inform timestamp generation.

An alternative approach may involve a special token which demarcates packet content from discretized (*i.e.,* tokenized) representations of the timestamp, allowing for training under the same cross-entropy loss function (though the benefits/detriments of discretizing

time must be considered). Future improvements to NetSSM's timestamp generation and broader efforts to model both packet contents and time in parallel should thus consider how to reason about the different modalities involved, and attempt to incorporate outside influences not present in the capture itself.

**Choosing generation granularity (single-flow versus multi-flow).** NetSSM is the first network traffic generator that can generate raw packet traces comprised of either a single flow, or multiple flows. However, it is important to consider which granularity is most appropriate when modelling a given networked workload. An example of this is in Section 4.4.3, where instructing NetSSM to generate a multi-flow trace for YouTube traffic, a workload whose ground truth is predominantly comprised of only a single audio and video downloading flow, results in a substantial number of "hallucinated flows." Alternatively, synthetic traces for Netflix traffic, a workload whose ground truth is predominantly comprised of five flows (one downloading audio and four downloading video) show moderate positive correlation to the ground truth. We provide this case study to provide more thorough analysis of NetSSM's behavior for generating video streaming traffic, and to view how NetSSM behaves given different sending and receiving dynamics (*i.e.,* YouTube's single server audio/video chunk transmission versus Netflix's multiple server transmission). Intuitively, using a single-flow-trained NetSSM model to learn and generate a predominantly single-flow workload will very likely yield substantially better results. Taken a step further, it may be more effective to learn from and independently generate multiple key single-flow traces for a given workload, before combining them in a unified, interleaved trace. However, determining how to order the arrival and interleaving of flows may be a non-trivial task.

**Generating and evaluating more diverse network data.** In this work, we generate single and multi-flow traces for various multimedia traffic. We choose these workloads as they are straightforward, and provide a solid starting point for evaluating NetSSM's synthetic data. Follow-up work should explore generating more diverse, and/or complicated traffic. One immediate direction is extending NetSSM's pre-processor to parse traces comprised of

additional transport and application layer protocols (*e.g.,* QUIC, RTP). This is easily implementable, only requiring writing an additional handler function within our Go parser; all ML-related operation of NetSSM remains the same. Additionally, we find that NetSSM's performance may vary based on generation hyperparameters to best suit a target workload. As such, additional modelling of different network workloads could help to better understand if different patterns of parameters possibly exist for different traffic.

# CHAPTER 5

# CONCLUSION AND FUTURE DIRECTIONS

## 5.1 Summary

The scarcity of high-quality network datasets remains a persistent obstacle in networking research, security evaluation, and machine-learning-driven network management. Real-world packet traces are difficult to collect and share due to privacy concerns, operational overhead, and limited accessibility. As a result, the ability to generate realistic synthetic network traffic is increasingly important for enabling experimentation, benchmarking, and training of ML systems for networking tasks.

This dissertation explored generative modeling approaches for synthesizing realistic packet-level network traffic. Specifically, it investigated how different classes of generative architectures can be adapted to model network communication and what trade-offs arise from those architectural choices. Two complementary systems were developed and evaluated: *NetDiffusion* and *NetSSM*. Together, these systems reveal distinct regimes of capability within the design space of synthetic network traffic generation.

**NetDiffusion.** NetDiffusion demonstrated that diffusion models can be effectively adapted for network traffic synthesis by representing packet captures as structured image data. Leveraging mature diffusion-based image generation ecosystems such as Stable Diffusion and ControlNet, NetDiffusion generates short single-flow packet traces with high distributional fidelity and strong protocol compliance. The generated traces can be reconstructed into PCAP format and used directly with conventional networking tools such as Wireshark and tcpreplay. Experimental results show that NetDiffusion produces synthetic traffic that closely matches real packet traces and improves downstream machine learning performance, including significant improvements for minority classes through synthetic data augmentation.

**NetSSM.** While diffusion models excel at generating highly controlled short traces, many real-world network behaviors unfold over much longer time horizons and involve interleaved flows within the same session. NetSSM addresses this challenge by modeling packet sequences directly using structured selective state-space models based on the Mamba architecture. Because state-space models scale linearly with sequence length and maintain a recurrent hidden state, they naturally align with the sequential and stateful nature of network communication.

NetSSM is able to train on context windows significantly longer than those used by prior transformer-based packet generators and produces traces orders of magnitude longer than previous systems. More importantly, it enables the generation of realistic multi-flow network sessions, capturing interactions between concurrent flows within a single trace. Experiments demonstrate that NetSSM achieves strong statistical similarity to real traffic and allows machine learning classifiers trained entirely on synthetic data to generalize effectively to real-world traffic.

## 5.2 Complementary Strengths of Generative Architectures

Beyond the individual systems themselves, the central insight of this dissertation is that different generative architectures occupy distinct regions of the synthetic traffic generation design space.

- **Diffusion models** excel at fine-grained *structural fidelity*. The image-based representation of packet traces allows diffusion models to capture detailed spatial relationships among packet header fields. Because diffusion models are trained on large image corpora and support conditional generation through mechanisms such as ControlNet, they offer strong control over protocol distributions and packet structure. This makes diffusion models particularly effective for generating short traces that must satisfy strict protocol constraints. The primary limitation of this approach is that generation operates over fixed-size representations, which restricts the length of traces that can be produced.

- **State-space models** excel at *long-range sequential modeling*. Their recurrent hidden state accumulates information about previous packets and session context, closely mirroring the stateful behavior of network protocols. This enables the generation of substantially longer traces and realistic multi-flow sessions, capturing the temporal dynamics of real network workloads. However, because these models operate directly on packet byte sequences, they offer less explicit control over the fine-grained distribution of individual protocol fields compared to diffusion-based generation.

These observations suggest that the choice of generative architecture should be guided by the intended application. Diffusion-based approaches are well suited for scenarios requiring short, highly controlled traces with strong protocol fidelity. In contrast, state-space models are more appropriate when modeling long sessions, stateful communication patterns, or interactions among multiple flows.

## 5.3 Open Challenges and Future Directions

Despite the progress demonstrated by NetDiffusion and NetSSM, several important research challenges remain.

**Timestamp generation.** Neither system directly models packet inter-arrival times during generation. NetSSM samples timestamps from empirical distributions, while NetDiffusion does not explicitly generate timing information. A key challenge is the multi-modal nature of the problem: packet contents are discrete byte sequences, whereas timestamps are continuous-valued signals influenced by network conditions such as latency and congestion. Future work may explore joint modeling approaches using discretized timestamp tokens or multi-head generation architectures that jointly predict packet content and timing.

**Combining fidelity and sequence length.** An important future direction is the development of hybrid architectures that combine the structural fidelity of diffusion models with the long-context modeling capabilities of state-space models. One possible approach is a hierar-

chical generation pipeline in which a sequence model generates session-level packet structure while a diffusion model refines individual packet representations. Such architectures could potentially unify the strengths of both paradigms.

**Multi-flow workload modeling.** Although NetSSM enables multi-flow generation, determining the appropriate granularity for modeling traffic sessions remains an open question. Future work may explore models that dynamically decide when to generate single flows versus multi-flow sessions, or systems that independently generate flows and learn to interleave them according to realistic workload dynamics.

**Protocol diversity and payload generation.** The current systems primarily focus on common protocol combinations such as IPv4/IPv6 with TCP or UDP. Extending these methods to additional protocols such as QUIC, RTP, or emerging application-layer protocols represents a natural extension. Generating semantically meaningful payloads remains a particularly challenging problem, especially for encrypted traffic. Latent representations learned through autoencoders or protocol-aware embeddings may offer a promising direction for modeling payload semantics.

**Foundation models for network traffic.** Both NetDiffusion and NetSSM are trained on specific traffic datasets. A long-term research direction is the development of large-scale *foundation models for network traffic*. Such models could be trained on diverse network datasets and later adapted to specific workloads through fine-tuning, similar to the paradigm used in modern language models. The ability of state-space models to efficiently model extremely long sequences makes them particularly promising candidates for such large-scale pre-training.

## 5.4 Closing Remarks

Synthetic network traffic generation is becoming an increasingly important capability for networking research and practice. As machine learning becomes more deeply integrated

into network management and security systems, the need for realistic, scalable, and privacy-preserving datasets will continue to grow.

This dissertation demonstrates that the generative architecture underlying a traffic generator fundamentally determines the types of network behavior it can capture. Diffusion models and state-space models occupy complementary positions in this design space, offering distinct advantages in terms of protocol fidelity, sequence length, and session complexity. Understanding these trade-offs provides a principled foundation for selecting or combining architectures in future synthetic network traffic generation systems.

## Bibliography


[1] User Datagram Protocol. RFC 768, August 1980.

[2] Internet Control Message Protocol. RFC 792, September 1981.

[3] Internet Protocol. RFC 791, September 1981.

[4] Transmission Control Protocol. RFC 793, September 1981.

[5] An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, November 1982.

[6] Kdd cup 1999 data, 1999. Accessed: 2023.

[7] The caida anonymized internet traces data, 2019. Accessed: 2023.

[8] The cisco trex tool, 2023. Accessed: 2023.

[9] Sebastian Abt and Harald Baier. Are we missing labels? a study of the availability of ground-truth in network security research. In *2014 third international workshop on building analysis datasets and gathering experience returns for security (badgers)*, pages 40–55. IEEE, 2014.

[10] Iman Akbari, Mohammad A Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. A look behind the curtain: traffic classification in an increasingly encrypted web. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(1):1–26, 2021.

[11] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[12] Fred Baker, Bill Foster, and Chip Sharp. Cisco architecture for lawful intercept in ip networks. *Internet Engineering Task Force, RFC*, 3924, 2004.

[13] Anand Balachandran, Geoffrey M Voelker, Paramvir Bahl, and P Venkat Rangan. Characterizing user behavior and network performance in a public wireless lan. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 195–205, 2002.

[14] Adrián Barahona-Rıos and Sandra Pauletto. Synthesising knocking sound effects using conditional wavegan. In *17th Sound and Music Computing Conference, Online*, 2020.

[15] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 71–82, 2002.

[16] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[17] Jay Beale, Angela Orebaugh, and Gilbert Ramirez. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[18] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, 2006.

[19] Andrea Bianco, Gianluca Mardente, Marco Mellia, Maurizio Munafo, and Luca Muscariello. Web user session characterization via clustering techniques. In *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, volume 2, pages 6–pp. IEEE, 2005.

[20] ROBERTR Boorstyn, Aaron Kershenbaum, Basil Maglaris, and Veli Sahin. Throughput analysis in multihop csma packet radio networks. *IEEE Transactions on Communications*, 35(3):267–274, 1987.

[21] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15):3531–3547, 2012.

[22] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Hyojoon Kim, Renata Teixeira, and Nick Feamster. Traffic refinery: Cost-aware data representation for machine learning on network traffic. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–24, 2021.

[23] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–25, 2019.

[24] Zhiyong Bu, Bin Zhou, Pengyu Cheng, Kecheng Zhang, and Zhen-Hua Ling. Encrypted network traffic classification using deep and parallel network-in-network models. *IEEE Access*, 8:132950–132959, 2020.

[25] Tobias Bühler, Roland Schmid, Sandro Lutz, and Laurent Vanbever. Generating representative, live network traffic out of millions of code repositories. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 1–7, 2022.

[26] Lelio Campanile, Marco Gribaudo, Mauro Iacono, Fiammetta Marulli, and Michele Mastroianni. Computer network simulation with ns-3: A systematic literature review. *Electronics*, 9(2):272, 2020.

[27] Herminio Chavez-Roman and Volodymyr Ponomaryov. Super resolution image generation using wavelet domain interpolation with edge extraction via a sparse representation. *IEEE Geoscience and remote sensing Letters*, 11(10):1777–1781, 2014.

[28] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[29] Yizhou Chen, Xu-Hua Yang, Zihan Wei, Ali Asghar Heidari, Nenggan Zheng, Zhicheng Li, Huiling Chen, Haigen Hu, Qianwei Zhou, and Qiu Guan. Generative adversarial networks in medical image augmentation: A review. *Computers in Biology and Medicine*, 144:105382, 2022.

[30] Zhitang Chen, Ke He, Jian Li, and Yanhui Geng. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *2017 IEEE International conference on big data (big data)*, pages 1271–1276. IEEE, 2017.

[31] Phillip Chlap, Hang Min, Nym Vandenberg, Jason Dowling, Lois Holloway, and Annette Haworth. A review of medical image data augmentation techniques for deep learning applications. *Journal of Medical Imaging and Radiation Oncology*, 65(5):545–563, 2021.

[32] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic data repository at the {WIDE} project. In *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000.

[33] Ana Cholakoska, Martina Shushlevska, Zdravko Todorov, and Danijela Efnusheva. Analysis of machine learning classification techniques for anomaly detection with nsl-kdd data set. In *Data Science and Intelligent Systems: Proceedings of 5th Computational Methods in Systems and Software 2021, Vol. 2*, pages 258–267. Springer, 2021.

[34] Jan Chorowski, Ron J Weiss, Samy Bengio, and Aäron Van Den Oord. Unsupervised speech representation learning using wavenet autoencoders. *IEEE/ACM transactions on audio, speech, and language processing*, 27(12):2041–2053, 2019.

[35] Andrew Chu, Xi Jiang, Shinan Liu, Arjun Bhagoji, Francesco Bronzino, Paul Schmitt, and Nick Feamster. Feasibility of state space models for network traffic generation. In *Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing*, pages 9–17, 2024.

[36] Kimberly C Claffy. Internet traffic characterization. 1995.

[37] Benoit Claise. Cisco systems netflow services export version 9. Technical report, 2004.

[38] Benoît Claise and Brian Trammell. Information Model for IP Flow Information Export (IPFIX). RFC 7012, September 2013.

[39] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[40] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[41] Susu Cui, Bo Jiang, Zhenzhen Cai, Zhigang Lu, Song Liu, and Jian Liu. A session-packets-based encrypted traffic classification using capsule neural networks. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 429–436. IEEE, 2019.

[42] Amit Damri, Mark Last, and Niv Cohen. Towards efficient image-based representation of tabular data. *Neural Computing and Applications*, pages 1–21, 2023.

[43] György Dán, Viktória Fodor, and Gunnar Karlsson. On the effects of the packet size distribution on the packet loss process. *Telecommunication Systems*, 32:31–53, 2006.

[44] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 10041–10071. PMLR, 21–27 Jul 2024.

[45] François De Keersmaeker, Yinan Cao, Gorby Kabasele Ndonda, and Ramin Sadre. A survey of public iot datasets for network security research. *IEEE Communications Surveys & Tutorials*, 2023.

[46] Tcpreplay Developers. Tcpreplay. https://tcpreplay.appneta.com/, 2023.

[47] Christian Dewes, Arne Wichmann, and Anja Feldmann. An analysis of internet chat systems. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 51–64, 2003.

[48] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018.

[49] Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with gans. 2018.

[50] Let's Encrypt. Let's encrypt stats, 2024. Accessed: 2024.

[51] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

[52] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[53] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[54] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher R é. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.

[55] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[56] Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10696–10706, 2022.

[57] Qiu Guan, Yizhou Chen, Zihan Wei, Ali Asghar Heidari, Haigen Hu, Xu-Hua Yang, Jianwei Zheng, Qianwei Zhou, Huiling Chen, and Feng Chen. Medical image augmentation for lesion detection using a texture-constrained multichannel progressive gan. *Computers in Biology and Medicine*, 145:105444, 2022.

[58] Selim Gurun and Boleslaw K Szymanski. Automating internet routing behavior analysis using public www traceroute services. In *Managing QoS in Multimedia Networks and Services: IEEE/IFIP TC6—WG6. 4 & WG6. 6 Third International Conference on Management of Multimedia Networks and Services (MMNS'2000) September 25–28, 2000, Fortaleza, Ceará, Brazil*, pages 47–59. Springer, 2000.

[59] Craig Gutterman, Katherine Guo, Sarthak Arora, Xiaoyang Wang, Les Wu, Ethan Katz-Bassett, and Gil Zussman. Requet: Real-time qoe detection for encrypted youtube traffic. In *Proceedings of the 10th ACM Multimedia Systems Conference*, pages 48–59, 2019.

[60] Guy Harris and Michael Richardson. PCAP Capture File Format. Internet-Draft draft-ietf-opsawg-pcap-06, Internet Engineering Task Force, September 2025. Work in Progress.

[61] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. Ieee, 2008.

[62] Hongyu Hè and Maria Apostolaki. Just-in-time logic enforcement.

[63] Hongyu Hè, Minhao Jin, and Maria Apostolaki. Learning constraints directly from network data. *arXiv preprint arXiv:2506.23964*, 2025.

[64] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[65] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[66] Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *The Journal of Machine Learning Research*, 23(1):2249–2281, 2022.

[67] Paul E. Hoffman and Patrick McManus. DNS Queries over HTTPS (DoH). RFC 8484, October 2018.

[68] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New directions in automated traffic analysis. CCS '21, page 3366–3383, New York, NY, USA, 2021. Association for Computing Machinery.

[69] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 3366–3383, 2021.

[70] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[71] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, May 2016.

[72] Kyle Jamieson and Hari Balakrishnan. Ppr: Partial packet recovery for wireless networks. *ACM SIGCOMM Computer Communication Review*, 37(4):409–420, 2007.

[73] Xi Jiang and Noah Apthorpe. Automating internet of things network traffic collection with robotic arm interactions. *arXiv preprint arXiv:2110.00060*, 2021.

[74] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. Netdiffusion: Network data augmentation through protocol-constrained traffic generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 8(1):1–32, 2024.

[75] Xi Jiang, Shinan Liu, Saloua Naama, Francesco Bronzino, Paul Schmitt, and Nick Feamster. Ac-dc: Adaptive ensemble classification for network traffic identification. *arXiv preprint arXiv:2302.11718*, 2023.

[76] Xi Jiang, Shinan Liu, Saloua Naama, Francesco Bronzino, Paul Schmitt, and Nick Feamster. Jiti: Dynamic model serving for just-in-time traffic inference. *Proceedings of the ACM on Networking*, 3(CoNEXT4):1–24, 2025.

[77] Xi Jiang, Saloua Naama Shinan Liu, Francesco Bronzino, Paul Schmitt, and Nick Feamster. Towards designing robust and efficient classifiers for encrypted traffic in the modern internet.

[78] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[79] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10124–10134, 2023.

[80] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 229–240, 2005.

[81] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

[82] S Kavitha, N Uma Maheswari, et al. Network anomaly detection for nsl-kdd dataset using deep learning. *Information Technology in Industry*, 9(2):821–827, 2021.

[83] Anthony Kenyon, Lipika Deka, and David Elizondo. Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets. *Computers & Security*, 99:102022, 2020.

[84] Myung-Sup Kim, Young J Won, and James W Hong. Characteristic analysis of internet traffic from the perspective of flows. *Computer Communications*, 29(10):1639–1652, 2006.

[85] Amit Klein and Benny Pinkas. From {IP}{ID} to device {ID} and {KASLR} bypass. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1063–1080, 2019.

[86] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[87] Craig Labovitz, S Iekel-Johnson, D McPherson, J Oberheide, and F Jahanian. Internet traffic and content consolidation. *77th Internet Engineering Task Force*, 2010.

[88] Mathieu Lacage and Thomas R Henderson. Yet another network simulator. In *Proceedings of the 2006 Workshop on ns-3*, pages 12–es, 2006.

[89] Tanja Lang, Grenville Armitage, Phillip Branch, and Hwan-Yi Choo. A synthetic traffic model for half-life. In *Australian Telecommunications Networks & Applications Conference*, volume 2003, 2003.

[90] Tanja Lang, Philip Branch, and Grenville Armitage. A synthetic traffic model for quake3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 233–238, 2004.

[91] Jianfeng Li, Hao Zhou, Shuohan Wu, Xiapu Luo, Ting Wang, Xian Zhan, and Xiaobo Ma. {FOAP}:{Fine-Grained}{Open-World} android app fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1579–1596, 2022.

[92] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proceedings of the ACM Web Conference 2022*, WWW '22. ACM, April 2022.

[93] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483, 2020.

[94] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[95] Weixing Liu, Bin Luo, and Jun Liu. Synthetic data augmentation using multiscale attention cyclegan for aircraft detection in remote sensing images. *IEEE Geoscience and Remote Sensing Letters*, 19:1–5, 2021.

[96] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.

[97] Qianli Ma, Wei Huang, Yanliang Jin, and Jianhua Mao. Encrypted traffic classification based on traffic reconstruction. In *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 572–576. IEEE, 2021.

[98] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[99] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 229–244, 2021.

[100] Matthew V Mahoney. Network traffic anomaly detection based on packet bytes. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 346–350, 2003.

[101] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[102] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. Profiliot: A machine learning approach for iot device identification based on network traffic analysis. In *Proceedings of the symposium on applied computing*, pages 506–509, 2017.

[103] Sebastian Meister, Nantwin Möller, Jan Stüve, and Roger M Groves. Synthetic image data augmentation for fibre layup inspection processes: Techniques to enhance the data set. *Journal of Intelligent Manufacturing*, 32:1767–1789, 2021.

[104] Xuying Meng, Chungang Lin, Yequan Wang, and Yujun Zhang. Netgpt: Generative pretrained transformer for network traffic. *arXiv preprint arXiv:2304.09513*, 2023.

[105] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018.

[106] Anthony Moi and Nicolas Patry. HuggingFace's Tokenizers, April 2023.

[107] Nour Moustafa, Jiankun Hu, and Jill Slay. A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, 128:33–55, 2019.

[108] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.

[109] Prashil Negandhi, Yash Trivedi, and Ramchandra Mangrulkar. Intrusion detection system using random forest on the nsl-kdd dataset. In *Emerging Research in Computing, Information, Communication and Applications: ERCICA 2018, Volume 2*, pages 519–531. Springer, 2019.

[110] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[111] Santosh Kumar Nukavarapu, Mohammed Ayyat, and Tamer Nadeem. Miragenet-towards a gan-based framework for synthetic network traffic generation. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 3089–3095. IEEE, 2022.

[112] Kushagra Pandey, Avideep Mukherjee, Piyush Rai, and Abhishek Kumar. Diffusevae: Efficient, controllable and high-fidelity generation from low-dimensional latents. *arXiv preprint arXiv:2201.00308*, 2022.

[113] Vern Paxson. Empirically derived analytic models of wide-area tcp connections. *IEEE/ACM transactions on Networking*, 2(4):316–336, 1994.

[114] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[115] Jian Qu, Xiaobo Ma, and Jianfeng Li. Trafficgpt: Breaking the token barrier for efficient long traffic analysis and generation. *arXiv preprint arXiv:2403.05822*, 2024.

[116] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[117] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. *URL https://arxiv.org/abs/2204.06125*, 7, 2022.

[118] M Mazhar Rathore, Awais Ahmad, and Anand Paul. Real time intrusion detection system for ultra-high-speed big data environments. *The Journal of Supercomputing*, 72:3489–3510, 2016.

[119] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017.

[120] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019.

[121] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019.

[122] R Rohith, Minal Moharir, G Shobha, et al. Scapy-a powerful interactive packet manipulation program. In *2018 international conference on networking, embedded and wireless systems (ICNEWS)*, pages 1–5. IEEE, 2018.

[123] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[124] Sujan Chandra Roy, Md Murad Ali, and Md Ashraful Islam. Analysis the effect of transmission cost on ttl and number of nodes in social aware routing protocols. *group*, 20(40):60–80, 2019.

[125] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

[126] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in Neural Information Processing Systems*, 35:25278–25294, 2022.

[127] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[128] Vikash Sehwag, Caner Hazirbas, Albert Gordo, Firat Ozgenel, and Cristian Canton. Generating high fidelity data from low-density regions using diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11492–11501, 2022.

[129] Colleen Shannon, David Moore, and K Claffy. Characteristics of fragmented ip traffic on internet links. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 83–97, 2001.

[130] Tal Shapira and Yuval Shavitt. Flowpic: Encrypted internet traffic classification is as easy as image recognition. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 680–687. IEEE, 2019.

[131] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[132] Taveesh Sharma, Tarun Mangla, Arpit Gupta, Junchen Jiang, and Nick Feamster. Estimating webrtc video qoe metrics without using application headers. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, IMC '23, page 485–500, New York, NY, USA, 2023. Association for Computing Machinery.

[133] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016.

[134] Hoo-Chang Shin, Neil A Tenenholtz, Jameson K Rogers, Christopher G Schwarz, Matthew L Senjem, Jeffrey L Gunter, Katherine P Andriole, and Mark Michalski. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In *Simulation and Synthesis in Medical Imaging: Third International Workshop, SASHIMI 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Proceedings 3*, pages 1–11. Springer, 2018.

[135] shramos. shramos/pcap-splitter. https://github.com/shramos/pcap-splitter, 2019.

[136] Pallavi Singhal, Rajeev Mathur, and Himani Vyas. State of the art review of network traffic classification based on machine learning approach. *International Journal of Computer Applications*, 975:8887, 2013.

[137] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[138] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[139] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[140] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[141] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[142] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon: a flow-level traffic generator for router and network tests. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):392–392, 2004.

[143] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[144] Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-nice-mc: Adversarial training for mcmc. *Advances in Neural Information Processing Systems*, 30, 2017.

[145] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[146] Jonathan Stone and Craig Partridge. When the crc and tcp checksum disagree. *ACM SIGCOMM computer communication review*, 30(4):309–319, 2000.

[147] Mark Sullivan and Andrew Heybey. A system for managing large databases of network traffic. In *Proceedings of USENIX*, 1998.

[148] Boyu Sun, Wenyuan Yang, Mengqi Yan, Dehao Wu, Yuesheng Zhu, and Zhiqiang Bai. An encrypted traffic classification method combining graph convolutional network and autoencoder. In *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2020.

[149] Matthew Swann, Joseph Rose, Gueltoum Bendiab, Stavros Shiaeles, and Nick Savage. Tools for network traffic generation–a quantitative comparison. *arXiv preprint arXiv:2109.02760*, 2021.

[150] Jonti Talukdar, Ayon Biswas, and Sanchit Gupta. Data augmentation on synthetic images for transfer learning using deep cnns. In *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 215–219. IEEE, 2018.

[151] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. Ieee, 2009.

[152] Rajesh Thomas and Deepa Pavithran. A survey of intrusion detection models based on nsl-kdd data set. *2018 Fifth HCT Information Technology Trends (ITT)*, pages 286–291, 2018.

[153] Runzhi Tian, Yongyi Mao, and Richong Zhang. Learning vae-lda models with rounded reparameterization trick. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1315–1325, 2020.

[154] Yves Vanaubel, Jean-Jacques Pansiot, Pascal Mérindol, and Benoit Donnet. Network fingerprinting: Ttl-based router signatures. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 369–376, 2013.

[155] ARi Vasudevan, E Harshini, and S Selvakumar. Ssenet-2011: a network intrusion detection system dataset and its comparison with kdd cup 99 dataset. In *2011 second asian himalayas international conference on internet (AH-ICI)*, pages 1–5. IEEE, 2011.

[156] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[157] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antô nio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[158] Kashi Venkatesh Vishwanath and Amin Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, 2009.

[159] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

[160] Maonan Wang, Kangfeng Zheng, Dan Luo, Yanqing Yang, and Xiujuan Wang. An encrypted traffic classification framework based on convolutional neural networks and stacked autoencoders. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 634–641. IEEE, 2020.

[161] Xin Wang, Shinji Takaki, Junichi Yamagishi, Simon King, and Keiichi Tokuda. A vector quantized variational autoencoder (vq-vae) autoregressive neural $f\_0$ model for statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:157–170, 2019.

[162] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32, 2019.

[163] Shengzhe Xu, Manish Marwah, Martin Arlitt, and Naren Ramakrishnan. Stan: Synthetic network traffic generation with generative neural models. In *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2*, pages 3–29. Springer, 2021.

[164] Haipeng Yao, Chong Liu, Peiying Zhang, Sheng Wu, Chunxiao Jiang, and Shui Yu. Identification of encrypted traffic through attention mechanism based long short term memory. *IEEE Transactions on Big Data*, 2019.

[165] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 458–472, 2022.

[166] Sebastian Zander, David Kennedy, and Grenville Armitage. Kute a high performance kernel-based udp traffic engine. 2005.

[167] Bowen Zhang, Shuyang Gu, Bo Zhang, Jianmin Bao, Dong Chen, Fang Wen, Yong Wang, and Baining Guo. Styleswin: Transformer-based gan for high-resolution image generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11304–11314, 2022.

[168] Chenshuang Zhang, Chaoning Zhang, Mengchun Zhang, and In So Kweon. Text-to-image diffusion model in generative ai: A survey. *arXiv preprint arXiv:2303.07909*, 2023.

[169] Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. *arXiv preprint arXiv:2310.09656*, 2023.

[170] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.

[171] Shiyuan Zhang, Tong Li, Depeng Jin, and Yong Li. Netdiff: A service-guided hierarchical diffusion model for network flow trace generation. *Proceedings of the ACM on Networking*, 2(CoNEXT3):1–21, 2024.

[172] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the characteristics and origins of internet flow rates. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 309–322, 2002.

[173] Shihao Zhao, Dongdong Chen, Yen-Chun Chen, Jianmin Bao, Shaozhe Hao, Lu Yuan, and Kwan-Yee K. Wong. Uni-ControlNet: All-in-One Control to Text-to-Image Diffusion Models, May 2023. arXiv:2305.16322 [cs].

[174] Weiping Zheng, Jianhao Zhong, Qizhi Zhang, and Gansen Zhao. Mtt: an model for encrypted network traffic classification using multi-task transformer. *Applied Intelligence*, pages 1–16, 2022.

[175] Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno Yepes. Image-based table recognition: data, model, and evaluation. In *European conference on computer vision*, pages 564–580. Springer, 2020.

[176] Yitan Zhu, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A Evrard, James H Doroshow, and Rick L Stevens. Converting tabular data into images for deep learning with convolutional neural networks. *Scientific reports*, 11(1):11325, 2021.

[177] Yuanzhi Zhu, Zhaohai Li, Tianwei Wang, Mengchao He, and Cong Yao. Conditional Text Image Generation With Diffusion Models.

# APPENDIX A

# ADDITIONAL DETAILS FOR CHAPTER 3

## A.1 Ablation Study - Fine-tuning and ControlNet

We evaluate the design choices of NetDiffusion, focusing on its fine-tuning and controlled generation aspects. Our experiments involve three configurations: (1) the non-fine-tuned Stable Diffusion 1.5 model, (2) the fine-tuned NetDiffusion model without ControlNet, and (3) the fine-tuned NetDiffusion model with ControlNet (excluding post-generation correction heuristic). All models are given the same prompt 'pixelated network traffic, type-0', representing Amazon traffic, to compare their generation outcomes. The Stable Diffusion 1.5 model used in Figure A.1 is employed in its original, off-the-shelf form, without any fine-tuning. The text prompt was directly inputted into the model as is. This non-fine-tuned model fails to meet our specific generation goals, producing outputs unrelated to pcap-based network traffic when prompted, thus demonstrating the necessity of fine-tuning. The fine-tuned NetDiffusion model shows significant improvement, generating results that more closely resemble real traffic. However, due to the intrinsic characteristics of diffusion models, as discussed in Section 3.3.2, its output lacks structural fidelity, specifically in mimicing the protocol and header value distributions. In contrast, incorporating controlled generation via ControlNet in NetDiffusion achieves image representations that more accurately reflects real traffic. These findings underscore the critical role of both fine-tuning and controlled generation in our framework.

## A.2 Generation Variation Assessment

We assess NetDiffusion's generation consistency by generating multiple synthetic flow instances using identical prompts ('pixelated network traffic, type-0', representing Amazon

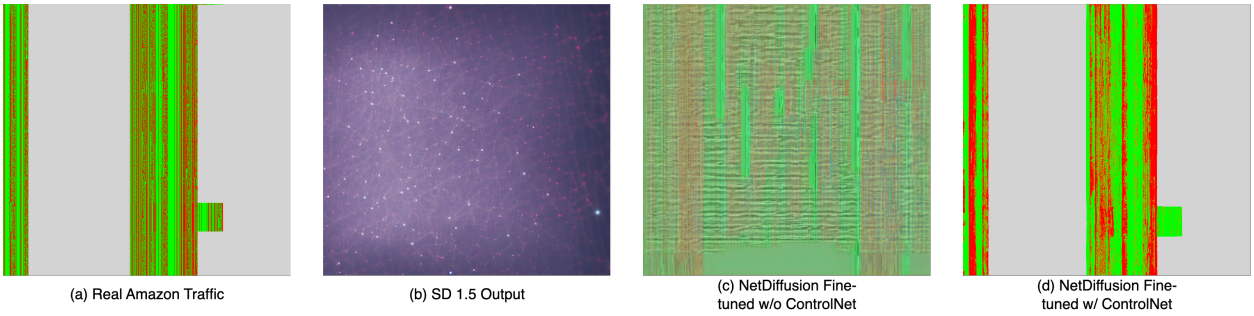



**Figure A.1:** Comparison of (a) real traffic with outputs from (b) non-fine-tuned SD 1.5, (c) fine-tuned NetDiffusion w/o ControlNet, and (d) fine-tuned NetDiffusion w/ ControlNet.

traffic) and ControlNet inputs and compare the generated instances. For every pair of generated flows, we compared each bit in the packet headers. If a bit differs between the two packets at the same position, it is counted as a variation. For each header bit position $b$, the difference percentage was calculated using the formula:

$$\text{Diff}_{\%}(b) = \left( \frac{\text{Number of differing bits at position } b}{\text{Total number of packets}} \right) \times 100$$

The average difference percentage for each aggregated bit position $B$ was then calculated by averaging across all pcap flow pairs:

$$\text{AvgDiff}_{\%}(B) = \text{Average of Diff}_{\%}(B) \text{ across all pcap flow pairs}$$

Our result reveals that the resultant synthetic traffic, exemplified by Amazon flow, exhibits an average variation of approximately 12.66%. Table A.1 show cases the top 20 header fields with the highest average percentage of difference across instances of NetDiffusion generated Amazon traffic.

| Header Field | Difference |
|---|---|
| ipv4_src | 50.34% |
| ipv4_id | 50.27% |
| ipv4_dst | 50.20% |
| tcp_ackn | 50.14% |
| ipv4_cksum | 49.95% |
| tcp_seq | 49.79% |
| tcp_sprt | 49.28% |
| tcp_dprt | 49.21% |
| ipv4_ttl | 34.90% |
| tcp_cksum | 32.36% |
| tcp_cwr | 26.49% |
| tcp_wsize | 22.44% |
| tcp_fin | 21.94% |
| tcp_ns | 21.46% |
| tcp_opt | 20.16% |
| ipv4_tl | 18.62% |
| tcp_rst | 15.56% |
| tcp_res | 14.98% |
| tcp_doff | 14.69% |
| tcp_psh | 13.99% |

**Table A.1:** Ranking of top 20 header fields by average percentage of difference across NetDiffusion generated Amazon traffic.

# APPENDIX B

# ADDITIONAL DETAILS FOR CHAPTER 4

## B.1 Comprehensive Results on Downstream Utilization

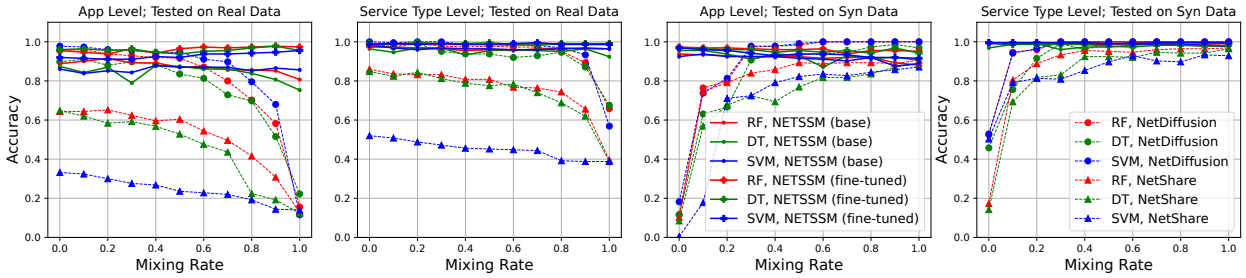
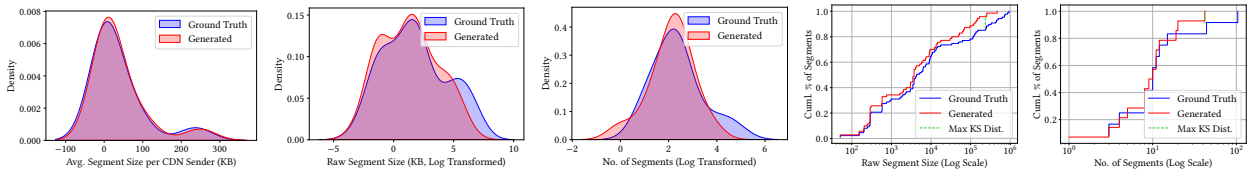



**Figure B.1:** Comparative ML performance across different model choices with mixed training data proportions.

## B.2 Additional Video Streaming Segment Results

The scenarios shown in all figures below have the following ground truth data bit rates: (1) 554 kbps, (2) 1,366 kbps, (3) 2,726 kbps, (4) 2,460 kbps, (5) 1,361 kbps, and (6) 1,450 kbps.




**(a)** **(b)** **(c)** **(d)** **(e)**

**Figure B.2: Distributions for downloaded segments.** KDE (log-transformed) and ECDF (non-log-transformed, displayed on log scale) plots for the number and size of downloaded segments sent per sender. The ground truth trace has a data bit rate of 1,366 kbps. NetSSM's distributions overlap significantly with the real data.

**Downloaded Segment Sizes.** Figure B.2 shows applying kernel density estimation (KDE) to the average segment sizes per sender (B.2a) and log-transformed sizes of all raw segment

sizes (B.2b), and the empirical cumulative distribution function (ECDF) for raw segment sizes (B.2d) for a ground truth and corresponding generated trace. All KDE plots are created using a Gaussian kernel with (ground truth, generated) bandwidths of$(40.62, 39.08)$, and $(1.07, 0.99)$ for Figures B.2a, and B.2b respectively, chosen using Scott's rule of thumb in the Python `scipy` library [127, 157]. These figures well illustrate the similarity in downloaded segment sizes, where Observing Figures B.2a and B.2b, clear overlap exists between the segment sizes of the ground truth and synthetic data, even when considering instances of larger tail values. Similarly, in the Figure B.2d ECDF, the generated trace segment sizes overlap with the ground truth, as illustrated by similar magnitudes in the 25th, 50th, and 75th quartiles: $(580.00, 4344.00, 41564.00)$ KB for ground truth and $(369.50, 3721.00, 14349.50)$ KB for generated, respectively. As such, we see that NetSSM generates traces with similar size magnitudes across all segments, and with small and medium-sized segments are with similar absolute size, as compared to the ground truth.

Figure B.3 shows additional visualizations for both the average downloaded segment sizes and raw downloaded segments sizes. Specifically: (a) KDE plots for the average downloaded segment sizes per sender, (b) KDE plots for the log-transformed average downloaded segment sizes per sender, (c) KDE plots for the sizes of all downloaded segments and (d) KDE plots for the log-transformed sizes of all downloaded segments.

**Number of Downloaded Segments.** We also evaluate the number of segments downloaded both in NetSSM's synthetic traces and in the ground truth. Similar to evaluation of segments' sizes, we find that NetSSM produces data that closely aligns with the ground truth traffic. In Figures B.2c and B.2e, there again exists clear overlap in the KDE plots for number of segments downloaded between the ground truth and synthetic data, though it appears NetSSM's traces may not completely capture the tail end cases of higher volume senders. The quartile values from the Figure B.2e ECDF further supports the overlap, with only small deltas between the ground truth $(7.00, 10.00, 12.75)$ and generated $(5.75, 9.50, 11.75)$ number of segments downloaded, respectively.

## B.3 Semantic Similarity

Figure 4 presents additional examples plotting the throughput in kilobits/100 packet slice for Netflix and YouTube traces. Table B.1 shows the generation hyperparameters used in Section 4.4.3.

**Table B.1:** Generation parameters for NetSSM single and multi-flow models. RP: repetition penalty; T: temperature; MP: min-p; TK: top-k; TP: top-p.

| **Dataset** | **RP** | **T** | **MP** | **TK** | **TP** |
|---|---|---|---|---|---|
| Netflix | 1.8 | 0.15 | 0 | 25 | 0.9 |
| YouTube | 1.8 | 0.75 | 0 | 25 | 0.9 |

## B.4 Memorization Analysis

**Table B.2:** NetSSM Memorization Analysis Overview. Table B.2a reports packet-level memorization and diversity metrics, while Table B.2b lists header fields with the largest real–synthetic changes.

**(a)** Packet-Level Memorization and Diversity Metrics

| Metric | Value | % |
|---|---|---|
| **Basic Comparison** | | |
| Identical Packets | – | 2.35 |
| Differing Bytes | – | 22.27 |
| Avg diff. per Packet | 44.78 bytes | – |
| **Intra-Set Diversity** | | |
| *Synthetic Packets:* | | |
| Avg Pairwise Dist | 0.359 (norm.) | – |
| Std. Dev | 0.206 (norm.) | – |
| *Real Packets:* | | |
| Avg Pairwise Dist | 0.680 (norm.) | – |
| Std. Dev | 0.250 (norm.) | – |
| **Diversity Ratio (Syn/Real)** | 0.528 (norm.) | – |
| **Nearest-Neighbor Memorization** | | |
| Overall Mean Dist | 0.186 (norm.) | – |
| Median Dist | 0.170 (norm.) | – |
| Std. Dev | 0.085 (norm.) | – |
| Min / Max Dist | 0.000 / 0.543 | – |
| *Thresholds:* | | |
| Within 5% | – | 3.83 |
| Within 10% | – | 10.67 |
| Within 15% | – | 40.48 |
| Within 20% | – | 66.82 |
| **Position-Aware Memorization** | | |
| *Packets 0–10:* | | |
| Avg Dist | 0.128 ± 0.041 | – |
| *Packets 10–50:* | | |
| Avg Dist | 0.175 ± 0.052 | – |
| *Packets 50–100:* | | |
| Avg Dist | 0.223 ± 0.083 | – |

**(b)** Header Fields with Largest Avg Change

| Header Field | Avg Change (bytes) |
|---|---|
| TCP_ack | 835.2M |
| TCP_seq | 758.8M |
| TCP_chksum | 19,885 |
| TCP_sport | 12,974 |
| TCP_dport | 12,971 |
| IP_id | 12,122 |
| IP_chksum | 11,436 |
| TCP_window | 2,543 |
| IP_len | 192 |
| TCP_urgptr | 24.68 |
| IP_ttl | 11.97 |
| IP_tos | 5.54 |
| Raw_load | 1.00 |
| Padding_load | 1.00 |
| TCP_options | 0.83 |

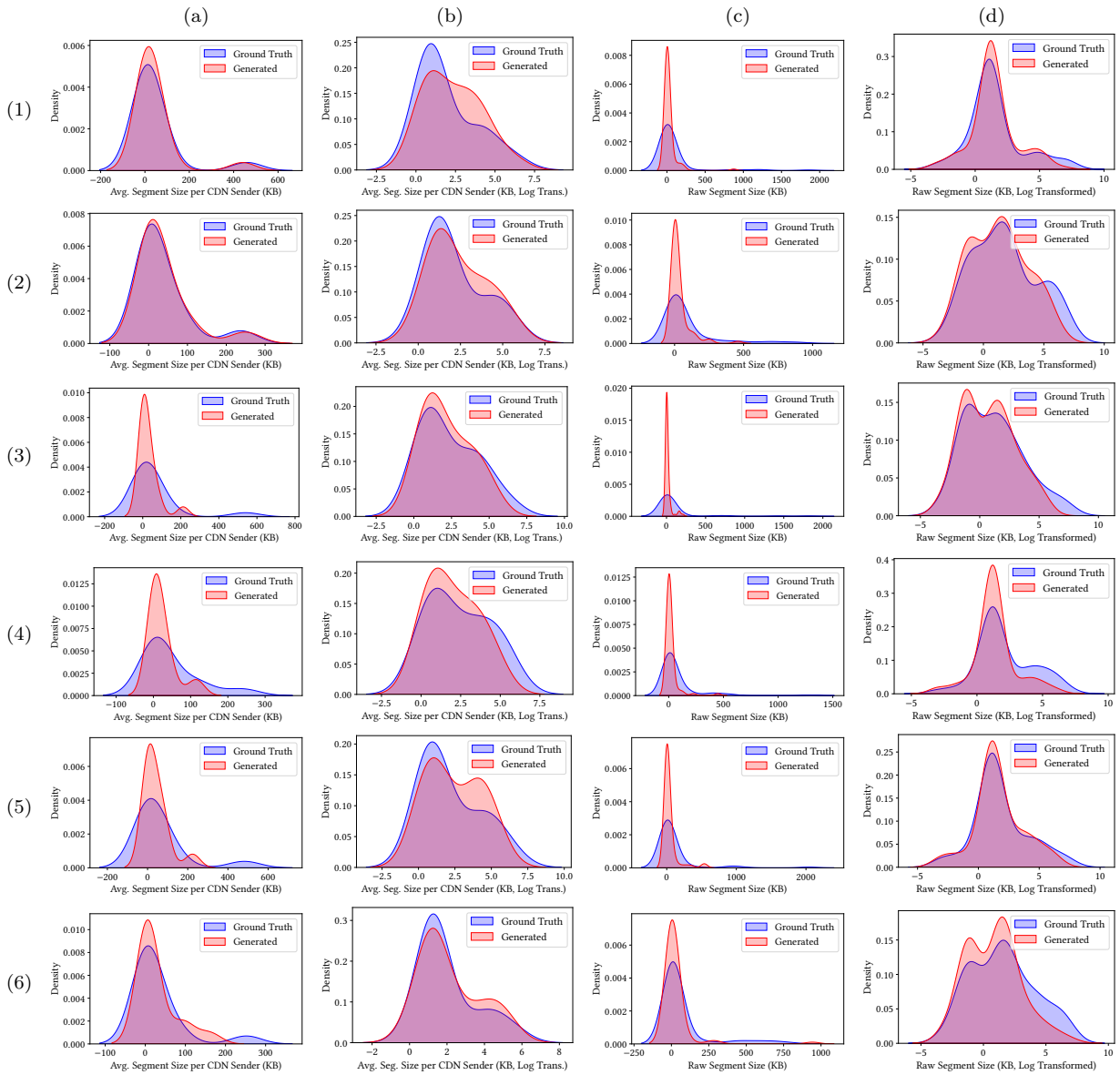



**Figure B.3:** KDE plots for downloaded segment sizes.

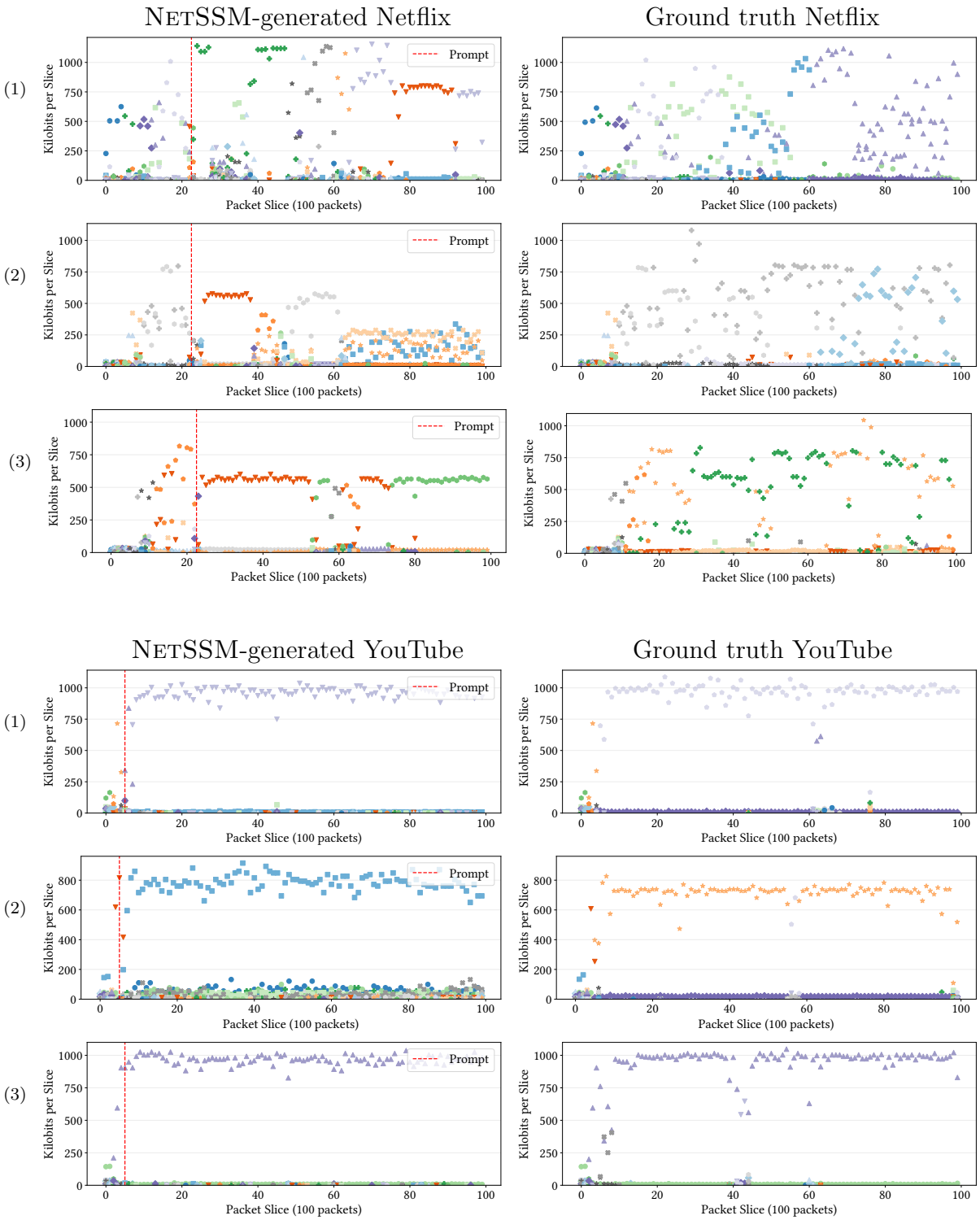



**Figure B.4:** Plots of throughput per flow for NetSSM-generated/ground truth Netflix and YouTube trace pairs.