NetSSM: Multi-Flow and State-Aware Network Trace Generation using State-Space Models

ANDREW CHU*, University of Chicago, USA
XI JIANG*, University of Chicago, USA
SHINAN LIU, University of Hong Kong, Hong Kong
ARJUN BHAGOJI, IIT Bombay, India
FRANCESCO BRONZINO, École Normale Supérieure de Lyon; Institut Universitaire de France, France
PAUL SCHMITT, California Polytechnic State University, USA
NICK FEAMSTER, University of Chicago, USA

Access to raw network traffic data is essential for many computer networking tasks, from traffic modeling to performance evaluation. Unfortunately, this data is scarce due to high collection costs and governance rules. Previous efforts explore this challenge by generating synthetic network data, but fail to reliably handle multi-flow sessions, struggle to reason about stateful communication in moderate to long-duration network sessions, and lack robust evaluations tied to real-world utility. We propose a new method based on state-space models called NetSSM that generates raw network traffic at the packet-level granularity. Our approach captures interactions between multiple, interleaved flows – an objective unexplored in prior work – and effectively reasons about flow-state in sessions to capture traffic characteristics. NetSSM accomplishes this by training with a context window more than 8× longer, and produces traces up to 78× longer than existing transformer-based raw packet generators. Evaluation results show that NetSSM generates high-fidelity traces that outperform prior efforts in existing benchmarks. We also find that NetSSM's traces have high semantic similarity to real network data regarding compliance with standard protocol requirements and flow and session-level traffic characteristics.

CCS Concepts: • **Networks** → **Network simulations**; • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: State space models, Network trace generation

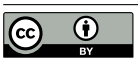
ACM Reference Format:
Andrew Chu, Xi Jiang, Shinan Liu, Arjun Bhagoji, Francesco Bronzino, Paul Schmitt, and Nick Feamster. 2026. NetSSM: Multi-Flow and State-Aware Network Trace Generation using State-Space Models. *Proc. ACM Netw.* 4, CoNEXT1, Article 6 (March 2026), 24 pages. https://doi.org/10.1145/3786289

1 Introduction

There is high demand for representative, scalable network data, driven by applications in security analysis, traffic modeling, and performance evaluation [2, 21, 34, 39, 42]. Unfortunately, acquiring large-scale, high-fidelity network data is difficult due to data governance policies, and high collection costs [1, 10]. In response, methods have been developed to generate synthetic network data

*Equal contribution.

Authors' Contact Information: Andrew Chu, University of Chicago, Chicago, Illinois, USA; Xi Jiang, University of Chicago, Chicago, Illinois, USA; Shinan Liu, University of Hong Kong, Hong Kong, Hong Kong; Arjun Bhagoji, IIT Bombay, Mumbai, India; Francesco Bronzino, École Normale Supérieure de Lyon; Institut Universitaire de France, Lyon, France; Paul Schmitt, California Polytechnic State University, San Luis Obispo, California, USA; Nick Feamster, University of Chicago, Chicago, Illinois, USA.



ACM 2834-5509/2026/3-ART6
https://doi.org/10.1145/3786289

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 [37], application fingerprinting [28]). 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 [46] and NetDiffusion [24], 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.

2 Related Work

Techniques for generating synthetic network data aim to replicate the characteristics of real-world communication between networked devices, either through higher-level traffic attributes about packets or a session, or raw packet captures. Traffic generators can be categorized into two main approaches: traffic attribute generators and raw packet generators.

2.1 Traffic Attribute Generators

Traffic attribute generators use simulation or machine learning to produce higher-level data describing networked communication. Simulation-based approaches were the earliest method for synthesizing network data, using user-defined templates to configure a simulated network (*e.g.,* topology, link specifications, workload), and replaying or emulating communication on this network to produce traffic attributes relevant to the simulated network. Notable efforts in this approach include NS-3 [19], TRex [8], and others [3, 5, 27], which remain popular due to their configurability, versatility, and relative inexpensiveness. Unfortunately, these methods' simulated traffic may not model the variability and unpredictability inherent in actual network conditions [6, 43], and thus may fail to capture the nuances of real-world traffic exchange.

Machine learning-based approaches adopt techniques for time-series forecasting to learn signals in a given continuous stream of input. These models isolate the fine-grained variations in one or more traffic attributes and produce data statistically similar to real-world traffic, further reinforced by offering improved performance when used in downstream ML-based tasks (*e.g.,* service recognition, anomaly detection). Additionally, this data can have arbitrary length, as the generating model learns from only a single or small set of continuous traffic attribute values. Early ML-based traffic attribute generators include Lin *et al.* 's DoppelGANger which uses a generative adversarial network (GAN) to produce sequences of single traffic attribute values [30], and Yin *et al.* 's NetShare which builds on DoppelGANger to output more expressive sets of aggregate traffic attributes (*e.g.,* duration, packet count), or more comprehensive sets of packet-level header field values (*e.g.,* time-to-live [TTL], protocol flags) [46]. Zhang *et al.* 's NetDiff uses both diffusion and transformers to try to better encode patterns in traffic attributes and use this encoding to better inform generation, specifically for mobile network data [48]. One limitation of these models is that when modeling raw packet contents, they only support learning and generating values from

Layer 3 and below (plus transport-layer port numbers) in the OSI model. Thus, they cannot model interactions or attributes in stateful protocols (*e.g.,* TCP).

2.2 Raw Packet Generators

Raw packet generators use simulation or machine learning to output synthetic network traffic in the form of verbose, raw PCAPs. The same simulation-based approaches used to produce traffic attributes can be used to produce raw traces, where the simulated communication between nodes is collected (versus summarized to yield traffic attributes) and written to a trace. Unfortunately, the same shortcoming in expressiveness also exists for these simulators for this output granularity.

Machine learning-based approaches train on raw packet data and generate the byte-level values that comprise the packets of a session. Whereas traffic attribute generators are designed to learn from and capture variations in values over time implicit in a given sequence, raw packet generators learn from and capture the inter- and intra-packet relationships contained in a trace's raw contents, from which traffic attributes can be extracted. Operating at the packet level, these generators can model protocols at any layer. Evaluated under the same metrics, raw packet generators have comparable or better statistical similarity and downstream ML-task performance than traffic attribute generators, and their verbose PCAP format may be more versatile for later analysis and feature extraction. For example, Jiang *et al.* 's NetDiffusion uses a text-to-image diffusion model trained on image representations of network traces to generate images with specific traffic characteristics and are convertible back to PCAP form [24]. Qu *et al.* 's TrafficGPT is a transformer decoder that trains on, and produces tokens corresponding to raw bytes of PCAP traces [35]. A key drawback to existing diffusion and transformer-based raw packet generators is their relatively short limit in training context and output length (*i.e.,* learning from and producing PCAPs with maximum lengths of 1,024 packets for NetDiffusion and $113-128$[1] packets for TrafficGPT), which may fail to capture target events in exchanged communication. Most recently, Chu and Jiang *et al.* proposed using SSMs, specifically Mamba-1, to generate synthetic traces [7]. Our work improves on this effort by using the Mamba-2 architecture (allowing for larger modeled state and faster training), training on longer contexts (100,000 versus 50,000 tokens), producing multi-flow traces (versus only single-flow) and presenting more detailed evaluation of generation quality.

3 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 [9, 12], a line of selective structured SSMs, accommodates these characteristics. In this section, we provide background on SSMs, specifically, the Mamba model (Section 3.1), and compare Mamba against the existing approaches in raw packet generators (Section 3.2).

3.1 Background: State Space Models and Mamba

SSMs are probabilistic graphical models built on the control engineering concept of a state space [26]. 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

[1]Using packet lengths of 94/106 tokens from our evaluation case studies, for TrafficGPT's max generation length of 12,032 tokens.

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 [13, 45]. Follow-up works by Gu *et al.* provide additional improvements to the SSM, improving training efficiency for practical use via convolutional kernel (S4 [14]) and sequence modeling performance via a selection mechanism and a fixed state matrix (Mamba, Mamba-2 [9, 12]).

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 [13], 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).

3.2 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 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 5.3.2), a flow's representative state for downloading audio/video segments may not be reached until after a few hundred or thousand packets.

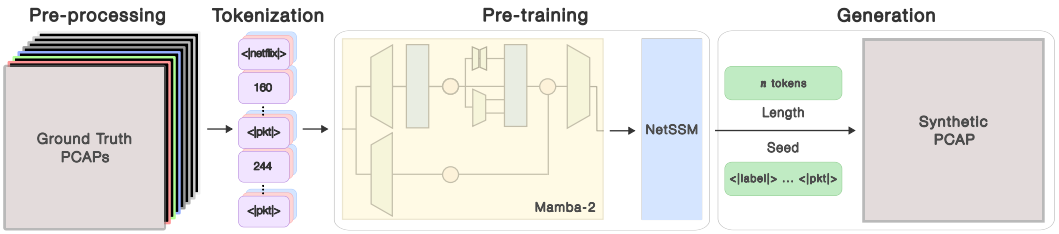


Fig. 1. Overview of the NetSSM pipeline.

4 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 communications, 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 1 provides an overview of the NetSSM pipeline, and we provide details for each pipeline step below.

4.1 Pipeline Overview

4.1.1 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 [16] 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 [33] 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 [29], as a mix of packet-level and flow attributes [35], or created using a tokenization algorithm that may map raw bytes to tokens using logic suited to a different domain (*i.e.,* Word-Piece from NLP) [32]. 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` [38] 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.

4.1.2 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).

4.1.3 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).

4.2 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 6.

5 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

Table 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.* [4]	§5.1, 5.2, 5.4	Video Streaming	4	6.36 GiB	273
	MacMillan *et al.* [31]	§5.1, 5.2, 5.4	Video Conferencing	3	17.36 GiB	339
	Jiang *et al.* [25]	§5.1, 5.2, 5.4	Social Media	3	5.40 GiB	151
Netflix Streaming	Bronzino *et al.* [4]	§5.3, 5.4	Video Streaming	1	216.36 GiB	5,882
YouTube Streaming	Gutterman *et al.* [15]	§5.3	Video Streaming	1	2.06 GiB	619

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.

5.1 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.

5.1.1 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, 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 1. We first pre-process the data using `pcap-splitter` [38], 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 4.1.1. 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 [11, 20, 23] and would be noise our model would not learn from. Next, we create a custom tokenizer following the configuration described in Section 4.1.1, 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 5×10^{-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 4.1.3, producing a corresponding synthetic trace

Table 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 [35]. †Post-generation correction applied.

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 statistics (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 [21]) 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.

5.1.2 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 [24, 30, 35, 46]. 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 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 2 shows only the average JSD for each field, though the TVD and HD are highly similar (±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 [21] 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.

5.2 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).

5.2.1 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 [22]. We extract these features from both the ground truth Multimedia Traffic dataset, and three sets of synthetic data generated by NetSSM, NetDiffusion, and NetShare 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 5.5) 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.

5.2.2 Results. Figure 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 A. We first examine our downstream models performance when tested on completely real data. Figures 2a and 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

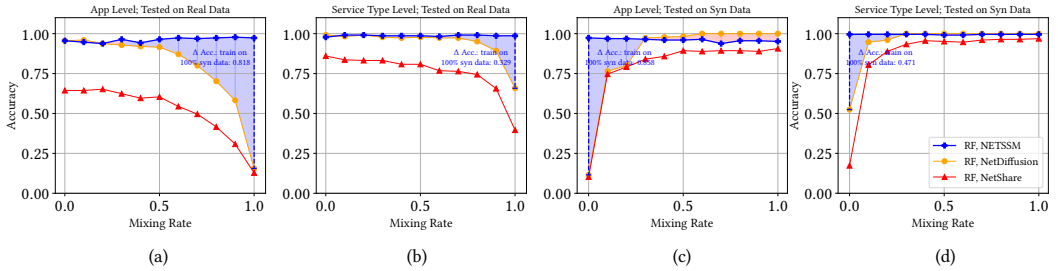


Fig. 2. **Performance of random forest classifiers trained on mixed real/synthetic data.** Models trained on NetSSM data perform best across baselines. Shading denotes the delta between the next best baseline.

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 2c and 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.

5.3 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 5.1.2 (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 5.3.2). 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 6.

5.3.1 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.* [4] to train our Netflix NetSSM model and the traces from Gutterman *et al.* [15] to train our YouTube NetSSM model. Table 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 [40] (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 5.1.1. 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 5.1.1 (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 5.1.1 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 × 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 C 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.

5.3.2 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

Table 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.

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."

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.

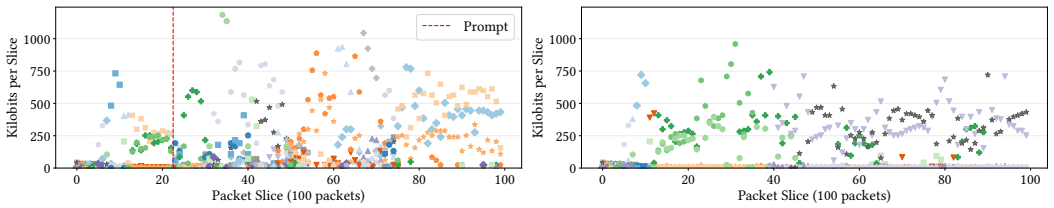
Segment Attributes. We compare the distributions and summary statistics for segments in regard to 1) raw sizes of all downloaded segments 2) average segment size per flow, and 3) number of segments downloaded per flow, for each ground truth and NetSSM-generated trace pair, for each generation granularity and application. Table 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. $\Delta := median\left(\sigma_{GT_i}(\texttt{eval}) - \sigma_{Gen_i}(\texttt{eval})\right)$ 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.

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. Δ 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 Δ is close to, or smaller than the ground truth respectively, while the median Δ 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 Δ. 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 4.1.3). 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 3 visualizes this throughput for both the ground truth (truncated to a matching



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

Fig. 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 3a/3b.

10,000 packets) and generated traces for a sample Netflix trace pair. Appendix Figure 4 contains additional visualizations for either 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.

5.4 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 5.1 and 5.2 and single and multi-flow Netflix streaming traffic from Section 5.3, 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

Table 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.

TCP options. Table 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 [17, 18]. 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 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.

5.5 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 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.

6 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 [41, 47] 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 5.3.2, 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.

7 Conclusion

In this paper, we presented NetSSM, an SSM-based raw packet generator. NetSSM's sequential, stateful architecture enables it to learn from, and produce sessions 8× and 78× longer, respectively, than the current state-of-the-art transformer-based raw packet generator. This allows it to capture key flow-state-dependent session events at both the single and multi-flow session granularities that only manifest after substantial setup. NetSSM outperforms all previous generators in measures of statistical similarity and as measured by the performance of downstream ML-for-networking models trained on NetSSM data. We additionally pose a new evaluation of semantic similarity that attempts to better reason about the empirical, practical similarities between synthetic output and real-world network data. We find that NetSSM can capture complex application dynamics of multi-flow networked communication. Finally, we verify that NetSSM's generated traces largely reproduce the TCP-adherent, and anomalous behaviors found in real traffic data. This paper does not raise any ethical concerns.

Acknowledgments

We thank our shepherd Alessandro Finamore and our anonymous reviewers for their feedback and suggestions. This work has been supported by grants from the Agence Nationale de la Recherche (project no. ANR-21-CE94-0001 [MINT]), the National Science Foundation (grant nos. CNS-2334996 and CNS-2319603), and the France and Chicago Collaborating in The Sciences program.

References

[1] Sebastian Abt and Harald Baier. 2014. 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)*. IEEE, 40–55.

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

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

[4] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. 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 (2019), 1–25.

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

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

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

[8] ciscotrex2023 2024. The CISCO TRex Tool. https://trex-tgn.cisco.com/. [Online; accessed 31-May-2024].

[9] Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 10041–10071. https://proceedings.mlr.press/v235/dao24a.html

[10] François De Keersmaeker, Yinan Cao, Gorby Kabasele Ndonda, and Ramin Sadre. 2023. A Survey of Public IoT Datasets for Network Security Research. *IEEE Communications Surveys & Tutorials* (2023).

[11] Let's Encrypt. 2024. Let's Encrypt Stats. https://letsencrypt.org/stats/ Accessed: 2024.

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

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

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

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

[16] Guy Harris and Michael Richardson. 2025. *PCAP Capture File Format*. Internet-Draft draft-ietf-opsawg-pcap-06. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcap/06/ Work in Progress.

[17] Hongyu Hè and Maria Apostolaki. [n. d.]. Just-in-Time Logic Enforcement. ([n. d.]).

[18] Hongyu Hè, Minhao Jin, and Maria Apostolaki. 2025. Learning Constraints Directly from Network Data. *arXiv preprint arXiv:2506.23964* (2025).

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

[20] Paul E. Hoffman and Patrick McManus. 2018. DNS Queries over HTTPS (DoH). RFC 8484. https://doi.org/10.17487/RFC8484

[21] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2021. New Directions in Automated Traffic Analysis *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 3366–3383. https://doi.org/10.1145/3460120.3484758

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

[23] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. 2016. Specification for DNS over Transport Layer Security (TLS). RFC 7858. https://doi.org/10.17487/RFC7858

[24] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2024. NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–32.

[25] Xi Jiang, Shinan Liu, Saloua Naama, Francesco Bronzino, Paul Schmitt, and Nick Feamster. 2025. JITI: Dynamic Model Serving for Just-in-Time Traffic Inference. *Proceedings of the ACM on Networking* 3, CoNEXT4 (2025), 1–24.

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

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

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

[29] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. 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. https://doi.org/10.1145/3485447.3512217

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

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

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

[33] Anthony Moi and Nicolas Patry. 2023. *HuggingFace's Tokenizers*. https://github.com/huggingface/tokenizers

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

[35] Jian Qu, Xiaobo Ma, and Jianfeng Li. 2024. TrafficGPT: Breaking the Token Barrier for Efficient Long Traffic Analysis and Generation. *arXiv preprint arXiv:2403.05822* (2024).

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

[37] Taveesh Sharma, Tarun Mangla, Arpit Gupta, Junchen Jiang, and Nick Feamster. 2023. Estimating WebRTC Video QoE Metrics Without Using Application Headers. In *Proceedings of the 2023 ACM on Internet Measurement Conference* (Montreal QC, Canada) *(IMC '23)*. Association for Computing Machinery, New York, NY, USA, 485–500. https://doi.org/10.1145/3618257.3624828

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

[39] Pallavi Singhal, Rajeev Mathur, and Himani Vyas. 2013. State of the Art Review of Network Traffic Classification based on Machine Learning Approach. *International Journal of Computer Applications* 975 (2013), 8887.

[40] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* 18, 4 (2011), 62–67. https://doi.org/10.1109/MMUL.2011.71

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

[42] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*. 305–316. https://doi.org/10.1109/SP.2010.25

[43] Matthew Swann, Joseph Rose, Gueltoum Bendiab, Stavros Shiaeles, and Nick Savage. 2021. Tools for Network Traffic Generation–A Quantitative Comparison. *arXiv preprint arXiv:2109.02760* (2021).

[44] 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. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

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

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

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

[48] Shiyuan Zhang, Tong Li, Depeng Jin, and Yong Li. 2024. NetDiff: A Service-Guided Hierarchical Diffusion Model for Network Flow Trace Generation. *Proceedings of the ACM on Networking* 2, CoNEXT3 (2024), 1–21.

A Comprehensive Results on Downstream Utilization

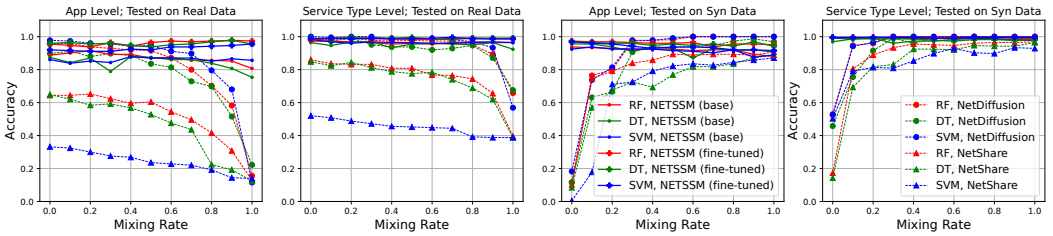
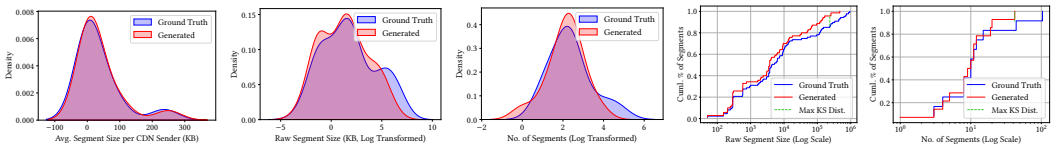


Fig. 1. Comparative ML performance across different model choices with mixed training data proportions.

B 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)

Fig. 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 2 shows applying kernel density estimation (KDE) to the average segment sizes per sender (2a) and log-transformed sizes of all raw segment sizes (2b), and the empirical cumulative distribution function (ECDF) for raw segment sizes (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 2a, and 2b respectively, chosen using Scott's rule of thumb in the Python `scipy` library [36, 44]. These figures well illustrate the similarity in downloaded segment sizes, where Observing Figures 2a and 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 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 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.

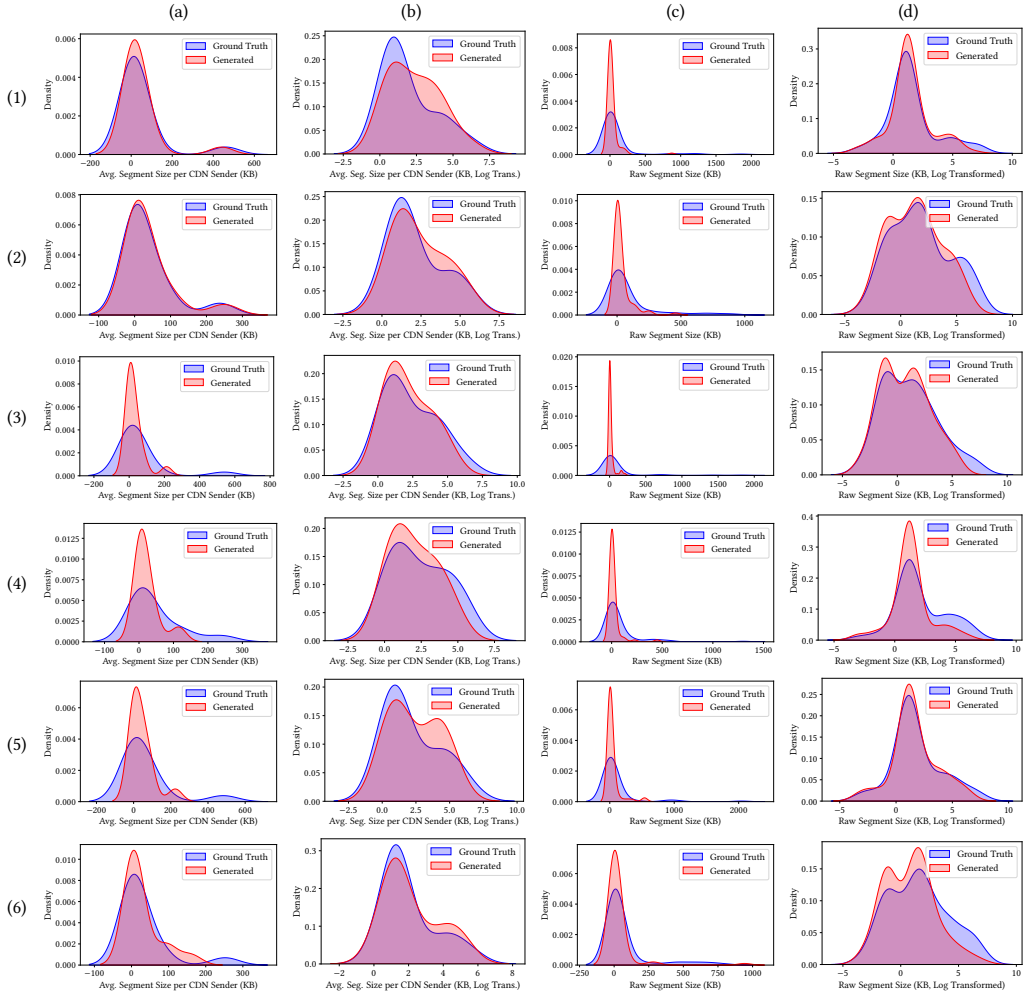


Fig. 3. KDE plots for downloaded segment sizes.

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 2c and 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 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.

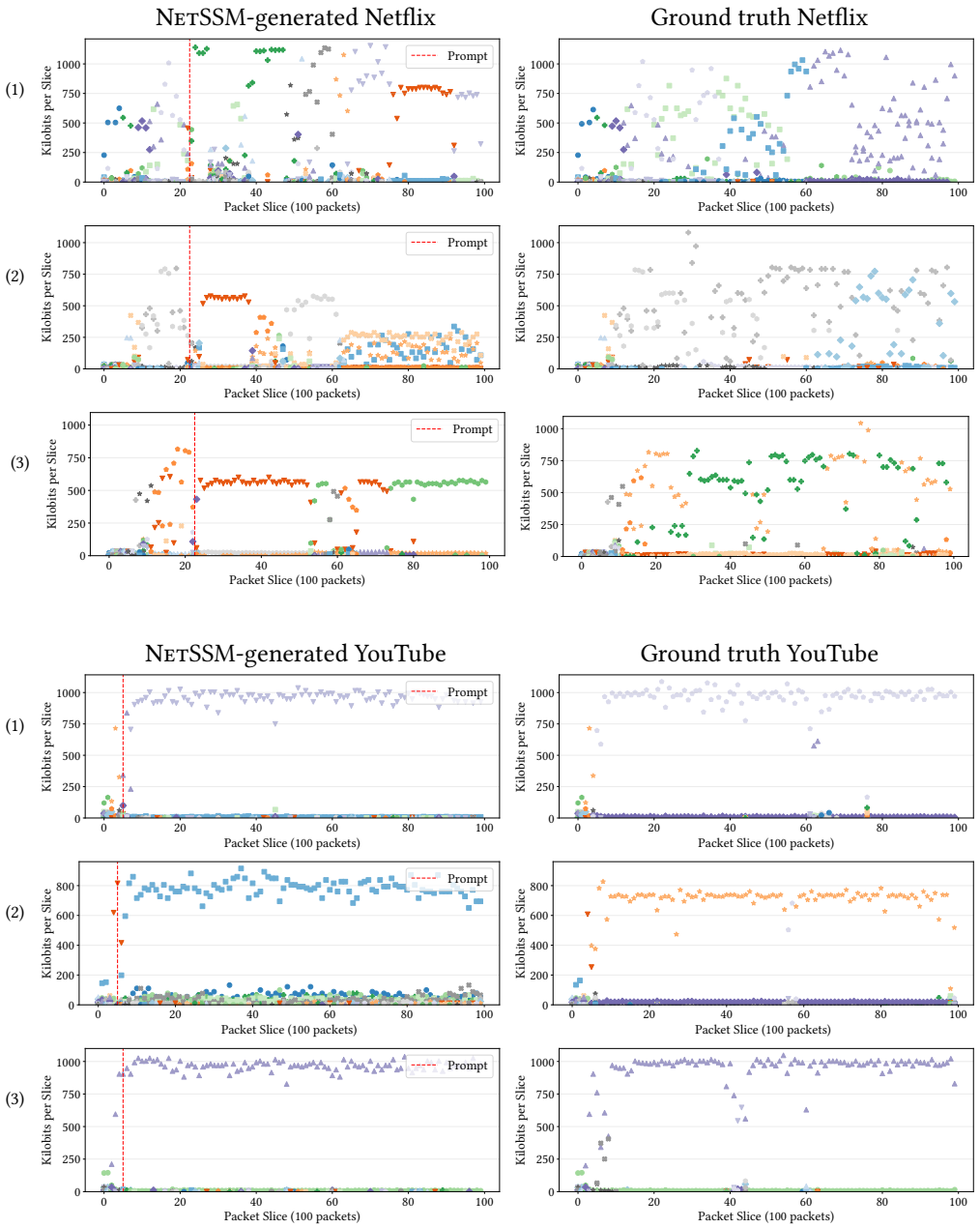


Fig. 4. Plots of throughput per flow for NetSSM-generated/ground truth Netflix and YouTube trace pairs.

C Semantic Similarity

Figure 4 presents additional examples plotting the throughput in kilobits/100 packet slice for Netflix and YouTube traces. Table 1 shows the generation hyperparameters used in Section 5.3.

Table 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

D Memorization Analysis

Table 2. NetSSM Memorization Analysis Overview. Table 2a reports packet-level memorization and diversity metrics, while Table 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

Received June 2025; revised November 2025; accepted December 2025