ARTIFACT EVALUATED
USENIX
AVAILABLE

Defeating Slow-and-Low Threats via Diffusion Model-based Generative Inference

Seyed Mohammad Mehdi Mirnajafizadeh
Dept. of Computer Science
Wayne State University

Prashant Khanduri
Dept. of Computer Science & AIDaS
Wayne State University

DaeHun Nyang
Division of AI and Software
Ewha Womans University

Rhongho Jang
Dept. of Computer Science & AIDaS
Wayne State University

Abstract

Content Delivery Networks (CDNs) are known to be vulnerable to slow-and-low threats that exploit trusted protocols while evading threshold-based defense at the edge. Our work addresses three limitations at edge defense: constrained resources, absence of a behavior monitor, and impractical assumptions for online detection. To defeat slow-and-low threats, we propose *SketchVision*, a vision-inspired detection framework that redefines flow behavior monitoring and attack detection under resource-constrained settings. We introduce a vision-inspired sketch that encodes packet-level temporal patterns of all flows into a compact image, a diffusion model tailored for sketch denoising, and a generative inference pipeline to forecast mature flow states from partial observations for early detection. Implemented with eBPF-enabled data planes and diffusion-based control, *SketchVision* achieves robust accuracy across 19 types of slow-and-low attacks, reaching an average AUC of 0.982 and F1 score of 0.913, improving detection by up to 29% over the state-of-the-art methods, while remaining efficient for large-scale CDN edge deployment.

1 Introduction

Modern production servers are positioned behind *cloud-hosted* content delivery networks (CDNs) to mitigate volumetric and application-layer attacks. However, studies have revealed that CDN infrastructure is not immune to exploitation [1, 2]. Recent attacks exploit the trusted protocol [3] and operate below the volumetric detection threshold [4], making CDN protection struggle to distinguish attacks from legitimate traffic. The CDN-native detections are primarily deployed at edge (entry-point) servers [5]. However, it encounters a dilemma triggered by the resource constraint nature.

On the one hand, edge servers are optimized for high throughput and low latency to serve a large population of clients. To monitor massive concurrent flows, the edge defense function must keep a short measurement window for flow monitoring (often just a few seconds) and finalize flow investigation quickly to save scarce memory resources [6, 7]. On the other hand, such resource constraints enabled slow-and-low attacks to operate at slow rates with low traffic volume, for evading current threshold-based detections [3, 4]. A flow behavior monitor is crucial to defend against the slow-and-low threats. However, the extensive resource consumption prohibits the approach from being deployed at the edge. Moreover, slow-and-low attacks exhibit a significantly higher off-time-to-lifetime ratio than the burst-and-volumetric attacks [8–13], meaning that these attacks are featured by *much lower flow activeness* (refer to §2 for attack analyses). Therefore, a much longer observation window is required to collect enough flow data for informed detection, which unavoidably adds extra burden to the edge servers. This phenomenon reflects the lack of robust behavior monitoring for slow-and-low traffic in the current cloud-hosted CDN infrastructure.

Prior research efforts have introduced a range of attack detection solutions tailored for different network attack types [14–28]. However, these works focus on burst-and-volumetric flow attacks only, by downplaying the impacts of slow-and-low threats. State-of-the-art defense solutions incorporate enriched features for finer-grained flow behavior monitoring and attack detection [20, 21, 29, 30]. However, their capacity for detection is insufficient to address effectively slow-and-low threats in the edge-server environment.

In this work, we identify and address three research gaps. *First*, current solutions fail to address the memory constraint and scalability issues by adopting a hash table-based stateful flow monitor for attack detection [20, 21]. These schemes employ a selective flow monitor with an eviction policy, with preference given to burst-and-volumetric flows, which caused invisibility of slow-and-low threats to the detection module. *Second*, we unveil that the behavioral features proposed in the state-of-the-art schemes [29, 30] ignore the temporal space behavior of flows to overcome the resource limitations. Lacking of temporal behavior monitoring capability opens a chance for adversaries to evade detection by mimicking a benign flow, while behaving dramatically differently in temporal space. *Lastly*, we shed light on a critical practicality issue of online

detection [20, 21, 29, 30], as prior works generally assume that network attack detection models are trained offline on completed flow datasets. The pitfall is that online detection must operate over continuous traffic streams, where flow completion (i.e., the availability of full data) is an uncertain event, especially in the case of slow-and-low attacks. The partial view of the attack flows triggers a critical logic flaw, that is, offline-optimized models with fully observed inputs struggle when supplied with partial or truncated flow observations online (e.g., 20% of packets of flows), resulting in high uncertainty and instability in detection. *Our analysis* indicates that such a mismatch leads to false negatives when attack flows are sparse, and to false positives when benign flows are at an early stage (refer to §2 for point-to-point discussions).

In this paper, we propose a vision-based slow-and-low attack monitoring and detection framework for edge servers, namely *SketchVision*. At its core, we introduce a vision-inspired sketch algorithm integrated with diffusion models (supported by widespread edge GPU availability [31]) for sketch denoising and generative inference. Our system aims to (i) enable per-flow temporal behavior monitoring by resolving resource constraints and (ii) realize a practical online detection to mitigate slow-and-low threats at an early stage. The essential building blocks are summarized as follows:

(1) Vision-inspired Stream Data Sketch Algorithm. We first overcome the edge resource constraint issue by advancing the probabilistic stream data sketch algorithm [32–46]. In particular, we propose a *geometric encoding strategy* that transforms packet-wise temporal patterns of each flow into a visual structure on an image, along with the per-packet metadata rendered in the pixels. To boost memory efficiency, we allow every single flow to be drawn into a shared image, namely *sketch universe*, with memory randomization.

The vision-inspired sketch is crucial for addressing the memory bottleneck for fine-grained behavior monitoring of ALL flows at edge servers, making slow-and-low attacks visible.

(2) Vision-based Sketch Denoising via Diffusion Model. The vision-inspired sketch, however, introduces a more complex noise issue, by inter-flow pixel overlaps in the image. We tackle the issue by incorporating a diffusion model for flow image (sketch) denoising. Since existing diffusion models designed for vivid natural images [47] are incompatible with the sparse network traffic pattern on sketch, we propose a flow-aware learning, aiming to enable finer-grained recognition of flow types based on visualized behaviors. Moreover, we feed the diffusion model with clean and noisy flow sketches concurrently for loss calculation. This design allows the model to exploit their structural discrepancy and enables the reverse diffusion process to act as a sketch-based denoiser [48–50].

The design trades GPU resources to maximize memory efficiency for memory-constrained edge servers. The integration of the diffusion model for sketch denoising is key to making vision-based slow-and-low attack detection viable.

(3) Generative Inference for Early Attack Detection. We then tackle the impracticability of online detection derived from the partial view issue by utilizing the diffusion model's generative capacity. We train the diffusion model to forecast a flow's final (or mature) state at its early stage via sketch inpainting. This time, the forward process of a diffusion model is fed by a partial flow sketch and its complete version concurrently; thereby, the model can generate a completed flow sketch based on a few packets and their temporal relations.

The diffusion model for generative inference enhances the practicality of online detection under stream data, where partial views of flows and slow-and-low attacks are norm.

We note that the denoiser and forecaster models do not work alone, but serve as building blocks of *SketchVision*'s attack detection pipeline, as *sketch*→ *denoiser* → *forecaster* → *classifier*. We refer to the process as generative inference.

We deployed *SketchVision*[1] in an edge server environment. With approximately 500 lines of code, we integrate the proposed sketches into the eBPF-enabled data plane (kernel space), deploy diffusion models into the control plane (user space), and use direct memory mapping for sketch data communication. Based on 19 types of slow-and-low attacks, we verified the feasibility of vision-inspired sketch and diffusion model-based denoiser by achieving an average AUC of 0.982 and an F1 score of 0.913, which is a 29% average improvement compared to state-of-the-art schemes [20, 21, 29, 30]. In the early detection, *SketchVision* achieves F1 score of 0.933 using only 20% of the flow's packets, up to 73% improvement compared to state-of-the-art schemes. Finally, our results show that the diffusion model, on average, takes ≈113 ms for sketch denoising and ≈11 ms for forecasting, which makes it viable for slow-and-low attack detection.

Organization. We first motivate *SketchVision* in §2. Then, we explain the threat model and system workflow in §3. Next, we describe function designs in §4 and §5, followed by the system implementation in §6. In §7, we give comprehensive system and security evaluations. Lastly, we described related works and concluded our work in §8 and §9, respectively.

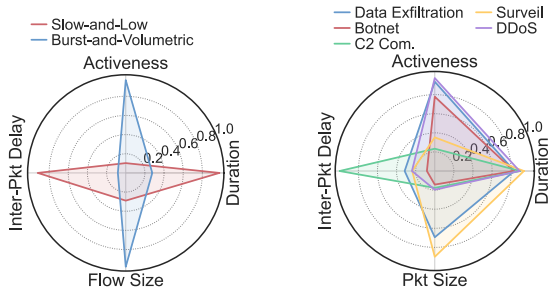
2 Motivating *SketchVision*

In this section, we motivate *SketchVision* via attack analysis. We then discuss three challenges for slow-and-low detection.

2.1 Analysis of Slow-and-Low Attacks

We analyzed a recent attack dataset [21], which consists of 80 types of malicious traces, categorized into *slow-and-low* and *burst-and-volumetric* attacks based on the flow duration and

[1]The source code is at https://github.com/NIDS-LAB/SketchVision



(a) Attack categories (b) Slow-and-low attacks

Figure 1: Comparison of attack types: slow-and-low vs. burst-and-volumetric, with slow-and-low variations.

activeness. *We stress that our categorization does not necessarily imply attack duration, since it is an attacker's choice.* Our perspective is the flow monitoring window required to collect enough packets for detection, considering both flow duration and activeness. We define activeness as the ratio of active bins (10 ms each) over the flow lifetime, and observed that burst-and-volumetric attacks last less than 5 seconds and are active for over 50% of bins, whereas slow-and-low attacks can last up to 40 seconds with much fewer active bins.

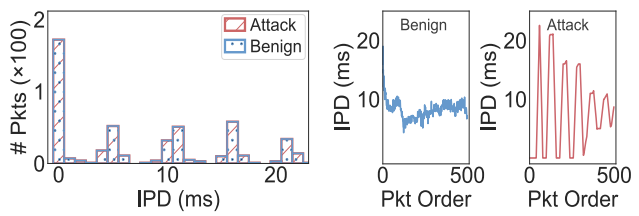
In Figure 1 (a), we compare slow-and-low (red) [51–56] and burst-and-volumetric (blue) [19, 57–61] attacks, in terms of activeness, duration, flow size, and inter-packet delay. As shown, the burst-and-volumetric exhibit a bursty nature with larger volume and higher activity throughout their lifetime. Therefore, a short measurement window at the edge is sufficient for threshold-based detection. In contrast, slow-and-low attacks, characterized by much lower activity and volume, can easily evade such detection. We further visualize slow-and-low attacks in Figure 1 (b), categorized into Data Exfiltration [19, 59], C2 Communication [62], DDoS [8, 10, 11], Botnet [63, 64], and Surveillance [59, 65]. As shown, slow-and-low attacks exhibit similar behaviors in terms of duration and activeness, but differ significantly in packet-level behaviors, such as packet size and inter-packet delay. Therefore, a binary classifier that treats all attack types as a single class is impractical given their substantial disparity. State-of-the-art solutions [20, 21, 66] train multiple independent models for binary classification of each attack type; however, this adds extra burden to the packet processing pipeline. These observations suggest that a scalable and packet-level flow behavior monitor with a unified multiclass recognition capacity is crucial to mitigate slow-and-low threats at the edge.

2.2 Challenges

Lacking Scalable Flow Behavior Monitor. Edge servers, as the entry point to CDNs and cloud infrastructures, sit close to clients and offer first-line defenses, such as filtering, rate limiting, and protocol sanitization before traffic reaches the central cloud. These defenses, relying on the threshold- and signal-based detections, are effective against burst-and-volumetric attacks, but fall short against slow-and-low attacks [3, 4]. Particularly, given limited capacity at the edge, computing resources are largely consumed by delivery, caching, and routing, leaving little capacity for deep inspection or feature extraction [67]. These constraints further force downgrading of granularity and duration for the flow observation (i.e., threshold or signal-based detections), resulting in the edge servers being ineligible for tracking slow-and-low attacks [68, 69]. Existing deep learning approaches, including LSTMs [70] and VAEs [71], offer promising anomaly detection capabilities but remain impractical for line-rate deployment. The requirement for sequential packet ingestion imposes an unsustainable memory burden on edge devices due to the need for extensive per-flow state preservation. State-of-the-art flow behavior monitoring systems [21, 22] employ a memory-intensive hash table to collect per-flow features at packet-level granularity, creating a severe bottleneck due to the ever-increasing edge flow rates, from 10 K flows/s in 2012 [72] to 50 K flows/s in 2024 [73]. With resources prioritized for user applications [74], such unbounded memory overhead exposes the defense itself to resource saturation attacks. The alternative selective approaches [19, 20, 66] reduce overhead but sacrifice flow coverage; thus, leaving loopholes for adversaries [29].

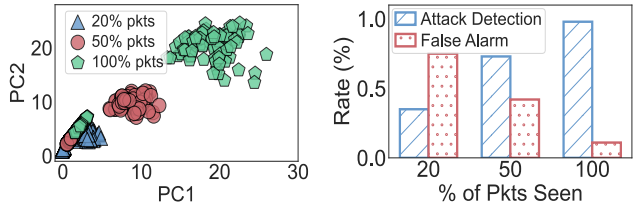
Evading Coarse Behavior-based Detection. The state-of-the-art approaches to generalizable flow measurement mainly rely on flow-level aggregated flow features, such as packet size distribution [20, 75, 76] and inter-packet delay distribution [19, 29, 77], neglecting packet-level temporal dependencies and metadata within flows. While this design circumvents resource constraints, the highly aggregated feature allows an attacker to forge attack flows to mimic benign patterns while hiding malicious behavior. In Figure 2, we show that flow-level distribution features cannot distinguish an attack flow that mimics a benign flow's distribution (see Figure 2 (a)), and also has a large room to behave differently over an extended temporal domain (see Figure 2 (b)). Existing solutions [21, 22] attempt to preserve temporal dependencies using per-flow hash tables for packet-level temporal relation logging, which is memory-intensive and unrealistic for real-world deployment. Therefore, it is crucial to enable a packet-level temporal behavior monitor to defeat advanced adversaries at the edge.

Partial View of Slow-and-Low Attacks. Training machine learning (ML) models offline on completed flows data and then applying them during online detection is a common practice in current approaches [20, 29, 66, 77, 78]. However, we stress that the dataset on completed flows is almost unavailable when online detection operates on a dynamic network stream. The issue centers on the design of the detection trigger. *First*, online detection that is triggered by flow completion signals introduces the other attack surface, as adversaries can randomly inject flow termination packets. *Second*, idle



(a) Flow-level distribution (b) Packet-level temporal behavior

Figure 2: Evading coarse behavior-based detection: as shown, malicious flow behavior mimics benign distribution-level patterns, concealing temporal malicious behavior.



(a) PCA feature analysis (b) ML performance

Figure 3: Partial view of slow-and-low attack: PCA reveals that flows at different stages form distinct clusters, corresponding to inconsistent ML performance across these stages.

timeout-triggered detection allows an attacker to exploit a continuous stream to evade the detection trigger. *Lastly* and therefore, most state-of-the-art systems employ hard timeout or threshold-based approaches to perform detection with a partial view of the flows [19–23].

Our *observation* is that models trained on full data will fail in either excessive false alarms or delayed detection due to the non-stationary statistical features of a network flow, i.e., features at different stages exhibit distinct behaviors. To verify, Figure 3 (a) illustrates principal component analysis (PCA) of network flows; the data distribution of an early-stage flow occupies a different feature space from the data of a completed one. This discrepancy causes a model to either miss the attack or incorrectly flag benign partial flows, as shown in Figure 3 (b). One might suggest training models on multiple window sizes to address the issue. However, such an approach introduces substantial training overhead and deployment complexity, making it impractical in real-world deployment [66]. Finally, we argue that the issue is particularly challenging for low-and-slow attacks since it lacks a scalable flow behavior monitor, and waiting for flow completion is unrealistic.

3 Vision-based Attack Detection Framework

To defeat ever-evolving adversaries, recent advances have inspired researchers to apply GPT-like large language models (LLMs) to network security problems [79–81]. Conceptually, LLMs designed for sequential text align well with the stream-oriented nature of network traffic. However, the per-flow data collection, tokenization, and vectorization at packet-level granularity trigger severe scalability issues. Therefore, deploying LLMs for online defense is infeasible, but available for offline security analyses. To enable AI for online detection, we designed *SketchVision* to encode temporal information into a compact image representation while leveraging the generative capacity of diffusion models for data recovery and early detection. In the following, we define the threat model for our system and describe the architecture and workflow.

3.1 Threat Model

As shown in Figure 4, we assume an adversary to control a botnet or malware to attack production servers behind a content delivery network (CDN). The attack exploits trusted protocol and slow-and-low attack patterns to evade current edge defenses designed for burst-and-volumetric attack mitigation. In this work, we focus on slow-and-low threats, illustrated in Figure 1, which demand packet-level temporal behavior monitoring. The burst-and-volumetric attacks are excluded, as volumetric edge defenses are well-established [14, 18, 22, 23, 78, 82]. We assume adversaries apply perturbations to packet sizes and timing to evade detection. Given the diverse temporal patterns inherent to these attacks, we emphasize the necessity of multi-class recognition. Lastly, a concept drift issue is not considered in this work.

3.2 System Design and Workflow

As shown in Figure 4, our framework is designed for deployment at edge servers (entry points) of CDNs, to protect production servers from slow-and-low threats. We assume the edge server runs an eBPF-based monitoring module [83] within the kernel space to boost network traffic processing speed, and equips at least a consumer-grade GPU for running AI-assisted tasks. Based on these edge settings, we propose *SketchVision* framework to enable a per-flow behavior monitor at packet-level granularity for slow-and-low attack detection. The essential building blocks and workflow are as follows:

❶ **Vision-inspired Sketch (Data Encode).** *SketchVision* starts with vision-inspired per-flow behavior sketching in the kernel space, which resides in the eBPF-enabled express data path (XDP). The proposed sketch encodes flows into a *compact* image representation by converting packet-level temporal relations into geometric structures that preserve primitive flow behavior, as well as the metadata with pixels. To achieve sublinear memory usage, we employ the conventional sketch concept to allow memory random sharing among visualized flows. System-wise, we maintain a hash table for per-flow metadata, including 1) *tuples* for IDs, 2) the *timestamp* of the first packet for temporal behavior sketching, and 3) per-flow *count* to trigger AI-based detection. We note that storing per-

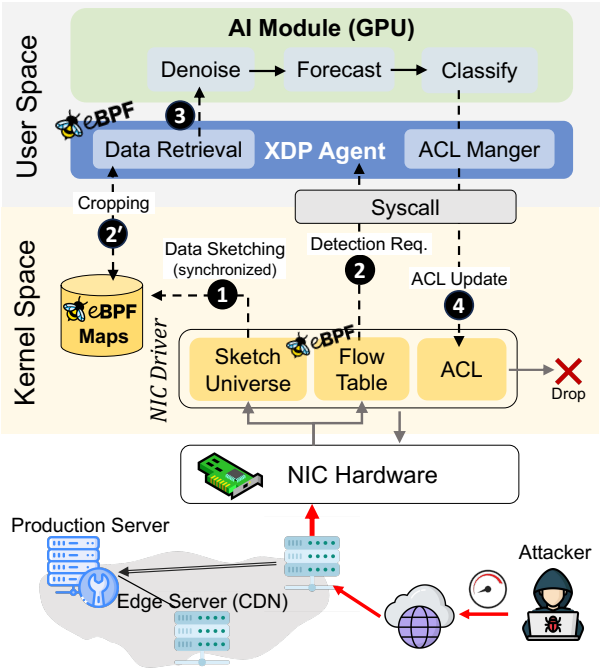


Figure 4: System workflow: *SketchVision* integrates with the NIC driver's eBPF XDP module, encoding per-packet features into eBPF memory shared between the user space and the kernel for subsequent sketch denoising and flow forecasting.

flow metadata only, but not features, adds a negligible burden to the edge server (e.g., 14 MB required per 1 million flows).

❷ **Memory Mapping for AI Modules (Data Retrieval).** Next, we map the kernel-space sketch memory to a reserved user-space region, enabling real-time data retrieval from the XDP and preparing for AI-driven inspection at the server's control plane. The detection trigger is embedded in the sketch encoding logic to contact the user-space XDP agent with the ID of active flows (srcIP and dstIP). We apply a small threshold based on the per-flow count of the hash table (i.e., partial view of flows) to trigger the detection, using the generative capacity of the proposed diffusion models (see below).

❸ **Sketch Denoising and Flow Forecasting (Data Process).** Our AI modules are placed in the user space (control plane) of the edge servers and listen to detection events triggered by the XDP agent. Based on the visualized flows, our first objective is to train a diffusion model on a diverse range of network traffic, including both malicious and benign flows. Therefore, the model learns to capture the underlying patterns and structures of diverse flows. With the flow-aware learning, we proposed two diffusion model-based function variations, namely 1) sketch denoiser and 2) flow forecaster. The former resolves the notorious sketch noise issue that is caused by flow-wise pixel sharing, and the latter performs generative inference of slow-and-low attacks with a partial view of flows for early detection (refer to §5 for diffusion model designs).

❹ **Detection and Action.** The proposed diffusion models serve as data processing units rather than a classifier, since our research interest is to prove the feasibility of generative inference for network applications. In our evaluation, we show that our diffusion-based data processing functions can be combined with an elementary convolutional neural network (CNN)-based classifier that is trained by the complete flows (current training and testing paradigm) for early detection, given a partial view of flows. System-wise, we employ a native access control list (ACL) module to take proactive actions for early mitigation of slow-and-low threats.

4 Vision-inspired Behavior Sketching

Our first technical contribution is to extend a conventional per-flow counting sketch concept to a vision-based per-flow behavior sketch. The proposed sketch is motivated by state-of-the-art works that leverage a hash table for logging entire and exact packet orders and arrival times for all flows [21,22], which wastes excessive memory space and is impractical even for a data center environment. Our vision-inspired sketch is the key to enabling a per-flow temporal behavior monitor at edge servers with a limited budget, by approximately encoding packet-level temporal behavior in an image for all flows.

Data Structure and Function Logic. As shown in Figure 5, *SketchVision* applies behavior sketching to a data structure organized as an $N \times N$ grid, referred to as the "Sketch Universe", with each cell containing three 8-bit slots. Visually, each cell is interpreted as a pixel in a standard 2D image. To visualize temporal behavior of flows, each flow is given a base point and then the temporal patterns of packets are converted into a geometric representation, using an angle (θ) and radius (R) relative to a flow's *base point*, while per-packet attributes (i.e., packet size in this work) are recoded in the corresponding pixels; we refer to this process as "per-flow painting". To locate the base point of flows within the Sketch Universe, we use a pairwise-independent hash function on the source-destination tuple to determine the flow location. This process is applied to all flows within the shared Sketch Universe. We note that geometric encoding in the shared universal memory triggers interference and overlap among flows, known as sketch noise. We address the issue in §5, which is referred to as diffusion model-based sketch denoising.

Encoding (Flow Painting). Algorithm 1 illustrates the encoding function. For an incoming packet of flow f, first, two pairwise independent hash functions are used to determine a base point (x, y) in the grid for the flow encoding (lines 1-3). As shown in Figure 5, subsequent packets of the flow f are then encoded based on this base point by converting the temporal patterns into angle (θ) and radius (R). Particularly, we leverage each packet's timestamp to vary the encoding location using the circular formula $R \cdot e^{i\theta}$, where the radius R represents the average inter-packet delay of the flow at each

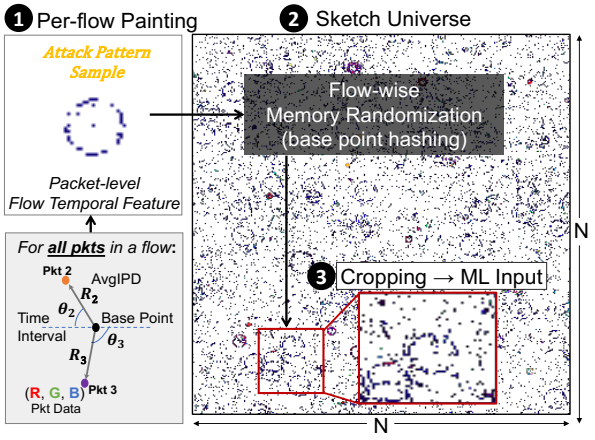


Figure 5: *SketchVision*'s flow behavior sketching: all per-packet features are encoded into the sketch universe via flow-wise base point hashing, approximately encoded regions are then cropped and fed to ML models for early attack detection.

Algorithm 1: Sketch Encoding

Input: Sketch Universe SU, Sketch Size $N \times N$

1 $tuple \leftarrow (\texttt{pkt.SrcIP, pkt.DstIP})$;
2 $key_1 \leftarrow \mathcal{H}(tuple)$, $key_2 \leftarrow \mathcal{H}'(tuple)$;
3 $x_{\text{base}} \leftarrow key_1 \bmod N$, $y_{\text{base}} \leftarrow key_2 \bmod N$;
4 $(\texttt{time}_{pkt}, \texttt{len}_{pkt}) \leftarrow \texttt{pkt}$;
5 $R \leftarrow \texttt{AvgIPD}(key_1, \texttt{time}_{pkt})$;
6 $\theta = 2\pi \cdot \sqrt{\frac{\texttt{time}_{pkt}}{T}}$;
7 $x_{pkt} \leftarrow (x_{\text{base}} + R \cdot \cos(\theta)) \bmod N$;
8 $y_{pkt} \leftarrow (y_{\text{base}} + R \cdot \sin(\theta)) \bmod N$;
9 $(R, G, B) \leftarrow SU[x_{pkt}][y_{pkt}]$;
10 $\texttt{pkt_cnt}_{collision} \leftarrow (R_0{:}R_7,\ G_0{:}G_4)_{13-bit}$;
11 $\texttt{pkt_len}_{avg} \leftarrow (G_5{:}G_7,\ B_0{:}B_7)_{11-bit}$;
12 $\texttt{pkt_len}_{avg} \leftarrow \frac{\texttt{pkt_cnt}_{collision} \times \texttt{pkt_len}_{avg} + \texttt{len}_{pkt}}{\texttt{pkt_cnt}_{collision} + 1}$;
13 $[R_0{:}R_7,\ G_0{:}G_4]_{13-bit} \leftarrow \texttt{pkt_cnt}_{collision} + 1$;
14 $[G_5{:}G_7,\ B_0{:}B_7]_{11-bit} \leftarrow \texttt{pkt_len}_{avg}$;

packet arrival and angle θ is mapped to each packet's arrival time directly (lines 4-6). Such a design transforms packet-level temporal relations into a geometric structure, where the eventual shape depends on the packet-level temporal dynamicity of the flow. We simply calculated the average inter-packet delay (i.e., AvgIPD, see Equation 1) by using a separate native eBPF hash table to store per-flow ID, the first timestamp, and the packet count. Worth noting that the hash table stores per-flow metadata only, and the resource-intensive packet-level temporal feature is encoded on the sketch. After determining the pixel (location) for each arriving packet (lines 7-8), we encode packet metadata into the 24-bit pixels. Specifically, 13 bits record the number of packet collisions in pixels, and 11 bits store the estimated (average) packet length upon collisions (lines 9-14). We note that the 11-bit assigned to the packet length covers the standard maximum transmission unit (i.e., 1500 bytes). Also, enlarging the collision counter is an option by employing 32-bit pixels for the sketch image.

Decoding (Image Cropping). In the proposed system, we do not decode flow data directly from the sketch, since retrieving packet-level numerical data is complex and imprecise due to the pixel overlaps among flows. Instead, we aim to use a diffusion model to interpret the visual representation of the flow's behavior directly. To prepare data for vision-based processing, we could estimate the flow's data region by calculating its central coordinates (x, y), derived from the flow's ID and its average inter-packet delay. To simplify computation and maintain consistent input dimensions for the model, we fixed a 64×64 projection bounding box considering the maximum attack volume. Consequently, a 64×64 patch, centered on the flow's coordinate, is cropped from the sketch universe and serves as a compact image-based representation of the flow's behavior for the downstream tasks.

Design Intuitions. The design of radius R and angle θ is empirical, based on temporal characteristics of flows. The radius R reflects a flow's current average inter-packet delay (IPD) upon arrival of a new packet. The design to encode each packet with the flow's average IPD enables measurement of the flow's temporal dynamicity, denoted as:

$$R = \overline{\Delta} = \frac{1}{n-1}\sum_{i=1}^{n-1}(t_{i+1} - t_i) = \frac{1}{n-1}(t_{n-1} - t_1) \quad (1)$$

where t_i is the timestamp of the i^{th} packet and n is the total packets. Our analyses indicate that slow flows exhibit large and variable inter-packet delays, whereas burst flows have small and stable delays. By encoding R with average inter-packet delay $\overline{\Delta}$, a flow's shape naturally adapts to its burst and activeness patterns. That is, slow-and-low flows with large-and-irregular $\overline{\Delta}$s map to outer space, expanding their geometric footprint and making irregular delays more distinguishable, whereas burst flows with small $\overline{\Delta}$ maps to inner space, producing compact representations to reduce memory overhead. This design significantly enhances the visibility of slow-and-low attacks, which are often ignored by threshold-based detection systems [14, 16, 19, 23, 78, 82, 84, 85] by allocating them sufficient space to reveal temporal irregularities.

The angle θ serves as the other factor to distribute each packet's timestamp across the geometric plane for capturing the temporal diversity of a flow. A straightforward linear time-to-angle projection $f_{time-to-angle}(t) = 2\pi \cdot t/T$, where T is a fixed time window mapped onto a full period of a circle. This projection produces a fixed repeating cycle over time, distributing packets uniformly. However, during the flow establishment stage, flows often burst with subtle timing differences [86], and in later stages, irregular IPD variation of slow-and-low attack samples uniformly from this projection, creating dense or sparse structure regions. To adapt to

such characteristics of slow-and-low attack, i.e., capturing subtle differences in early stage while efficiently representing sparse data over time, we use $f_{time-to-angle}(t) = 2\pi \cdot \sqrt{t/T}$, where T defines the first cycle and evolves according to $T(t) = 2\sqrt{T \cdot t} + T$. Initially, the first cycle progresses slowly, magnifying the flow's early stage; in later stages, the cycles stretch, with longer periods compressing the sparse data points. This design reveals early-stage activity with a lower collision rate and gracefully handles sparse data, achieving both sensitivity in the early phase and memory-efficient representation over extended periods.

Worth noting that the eventual shape of flows is determined by their temporal dynamics (i.e., inter-packet delay variations across stages). A near-perfect circle can be observed when a flow exhibits highly stable inter-packet delays combined with high volume, matching the profile of burst-and-volumetric flows. Our interest is a distorted and incomplete circle corresponding to the profile of slow-and-low attacks that is sparse and unpredictable. Lastly, an adversary may add packet-level perturbation for its own flow (see §7.3 for Generalizability), but triggering flow-level collisions is hard due to the unawareness of hash algorithms and seed values of the sketch.

5 Diffusion Model as Denoiser & Forecaster

Utilizing the cropped image containing the flow behavior, we next aim to recover the flow's behavior from sketch noise and classify the flow. In the presence of sketch noise, directly classifying the cropped image confuses the classifier, leading to misclassifications. To address this, we first propose a diffusion-based method to denoise the sketch. Moreover, leveraging temporal information and recent advances in diffusion forecasting, we then predict the future flow states using a learned generative prior over temporal dynamics for early detection of slow-and-low attacks.

Preliminaries. In this paper, we follow the DDPM formulation [47], modeling the forward process as a Markov chain with additive Gaussian noise and learning a reverse process that removes the added noise to recover the data. The forward process progressively corrupts an initial image $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, sampled from the data distribution q. At timestamp t the corrupted data $\mathbf{x}_t$ is obtained as:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \mathbf{I}), \tag{2}$$

where $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$, and $\beta_t \in (0,1)$ is a predefined noise schedule controlling the noise at each stage. Equation 2 is equivalent to $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t \mathbf{I})$. Subsequently, we train a neural network ε_θ to reverse the forward process, modeling the distribution:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)). \tag{3}$$

We train the model to minimize the error in estimating the original clean sample $\mathbf{x}_0$ using $\varepsilon_\theta(\mathbf{x}_t, t)$. During inference, we start from a noisy sample $\mathbf{x}_T$ and iteratively denoise it to recover $\mathbf{x}_0$ from distribution q. At each step, we compute $\mathbf{x}_{t-1}$ using the mean of parameterized reverse process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$, an approximation of the true posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, which is a tractable Gaussian distribution with mean:

$$\hat{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t, \tag{4}$$

where the coefficients are determined by the forward diffusion schedule. This training/inference is called unconditional, since the model learns the full distribution q without additional information. To distinguish between different classes or attributes within q, conditional diffusion models incorporate external guidance or auxiliary signals to guide the generative process toward specific targets [87, 88]. Here, the model is expressed as $\varepsilon_\theta(\mathbf{x}_t, t, c)$, where c denotes external condition.

5.1 Diffusion Model for Sketch Denoising

We explain our diffusion model design for sketch denoising.

Problem Definition. Let $\tilde{\mathbf{x}}$ denote the noisy sketch image of a flow cropped from the sketch universe and $\mathbf{x}$ the clean noise-free image, which are available during offline training. The goal is to learn $\tilde{\mathbf{x}} \mapsto \mathbf{x}$. However, learning a direct mapping is non-trivial, as the sketch noise is highly nonlinear and exhibits a complex pattern. Instead, we propose to blend Gaussian noise with sketch noise and let a diffusion model progressively learn the flow structure and sketch noise under corruptions, ultimately reverse the noisy image to a noise-free image.

The *first* challenge is that the standard diffusion model [47] is designed to reverse the input back to an anchor point (i.e., $\tilde{\mathbf{x}}_T \mapsto \tilde{\mathbf{x}}$), which is a noisy sketch, but not the clean one (i.e., $\tilde{\mathbf{x}} \mapsto\!\!\!\!\!\!\not\;\;\; \mathbf{x}$). The *second* challenge is that existing diffusion-based denoisers [89–95] are designed for vivid natural images with predictable noise, which is incompatible with the sparse slow-and-low flows. Especially when multiple flows coexist within the same image, the inference (i.e., sketch noise) that occurs among sparse flows requires superior flow-awareness for the denoiser to recognize distinct flows within sketch noise.

Gaussian Noise Blending for Sketch Noise Learning. While we expect direct mapping $\tilde{\mathbf{x}}_T \mapsto \mathbf{x}$, sketch noise makes this task non-trivial. Our core assumption is that sketch noise is primarily driven by super-mice flows in heavy-tailed traffic [32–46]. With randomized flow mapping within the sketch universe, the super-mice flows converge toward a Gaussian distribution per the Central Limit Theorem [96]. Thereby, diffusion models serve as an ideal denoising mechanism, leveraging their inherent capability to reconstruct signals from Gaussian-corrupted data [47]. As shown in Figure 6, while the standard diffusion model $\mathcal{M}_{std}$ is capable of learning these distribution and even more complex types of corruption [97], the reverse process conceptually towards the anchor point ($\tilde{\mathbf{x}}_T \mapsto \tilde{\mathbf{x}}$), which cannot serve as a denoiser $\mathcal{M}_{goal}$. To tackle the issue, we propose a novel diffusion model-based denoising function $\mathcal{M}_{denoiser}$, which remodels the standard training

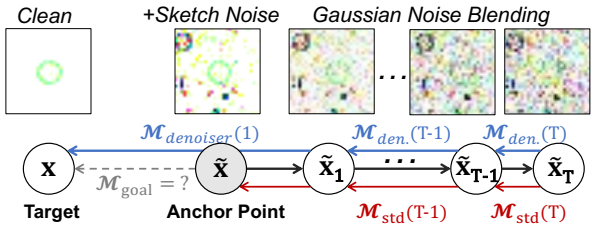


Figure 6: Direct sketch denoising is challenging; the standard diffusion model moves toward the anchor point via Gaussian denoising. Using both clean and noisy sketches, the model learns to remove both types of corruption.

process with a novel loss function. At its core, we incorporate both noisy and clean sketch images of all flows to derive the loss, allowing the model to learn the distribution of clean image $q(\mathbf{x})$ referencing noisy image $q(\tilde{\mathbf{x}})$.

Given the sketch image $\tilde{\mathbf{x}}$, we corrupt it with Gaussian noise according to the forward process (Equation 2), which is defined for T diffusion steps. While the forward process is defined up to T steps, training the denoiser on all T steps can be computationally expensive and may even be unnecessary, since timestamps near T correspond to pure Gaussian noise and are less informative for sketch denoising. To reduce the training overhead, we instead sample timestamps from a truncated range $t \sim \mathcal{U}\{1,\ldots,\tau\}$ with $\tau \leq T$. This choice, inspired by [98], allows the model to focus on a representative spectrum of Gaussian noise levels without the full cost of training across all steps. The proposed loss function is denoted as:

$$\mathcal{L} = \mathbb{E}_{x,t,\varepsilon,c}\left[\|\varepsilon_\theta(\tilde{\mathbf{x}}_t, t, c) - \mathbf{x}\|^2\right], \tag{5}$$

where $\mathbf{x}$ is indirectly referenced when deriving losses based on $\tilde{\mathbf{x}}$; thereby, the reverse process moves towards the distribution of the clean image, rather than the noisy anchor point. With the proposed loss design, the model provides an approximation of the clean image with a diverse set of corruptions; however, under sketch noise domination, the model tends to over-denoise and inadvertently remove important details. During inference, we follow the standard backward process, which was found to be effective in resolving the sketch noise issue. In particular, during the backward process and starting from $\tilde{\mathbf{x}}_\mathbf{t}$, we first estimate the clean image $\hat{\mathbf{x}}$, which is then used to guide the next less noisy input $\tilde{\mathbf{x}}_{t-1}$ derived as a combination of estimated clean image $\hat{\mathbf{x}}$ and the current noisy sketch $\tilde{\mathbf{x}}_t$ using Equation 4. This iterative denoising results in consistent and robust denoising, where the contribution of $\tilde{\mathbf{x}}_t$ helps restore the lost information, while the influence of $\hat{\mathbf{x}}$ guides the process towards the clean image.

Flow-aware Unconditional Denoising. To boost the flow-awareness during training, we condition the model on the traffic class label $l \in \{0,1,\ldots,L\}$, which specifies one benign and L attack types. The label l is projected into an embedding and

Algorithm 2: Training **Denoiser** and **Forecaster**

Input: Flow input $\mathbf{x}^1_{\text{obs}}$, target flow $\mathbf{x}_{\text{target}}$, condition c (e.g., p, l, $\mathbf{X}_{past}$), # diffusion steps $\tau \leq T$

// 1 `Denoiser:` $\mathbf{x}_{\text{obs}} = \tilde{\mathbf{x}}^p$, $\mathbf{x}_{\text{target}} = \mathbf{x}^p$, $c = (l, p)$

// `Forecaster:` $\mathbf{x}_{\text{obs}} = \mathbf{x}^p$, $\mathbf{x}_{\text{target}} = \mathbf{x}^{p+s}$, $c = (l, p, \mathbf{X}_{\text{past}})$

1 **for** *each training step* **do**
2 $t \sim \text{Uniform}(\{1,\ldots,\tau\})$;
3 $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$;
4 $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_{obs} + \sqrt{1-\bar{\alpha}_t}\,\varepsilon$;
5 $u \sim \text{Uniform}(0, 1)$;
6 **if** $u < p_{cond}$ **then**
7 $\hat{\varepsilon} = \varepsilon_\theta(\mathbf{x}_t, t, \text{None})$;
8 **else**
9 $\hat{\varepsilon} = \varepsilon_\theta(\mathbf{x}_t, t, c)$;
10 $\mathcal{L} = \|\hat{\varepsilon} - \mathbf{x}_{\text{target}}\|^2$;
11 Backpropagate and update model parameters θ;
12 **return** θ;

fed into the denoising network to learn attack class-specific distributions. However, a unique challenge is triggered for the flow-aware training, since the true label is unknown during online denoising (i.e., inference phase). Our solution to the problem is to train the model with occasionally provided condition l among flow instances, and otherwise with a "null" label, allowing it to denoise unconditionally when label information is unavailable [87]. We further construct our training dataset with snapshots of flows at different stages. Accordingly, an additional conditioning variable p, which represents the number of packets, is given during training to generalize the model beyond a fixed-snapshot denoiser.

Algorithm 2 highlights two key modifications of the diffusion process: 1) occasional conditioning (lines 6-9) and 2) loss calculation with paired inputs (line 10). We use pair-wise input $(\tilde{\mathbf{x}}^p, \mathbf{x}^p)$ denoting a flow's sketch and its clean image with p encoded packets, with $c = (p, l)$ as conditioning variables. During each training step, we sample a timestamp t uniformly from 1 to τ and blend Gaussian noise with the sketch image across different flow stages p and class conditions l. Depending on a given probability, the model is conditioned on both flow type and temporal embedding c to predict the clean image using the mean squared error (MSE).

5.2 Diffusion Model for Flow Forecasting

Next, we explain how to train a diffusion model for forecasting a flow's mature state using a partial view of the flow.

Problem Definition. Let $\mathbf{x}^p$ denote a flow's sketch image rendered with the first p packets, and let $\mathbf{x}^{p+s}$ be the future flow state with s packets ahead. Ideally, the objective is to learn a forecasting function $M_{forecaster}$, such that

$M_{forecaster}(\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^p) \mapsto \mathbf{x}^{p+s}$, where the forecasting is conditioned on the past observations. For achieving precise generation, forecasters in other domains often rely on the entire history of the previous data [99–101]. However, during online inference, storing and processing per-packet flow progression is impractical due to the significant overhead added to the system. Therefore, it is crucial to achieve forecasting accuracy with computational efficiency. Moreover, when training a diffusion model with per-packet flow progress (i.e., $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^p$), the sparse network flow representation at the early stage increases learning difficulty, leading to incorrect forecasting, emphasizing the importance of flow-aware forecasting over the entire flow progression.

Iterative Forecasting with Partial View. Our solution is to predict the future state $\mathbf{x}^{p+s}$ from a sliding window of w historical snapshots sampled every s packets, rather than updating on a per-packet basis. Formally, we defined the forecasting function as $\mathcal{M}_{forecaster}(\mathbf{X}_{past}, \mathbf{x}^p) \mapsto \mathbf{x}^{p+s}$, where the forecasting is conditioned on a window of w past observations $\mathbf{X}_{past} = \mathbf{x}^{p-(w)\cdot s}, \mathbf{x}^{p-(w-1)\cdot s}, \ldots, \mathbf{x}^{p-s}$. The window parameter w controls how far into the past the model looks, while the step size s determines the sampling frequency of that history. Accordingly, a small s provides a high-resolution view of the past at a high computational cost, whereas a large s is efficient but might fail to capture rapid or subtle temporal dynamics. Our experiments indicate that the model captures essential flow dynamics with larger step sizes. Furthermore, an autoregression approach [101] can generalize the forecasting to states of $\{p+2s, p+3s, \ldots\}$, using each predicted state as the conditioning input for the next step.

Algorithm 2 illustrates the training procedure for our forecaster. Unlike our denoiser that pairs clean- and noisy- images, we use pair ($\mathbf{x}^p$, $\mathbf{x}^{p+s}$) denoting the current flow state and the future state at $p+s$, with conditioning variables $c = (p, l, \mathbf{X}_{past})$. The training process follows a procedure similar to our denoiser, with the addition of conditioning on $\mathbf{X}_{past}$. We also initialize the diffusion model input by adding noise to the current sample $\mathbf{x}^p$, keeping past observations $\mathbf{X}_{past}$ as-is, which has been shown to accelerate generation compared to starting from pure Gaussian noise [102].

One-step Forecasting. We note that the autoregressive model is not our eventual goal, but a foundation for a one-step forecaster that predicts the final state $\mathbf{x}^N$ of a flow directly from partial view $\mathbf{x}^p$, i.e., $P(\mathbf{x}^N|\mathbf{x}^p)$, where N is the total packets of flows in the training dataset. The one-step forecaster significantly reduces computation and memory consumption, and also eliminates the forecasting error accumulation that appears in autoregressive forecasting. We report results of both one-step (§7.3) and autoregressive models (Appx. A.1).

Joint Forecasting and Denoising. We further explore a unified model for simultaneous denoising and forecasting. The model is trained to predict the final clean state from a sketch image $P(\mathbf{x}^N \mid \tilde{\mathbf{x}}^p)$ (refer to Appx. A.2 for details).

6 System Implementation

We implemented *SketchVision* in three main components in the edge server: (1) a vision-inspired behavioral sketching module within an eBPF XDP kernel written in `C`, enabling real-time encoding of temporal traffic behavior directly at the NIC level; (2) a user-space program, XDP Agent in `C`, serving as a bridge to retrieve data, feed it to the ML pipeline, collect results, and install ACL rules; (3) a Python-based user-space program that leverages the data from XDP Agent for denoising and forecasting. Building *SketchVision* in the constrained eBPF/XDP environment introduces several technical challenges, where floating-point operations and the use of standard mathematical functions such as sqrt, sin/cos are not supported. Additionally, all loops must be statically bounded, and memory accesses must be provably safe and within predefined ranges to satisfy the kernel's strict verifier. To overcome the lack of built-in mathematical functions, we precomputed the required operations over expected input ranges and stored them in lookup tables accessed through BPF maps at runtime [103]. Another challenge we faced was the absence of a built-in hash function for flow-based point mapping. Due to loop bounding constraints in eBPF, we resorted to using a single round of MurmurHash3 [104], a widely adopted lightweight alternative for the eBPF environment.

7 Experimental Evaluation

This section evaluates *SketchVision* compared with state-of-the-art systems in the early detection of slow-and-low attacks, demonstrates the effectiveness of diffusion-based denoisers, and shows the latency analysis to prove the feasibility.

7.1 Experimental Setup

Testbed. We implemented *SketchVision* in eBPF, comprising $\sim$ 500 lines of `C` code, complemented by Python scripts for ML inference. All experiments and model training were conducted on a server equipped with an Intel(R) Xeon(R) CPU @ 2.60 GHz, 1 TB DRAM, and an NVIDIA A100 GPU.

Dataset. We used 19 types of slow attacks grouped into five categories, as shown in Figure 1 (b), selected from the recent study [21], comprising 80 attack variants (details are provided in Appx. B). The selected attack traffic was split into training and testing sets using an 80:20 ratio. Each attack segment was mixed with benign background traffic from the real-world MAWI dataset [105] over a one-minute monitoring period. For training, we employed an autoencoder-based augmentation strategy [106] to generate 2 K flows per attack type by injecting varying levels of noise into the autoencoder's latent space, diversifying network traffic, and strengthening diffusion-based training.

ML Models and Parameters. We trained a conditional U-Net diffusion model using OpenAI's codebase [107], incorporating embeddings for timestamp, flow progression, and flow class labels (details in Table 3). For classification, we used a multiclass CNN for fine-grained learning, reporting binary results to highlight overall performance. The training dataset consists of image sequences generated by our behavioral encoding framework: for each flow, we use ten noisy sketch variants per packet to train the denoiser/forecaster and the corresponding final clean image to train the classifier.

7.2 Metrics

Area Under the ROC Curve (AUC). We used AUC to quantify the performance of attack detection models by comparing the True Positive Rate (TPR) and False Positive Rate (FPR) at various classification thresholds (ROC curve).

Accuracy. We used Accuracy to measure the correctness of a model's predictions compared to the actual values.

F1 Score. We used the F1 score to evaluate the accuracy of the binary classification model. It is defined as the harmonic mean of precision (the proportion of true positive results in all positive predictions) and true positive rate (TPR) to denote the trade-off between precision and TPR.

False Negative and Positive Rates (FNR and FPR). We exam the attack detection performance of the proposed system using FNR (attack as benign) and FPR (benign as attack).

Structural Similarity Index (SSIM). We used SSIM, a perceptual metric that evaluates the similarity between two images for a context-aware assessment of image quality.

7.3 Early Detection via Generative Inference

This section compares *SketchVision* with state-of-the-art ML-based attack detection systems for early detection with a partial view of attack flows.

Setting. We selected one attack type from each category, including Botnet (telnetpwdla) [63, 64], C2 Communication (mazarbot) [62], DDoS (ssdprdos) [8, 10, 11], Data Exfiltration (koler) [19, 59], and Surveillance (adload) [59, 65], and mixed them with benign background traffic, comprising 11.6 M packets and 64 K flows in total over a one-minute monitoring period. We report results of all attack types and detailed configuration of the compared schemes in Appx. C. We allocated 12 MB to sketch-based schemes: *SketchVision* and SketchFeature [29]. In contrast, FlowLens [20] and FlowPic [76] consumed ≈13 MB and ≈256 MB, respectively, proportional to traffic volume. nPrintML [108] used negligible memory for its per-packet detection. For early detection, we apply a small threshold of 5 packets for prediction. *SketchVision* follows *sketch* → *denoiser* → *forecaster* → *classifier* pipeline to derive early detection performance.

Early Detection Performance. Table 1 illustrates the performance of each method. nPrintML [108] adopts a packet-level approach using bit-aligned header representations. However, its reliance on static content ignores temporal flow dynamics, resulting in poor detection performance. FlowLens [20] and SketchFeature [29] rely on per-flow distribution features. While SketchFeature improves memory efficiency over the unbounded FlowLens, neither effectively identifies malicious flows from partial observations (F1 ≈ 0.74 and 0.70). The lack of a forecasting mechanism leads to poor accuracy, as features extracted from early-stage traffic occupy a distinct feature space compared to complete flows (see §2). FlowPic [76] improves performance (F1 = 0.874) by aggregating IPD and PS distributions into a 2D histogram. However, this incurs unbounded memory costs (4 KB/flow). Moreover, the lack of a forecasting mechanism similarly constrains its effectiveness, resulting in 22% of attack flows being missed. *SketchVision* performs per-packet behavioral monitoring (via temporal-aware features) in sublinear memory and utilizes generative inference to forecast from partial early flow data. Consequently, it outperforms all other approaches, achieving an AUC of 0.997 and an F1 score of 0.933 while observing only 20% of the packets, performance unmatched by any other scheme in both feature effectiveness and memory efficiency.

Table 1: Comparing SketchVision with state-of-the-art (supervised) schemes in early detection of low-and-slow attacks.

Schemes	Feature (Grnlrty)	AUC	ACC	F1	FNR	FPR
nPrintML[1] [108]	Hdr Info. (packet)	0.905	0.985	0.202	0.885	0.000
FlowLens [20]	PS Dist. (flow)	0.970	0.964	0.738	0.420	0.000
SketchFeat. [29]	IPD Dist. (flow)	0.992	0.963	0.695	0.460	0.001
FlowPic [76]	PS&IPD Dist. (flow)	0.999	0.982	0.874	0.220	0.000
SketchVision	PS&IPD (packet)	**0.997**	**0.989**	**0.933**	**0.041**	**0.008**

IPD/PS: Inter-Packet Delay/Packet Size **FlowLens, nPrintML**[1] use RF, others CNN

Visual Examples. Figure 7 presents visual examples of representative attacks, for each category of Botnet, C2 Communication, DDoS, Data Exfiltration, and Surveillance, demonstrating how an early partially encoded flow behavior in sketch universe can be denoised and effectively be used to predict final flow behavior using our diffusion model-based forecaster. As shown, the attack flows, encoded from a small early portion of packets, are obscured by sketch noise, making it difficult for humans to discern the true pattern. The diffusion model first captures the context and removes sketch noise, after which the forecasting model predicts the final state of the flow. As shown in the reference data, the predictions are quite close, allowing the classifier to classify them reliably.

Generalizability. Figure 8 evaluates the generalizability of SketchVision and FlowPic under perturbations. While these do not represent targeted adversarial attacks, they test the models' resilience to noise. During the inference phase, we introduce random noise to packet timing and packet size by varying the perturbation rate r from 0 to 1.2. These perturbations are applied independently to each packet with a probability of 0.5; when applied, the original Inter-Packet Delay (IPD) or Packet

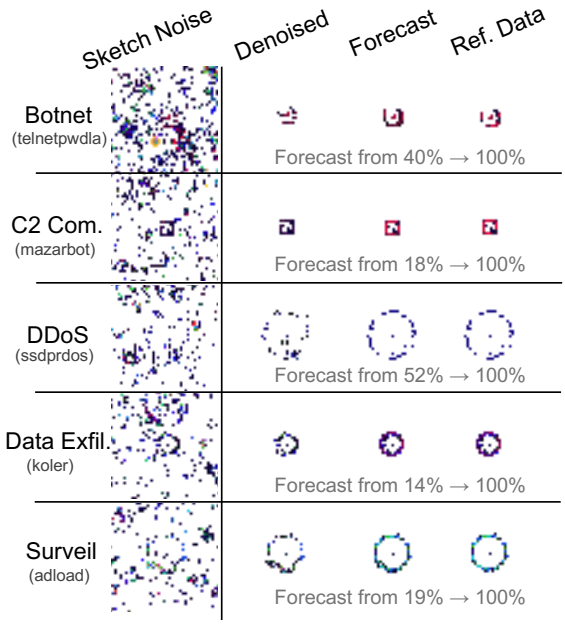
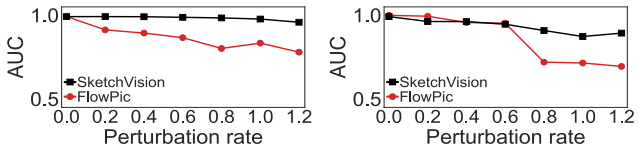


Figure 7: Visual example of representative attack per category, showing sketch denoising and final flow state forecasting.



(a) PS perturbation (b) IPD perturbation

Figure 8: Generalizability analysis under random perturbations of packet size and inter-packet timing.

Size (PS) values are scaled by a factor of $(1+r)$. As shown, SketchVision exhibits superior robustness compared to FlowPic, revealing the vulnerability of flow-level statistics (used by FlowPic) to variations in IPD and PS distributions [29]. This highlights the advantage of finer-grained, packet-level encoding. Moreover, SketchVision maintains high AUC stability thanks to the generative inference of diffusion models, which are known for their resilience to such perturbations [109].

7.4 Diffusion-based Sketch Denoising

This section analyzes the performance of the diffusion model-based sketch denoiser in terms of detection accuracy and feature similarity after denosing.

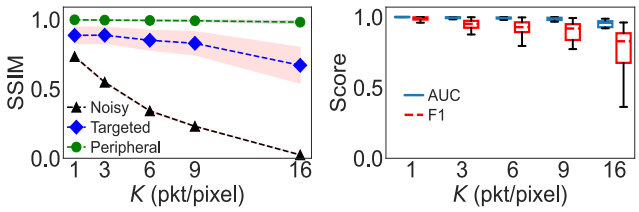
Setting. To evaluate denoising performance, we focus on the final image of each flow to match the input of "denoiser → classifier" pipeline. We mixed each of the 19 attack types with benign background traffic over a one-minute monitoring period, then encoded each resulting set into its own sketch universe. Next, we apply base-point cropping to each flow, retaining only those with at least 20 packets to filter out super-mice benign flows and pass the resulting images to 1) the final classifier (with noise) and 2) the diffusion model for denoising, which is then fed into the classifier (denoised). We assess sketch denoising robustness by varying the **packet-to-pixel ratio** K, where a higher value implies more collisions.

Table 2: Classification performance of the copped image from sketch universe with 1) noise present and 2) after denoising.

Category	Input Data	AUC	ACC	F1	FNR	FPR
Botnet	Baseline	0.999	0.998	0.976	0.021	0.000
	With noise	0.504	0.978	0.000	1.000	0.000
	Denoised	**0.990**	**0.996**	**0.905**	**0.035**	**0.004**
C2 Com.	Baseline	0.999	0.998	0.980	0.009	0.001
	With noise	0.500	0.976	0.000	1.000	0.000
	Denoised	**0.977**	**0.994**	**0.894**	**0.044**	**0.005**
DDoS	Baseline	1.000	0.999	0.999	0.000	0.000
	With noise	0.519	0.959	0.065	0.963	0.000
	Denoised	**0.998**	**0.997**	**0.968**	**0.002**	**0.003**
Data Exfil.	Baseline	0.999	0.999	0.992	0.003	0.000
	With noise	0.500	0.966	0.000	1.000	0.000
	Denoised	**0.953**	**0.986**	**0.848**	**0.091**	**0.012**
Surveil	Baseline	0.999	0.999	0.991	0.002	0.000
	With noise	0.500	0.975	0.000	1.000	0.000
	Denoised	**0.994**	**0.998**	**0.953**	**0.018**	**0.002**

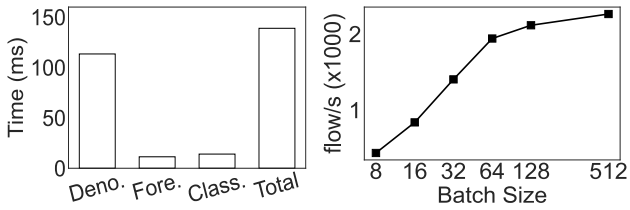
Performance of Sketch Denoising (Ablation Study). To assess performance, we report the average results of all attacks in five categories. Table 2 compares the performance of the classifier in three scenarios, when $K = 3$. The *Baseline* (collision-free scenario without sketch memory sharing) achieves near-perfect detection on clean images, confirming the effectiveness of the temporal-aware feature representation. Conversely, *Without Denoising* fails completely (AUC $= 0.5$) due to the presence of sketch noise. However, *With Denoising* restores near-optimal accuracy (AUC > 0.98, F1 > 0.8), demonstrating the robustness of our data recovery approach.

Effectiveness of Sketch Denoising (Feature Similarity). To further demonstrate the impact of sketch noise, we use the Structural Similarity Index Measure (SSIM) to compare the structural and contextual similarity between the pairs (clean, sketch) and (clean, denoised) images. We note that the choice of SSIM is to restore data distribution, not exact pixel values in diffusion-based denoising. As shown in Figure 9 (a), when sketch noise is present (Noisy), the SSIM is very low up to 0.5 for $K = 3$ and decreases to 0 when $K = 16$, explaining why the classifier struggled to classify flows. However, with the applied denoising model, we observed a significant improvement in data recovery in both the targeted region, where the flow data are encoded, and its peripheral area. The denoising model effectively removed most of the sketch noise, achieving near-perfect SSIM values. In the targeted region, SSIMs



(a) Recovered sketch data (b) ML performance

Figure 9: SSIM of encoded sketches pre- and post-denoising, along with classifier performance under memory pressure.



(a) Per-model latency (b) Overall throughput

Figure 10: Per-model processing time breakdown and inference performance varying batch size.

approach 0.95 as K decreases, enabling robust classification.

Denoiser Robustness varying Sketch Memory. Figure 9 (b) illustrates the classifier performance after sketch denoising, varying packet-to-pixel (K) settings. The boxplots show the AUC and F1 distribution of all 19 attack types. As shown, when $K \leq 9$, the denoiser maintains SSIM scores above 0.8, ensuring robust detection with AUC above 0.98 and F1 scores above 0.8 for all attack types. However, at $K = 16$, the performance drops for half of the attack types with sparse, intermittent signals, which are easily overwhelmed by dense sketch noise. Under such extreme memory pressure, reliable reconstruction becomes untenable, limiting F1 scores to 0.4-0.8.

7.5 Latency and Throughput

Data Plane Module. The complexity of *SketchVision*'s data plane is comparable to the state-of-the-art count-min sketch implementation in the eBPF environment [110], which requires two hash operations and four memory accesses per packet. In particular, our *SketchVision* implementation performs four memory accesses to read precomputed values for functions such as sqrt and sin/cosine, along with two hash operations to locate packets in the sketch. We note that compared to the native eBPF, the sketch overhead is negligible. eBPF can further scale packet processing using multi-core computing, a standard technique [110, 111].

ML Module. Figure 10 (a) shows that *SketchVision* exhibits $\approx$ 138 ms latency for a single image, including denoiser (113 ms), forecaster (11 ms), and CNN (14 ms). We note that the diffusion model's overhead is mainly determined by the number of processing steps, which forms a trade-off with data processing quality (see Appx. D for details). The forecaster model is lightweight with a 1-step forecasting. Meanwhile, the denoiser employs a 10-step denoising for complex sketch noise. The diffusion models are an unavoidable cost to realize *SketchVision*; however, we show that the overall throughput can be improved by using a larger batch size, as illustrated in Figure 10 (b). Targeted low-and-slow threats relax latency requirements compared to a high-volume scenario (e.g., 50 K/s pick flows arrival rate in 2024 [73]), providing edge defense with the necessary computing windows for analysis. Lastly, we note our diffusion model is trained based on a standard and unoptimized reference model [107], for baseline setting, also leaves a huge room for acceleration via model- and hardware-aware optimizations, which is the norm in practice [112–115].

8 Related Work

ML-based detection algorithms [16, 20–23, 30, 82] have boosted the performance of traditional signature- and threshold-based [14, 17, 18, 23, 116] solutions. However, limited memory resources for per-flow data collection pose a trade-off between feature granularity and collection scalability. Aggregation-based approaches [30, 66, 77, 82] sacrifice detail for scale, while distribution-based methods [20, 29, 117] capture richer features but incur unbounded memory usage and ignore temporal dynamics [19, 22, 29]. CPU-based solutions [21, 22, 78] relax memory constraints to capture per-packet temporal features but scale poorly under high-volume traffic. Recent Transformer-based payload analyses [118, 119] are ineffective even for application identification [120], making their extension to attack detection impractical due to the prohibitive memory cost of long packet sequences. Moreover, these post-collection approaches lack early detection capabilities for slow-and-low attacks. In contrast, *SketchVision* enables a scalable per-flow detection from partial observations via rich temporal-aware features.

9 Conclusion

In this paper, we aim to defeat slow-and-low threats in CDN networks, motivated by 1) the lack of a scalable solution for long-term monitoring, 2) the neglect of temporal information in prior work, and 3) unrealistic assumptions in offline training and online detection. We propose *SketchVision*, a vision-inspired approach that encodes packet-level temporal patterns of all flows into a sketch universe. We then leverage diffusion-based denoising to remove sketch noise for each flow, and then perform diffusion-based forecasting to predict the final flow state from partial observations. Experiments verify *SketchVision*'s scalability feasibility of early detection with negligible delay in the edge setting.

Acknowledgments

We would like to thank the anonymous reviewers and our Shepherd, Kevin Hsieh, for their valuable comments and suggestions. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00222385) and (No. RS-2023-NR077143). Lastly, we thank the generous support from the Department of Computer Science and the James and Patricia Anderson College of Engineering at Wayne State University.

References

[1] L. Zheng, X. Li, C. Wang, R. Guo, H. Duan, J. Chen, C. Zhang, and K. Shen, "Reqsminer: Automated discovery of CDN forwarding request inconsistencies and dos attacks with grammar-based fuzzing," in *Proc. of ISOC NDSS*, 2024.

[2] R. Guo, W. Li, B. Liu, S. Hao, J. Zhang, H. Duan, K. Sheng, J. Chen, and Y. Liu, "CDN judo: Breaking the CDN dos protection with itself," in *Proc. of ISOC NDSS*, 2020.

[3] Z. Lin, Z. Lin, X. Liu, J. Chen, R. Guo, C. Chen, and S. Xiao, "CDN cannon: Exploiting CDN back-to-origin strategies for amplification attacks," in *Proc. of USENIX Security*, 2024.

[4] R. Guo, J. Chen, Y. Wang, K. Mu, B. Liu, X. Li, C. Zhang, H. Duan, and J. Wu, "Temporal cdn-convex lens: A cdn-assisted practical pulsing ddos attack," in *Proc. of USENIX Security*, 2023.

[5] S. Hao, Y. Zhang, H. Wang, and A. Stavrou, "End-users get maneuvered: Empirical analysis of redirection hijacking in content delivery networks," in *Proc. of USENIX Security*, 2018.

[6] Y. Gilad, A. Herzberg, M. Sudkovitch, and M. Goberman, "Cdn-on-demand: An affordable ddos defense via untrusted clouds." in *Proc. of ISOC NDSS*, 2016.

[7] U. Naseer and T. A. Benson, "Configanator: A data-driven approach to improving CDN performance." in *Proc. of USENIX NSDI*, 2022.

[8] N. Muraleedharan and B. Janet, "Behaviour analysis of http based slow denial of service attack," in *Proc. of IEEE WiSPNET*, 2017.

[9] X. Luo, R. K. Chang *et al.*, "On a new class of pulsing denial-of-service attacks and the defense." in *Proc. of ISOC NDSS*, 2005.

[10] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE transactions on emerging topics in computational intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[11] Y. Zhou, G. Cheng, Z. Ouyang, and Z. Chen, "Resource-efficient low-rate ddos mitigation with moving target defense in edge clouds," *IEEE Transactions on Network and Service Management*, 2024.

[12] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *Proc. of IEEE ICCST*, 2018.

[13] H. Zhang, D. D. Yao, and N. Ramakrishnan, "Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery," in *Proc. of ACM ASIA-CCS*, 2014.

[14] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches," in *Proc. of USENIX Security*, 2021.

[15] Z. Zhang, G. Xiao, S. Song, R. C. Aygun, A. Stavrou, L. Zhang, and E. Osterweil, "Revealing protocol architecture's design patterns in the volumetric ddos defense design space," *IEEE Communications Surveys & Tutorials*, 2024.

[16] C. Misa, R. Durairajan, A. Gupta, R. Rejaie, and W. Willinger, "Leveraging prefix structure to detect volumetric ddos attack signatures with programmable switches," in *Proc. of IEEE SP*, 2024.

[17] Z. Xu, S. Ramanathan, A. Rush, J. Mirkovic, and M. Yu, "Xatu: Boosting existing ddos detection systems using auxiliary signals," in *Proc. of ACM CoNEXT*, 2022.

[18] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *Proc. of ISOC NDSS*, 2020.

[19] J. Xing, Q. Kang, and A. Chen, "Netwarden: Mitigating network covert channels while preserving performance," in *Proc. of USENIX Security*, 2020.

[20] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications," in *Proc. of ISOC NDSS*, 2021.

[21] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," in *Proc. of ISOC NDSS*, 2023.

[22] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proc. of ACM CCS*, 2021.

[23] S. M. M. Mirnajafizadeh, A. R. Sethuram, D. Mohaisen, D. Nyang, and R. Jang, "Enhancing network attack detection with distributed and in-network data collection system," in *Proc. of USENIX Security*, 2024.

[24] S. Yoo, X. Chen, and J. Rexford, "Smartcookie: Blocking large-scale syn floods with a split-proxy defense on programmable data planes," in *Proc. of USENIX Security*, 2024.

[25] R. Guo, J. Chen, Y. Wang, K. Mu, B. Liu, X. Li, C. Zhang, H. Duan, and J. Wu, "Temporal CDN-Convex lens: A CDN-Assisted practical pulsing DDoS attack," in *Proc. of USENIX Security*, 2023.

[26] B. Nugraha and R. N. Murthy, "Deep learning-based slow ddos attack detection in sdn-based networks," in *Proc. of IEEE NFV-SDN*, 2020.

[27] C. Jung, S. Kim, R. Jang, D. Mohaisen, and D. Nyang, "A scalable and dynamic ACL system for in-network defense," in *Proc. of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 1679–1693.

[28] R. Jang, S. Moon, Y. Noh, A. Mohaisen, and D. Nyang, "Instameasure: Instant per-flow detection using large in-dram working set of active flows," in *Proc. IEEE ICDCS*, 2019.

[29] S. Kim, S. M. M. Mirnajafizadeh, B. Kim, R. Jang, and D. Nyang, "Sketchfeature: High-quality per-flow feature extractor towards security-aware data plane," in *Proc. of ISOC NDSS*, 2025.

[30] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[31] Akamai Technologies, "Akamai inference cloud transforms AI from core to edge with NVIDIA," October 2025. [Online]. Available: https://www.akamai.com/newsroom/press-release/akamai-inference-cloud-transforms-ai-from-core-to-edge-with-nvidia

[32] R. B. Basat, G. Einziger, M. Mitzenmacher, and S. Vargaftik, "Salsa: self-adjusting lean streaming analytics," in *Proc. of IEEE ICDE*, 2021.

[33] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "Fcm-sketch: Generic network measurements with data plane support," in *Proc. of ACM CoNext*, 2020.

[34] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig, "Cold filter: A meta-framework for faster and more accurate stream processing," in *Proc. of ACM SIGMOD*, 2018.

[35] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li, "Pyramid sketch: A sketch framework for frequency estimation of data streams," *Proc. of VLDB Endowment*, vol. 10, no. 11, pp. 1442–1453, 2017.

[36] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[37] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*, 2002, pp. 693–703.

[38] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. of ACM SIGCOMM*, 2002.

[39] M. Chen, S. Chen, and Z. Cai, "Counter tree: A scalable counter architecture for per-flow traffic measurement," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1249–1262, 2016.

[40] G. Gao, Z. Qian, H. Huang, and Y. Du, "An adaptive counter-splicing-based sketch for efficient per-flow size measurement," in *Proc. of IEEE/ACM IWQoS*, 2023.

[41] H. Li, Q. Chen, Y. Zhang, T. Yang, and B. Cui, "Stingy sketch: a sketch framework for accurate and fast frequency estimation," *Proc. of VLDB Endowment*, vol. 15, no. 7, pp. 1426–1438, 2022.

[42] R. Jang, D. Min, S. Moon, D. Mohaisen, and D. Nyang, "Sketchflow: Per-flow systematic sampling using sketch saturation event," in *Proc. of IEEE INFOCOM*, 2020.

[43] D. Nyang and D. Shin, "Recyclable counter with confinement for real-time per-flow measurement," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 3191–3203, 2016.

[44] S. Yang, S. M. M. Mirnajafizadeh, S. Kim, R. Jang, and D. Nyang, "Enhancing resiliency of sketch-based security via lsb sharing-based dynamic late merging," *arXiv preprint arXiv:2503.11777*, 2025.

[45] S. Kim, C. Jung, R. Jang, D. Mohaisen, and D. Nyang, "A robust counting sketch for data plane intrusion detection," in *Proc. of ISOC NDSS*, 2023.

[46] D. Dao, R. Jang, C. Jung, D. Mohaisen, and D. Nyang, "Minimizing noise in hyperloglog-based spread estimation of multiple flows," in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022*. IEEE, 2022, pp. 331–342.

[47] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[48] C. Saharia, W. Chan, H. Chang, C. Lee, J. Ho, T. Salimans, D. Fleet, and M. Norouzi, "Palette: Image-to-image diffusion models," in *Proc. of ACM SIGGRAPH*, 2022.

[49] R. Gandikota, J. Materzynska, J. Fiotto-Kaufman, and D. Bau, "Erasing concepts from diffusion models," in *Proc. of the IEEE/CVF*, 2023.

[50] S. Chen, P. Sun, Y. Song, and P. Luo, "Diffusiondet: Diffusion model for object detection," in *Proc. of the IEEE/CVF*, 2023.

[51] Y. Gilad and A. Herzberg, "Fragmentation considered vulnerable," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 4, pp. 1–31, 2013.

[52] S. DeLaughter and K. Sollins, "Syn proof-of-work: Improving volumetric dos resilience in tcp," in *Proc. of IEEE SP*, 2025.

[53] Y. Zhang, Z. M. Mao, and J. Wang, "Low-rate tcp-targeted dos attack disrupts internet routing." in *Proc. of ISOC NDSS*, 2007.

[54] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants," in *Proc. of ACM SIGCOMM*, 2003.

[55] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks," in *Proc. of ACM IMC*, 2014.

[56] K. Bock, A. Alaraj, Y. Fax, K. Hurley, E. Wustrow, and D. Levin, "Weaponizing middleboxes for TCP reflected amplification," in *Proc. of USENIX Security*, 2021.

[57] H. Yuwen, L. Zhang, Z. Wang, and Y. Kong, "Probability-based delay scheme for resisting sdn scanning," in *Proc. of IEEE ICCC*, 2016.

[58] G. Xie, H. Hang, and M. Faloutsos, "Scanner hunter: Understanding http scanning traffic," in *Proc. of ACM ASIA-CCS*, 2014.

[59] A. Saha, J. Mattei, J. Blasco, L. Cavallaro, D. Votipka, and M. Lindorfer, "Expert insights into advanced persistent threats: Analysis, attribution, and challenges," in *Proc. of USENIX Security*, 2025.

[60] M.-H. Chung, Y. Yang, L. Wang, G. Cento, K. Jerath, A. Raman, D. Lie, and M. H. Chignell, "Implementing data exfiltration defense in situ: a survey of countermeasures and human involvement," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–37, 2023.

[61] M. Guri, A. Kachlon, O. Hasson, G. Kedma, Y. Mirsky, and Y. Elovici, "Gsmem: Data exfiltration from air-gapped computers over gsm frequencies," in *Proc. of USENIX Security*, 2015.

[62] V. Jain, S. M. Alam, S. V. Krishnamurthy, and M. Faloutsos, "C2store: C2 server profiles at your fingertips," *Proc. of the ACM on Networking*, vol. 1, no. CoNEXT3, pp. 1–21, 2023.

[63] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *Proc. of USENIX security*, 2017.

[64] T. Shekari, A. A. Cardenas, and R. Beyah, "Madiot 2.0: Modern high-wattage iot botnet attacks and defenses," in *Proc. of USENIX security*, 2022.

[65] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. X. Song, "Dynamic spyware analysis," in *Proc. of USENIX ATC*, 2007.

[66] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *Proc. of USENIX Security*, 2023.

[67] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[68] CrowdStrike, "What is a command and control (c&c) attack?" 2023. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/command-and-control-cac-attack/

[69] Cloudflare, "What is a low-and-slow attack?" 2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-low-and-slow-attack/

[70] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. of ESANN*, 2015.

[71] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[72] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. of the ACM SIGCOMM on IMC*, 2010.

[73] S. Liu, T. Shaowang, G. Wan, J. Chae, J. Marques, S. Krishnan, and N. Feamster, "Serveflow: A fast-slow model architecture for network traffic analysis," *arXiv preprint arXiv:2402.03694*, 2024.

[74] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. of USENIX NSDI*, 2022.

[75] K. A. Ross, "Efficient hash probes on modern processors," in *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 2007, pp. 1297–1301.

[76] T. Shapira and Y. Shavitt, "Flowpic: A generic representation for encrypted traffic classification and applications identification," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 2, pp. 1218–1232, 2021.

[77] J. Yan, H. Xu, Z. Liu, Q. Li, K. Xu, M. Xu, and J. Wu, "Brain-on-Switch: Towards advanced intelligent network data plane via NN-Driven traffic analysis at Line-Speed," in *Proc. of USENIX NSDI*, 2024.

[78] C. Fu, Q. Li, M. Shen, and K. Xu, "Detecting tunneled flooding traffic via deep semantic analysis of packet length patterns," in *Proc. of ACM CCS*, 2024.

[79] T. Wang, X. Xie, L. Zhang, C. Wang, L. Zhang, and Y. Cui, "Shieldgpt: An llm-based framework for ddos mitigation," in *Proc. of the 8th Asia-Pacific Workshop on Networking*, 2024.

[80] Y. Li, Z. Xiang, N. D. Bastian, D. Song, and B. Li, "IDS-agent: An LLM agent for explainable intrusion detection in iot networks," in *Proc. of NeurIPS Workshop on Open-World Agents*, 2024.

[81] X. Meng, C. Lin, Y. Wang, and Y. Zhang, "Netgpt: Generative pretrained transformer for network traffic," *https://arxiv.org/abs/2304.09513*, 2023.

[82] Y. Dong, Q. Li, K. Wu, R. Li, D. Zhao, G. Tyson, J. Peng, Y. Jiang, S. Xia, and M. Xu, "HorusEye: A realtime IoT malicious traffic detection framework using programmable switches," in *Proc. of USENIX Security*, 2023.

[83] "XDP," https://www.iovisor.org/technology/xdp.

[84] E. Cambiaso, M. Aiello, M. Mongelli, and I. Vaccari, "Detection and classification of slow dos attacks targeting network servers," in *Proc. ACM ARES*, 2020.

[85] A. G. Alcoz, M. Strohmeier, V. Lenders, and L. Vanbever, "Aggregate-based congestion control for pulse-wave ddos defense," in *Proc. of the ACM SIGCOMM Conference*, F. Kuipers and A. Orda, Eds., 2022.

[86] S. Shakkottai, N. Brownlee, and K. Claffy, "A study of burstiness in tcp flows," in *International Workshop on Passive and Active Network Measurement*, 2005.

[87] J. Ho and T. Salimans, "Classifier-free diffusion guidance," *arXiv preprint arXiv:2207.12598*, 2022.

[88] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," in *Proc. of NeurIPS*, 2021.

[89] K. Song, H. Lai, Y. Pan, and J. Yin, "Mimicdiffusion: Purifying adversarial perturbation via mimicking clean diffusion model," in *Proc. of IEEE/CVF*, 2024.

[90] M. Lee and D. Kim, "Robust evaluation of diffusion-based adversarial purification," in *Proc. of IEEE/CVF*, 2023.

[91] L. Lu, R. Achddou, and S. Süsstrunk, "Noise synthesis for low-light image denoising with diffusion models," *arXiv preprint arXiv:2503.11262*, 2025.

[92] C. Yang, L. Liang, and Z. Su, "Real-world denoising via diffusion model," *arXiv preprint arXiv:2305.04457*, 2023.

[93] W. Nie, B. Guo, Y. Huang, C. Xiao, A. Vahdat, and A. Anandkumar, "Diffusion models for adversarial purification," *arXiv preprint arXiv:2205.07460*, 2022.

[94] J. Wang, Z. Lyu, D. Lin, B. Dai, and H. Fu, "Guided diffusion model for adversarial purification," *arXiv preprint arXiv:2205.14969*, 2022.

[95] M. Kang, D. Song, and B. Li, "Diffattack: Evasion attacks against diffusion-based adversarial purification," *Advances in Neural Information Processing Systems*, vol. 36, pp. 73 919–73 942, 2023.

[96] W. Feller, *An Introduction to Probability Theory and Its Applications, Vol. 1*, 3rd ed. Wiley, 1968.

[97] A. Bansal, E. Borgnia, H.-M. Chu, J. Li, H. Kazemi, F. Huang, M. Goldblum, J. Geiping, and T. Goldstein, "Cold diffusion: Inverting arbitrary image transforms without noise," in *Proc. of NeurIPS*, 2023.

[98] M. Kim, D. Ki, S.-W. Shim, and B.-J. Lee, "Adaptive non-uniform timestep sampling for accelerating diffusion model training," in *Proc. of IEEE CVF*, 2025.

[99] K. Rasul, C. Seward, I. Schuster, and R. Vollgraf, "Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting," in *Proc. of ICML*, 2021.

[100] R. Feng, Y. Gao, T. H. E. Tse, X. Ma, and H. J. Chang, "Diffpose: Spatiotemporal diffusion model for video-based human pose estimation," in *Proc. of the IEEE/CVF*, 2023.

[101] X. Guo, Y. Zhang, B. Chen, H. Xu, J. Wang, and M. Long, "Dynamical diffusion: Learning temporal dynamics with diffusion models," *arXiv preprint arXiv:2503.00951*, 2025.

[102] C. Meng, Y. He, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon, "Sdedit: Guided image synthesis and editing with stochastic differential equations," *arXiv preprint arXiv:2108.01073*, 2021.

[103] Y. Zhou, X. Xiang, M. Kiley, S. Dharanipragada, and M. Yu, "DINT: Fast In-Kernel distributed transactions with eBPF," in *Proc. of USENIX NSDI*, 2024.

[104] A. Appleby, "Original murmurhash3 c++ code," 2016. [Online]. Available: https://github.com/aappleby/smhasher/tree/master

[105] WIDE Project, "MAWI Working Group Traffic Archive," http://mawi.wide.ad.jp/mawi/.

[106] T.-H. Cheung and D.-Y. Yeung, "Modals: Modality-agnostic automated data augmentation in the latent space," in *Proc. of ICLR*, 2020.

[107] OpenAI, "Guided diffusion," https://github.com/openai/guided-diffusion, 2021.

[108] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proc. of ACM SIGSAC*, 2021.

[109] H. Chen, Y. Dong, S. Shao, Z. Hao, X. Yang, H. Su, and J. Zhu, "Diffusion models are certifiably robust classifiers," in *Proc. of NeurIPS*, 2024.

[110] S. Miano, X. Chen, R. B. Basat, and G. Antichi, "Fast in-kernel traffic sketching in eBPF," *ACM SIGCOMM Computer Communication Review*, vol. 53, no. 1, pp. 3–13, 2023.

[111] P. Enberg, A. Rao, and S. Tarkoma, "Partition-aware packet steering using xdp and ebpf for improving application-level parallelism," in *Proc, of ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms*, 2019.

[112] X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Kang, S. Zhang, and K. Keutzer, "Q-diffusion: Quantizing diffusion models," in *Proc. of IEEE/CVF*, 2023.

[113] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," in *Proc. of NeurIPS*, 2022.

[114] X. Ma, G. Fang, and X. Wang, "Deepcache: Accelerating diffusion models for free," in *Proc. of IEEE/CVF*, 2024.

[115] H. Wu, Y. Yu, J. Deng, S. Ibrahim, S. Wu, H. Fan, Z. Cheng, and H. Jin, "StreamBox: A lightweight GPU SandBox for serverless inference workflow," in *Proc. of USENIX ATC*, 2024.

[116] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized Link-Flooding defense against adaptive adversaries," in *Proc. of USENIX Security*, 2021.

[117] J. Guo, Y. Hong, Y. Wu, Y. Liu, T. Yang, and B. Cui, "Sketchpolymer: Estimate per-item tail quantile using one sketch," in *Proc. of ACM SIGKDD*, 2023.

[118] R. Zhao, M. Zhan, X. Deng, Y. Wang, Y. Wang, G. Gui, and Z. Xue, "Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation," in *Proc. of the AAAI*, 2023.

[119] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proc. of ACM WWW*, 2022.

[120] Y. Zhao, G. Dettori, M. Boffa, L. Vassio, and M. Mellia, "The sweet danger of sugar: Debunking representation learning for encrypted traffic classification," in *Proc. of the ACM SIGCOMM*, 2025.

[121] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *Proc. of ICML*, 2021.

A Forecaster Variants and Extension

This section evaluates the performance of the diffusion model for forecasting using three forecasters: autoregressive, one-step, and joint unified denoising and forecasting. Table 3 lists the hyperparameters for diffusion models.

Table 3: Diffusion model training hyperparameters.

Hyperparameter	**Value**	**Hyperparameter**	**Value**
Image Size	64×64	Batch Size	32
Epochs	4000	Diffusion Steps (T)	2000
Base Channels	64	Beta Schedule	Cosine
Loss Type	L2	Learning Rate	1×10^{-4}
Attention Resolutions	32, 16, 8	Num. Attention Heads	4
Number of Classes	20	Training Cutoff τ	800

A.1 Autoregressive and One-step Forecasting

To train the model $P(\mathbf{x}^{p+s} \mid \mathbf{X}_{past})$, where $\mathbf{X}_{past} = \mathbf{x}^{p-w\cdot s}, \mathbf{x}^{p-(w-1)\cdot s}, \ldots, \mathbf{x}^{p-s}$, we set the fixed intervals as $s = 5$ packets, with a window of $w = 5$ observations. This corresponds to observing 25 past packets at intervals of 5. We extend the input of the diffusion model to include $\mathbf{X}_{past}$ and apply channel-wise convolution for feature extraction, which serves as the latent conditioning for denoising. Training is then performed with input pairs $(\mathbf{x}^{p+s}, \mathbf{x}^{p})$, conditioned on $c = (l, p, \mathbf{X}_{past})$. Each flow forecasting pipeline is performed at fixed intervals of $s = 5$ packets, with a window of $w = 5$ past observations to predict the flow state in the next 5 packets. Then, each new 5-packet segment arrives, a new observation is formed, and predictions continue autoregressively up to 100 future packets (20 generative inferences in total), with an upper limit of 150 packets, meaning that no forecasting is performed for flows larger than 150 packets, as we found this number already gives optimal performance.

Results. At each step, flow classification is performed, and if a flow is classified as malicious with confidence above 0.9, the flow is labeled as an attack; otherwise, the process repeats over the next arrival of s packets. As shown, a clear drawback of the autoregressive approach is that we need to maintain the history for each flow to use in the next forecasting, which adds overhead. In terms of performance, however, we found that the overall classification results are comparable to the one-step forecaster and the ideal, noise-free condition, as shown in Figure 11. However, due to the accumulation of noise over the forecasting period, the autoregressive approach detects traffic later, as shown in Figure 12 when $K = 9$, with 90% of attacks detected after observing 50% of the packets.

A.2 Joint Denoising and Forecasting

In this section, we aim to determine whether a model can naturally filter out sketch noise and, even in its presence, forecast the mature flow state, i.e., $P(\mathbf{x}^{N} \mid \tilde{\mathbf{x}}^{p})$. For this purpose, we employ Algorithm 2 to train on pairs $(\mathbf{x}^{N}, \tilde{\mathbf{x}}^{p})$ conditioned on (l, p). Therefore, we reduce the forecasting pipeline to *skerch* $\rightarrow$ *denoise-and-forecast* $\rightarrow$ *classify*, improving efficiency and reducing computational overhead.

Results. We use the same setup as the generative inference experiment using our one-step forecaster in Appx. C. That is, the cropped sketch images are fed directly into the diffusion model to denoise and predict the flow's final-stage behaviors. Figure 13 demonstrates the performance. We observed that performance declined compared to having a separate denoiser due to the complex sketch noise pattern. However, the model is still capable of learning meaningful patterns and achieving strong forecasting performance when K is smaller than 6. As shown in Figure 14, compared to the one-step forecaster, it detects 90% of attacks after observing 40% of packets, whereas the one-step model requires 25%. The main advantage of this unified approach is faster inference in one diffusion step, reducing total denoising and forecasting time to 11 ms using a single diffusion model.

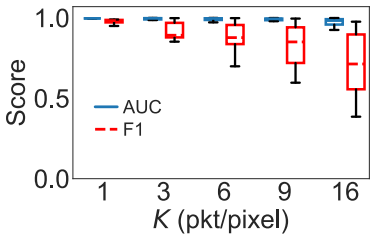
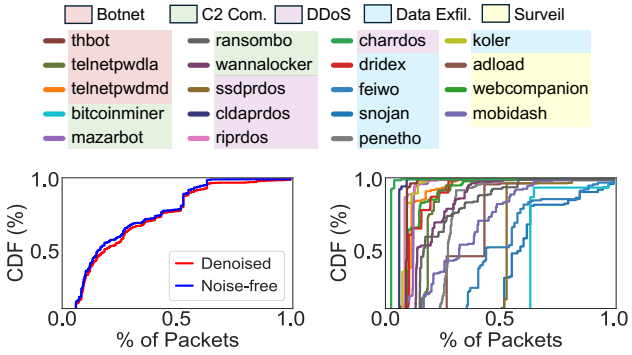


Figure 11: Autoregressive forecasting performance.



(a) Denoised vs. Noise-free (b) Attack-wise CDF

Figure 12: Early attack detection using autoregressive model: (a) shows detection timelines for noise-free vs. denoised images; (b) shows detection timing across diverse attack types.

B Dataset Selection

From the 80 attack variants introduced in [21], we identified 30 types exhibiting slow-and-low characteristics based on two criteria: (i) a flow duration exceeding 5 seconds, and (ii) an activeness ratio below 0.5 (indicating $>$ 50% idle time). We excluded the remaining 11 categories from analysis due to insufficient sample size or because the attacks were too small to be considered significant, resulting in the final 19 types. The benign background traffic was sourced from the WIDE MAWI project (Vantage Point G, AS 2500), collected between January and June 2020. We applied an 80:20 train-test split exclusively to the attack traffic. The 80% training partition served as seed data for autoencoder-based augmentation, generating 2 K synthetic flows per attack type. These were combined with 50 K benign flows for the training dataset.

C Early Detection via Generative Inference (Detailed)

The detailed description of generative inference is as follows.

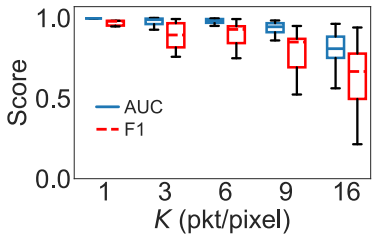
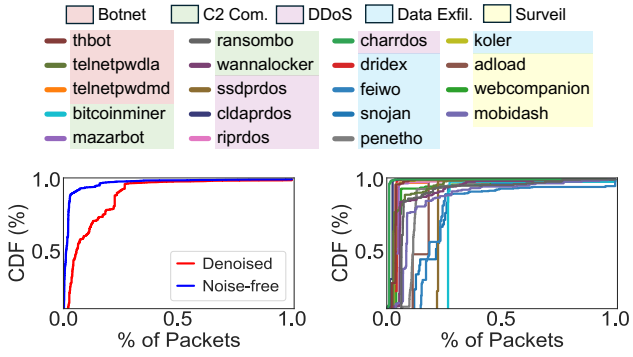


Figure 13: Joint denoising-forecasting performance.



(a) Denoised vs. Noise-free (b) Attack-wise CDF

Figure 14: Early attack detection using joint model: (a) shows detection timelines for noise-free vs. denoised images; (b) shows detection timing across diverse attack types.

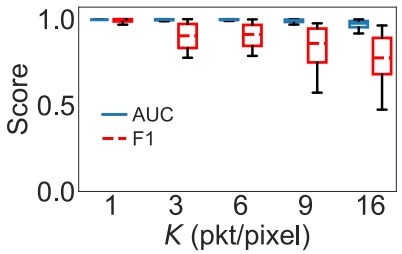
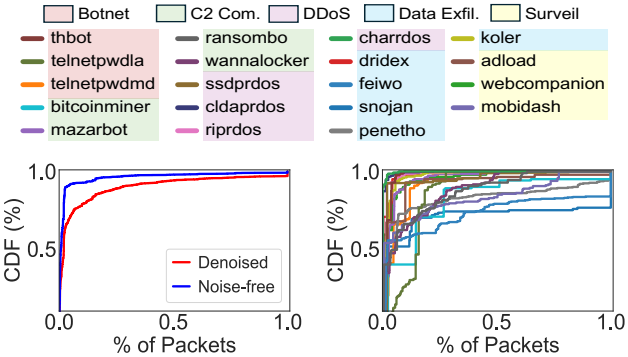


Figure 15: One-step forecasting performance.



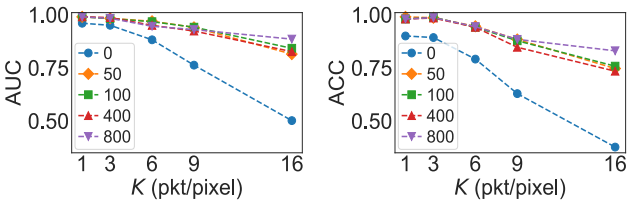
(a) Denoised vs. Noise-free (b) Attack-wise CDF

Figure 16: Early attack detection using one-step forecasting: (a) shows detection timelines for noise-free vs. denoised images; (b) shows detection timing across diverse attack types.

Early Detection Configuration. Since none of these works are designed for early detection, we report the result when collecting data for the first 150 packets of a flow, by sending the decoded information directly to the classifier (trained on full data). For *SketchVision*, we perform early detection every 5 packets once the flow size reaches 20, and limit per-flow observation up to 150 packets to reduce overhead. Specifically, we crop the image from the sketch universe and process it through *denoiser* $\rightarrow$ *forecaster* $\rightarrow$ *classifier* pipeline. For *SketchVision*, we use the autoregressive forecaster model described in §5 to derive one-step forecaster; specifically, we set $s = N - p$ and $w = 0$, which corresponds to predicting the final state $\mathbf{x}^N$ of a flow given the first p encoded packets, i.e., $P(\mathbf{x}^N|\mathbf{x}^p)$. To ensure high-confidence predictions, we use a decision threshold of 0.9. Additionally, we apply the average score over a sliding window of size 5: if the average score within the window indicates an attack, the sample is classified as an attack; otherwise, it is considered benign. This method helps mitigate fluctuations in classification for certain samples, including occasional misclassifications between attacks and benign flows during early stage forecasting (see §5).

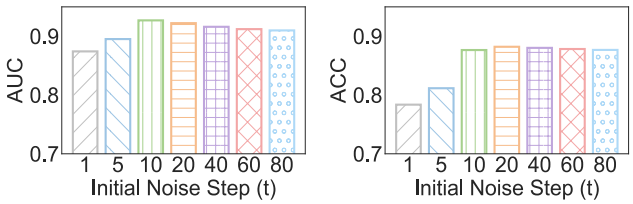
Performance varying Memory. Figure 15 presents boxplots of per-attack-type early detection performance as K varies. In general, when $K \leq 6$, the early-stage behaviors of different types of attacks are very similar, and performance is comparable to that of the autoregressive model. As K increases beyond 6, classifier performance degrades noticeably. Unlike the autoregressive approach, which leverages five historical images, the one-step approach relies on a single observation. Consequently, due to the similarity in early-stage attack behaviors, we observe that the diffusion model forecasts some attack flows into other attack types; nevertheless, these flows are still labeled and counted as attacks in our binary evaluation.

Attack-wise Detection. Figure 16 (a) compares early detection performance of *SketchVision* using the ground-truth clean image (noise-free) versus the denoised image as input to the forecaster in the "forecaster $\rightarrow$ classifier" pipeline. As shown, predictions based on denoised data closely follow the ground-truth, emphasizing the effectiveness of the denoiser. Notably, 86% of attacks are detected with only 20% of a flow's packets, rising to 95% with 50% observed packets, highlighting a key milestone in proactive network defense. Figure 16 (b) shows the CDF of early detection per 19 attack types. Most attack flows are detected within the first 10-30% of packets, though a few (Feiwo, Snojan, Penetho, Bitcoinminer) require more due to sparse or benign-like early behavior. Overall, we can conclude that slow-and-low attacks can be predicted using only a small fraction of packets, enabling a practical online detection under stream data.



(a) Attack detection AUC (b) Attack detection ACC

Figure 17: Denoising performance of the diffusion process under varying levels of Gaussian noise added during training.



(a) Attack detection AUC (b) Attack detection ACC

Figure 18: Denoising performance of the diffusion model with different levels of initial Gaussian noise.

D Gaussian Noise Impact

We study the impact of Gaussian noise on the diffusion model's ability for effective denoising and examine sampling choices during denoising at inference time.

Training Noise Step Limit. By selecting a large T (e.g., 2000), Gaussian noise is gradually added at each step of the forward process, which promotes stable training. However, this introduces additional unnecessary steps for the denoising task, since steps near T primarily contribute to pure image generation [121]. Consequently, reducing the number of training steps can significantly save computation time without degrading denoising performance. We define $\tau \in [0,T]$ as the maximum timestamp for the amount of Gaussian noise added during training and investigate how this choice impacts the diffusion model's performance. To achieve this, we trained multiple diffusion models using the same number of epochs while varying τ, and evaluated their performance under different levels of sketch noise by adjusting the K (packet-to-pixel ratio) from 1 to 16. Figure 17 illustrates the performance of a binary classifier when the denoiser is trained with various τ. Notably, when $\tau = 0$, no Gaussian noise is added during training, reducing the model to a pure U-Net or autoencoder. As shown, this configuration leads to poor denoising performance, as the model fails to learn robust representations necessary for effective denoising. Moreover, we observe that the impact of τ becomes more pronounced under high sketch noise, i.e., when K becomes bigger than 9. In these high-noise settings, higher values of τ improve denoising performance, suggesting that exposing the model to more noise during training enhances its robustness. In contrast, when exposed to a less noisy sketch, the choice of τ has a smaller effect on performance. Based on these results, we selected an upper bound of 800 for training the diffusion model and Gaussian noise sampling, which provides the best performance.

Sampling Noise Initialization. During sampling, when varying the sketch noise, the question is how much Gaussian noise should be added to the input to initiate the backward process and achieve optimal denoising. Additionally, does this choice affect the detection performance? To explore this, we focused on the case where $K = 9$ and analyzed how different amounts of added noise affect the results. Figure 18 shows the classifier's performance as the initial noise level (timestamp) varies from 1 to 80. Performance improves with increasing noise levels up to $t = 10$, after which gains become negligible. Since lower timestamps reduce computation and speed up inference, and $t = 10$ provides a good balance between accuracy and efficiency, we selected this value for denoising the sketch. For forecasting during flow progression, we used $t = 1$, which yielded the best results. Further increasing the timestamp provided only marginal improvements relative to the computational cost.