CARAVAN: Practical Online Learning of In-Network ML Models with Labeling Agents

Qizheng Zhang, Ali Imran[†], Enkeleda Bardhi[‡], Tushar Swamy, Nathan Zhang,
Muhammad Shahbaz[†⋆], Kunle Olukotun
Stanford University [†]*Purdue University* [‡]*Sapienza University of Rome* [⋆]*University of Michigan*

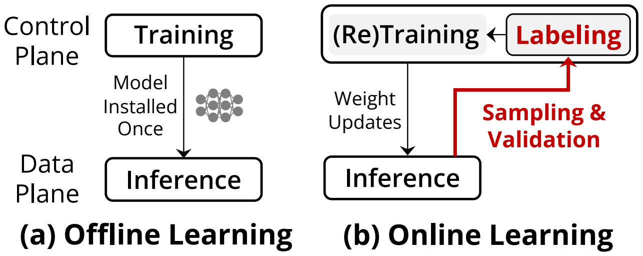
Abstract

Recent work on in-network machine learning (ML) anticipates offline models to operate well in modern networking environments. However, upon deployment, these models struggle to cope with fluctuating traffic patterns and network conditions and, therefore, must be validated and updated frequently in an online fashion.

This paper presents CARAVAN, a practical online learning system for in-network ML models. We tackle two primary challenges in facilitating online learning for networking: (a) the automatic labeling of evolving traffic and (b) the efficient monitoring and detection of model performance degradation to trigger retraining. CARAVAN repurposes existing systems (e.g., heuristics, access control lists, and foundation models)—not directly suitable for such dynamic environments—into high-quality labeling sources for generating labeled data for online learning. CARAVAN also introduces a new metric, *accuracy proxy*, to track model degradation and potential drift to efficiently trigger retraining. Our evaluations show that CARAVAN's labeling strategy enables in-network ML models to closely follow the changes in the traffic dynamics with a 30.3% improvement in F1 score (on average), compared to offline models. Moreover, CARAVAN sustains comparable inference accuracy to that of a continuous-learning system while consuming 61.3% less GPU compute time (on average) via accuracy proxy and retraining triggers.

1 Introduction

Machine learning (ML) is being increasingly leveraged to better manage and operate networks today [27, 30, 35, 37, 45, 49, 54, 77, 80, 82, 97, 101, 104, 105, 111, 114, 116]. In academia, several proposals make a case for using ML to improve systems security through anomaly and intrusion detection [30, 54, 105] and to optimize systems performance through inference, diagnosis, and forecasting of systems' behavior [49, 76, 80, 81]. Correspondingly, in industry, ML is being deployed to detect threats and bots in public and enterprise-scale cloud networks [1, 9, 10] for securing physical and virtual infrastructure and for providing better user experience by predicting network incidents and congestion early on [8]. Moreover, to operate at scale, with high throughput and low latency, the model inference is being offloaded to the data plane (e.g., programmable switches [15, 114] and SmartNICs [17, 101]) in the network (i.e., in-network ML)—to perform decision-making on a per-packet basis [37, 104, 122].



(a) Offline Learning (b) Online Learning

Figure 1: Comparison of in-network model learning. (a) Offline learning: trained and deployed once; (b) Online learning: trained and updated over time—requires iterative sampling, labeling, and validation.

Unlike conventional approaches (e.g., hand-crafted heuristics and static rulesets), ML models are better at revealing hidden patterns and characteristics in vast amounts of high-dimensional data—such as network traffic [35, 54, 71, 104, 111, 116, 117]. However, most efforts on replacing traditional approaches (e.g., heuristics and access control lists) with ML [35] are limited to using static models (aka offline learning, Figure 1a) [30, 35, 37, 49, 54, 82, 97, 101, 104, 105, 111, 114, 116]. These models are trained once using synthetic or controlled network traces and are expected to operate well in the real environment without further guidance (or retraining). While showing significant promise in stable (less volatile) environments, these static models perform poorly in the presence of fluctuations and unforeseen events—not captured by the traffic during the initial training phase [33, 115, 117]. These manifest as model drift either (a) when the network environment gradually evolves or suddenly changes due to traffic bursts, time-of-day, or rare events (called concept drift) [117] or (b) when new data patterns arrive or data distribution changes (called data drift) [33]. This model drift is shown to be prevalent in many online ML applications [33, 94, 100, 115].

To keep these models up-to-date with new patterns and network behavior, one approach is to train and update them continuously on the incoming traffic—referred to as online learning or continuous learning [33, 94]. For example, a (re)training pipeline in the control plane can continuously sample packets from the network (e.g., using INT [68] or NetFlow [7]), label them, and pass them to the model for retraining (Figure 1b). It then updates the weights on the data-plane device, performing model inference. As we show in our evaluations (§5), keeping the model current through online training allows it to handle new incidents with much higher accuracies compared to the static offline models (i.e., the average difference in accuracy is as high as 67%).

However, there are a number of challenges when it comes to enabling continuous learning in modern networking environments (processing Tbps of traffic for varying tenants and workloads) [32, 96, 119, 121]. First, unlike traditional online learning systems in other domains (e.g., recommendation systems and financial systems) where the new retraining data either contains labels (ground truth) [59, 103, 113] or can be easily labeled using existing approaches (like Data Programming [93] or Weak Supervision [91, 92]), in networking the incoming data is raw (sampled) traffic with no labels. *Challenge #1: How can we prepare (and label) traffic data for retraining in-network models?* Second, we cannot rely on fixed interval-based or periodic retraining to ensure the installed models perform well. The network conditions are highly dynamic and erratic; a large interval will miss such variations, whereas frequent updates would be too costly in terms of resource usage (CPU/GPU cycles and network bandwidth). *Challenge #2: How to decide when to trigger retraining?*

In this paper, we present CARAVAN, an online learning system for in-network ML to tackle these challenges. To label new incoming network traffic, CARAVAN relies on labeling agents that use different user-defined knowledge sources to assist with labeling. In networking, many existing systems, such as heuristics, access control lists, deep learning, or even foundation models (e.g., GPT-4 [87], Gemini [106], Llama 3 [16]), fare poorly when used for real-time decision-making—they either fail to adapt to changing network conditions or take too much time to process. However, we observe that these can be used as knowledge sources to label incoming traffic for online learning of in-network models. For instance, using foundation models (which encode a broad spectrum of information about the environment [89, 95, 112]) and guidance from users (e.g., prompts [28] and document retrievals [72]), we can generate application-specific, weak-supervision labels to (re)train these models. We also introduce a new metric, *accuracy proxy*, to decide when to trigger retraining. Instead of relying on ground-truth labels to compute model accuracy, we compute accuracy proxy based on generated labels we receive from the labeling agents for model (re)training. Doing so allows CARAVAN to track degradation in model behavior through relative changes in the accuracy level on a temporal scale, and to trigger retraining. More specifically, if there is an abrupt change in the accuracy proxy (i.e., model drift exceeds a certain threshold), CARAVAN uses this as a signal to trigger retraining. This limits CARAVAN from excessively retraining the model under normal conditions.

We evaluate our CARAVAN system both in simulation (for microbenchmarks) and with a Taurus FPGA-based switch [104] (for end-to-end results). Our simulation results show that labels generated using knowledge sources perform similarly to ground-truth labels in terms of inference accuracy when used to label incoming traffic for retraining. Moreover, our accuracy proxy and retraining triggers save up to 74.55% GPU compute time compared to continuous online training while sustaining similar accuracy gains. With our Taurus FPGA testbed, we show that CARAVAN maintains 30% higher accuracy on average compared to offline models while using 56.23% less CPU and with similar memory footprint compared to continuous retraining baselines—with CARAVAN, the model operates at line rate while adapting to changing traffic dynamics.

In summary, we make the following contributions:

- We present CARAVAN, a practical online learning system for in-network ML. CARAVAN's labeling-agent strategy allows the use of existing network systems (e.g., heuristics, access control lists) and emerging foundation models (e.g., GPT-4, Gemini, and Llama 3) as knowledge sources to label incoming traffic. Using accuracy proxy further allows CARAVAN to efficiently retrigger the training pipeline while closely tracking changes in the network conditions.
- We implement CARAVAN as a software logic running in the control plane, and test it both in a simulation setting and using a real testbed with Taurus FPGA-based switches. Our CARAVAN prototype is available as open-source.[1]
- Our evaluations show that CARAVAN allows in-network models to track changes in the network at line rate while sustaining 30.3% higher F1 score (on average) compared to offline systems. Moreover, it consumes 61.3% less GPU compute time (on average) than a continuous-learning system by selectively triggering retraining via accuracy proxy.

2 Background & Motivation

In-network Machine Learning. Network operators face many challenges with managing the size and complexity of modern networks while maintaining their stringent (and ever-increasing) performance requirements [32, 96, 119, 121]. Over time, the networking community has developed a plethora of hand-tuned heuristics permeating the network, which continuously introduce new parameters that must then be tuned to the given network (and workload). We see this with the constant iterations of congestion-control variants [55, 73], active-queue management [23], load balancing algorithms [26, 66], anomaly detection [25, 34, 38] and more. Relying on network developers and researchers to keep adding new parameters to each algorithm being used throughout the network, as the workloads change and evolve, has limited scalability as networks grow. Networking researchers have, therefore, begun to turn toward data-driven algorithms, in the form of ML, particularly deep-learning and neural networks [35, 40, 63, 80, 83, 105, 111, 116]. Rather than tuning individual model weights by hand, ML algorithms take training data as input and learn model weights to optimize for performance metrics (e.g., prediction accuracy).

To operate at scale with high throughput and low latency, these models are further offloaded to the network data plane

[1] Artifact: https://github.com/Per-Packet-AI/Caravan-Artifact-OSDI24

(e.g., programmable switches and SmartNICs) [17, 101, 104, 114, 125]. Doing so allows more fine-grained control over the traffic, with decision-making (and model inference) taking place at or near the packet level. For example, programmable switches (e.g., Intel Tofino series [15]) with match-action tables (MATs) can perform ML algorithms (such as SVMs and decision trees) [37, 114], with more recent data-plane devices incorporating MapReduce-based processing blocks to run deep neural networks (DNNs) directly in the network [104]. Likewise, emerging SmartNIC devices (e.g., Marvell Octeon 10 [17] and Xilinx SN1000 [2]) come equipped with on-board ML inference engines for per-packet inference. Similarly, data/infrastructure processing units (DPUs/IPUs) from Nvidia [6], AMD [4], and Intel [14] also provide computational resources capable of running ML inference alongside the packet-processing pipelines.

Online Learning and Model Drifts. Recent work on applying online learning in networking domains (such as video analytics and edge monitoring) shows promising results. For example, Ekya [33] and RECL [67] demonstrate that retraining computer vision models for video analytics applications with new video frames can effectively mitigate data drift for compressed ML models. Nazar [58] features online monitoring and adapts various ML models on mobile devices to relieve the problem of potential model drifts.

Through retraining ML models with new incoming data, online learning addresses two common issues these models face post-deployment: concept drift [117] and data drift [33]. Concept drift occurs as networks and traffic are subjected to dynamic signal interference due to environmental changes [123] (e.g., weather, temperature, or time-of-day), as well as changes in the network and user behavior (e.g., increased online activity during COVID-19 [46], addition/removal of devices and software due to upgrades or failures [75]). For example, a large file download may be classified as benign during the day when networks are more active but are marked as malicious during the night when the number of high-volume flows is smaller. On the other hand, data drift happens when the live traffic (or data) distribution diverges from the training data distribution after the model is deployed [33, 94]. For classification models, in particular, the arrival of new data classes (not already present in the offline training set) or a change in data class distribution could cause an ML model to perform poorly [33, 94]. For example, in network security, new attacks come up without warning, and it becomes challenging for a static ML model to detect such an attack since it was not trained on data featuring the new attack.

Network Data Labeling. The emerging interest in training and testing ML models for networking applications sparks extensive research in the area of obtaining labeled network data [42, 60, 98, 99]. Most recent work falls into three different categories: generating labeled network data in a controlled environment, synthetic data generation, and manual labeling through domain experts (i.e., network operators).

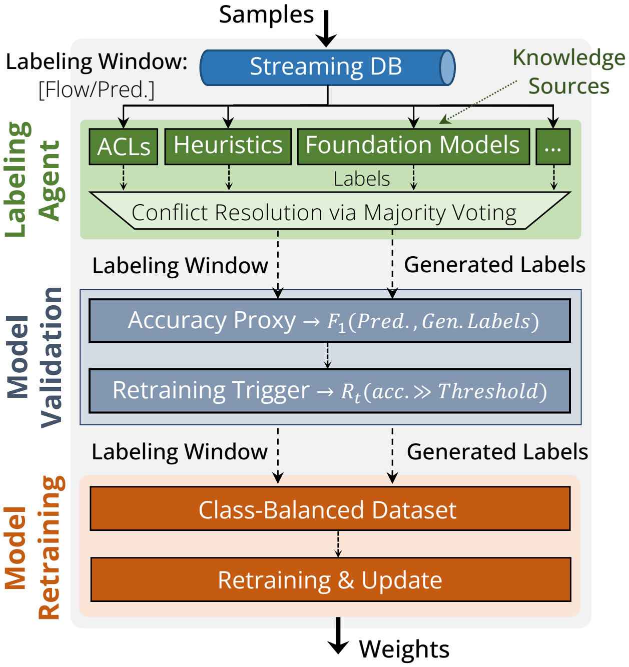


Figure 2: High-level design of Caravan. The three key components, Labeling Agent (§3.1), Model Validation (§3.2), and Retraining, work in tandem to keep the in-network ML model up-to-date.

Efforts like NetUnicorn [31] propose to collect and actively label network data in a controlled environment where operators can access different nodes (switches and hosts) in the network. Though labeling accuracy would be high since operators can choose to generate and collect selected traffic classes, this approach might not offer representative labeled data in real networks [53]—limiting its use in online learning. Other efforts feature synthetic data generation, where models like GANs [118] or diffusion models [65] are used to produce packet traces that match the feature distribution of input network data. However, the generation process takes a lot of time and cannot explicitly label new incoming data, making it impractical in an online setting. Also, it is unclear how closely the synthetic data reflects the traffic in a real environment (an open area of research [57, 107, 109]). The last resort is to ask human experts with domain knowledge to label all or selected network data. Though there are many efforts featuring selecting sampled data for human experts to label [53], this still requires a human-in-the-loop and may not operate at the timescales needed for automatic data labeling in networks.

3 Design of Caravan

We present Caravan, a system for practical online learning of in-network ML models deployed in the data plane. Caravan is designed to satisfy the following requirements: (1) generation of effective label datasets for retraining and (2) efficient monitoring and detection of model performance.

Overview. Figure 2 shows the high-level architecture of CARAVAN. The system periodically collects a window of samples, arriving from the data-plane device running in-network ML inference. Each sample contains a set of header fields (called flow) along with the prediction made by the deployed model. Once a window is full, a labeling agent (§3.1) generates application-specific (e.g., security) labels for each sample in the window in the form of class predictions (e.g., type of network attack) or confidence scores (e.g., the likelihood a flow being malicious). The agent relies on a collection of knowledge sources (§3.1.1), each generating its own labels. The label with the most votes (i.e., occurrences) is added to the final label set. Next, the validation stage (§3.2) monitors and detects the degradation in the model performance using a new metric, called *accuracy proxy* (§3.2.1), which uses predicted values and generated labels to measure the model's accuracy on the received samples in the window. Based on the accuracy values (e.g., exceeding a certain threshold), the retraining trigger stage decides whether retraining is necessary (e.g., in the presence of new types of attacks missed by the in-network ML model) for the current window of incoming samples. If an update is required, the final model retraining stage will generate a balanced dataset from the window of samples received, i.e., a mix of malicious and benign flows and generated labels. After training is complete, the in-network model is updated with the new weights to detect the new types of missed attacks.

3.1 Labeling Agent

The first component in CARAVAN is the labeling agent. It generates application-specific labels that can be used in the later stages of model validation and online retraining for: (1) computing an accuracy proxy that can signal potential model accuracy degradation to efficiently trigger retraining, and (2) generating a class-balanced labeled dataset for retraining when necessary. To generate labels for new incoming network traffic automatically and accurately, the labeling agent relies on external knowledge sources. Knowledge sources (§3.1.1) are defined to be entities or applications that can be repurposed to assist with data labeling (e.g., access control lists, heuristics, foundation models). They can be defined and provided by users through a user interface (§3.1.2).

When a full window of samples from the data plane is available, the labeling agent reads these samples and the associated inference results from a streaming database (e.g., InfluxDB [13] or Apache Kafka [5]). Then, it sends a labeling request to every knowledge source it relies on. With one label from each knowledge source, the labeling agent would do a majority voting to determine the final set of labels to be used (also called *generated labels*) for the current window of data samples. These generated labels will be sent to the next stage of CARAVAN for model validation (Figure 2).

3.1.1 Knowledge Sources.

The labeling agent relies on knowledge sources for labeling. We define knowledge sources to be any entities or services that contain useful information about the user-defined application and can be repurposed to assist data labeling. Take network intrusion detection as an example. A common knowledge source is IP-based access control lists (ACLs) [52] that can block network flows or packets from certain source IP addresses considered to be of malicious origins. With IP-based ACLs as a knowledge source, the labeling agent is able to label a flow as *malicious* if its source IP is on the list. Another example is foundation models. With appropriate adaptation, a foundation model can assist with downstream tasks in networking, such as traffic classification, by functioning as a multi-class classifier [54].

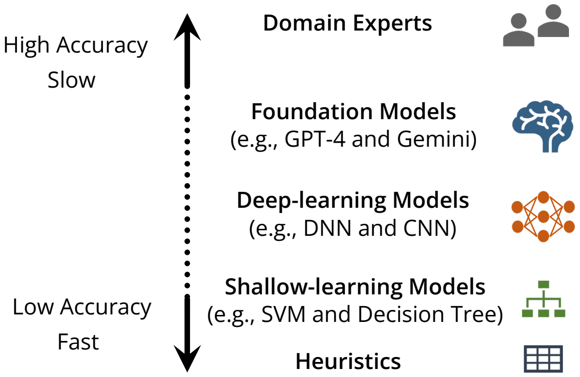


Figure 3: Classifying network knowledge sources across a spectrum based on accuracy and speed.

CARAVAN repurposes different knowledge sources in different ways for them to be used in the system for accurate and efficient labeling. In particular, CARAVAN focuses on two metrics of a knowledge source: (1) accuracy, which refers to how accurate the knowledge source is after being repurposed for labeling, and (2) speed (throughput and latency), which refers to how fast a knowledge source can be used to label data. Different knowledge sources can vary dramatically in these two metrics, and as illustrated in Figure 3, there exists a trade-off between these two metrics for a given knowledge source: sources with high accuracy (e.g., domain experts and foundation models) usually operate with lower throughput due to extra time needed for in-depth analysis, while high-speed sources (e.g., heuristics, IP-based ACLs) might not be able to provide accurate labels. The aim of CARAVAN is that through online learning, an in-network ML model can be turned into a model with both *high accuracy* and *high speed*, so it would be a good fit for real-time decision making in the data plane.

Low-Accuracy, Fast Knowledge Sources. Knowledge sources with a low accuracy but high speed (e.g., heuristics, IP-based ACLs) are a good fit for labeling large volumes of incoming data [24, 51, 108]. However, the main issue is that generated labels would be extremely noisy in this case. If we use these labels to retrain the in-network ML model, the accuracy of the retrained model can be even worse than that of the existing one in the data plane. To tackle this challenge, CARAVAN adopts the following solution: Instead of letting these low-accuracy knowledge sources provide a label for every unit of data in the current window, CARAVAN's labeling agent will ask these knowledge sources to label parts of data. These generated labels that cover a part of the dataset are usually called weak-supervision labels in the machine-learning community, and can reduce the amount of noise in the labels [92]. For example, when we use an IP-based ACL as a knowledge source for labeling, a straightforward way of repurposing it into a labeler is that we would label every flow with IP not on the blacklist as a benign flow. However, we could have mislabeled a lot of malicious flows as benign in this case and manually introduced a lot of noise into the final set of labels. With CARAVAN's solution of generating weak-supervision labels, we would only generate labels for flows whose IPs are on the blacklist (as we are more confident they would be malicious). Even though we would only be able to obtain a much smaller set of labeled data for model validation and retraining, with a large volume of incoming data, we would end up with a reasonable amount of labeled data with good accuracy (§5).

> **Insight 1:** *Low-accuracy but fast knowledge sources, such as heuristics and IP-based ACLs, can provide weak-supervision labels for training high-accuracy models.*

High-Accuracy, Slow Knowledge Sources. Knowledge sources characterized by low speed but high accuracy (e.g., domain experts and foundation models) are well-suited for labeling a small to medium amount of incoming data considered important or representative of the network (e.g., sampled data with network telemetry algorithms). For example, foundation models, like NetFound [54] and ChatGPT [20], are shown to be capable of solving downstream traffic analysis tasks with high accuracy and generalizing well across diverse network environments with no extra retraining. The main issue, however, is that they might either be too slow or use too many system resources (e.g., GPU/CPU memory and API costs) and thus cannot be activated frequently (for instance, at the end of every window of sampled data).

One insight that CARAVAN takes advantage of is that these knowledge sources can usually be transformed into cheaper rulesets or heuristics that are able to offer much higher throughput due to low latency or cost-effectiveness. In the machine-learning community, this insight was originally used to interpret black-box ML models [48, 62]. In CARAVAN, to avoid the costs associated with calling expensive knowledge sources at every labeling window, we introduce a *labeling rule cache*. Each time the knowledge source is activated for labeling, it is also asked to generate an ensemble of rulesets that will be stored in the labeling rule cache for fast and cheap labeling at the end of the next few labeling windows. For example, though foundation models, like GPT-4 [87], can be repurposed as a labeling source, the fees incurred by calling the GPT-4 APIs for inference can grow prohibitively expensive if we call these APIs at the end of every labeling window—GPT-4 turbo [11] can cost as high as $144 an hour for 1000-token labeling request/response per second (on average). CARAVAN specifically asks the language model to generate rules it relies on for decision-making and stores these in the rule cache for labeling the next few windows of data. Note that, in the evaluation section (§5), though we demonstrate that using the rule cache for labeling could save a lot on cost and system resources with little performance penalty, we also find that these rules could go stale quickly (Figure 7) and, therefore, must be updated occasionally.

> **Insight 2:** *High accuracy but slow knowledge sources, such as ChatGPT [20] and NetFound [54], can transform into rulesets or heuristics to facilitate fast and resource-efficient labeling for a limited duration, before becoming stale.*

3.1.2 User Interface. CARAVAN's labeling agent exposes an interface where users can conveniently specify what knowledge sources they would like to use for the labeling agent and how the labeling sources should be defined. To support a new knowledge source, the user only needs to complete a function called `label()`, which takes a window of data samples as input and returns a set of labels on this window as output.[2]

3.2 Model Validation

The model validation stage periodically monitors and evaluates the performance of the in-network ML model. It is also responsible for triggering online training when necessary, e.g., in the case of a potential concept drift or data drift when the performance of the model degrades due to changes in the network environment or due to new incoming classes. These actions take place at the end of a labeling window, after the labeling agent has generated labels for all data (samples) in the current window.

Next, we introduce two components for model validation that the user can define to express their intent or performance goal of the chosen application. (a) Accuracy proxy (§3.2.1) allows the user to specify what *signals* they would like to capture on a temporal scale from the generated labels and the inference results (e.g., drop in overall classification accuracy, the appearance of a particular type of new class, and more). (b) Retraining trigger (§3.2.2) allows the user to specify at what *occasion* they would like to initialize online training based on the observed signals through the accuracy proxy (e.g., retrain when model performance degrade or retrain when certain types of attacks show up).

3.2.1 Accuracy Proxy. We define accuracy proxy as the inference accuracy computed with generated labels as the reference ground truth, which we describe in detail below.

Ideally, for a given sample of incoming data (e.g., a network flow or a packet), the corresponding inference result

[2] As a case study, we show how to construct a new knowledge source for intrusion detection using LLMs in §4.1.

(e.g., in the form of a class label prediction, noted as ML_{labels} below) from the in-network ML model would be compared with the ground truth label in the validation stage. Ground truth labels (noted as $GndT_{labels}$ below), also called "golden" labels, are objectively correct reference results for the given application and are usually used to compute the performance accuracy of ML models. Acquiring such labels is typically challenging in practice as it necessitates domain knowledge from human experts, requiring a costly and time-consuming labeling process [53]. Moreover, during the online stage of in-network ML, where the volume of data for validation is immense, it is infeasible to obtain the ground truth labels for all new incoming data and calculate the actual performance accuracy of the in-network ML model. In CARAVAN, we instead utilize generated labels ($GenL_{labels}$) and compute the accuracy proxy for the current window of incoming data. For instance, in the intrusion detection case, using F1 score [50] as the performance metric, the real accuracy Acc_{real} is computed as follows:

$$Acc_{real} = F_1(ML_{labels}, GndT_{labels}) \tag{1}$$

The accuracy proxy, on the other hand, is computed with generated labels as ground-truth labels:

$$Acc_{proxy} = F_1(ML_{labels}, GenL_{labels}) \tag{2}$$

The accuracy proxy does not need to be defined in terms similar to the real accuracy. The user has the flexibility to define accuracy proxy to be any function as long as its definition is consistent with the user's intent or the application's performance goal, e.g., to signal potential concept or data drifts.

Without access to real accuracy values, we are unable to know the absolute accuracy level of the in-network ML model at the end of a labeling window. However, in our design, the primary responsibility of the validation stage is monitoring: it is expected to reveal potential model performance degradation and trigger online training, instead of giving users or operators the exact accuracy numbers of the in-network ML model.

In particular, we observe that accuracy proxy, though not numerically the same as the real performance accuracy, could signal a potential change in data distribution or class distribution based on its trend on a temporal scale. In our evaluation using the intrusion-detection example (§5.2.2), we observe that the arrivals of new types of attacks (unseen by the in-network ML model before) cause a drop in the relative level of accuracy on a temporal scale, and the values from accuracy-proxy can reveal that incident (Figure 8).

> **Insight 3:** *The accuracy proxy reveals potential concept and data drifts by capturing similar patterns of relative changes in accuracy levels as observed in real accuracy.*

3.2.2 Retraining Trigger. The goal of continuous model validation is to enable updating the in-network ML model through online training as and when necessary. The model validation stage will activate online training through a user-defined retraining trigger. A retraining trigger can take one of the following three forms, as pre-specified by the user of CARAVAN:

- **Window-based:** Retrain periodically once every X labeling windows. When $X = 1$, CARAVAN will perform continuous training for every window, similar to the approach in prior works [33, 85]. For window-based triggers, the validation stage will skip accuracy proxy since the trigger does not use it.
- **Accuracy-based:** Retrain if the values of accuracy proxy satisfy a certain pattern on a temporal scale. For example, users can set certain accuracy thresholds, and the retraining trigger will initialize retraining if the values of accuracy proxy continuously stay below the threshold.
- **Event-based:** Retrain when a particular event takes place, e.g., when the labeling agent or the human operator detects a particular type of attack.

The retraining trigger should ideally be defined together with accuracy proxy by the user: While accuracy proxy is able to catch meaningful signals (e.g., F1 score drop) on a temporal scale, the retraining trigger explicitly expresses at what occasions the user would like online training to happen, which can be very different given the particular user application in consideration.

In CARAVAN, we mainly focus on accuracy-based retraining triggers, in which we use values of accuracy proxy to determine if online training should occur. There are two types of decisions that the retraining trigger will need to make: (a) If we do not retrain at the end of the last labeling window, should we retrain for this window? CARAVAN's retraining trigger will initialize retraining if it observes an abrupt drop in the value of accuracy proxy in the current labeling window compared to the last one, since that could be an explicit signal of potential concept or data drifts. (b) If we retrain at the end of the last labeling window, should we stop retraining for this window? As we demonstrate in the evaluation section (Figure 8), if we continuously retrain for several windows, the marginal inference accuracy gain would gradually decrease, assuming that there are no new drifts that show up in this period. As a result, the retraining trigger stops retraining if we have retrained for the last few windows and obtained decent inference accuracy gain.

> **Insight 4:** *The marginal inference accuracy gain of online training would quickly diminish if no new sources of drifts are present (i.e., the network is stable).*

3.3 How to Select CARAVAN's Elements?

For knowledge sources, we can choose existing systems (e.g., IDS) or construct new ones (e.g., fine-tuned foundation models). It is important to evaluate the labeling accuracy and approximate speed of a knowledge source using an offline labeled dataset before deploying it in Caravan. When selecting accuracy proxy and retraining trigger, we should consider the application's performance objectives (e.g., low false-positive rate) and identify signals or events from the system that might indicate performance degradation (e.g., increased rebuffering events in video streaming).

4 Implementation

We implement an end-to-end version of Caravan using Python. To interact with in-network ML models, Caravan stores the samples of the arriving flows in a streaming database, InfluxDB [13]. We initialize InfluxDB with a predefined schema consisting of various header/feature fields and metadata of the arriving packet (e.g., duration, data rate, and 5-tuple) as well as the inference results (prediction) from the deployed in-network ML model (for validation purposes). Upon the arrival of a labeling window's worth of samples, the labeling agent queries these data samples from InfluxDB to generate labels.

For knowledge sources (e.g., heuristics, DNN-based classifiers, and foundation models), we define how it labels data by completing its `label()` function (as described in §3.1). Heuristics come in the form of labeling functions [91] and are easily defined by the user. The DNN-based classifiers load a pre-trained DNN classifier, and run batched inference upon calls of `label()` for labeling. For foundation models, we use GPT-4 API [87] for sending labeling requests in the form of prompts. In this setup, labeling is modeled as a text completion task, and we explicitly prompt the language model to produce a label for each input data sample. We present a case study of implementing a foundation model, LLM-based knowledge source in §4.1. With individual knowledge sources defined, we build a labeling agent by specifying what knowledge sources it will be using. The labeling agent calls each knowledge source's `label()` function to obtain all labels and selects the best ones (with the most occurrences) as final labels.

For model validation and retraining, we define a model validator that runs `compute_accuracy_proxy()` to compute the accuracy proxy (in §3.2.1) with generated labels and inference results (from InfluxDB) as input arguments. The retraining trigger is defined as a function that checks if we have retrained for the last window. If not, we check if there is a significant drop in accuracy proxy value to initialize retraining; if yes, we then check if the increase in accuracy proxy value is small enough to stop retraining. If retraining is necessary, we go on to form a class-balanced dataset based on iCaRL [94], keeping the same number of data samples from each class and maintaining a fixed upper bound for the size of the dataset (which can be specified by the user). For training ML models, we use PyTorch [88] and one Nvidia V100 Tensor Core GPU from AWS.

Caravan maintains a busy-waiting process for data labeling, model validation, and online learning. This process will periodically read data from InfluxDB and initialize data labeling as well as model validation at the end of a labeling window (determined by time or number of data samples). If retraining is necessary, it will conduct retraining and send out the weights to the in-network ML model as gRPCs [12] or PCIe writes.

4.1 `label()` with Foundation Model (LLM)

We now present a case study of developing a new knowledge source using large language models (LLMs). We use commercial off-the-shelf LLM, more specifically ChatGPT [20]. ChatGPT is not explicitly fine-tuned on network traffic data; but, as a foundation model, may have been trained on openly available data from the Internet. Please refer to §A.1 for details on the specific model (and snapshot) we use for labeling.

• **Instruction Following.** To ensure the LLM understands the structure of input data and properly follows the subsequent instructions, we compose system prompts §A.2 that are shared by all incoming inference requests (including labeling and rule extraction). The system prompts precisely state the objective of the application (e.g., flag malicious traffic for intrusion detection) and enumerate the names and meanings of each feature in the network dataset.

> **In-context Learning Prompt (P1):** To begin with, here are some labeled flows for your reference later. The last field is the binary label (0 for benign and 1 for malicious): `[Flows, their features and labels go here]`. Next, I will give you some unlabeled flows for labeling. Please let me know if you understand the requirement by answering yes or no.

• **In-context Learning.** We take advantage of in-context learning [36, 90] to improve LLM's ability to label network data (or packets) with higher accuracy without (re)training or fine-tuning the original model. We provide a few labeled examples from the CIC-IDS2018 dataset [98]. The network traffic in these examples contains similar attack types (such as brute force attacks and DDoS attacks) to those present in the evaluation dataset (CIC-IDS2017). However, it is collected from a different network and at a different time. Using these labeled examples, we construct an in-context learning prompt (P1) shared by all subsequent inference requests.

> **Data Labeling Prompt (P2):** Please give me a label for each of these unlabeled flows. No explanation or analysis needed, label only; one flow on each line. Format for each line: (flow number) label. `[Flows and their features go here]`.

• **Data Labeling.** Whenever we invoke the `label()` function, we first compose a labeling prompt (P2). This prompt specifies the expected response format, facilitating easy parsing of responses for per-packet labels. Additionally, it includes all the data to be labeled, and structured in accordance with the system prompt. We concatenate the system prompt, the in-context learning prompt, and the labeling prompt, and submit an API request to the LLM.

> **Rule Extraction Prompt (P3):** To begin with, here are some example input flows for your reference later. `[Flows and their features go here]`. Based on these example input flows, can you do some analysis and help me come up with some rules or heuristics (in the form of a Python function) to determine if an unlabeled flow is benign or not? Make sure that in the Python function, you label a flow as malicious only when you are very confident. Name the function `label_flow_with_rule_cache()`, and pass it in a format that can be executed by `exec()`. The input of the function should be the 16 features in the system prompt (in order), and the output should be 0 (benign) or 1 (malicious).

• **Rule Extraction.** To extract rules to store in the labeling rule cache for fast and resource-efficient labeling, we construct a rule-extraction prompt (P3). This prompt explicitly requests the LLM to generate rules and heuristics for data labeling as a Python function, specifying the expected input/output formats to simplify the parsing of the generated responses. §A.3 shows an example function generated by the LLM for fast labeling.

5 Evaluation

In our evaluation, we show: (a) using three different choices of knowledge sources, CARAVAN is able to efficiently label new incoming network traffic for the purpose of model validation and retraining, and can achieve almost the same level of inference accuracy gains compared to using ground-truth labels (§5.2.1). (b) CARAVAN's accuracy proxy and retraining trigger allow us to efficiently determine when to initialize or stop retraining. As compared to continuous retraining, the use of accuracy proxy and retraining trigger has the potential to reduce GPU compute cost by an average of 74.55% without significantly hurting inference accuracy gain (§5.2.2). (c) In software simulation (§5.3.1), CARAVAN is able to achieve a 30.3% improvement in F1 score (on average) compared to static offline models across three chosen applications. CARAVAN's accuracy proxy and retraining trigger enable 61.3% saving in GPU compute time (on average) for retraining without significantly compromising inference accuracy gains. (d) In the end-to-end Taurus FPGA testbed (§5.3.2), CARAVAN continuously keeps in-network ML models up-to-date with changing traffic dynamics and maintains high inference accuracy at network line-rate. With accuracy proxy and retraining trigger, CARAVAN improves over static models in terms of F1 score by an average of 30% with 56.23% less CPU usage and similar memory footprint as continuous retraining baselines.

5.1 Experiment Setup

Use Cases. To evaluate CARAVAN, we select two network traffic analysis applications widely used and evaluated by prior work in the domain of in-network ML (Table 1). (a) *Network Intrusion Detection*: The goal is to flag network flows or network packets regarding whether they involve malicious activities. We expect the in-network ML model to offer a preliminary analysis of the network flows through binary classification before running more expensive downstream security analysis instead of providing complete end-to-end protection of a networked system. This application is an example of how in-network ML could improve the *security* of networked systems. (b) *IoT Traffic Classification*: The goal is to assign an IoT device type to a network flow or packet. Classification results from the in-network ML model enable operators to know what different flows might entail (e.g., application or data type) early in the network, and to act correspondingly based on different devices, applications, or data types to optimize for the quality of service (QoS) or user quality of experience (QoE). For example, network flows from video cameras might require allocation to a less congested network path, since the user will likely be in a live video conference. In this case, fewer packet retransmissions and lower latency are critical to good user perception of video and service quality. This application is an example of how in-network ML could improve the *performance* of networked systems.

Datasets and In-network ML Models. We closely follow prior work in the domain of in-network ML when choosing datasets and in-network ML models. A summary of these datasets and related statistics is available in Table 1.

For network intrusion detection, we follow prior work [101, 125] to use CIC-IDS2017 [98] and UNSW-NB15 [84]. With CIC-IDS2017, we use the same features from pForest [37] and a deep neural network with similar architecture to the one from Taurus [104]. With UNSW-NB15, we use the same features and one of the deep neural networks from the intrusion detection example of N3IC [101]. For IoT traffic classification, we follow prior work [101,125] to use UNSW-IoT [102]. For the in-network ML model, we follow the IoT traffic classification example of N3IC [101] in terms of feature selection and model architecture. For multi-class classification, we use one of N3IC's four-layer deep neural networks, which have 16, 64, 32, and 10 neurons on each layer, respectively; we replace the binary weights with 32-bit weights.

Choices and Configuration of Knowledge Sources. We choose three knowledge sources for the labeling agent to use. (a) A *DNN-based classifier* for intrusion detection on CIC-IDS2017: The DNN-based classifier has 8 layers and 13,222 parameters in total. The architecture is similar to a stacked autoencoder in DeepPacket [78]. It is trained on a small part of CIC-IDS2017 (a subset that is not used during testing) and a small part of CIC-IDS2018 [98] (a different intrusion detection dataset from the same publisher). (b) A *large language model* for intrusion detection on UNSW-NB15: The large language model is based on GPT-4 [87] text completion APIs. We program the user prompts properly so the language

Application	Dataset	# Samples	# Features	# Classes	# Drifts
Network Intrusion Detection	CIC-IDS2017 [98]	7,000	16	2	7
	UNSW-NB15 [84]	5,000	20	2	5
IoT Traffic Classification	UNSW-IoT [102]	108,000	16	10	9

Table 1: Network applications and datasets used in our evaluation with input features listed in §A.2 and [101]. A drift occurs in intrusion detection with the arrival of new attack traffic, and in IoT classification with unseen IoT device traffic.

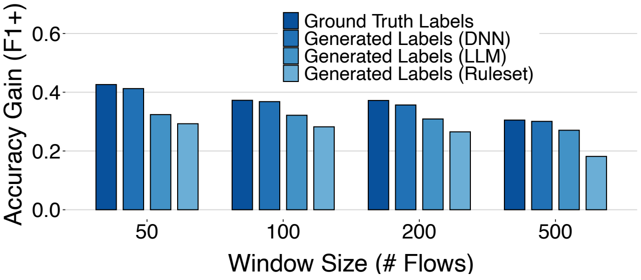


Figure 4: CARAVAN's labeling agent generates labels for online training, bringing comparable levels of accuracy gain as ground-truth labels across three different knowledge sources.

model can understand the particular format of our input network flows and generate labels in a format easily parsed by the labeling agent. To improve labeling accuracy, we take advantage of *in-context learning* and give the language model 10–20 labeled flows (not used during testing) for reference. (c) An *IoT device list* for IoT traffic classification on UNSW-IoT: We use the device list provided by the original dataset publishers. To ensure that the device list will generate strictly worse labels than the ground-truth labels, we modify the MAC address of some network flows so that the device list is unable label them. Overall, the device list can identify and label 10% of all the network flows in the dataset.

Quality and Usage Metrics. For accuracy, we use the F1 score [50] as the performance metrics for evaluating the quality of an in-network ML model. In machine learning, the F1 score is often preferred over basic metrics like classification accuracy. It provides a more nuanced measure of a model's performance, especially when class distributions are imbalanced or when the costs of false positives and false negatives differ. This preference for accuracy metrics aligns with previous research in the field [37, 101, 104, 122]. To better model the performance gain of the validation and online learning processes, we use the metrics of *accuracy gain*, defined as the increase in the F1 score of the retrained in-network ML model compared to that of the offline one. To determine the accuracy of a specific experiment, we first calculate an F1 score based on the model predictions and ground-truth labels at the end of each labeling window using the data from that window. Ultimately, we report the average F1 score, or the average increase in F1 score (i.e., F1+) compared to the offline model, as the final accuracy metric or accuracy gain.

To quantify the system resource usage for online training, we use the metrics of *GPU compute time*, defined as the time spent on the GPU during online training. When using large language models as a knowledge source, we also use *tokens used for labeling* to demonstrate the cost of using an expensive knowledge source for labeling, defined to be the aggregate number of tokens (an addition of prompt tokens by the user and completion tokens by the language model) used for the labeling task.

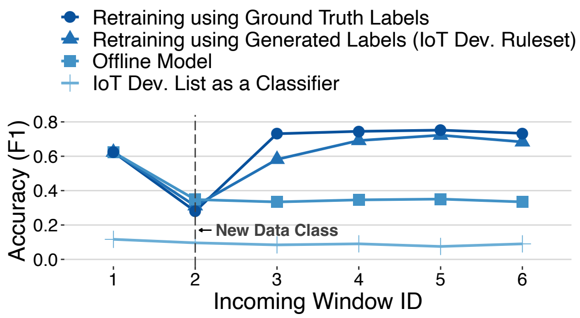


Figure 5: A comparison of generated labels using CARAVAN's labeling agent versus ground-truth labels for a low-accuracy and fast knowledge source (IoT device list) during data drift (i.e., encountering new data classes).

End-to-End Testbed. We use the Taurus FPGA-based testbed [104] for end-to-end evaluation. A 32-port programmable Tofino Wedge100BF-32x switch [21] is used to sample packets for the control plane and manage the Taurus ML core, which is emulated as a bump-in-the-wire FPGA. The switch bypasses its internal traffic through the Xilinx Alveo U250 FPGA [3], which is used to emulate the in-network ML model. The control plane runs a process to perform model validation and retraining on the sampled packets and update the model weights in the FPGA via PCIe. It also runs the ONOS controller [18] and a Python REST API to install forwarding rules on the switch. Two 80-core Intel Xeon servers generate and receive traffic via ScaPy [19] or MoonGen [44]. The in-network ML model has been compiled to Verilog using the Spatial [70] compiler and installed on the FPGA for evaluations.

5.2 Microbenchmarks

5.2.1 Effectiveness of the Labeling Agent.

We find that noisy labels and partial-coverage labels generated by imperfect knowledge sources can still lead to decent inference accuracy gains after online training.

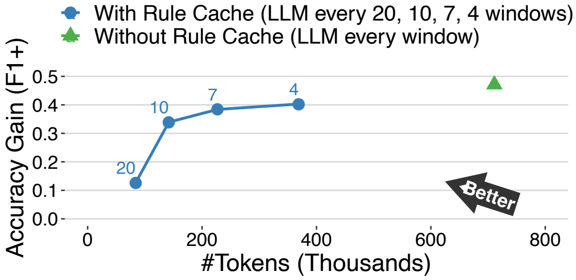


Figure 6: With a validation rule cache, CARAVAN conserves language model request tokens used for labeling, without significantly compromising the accuracy gains from retraining.

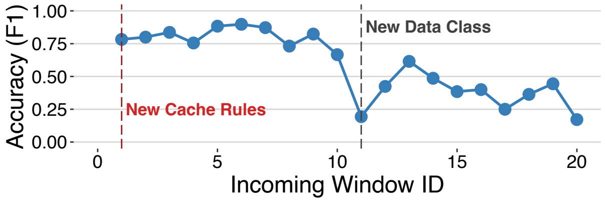


Figure 7: Though labeling rule cache generated by LLMs are subject to data drift, they generate accurate labels in a short local period of time.

Effectiveness of Noisy Labels. Noisy labels are defined to be labels that might be incorrect, and may be generated by knowledge sources like DNN-based classifiers or large language models in our case. Though these two knowledge sources (DNN-based classifier and language model) can generate a label for every sample of new incoming window when requested, we find that the overall quality of generated labels is around 0.7 to 0.8 in terms of F1 score on a small development set, indicating that there is a non-trivial level of noise in generated labels. We use these generated labels for a simple experiment of continuous online training, in which we skip validation and retrain at the end of every labeling window. We find that even with noisy labels, we are able to obtain a level of inference accuracy gain that is similar to the gain if we retrain with ground-truth labels under different labeling window sizes (Figure 4). The reason accuracy gain tends to decrease as window size increases is that we use a fixed number of 30 epochs for training; with larger training data sizes, it generally takes longer for the model to converge.

Effectiveness of Weak Supervision Labels. In the case of CARAVAN, weak supervision labels are defined to be labels that only cover a subset of all the samples in a labeling window and can be generated by low-accuracy but fast knowledge sources (e.g., an IoT device list) as discussed in §3.1.1. In our setup, the IoT device list can only label around 10% of all network flows in the dataset. To verify whether such a knowledge source can effectively mitigate model drift, we continuously retrain an ML model when new types of devices are present in the incoming data. We find that even with weak

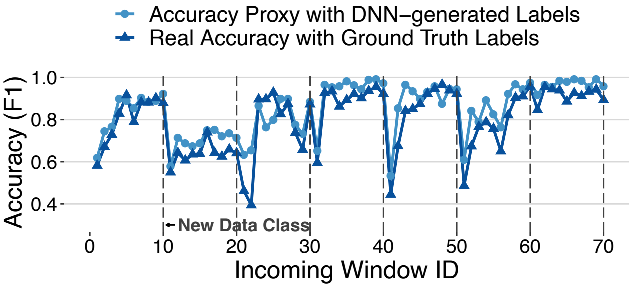


Figure 8: CARAVAN's accuracy proxy F1 scores align with the real F1 scores in terms of relative changes in accuracy on a temporal scale, particularly in instances of data drift.

supervision labels that have partial coverage, we can achieve a comparable level of inference accuracy gain when data drift occurs (after the arrival of a new class) to that of retraining with ground-truth labels (Figure 5). At the same time, we find that the device list cannot be used independently to classify incoming data with high accuracy due to partial coverage, as depicted in Figure 5.

5.2.2 Effectiveness of Labeling Rule Cache, Accuracy Proxy, and Retraining Trigger.

Labeling Rule Cache. As discussed in §3.1.1, when using expensive knowledge sources like large language models, we can request the knowledge source to generate temporary rules or heuristics in a rule cache that can be used for fast and cost-effective labeling for the following few labeling windows. In our experiment, we call language models for labeling and rule generation (in the form of a simple executable function) every 4, 7, 10, and 20 labeling windows. We use the generated function as the rule cache for labeling at the end of all other windows. By invoking the language model every 4, 7, or 10 windows, we achieve nearly the same level of inference accuracy gain after online training compared to employing language models for labeling at every window, while utilizing 65.4% fewer tokens on average (Figure 6). Note that the rules or heuristics in the rule cache can quickly go stale, especially in the case of a concept or data drift (Figure 7), so the rule cache should be updated frequently to avoid the generation of highly noisy labels.

Accuracy Proxy. We set up accuracy proxy in the same way as defined in §3.2.1, and verify if it is consistent with our insight that it can be used to reveal potential concept or data drifts even though it is not numerically equivalent to the real accuracy. In an incremental-class learning setup, where a new data class shows up in the incoming data every 10 labeling windows, we find that accuracy proxy is consistent with the real accuracy in terms of overall trend and relative changes in accuracy level on a temporal scale (Figure 8).

Retraining Trigger. To demonstrate the potential of using retraining triggers to avoid excessive retraining and save GPU compute time, we set up a window-based retraining trigger

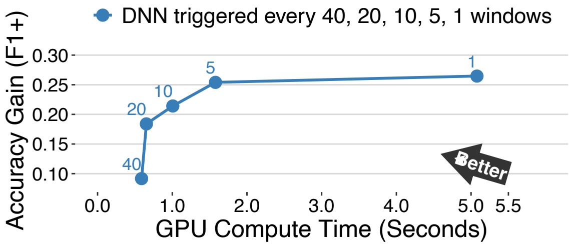


Figure 9: With a window-based retraining trigger, CARAVAN saves GPU compute time without significantly compromising retraining accuracy gain.

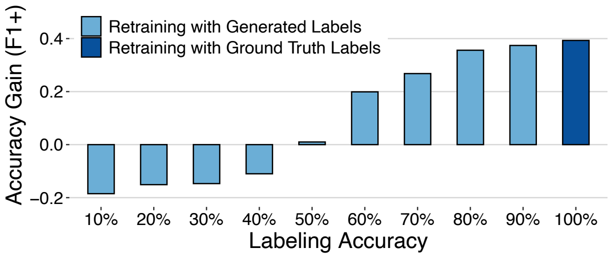


Figure 10: The relationship between CARAVAN's retraining accuracy gain and the labeling accuracy of the knowledge source. (Labeling accuracy is the percentage of data that can be correctly labeled by the knowledge source, compared to ground truth labels.)

that reduces the frequency of retraining from once every labeling window to once every 5, 10, 20, and 40 labeling windows. We observe that even with this straightforward retraining trigger, we manage to save an average of 74.55% GPU compute time, with at most a 0.05 reduction in inference accuracy gain in terms of F1 score when retraining occurs every 5 or 10 windows (Figure 9).

5.2.3 Sensitivity to External Knowledge Sources. CARAVAN assumes that users will be able to provide reliable knowledge sources that be adapted for data labeling. When inaccurate knowledge sources are used, the accuracy gain from CARAVAN's retraining may decrease and sometimes even drop below zero, as illustrated in Figure 10. We discuss potential solutions to this issue in §6.

5.3 End-to-End Improvement

We evaluate the end-to-end improvements of CARAVAN in software simulation and on the Taurus FPGA testbed [104].

5.3.1 Software Simulation. In software simulation, we find that CARAVAN is able to achieve a 30.3% improvement in F1 score (on average) as compared to static offline models across three chosen applications (Figure 12). We also find that the gap between inference accuracy gain of continuous online learning with ground-truth labels and with labeling-agent generated labels stays as little as 0.4–2.1% for intrusion detection with DNNs as knowledge source, and 0.5–1.8% for IoT traffic classification with device lists as knowledge source. Though that gap can be as large as 11% for intrusion detection with a large language model as a knowledge source, we believe that performance can be further improved when specialized network foundation models are used as knowledge sources in the future. Moreover, CARAVAN's accuracy proxy and retraining trigger enable 61.3% savings in GPU compute time (on average) for retraining without significantly compromising inference accuracy gain.

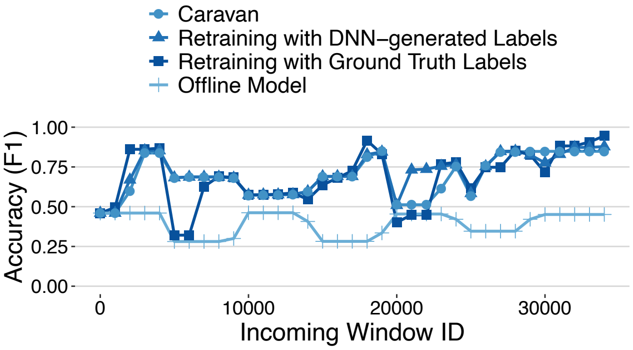


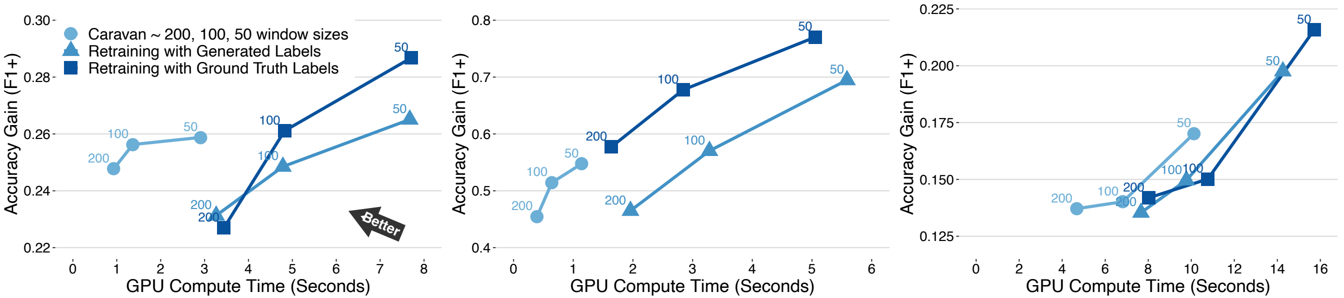
Figure 11: End-to-end results on the Taurus FPGA testbed. CARAVAN keeps in-network ML models up-to-date against changing traffic when operating at line rate.

System	LUT%	FFs%	BRAM%	Power (W)
Taurus: Offline	6.49	4.35	4.15	16.86
CARAVAN	6.81	4.71	4.15	17.16

Table 2: Resource usage of CARAVAN's in-network model for intrusion detection on the Taurus FPGA testbed.

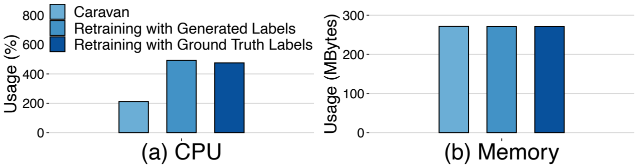
5.3.2 FPGA-based Experiments. In the Taurus FPGA testbed [104], we run an intrusion detection application with the same in-network model as in software simulation, programmed with Spatial [70]. We generate traffic by sampling 35 M packets from the CIC-IDS2017 dataset, while ensuring a uniform distribution of the seven attacks present in the dataset (i.e., 5 M packets for each attack). We preserve the order of the attacks as in the original dataset. Using Moongen [44], we send packets at 0.5 Million packets per second, and set the sampling rate to about 0.1%. Each labeling window receives about 500 packets. We find that CARAVAN can continuously keep in-network ML models up-to-date with changing traffic dynamics and maintain high inference accuracy under network line rate on a temporal scale (Figure 11). With accuracy proxy and retraining trigger, CARAVAN further improves upon static models in terms of F1 score by an average of 30%. It is worth noting that at times, the accuracy of CARAVAN can surpass that of the continuous retraining baselines. This is because continuous retraining for small in-network models may lead to overfitting, whereas CARAVAN's retraining trigger helps mitigate this issue.

5.3.3 Resource Usage & Latency Breakdown. Table 2 shows the FPGA's percentage resource count in terms of lookup tables (LUTs), flip flops (FFs), on-chip memory



(a) IDS with a DNN labeler on CIC–IDS2017 (b) IDS with an LLM labeler on UNSW–NB15 (c) IoT Classif. with Ruleset on UNSW–IoT

Figure 12: Tradeoff between inference accuracy gain and GPU compute time for CARAVAN and continuous retraining baselines across two applications (intrusion detection and IoT classification), three datasets, and three knowledge sources.



(a) CPU (b) Memory

Figure 13: CPU and memory usage of CARAVAN's busy-waiting process for labeling data, retraining model, and updating model weights.

Retraining Step	Latency (ms)
- Retrieving data from InfluxDB	6.041 ± 1.114
- Labeling data with DNN-based IDS	1.015 ± 1.238
- Computing accuracy proxy	1.732 ± 0.073
- Retraining in-network ML model	14.775 ± 0.982
- Installing new model weights	46.145 ± 0.507

Table 3: Latency breakdown of CARAVAN's retraining steps on a window of 100 packets for the network intrusion detection application.

(BRAM), and power (W). In contrast to the vanilla Taurus implementation (i.e., Offline ML) [104], which lacks support for online weight updates, CARAVAN introduces minimal additional overhead in FPGA resource usage while supporting live weight updates.

We also measure the CPU and memory usage of the CARAVAN's busy-waiting process that retrieves incoming data from a streaming database (i.e., InfluxDB), labels it, computes accuracy proxy, retrains models, and issues weight updates (§4). We see that CARAVAN reduces CPU usage by an average of 56.23% compared to continuous retraining baselines, without incurring any additional memory overhead (Figure 13). Table 3 further shows a breakdown of each these retraining steps in CARAVAN.

6 Limitations & Future Work

Optimizing Sample Selection for Online Learning. CARAVAN employs random sampling to reduce the volume of input data sent to the labeling agent. Existing research in online learning systems shows that network traffic is heavy-tailed and empirically variable [115], which could undermine the effectiveness of online learning in real-world deployments if training samples are not carefully selected [43]. While it is straightforward to modify the input data sampling and retraining data formation logic in CARAVAN, developing efficient and effective algorithms for online sample selection remains a future research direction that requires further understanding of both machine learning techniques and the characteristics of network traffic.

Reverting In-network ML Models. CARAVAN focuses on updating and improving models using continuous sampling of and selective retraining on the network's data, which lets them adapt to new events. However, in scenarios where data is compromised, it would be necessary to revert or reset these models to a previous good state. If online data (such as network traffic) is being used to retrain and update models, bad actors can poison training data by intentionally feeding bad traffic in the network. Future research may detect and protect against these attacks and restore models to a clean state.

Network Telemetry Data for ML. CARAVAN focuses on retraining models with sampled data but does not dictate how the collection of such data is performed. However, extensive research is needed on how to collect and sample data for the express purpose of retraining ML models. For instance, some data may not contribute to an increase in the fidelity of the model, even with further training iterations. In these cases, the data may simply be orthogonal to the task the ML model is built for. On the other hand, the system may need to sample more frequently in cases where notable network events are detected. For example, a server running out of resources may indicate a network attack that breaches security. Packets must be collected so as to classify and inoculate future ML models to these attacks. In short, collection systems for online training need to leverage dynamic sampling rates at various points throughout the network in order to ascertain when and where to get the best training data.

Creating Domain-Specific Knowledge Sources. In this paper, we repurpose GPT-4 as a knowledge source for network intrusion detection. We recognize that GPT-4 was not originally designed or trained for cybersecurity applications; instead, it is used primarily as a proof-of-concept foundation model for data labeling. An emerging research direction involves pre-training or fine-tuning domain-specific foundation models for networking or security on larger traffic traces (e.g., NetLLM [112], NetFound [54], Lens [110]). Another direction from the machine-learning community aims to enhance foundation models to better follow human intents and self-improve through feedback, whether human-generated or model-generated (e.g., constitutional AI [29] and self-improving LLMs [56, 61, 120]). These efforts could lead to developing knowledge sources that can generate accurate labels and better align with human expertise and intentions.

Evaluating and Validating Knowledge Sources. CARAVAN assumes that the provided labeling sources are sufficient to cover the space of input data for a given networking use case. As a next step, these labeling sources must be vetted further to ensure high-quality label generation. Common accuracy metrics such as F1, precision, or recall are all valid for assessing how well these labeling sources are performing (on a given dataset), but additional metrics are required to assess the full coverage of application space. For example, in a security context, how many of the commonly seen network attacks can the labeling source cover? Furthermore, the network community should start making its labeling sources public to allow retraining systems more effectively—similar to how various ML communities have put forth public collections of data and benchmarks. For instance, in the case of foundation models, public benchmarks feature open and comprehensive evaluations of models on specific applications, such as chat [124], code generation [74], and question answering [41]; these benchmarks help users select the best model for their particular use case. Finally, as suggested in Snorkel [92], multiple labeling sources can be aggregated for greater coverage and fidelity. In this way, aggregate labeling sources can generate more accurate labels than individual sources, effectively allowing a given source to cover the blind spots of another source.

Generalizing to Larger Control-Plane ML Models. Although CARAVAN is designed for online learning of in-network ML models, we believe that its core insights and techniques—such as using weak supervision for labeling data in an online setup, employing accuracy proxies, and utilizing retraining triggers to detect and mitigate model quality degradation—can be generalized to larger ML models deployed in the control plane. These control-plane ML models also face similar challenges like data or concept drifts [75] and a lack of labels for model monitoring and retraining in an online setup [53].

7 Related Work

Systems for Online Learning. Ekya [33] and RECL [67] discuss how online learning can be done for computer vision models on an edge server jointly with inference, while CARAVAN studies the case of in-network ML models in which data-plane inference does not interfere with control-plane online learning. Nazar [58] features how to mitigate data drift for ML models on mobile devices, and differs from CARAVAN as it does not address essential components of online learning (e.g., data labeling).

Data Collection and Generation for Networking. The emerging need to train ML models for networking tasks and design new network telemetry algorithms sparks extensive research in designing better tools for network data collection and network data generation. NetUnicorn [31] is a platform for collecting and actively labeling network data for developing offline generalizable ML models. It features a human-in-the-loop approach where users can select what data to collect and label, and it is different from our focus since CARAVAN features automatic online data labeling after ML models have been deployed. NetShare [118] enables synthetic IP-header generation for network flows but has a different focus from CARAVAN and does not study data labeling for downstream traffic analysis tasks.

Interpretability of ML Models. With the growth of ML models in networking, many existing efforts focus on the interpretation of these black-box models to make their decision-making logic transparent to network operators. For example, Trustee [62] proposes a framework that determines whether or not a given ML model suffers inductive biases by extracting a high-fidelity decision tree from the model being analyzed. However, such diagnosis of the ML models is not yet automatized and needs a human-in-the-loop. Indeed, CARAVAN can use Trustee as an orthogonal system component for diagnosing the behavior of the online learning model.

Programmatic Data Labeling. CARAVAN complements and augments (rather than competes with) existing data programming systems, such as Snorkel [92]. Snorkel uses generative models to estimate the accuracies of different knowledge sources, and can potentially be used for conflict resolution in CARAVAN's labeling agent. CARAVAN is similar to Snorkel in the aspect that both point out that weak knowledge sources can be used for labeling data and training ML models instead of using them for independent decision-making. However, CARAVAN focuses on how automatic data labeling helps online learning of ML models and mitigates drifts (by incorporating knowledge sources, accuracy proxy, and retraining trigger), while Snorkel focuses on enabling users to label datasets with multiple knowledge sources for training better ML models offline.

Weak Supervision in Networking. The concept of weak supervision has been extensively applied in networking, particularly in cybersecurity and internet measurement applications [47, 69, 86]. CARAVAN differs from these works by focusing on enabling weak supervision in an online setup to detect model quality degradation and retrain outdated models.

Label-free Data Drift Mitigation. Recent efforts in networking and security domains feature data drift mitigation with no need for labels. For example, CADE [117] proposes to train a neural network that can help determine if new incoming data has drifted away from training data. However, Caravan focuses on the continuous adaptation of an online model, where the training data are constantly evolving. Moreover, CADE uses root cause analysis to fix drifted models offline when there is no explicit requirement on how fast model update needs to happen, which is in contrast to Caravan's focus on the online setting when model updates must be done fast and automatically to keep up with the high inference rate. In summary, Caravan aims to be a more generalized framework designed for various in-network ML applications.

8 Conclusion

Once deployed online, in-network machine learning (ML) models can experience accuracy degradation owing to fluctuations in traffic patterns and changes in online data distribution. While online learning is a promising solution, it is challenging in practice due to the need for automatic labeling of evolving network traffic and the efficient monitoring of model performance degradation. To overcome these challenges, we present Caravan, the pioneering system for practical online learning of in-network ML models. Caravan addresses the issue of labeling new incoming traffic data for retraining by leveraging diverse knowledge sources that, otherwise, are unsuitable for real-time decision-making. Moreover, Caravan introduces the accuracy proxy metric to monitor model degradation and potential data drifts, providing an effective signal to trigger model retraining. Our evaluation shows that Caravan can keep in-network ML models up-to-date, achieving a 30.3% improvement in F1 score (on average) and reducing GPU compute time for training by 61.3% (on average), while achieving similar accuracy gains as continuous retraining. We hope the development of such a system will not only contribute to the domain of ML for networking and traffic analysis applications but also influence the design of practical and efficient machine-learning systems in general.

Acknowledgements

We thank our anonymous shepherd and reviewers for their invaluable feedback, which significantly enhanced the quality of this paper. We also thank Gerry Wan, Gautam Akiwate, Shinan Liu, Shiv Sundram, Tian Zhao, Alex Ratner, James Hong, Azalia Mirhoseini, Haijie Wu, Junchen Jiang, and Vyas Sekar for their insightful discussions. This research was supported by ACE, one of the seven centers in JUMP 2.0, an SRC program sponsored by DARPA; NSF awards CAREER-2338034, CNS-2211381, and CNS-2211384; and "SERICS" (PE00000014) under the NRRP MUR program funded by the EU-NGEU. Support also came in part from affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Intel, Microsoft, NEC, SAP, Teradata, and VMware.

References

[1] AI and ML: The New Frontier for Data Center Innovation and Optimization. https://www.techradar.com/news/data-centres-in-an-ai-and-ml-driven-future.

[2] Alveo SN1000 SmartNIC Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/sn1000.html.

[3] Alveo U250 Data Center Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/u250.html.

[4] AMD Pensando. https://www.amd.com/en/accelerators/pensando.

[5] Apache Kafka. https://kafka.apache.org/.

[6] Bluefield Data Processing Units (DPUs). https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[7] CISCO NetFlow. https://www.cisco.com/c/en/us/tech/quality-of-service-qos/netflow/index.html.

[8] Creating a Predictive Network for the Human Mind. https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2022/m05/creating-a-predictive-network-for-the-human-mind.html.

[9] Data Centers in an AI and ML Driven Future. https://www.techradar.com/news/data-centres-in-an-ai-and-ml-driven-future.

[10] Every Request, Every Microsecond: Scalable Machine Learning at Cloudflare. https://blog.cloudflare.com/scalable-machine-learning-at-cloudflare/.

[11] GPT-4 Turbo in the OpenAI API. https://help.openai.com/en/articles/8555510-gpt-4-turbo-in-the-openai-api.

[12] gRPC. https://grpc.io/.

[13] InfluxDB. https://www.influxdata.com/.

[14] Infrastructure Processing Unit (Intel IPU) and SmartNICs. https://www.intel.com/content/www/us/en/products/network-io/smartnic.html.

[15] Intel Tofino 2. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino-2.html.

[16] Introducing Meta Llama 3: The Most Capable Openly Available LLM to Date. https://ai.meta.com/blog/meta-llama-3/.

[17] Marvell OCTEON 10 DPU Platform. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-10-dpu-platform-product-brief.pdfe/.

[18] ONOS: Open Network Operating System. https://opennetworking.org/onos/.

[19] Scapy. https://scapy.net/.

[20] Three LLMs Walk into a Network Operations Center.... https://www.bigpanda.io/blog/three-large-language-models-walk-into-a-network-operations-center/.

[21] WEDGE 100BF-32X: 100GBE Data Center Switch. https://www.edge-core.com/cloud-data-center-100g/.

[22] What Is LLM Temperature? https://www.iguazio.com/glossary/llm-temperature/.

[23] Richelle Adams. Active Queue Management: A Survey. *IEEE Communications Surveys & Tutorials*, 15(3):1425–1476, 2012.

[24] Kazeem B Adedeji, Adnan M Abu-Mahfouz, and Anish M Kurien. DDoS Attack and Detection Methods in Internet-Enabled Networks: Concept, Research Perspectives, and Challenges. *Journal of Sensor and Actuator Networks*, 12(4):51, 2023.

[25] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A Survey of Network Anomaly Detection Techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[26] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In *ACM SIGCOMM*, 2014.

[27] Nahla Ben Amor, Salem Benferhat, and Zied Elouedi. Naive Bayes vs Decision Trees in Intrusion Detection Systems. In *ACM Symposium on Applied Computing*, 2004.

[28] Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, and Christopher Re. Ask Me Anything: A Simple Strategy for Prompting Language Models. In *ICML*, 2022.

[29] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: Harmlessness from AI Feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[30] Diogo Barradas, Nuno Santos, Luıs Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. FlowLens: Enabling Efficient Flow Classification for ML-Based Network Security Applications. In *NDSS*, 2021.

[31] Roman Beltiukov, Wenbo Guo, Arpit Gupta, and Walter Willinger. In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems. In *ACM CCS*, 2023.

[32] Theophilus Benson, Aditya Akella, and David A Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *ACM IMC*, 2010.

[33] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *USENIX NSDI*, pages 119–135, 2022.

[34] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys & Tutorials*, 16(1):303–336, 2013.

[35] Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities. *Journal of Internet Services and Applications*, 2018.

[36] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

[37] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pForest: In-Network Inference with Random Forests. *arXiv preprint arXiv:1909.05680*, 2019.

[38] Brian Caswell, Jay Beale, and Andrew Baker. *Snort Intrusion Detection and Prevention Toolkit*. Syngress, 2007.

[39] Lingjiao Chen, Matei Zaharia, and James Zou. How Is ChatGPT's Behavior Changing over Time? *arXiv preprint arXiv:2307.09009*, 2023.

[40] David D Clark, Craig Partridge, J Christopher Ramming, and John T Wroclawski. A Knowledge Plane for the Internet. In *ACM SIGCOMM*, pages 3–10. ACM, 2003.

[41] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think You Have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[42] L. Dhanabal and S.P. Shantharajah. A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(6):446–452, 2015.

[43] Alexander Dietmüller, Romain Jacob, and Laurent Vanbever. On Sample Selection for Continual Learning: a Video Streaming Case Study. *arXiv preprint arXiv:2405.10290*, 2024.

[44] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *ACM IMC*, 2015.

[45] Alice Este, Francesco Gringoli, and Luca Salgarelli. Support Vector Machines for TCP Traffic Classification. *Computer Networks*, 2009.

[46] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic. In *ACM IMC*, 2020.

[47] Vojtech Franc, Michal Sofka, and Karel Bartos. Learning Detector of Malicious Network Traffic from Weak Labels. In *ECML PKDD*, pages 85–99. Springer, 2015.

[48] Nicholas Frosst and Geoffrey Hinton. Distilling a Neural Network into a Soft Decision Tree. *arXiv preprint arXiv:1711.09784*, 2017.

[49] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks. In *USENIX NSDI*, 2019.

[50] Cyril Goutte and Eric Gaussier. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[51] Vic Grout, John McGinn, and John Davies. Real-Time Optimisation of Access Control Lists for Efficient Internet Packet Filtering. *Journal of Heuristics*, 13:435–454, 2007.

[52] Andreas Grünbacher. POSIX Access Control Lists on Linux. In *USENIX ATC*, 2003.

[53] Jorge Luis Guerra, Carlos Catania, and Eduardo Veas. Datasets Are Not Enough: Challenges in Labeling Network Traffic. *Computers & Security*, 120:102810, 2022.

[54] Satyandra Guthula, Navya Battula, Roman Beltiukov, Wenbo Guo, and Arpit Gupta. netFound: Foundation Model for Network Security. *arXiv preprint arXiv:2310.17025*, 2023.

[55] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 2008.

[56] Patrick Haluptzok, Matthew Bowers, and Adam Tauman Kalai. Language Models Can Teach Themselves to Program Better. *arXiv preprint arXiv:2207.14502*, 2023.

[57] James Halvorsen, Clemente Izurieta, Haipeng Cai, and Assefaw H Gebremedhin. Applying Generative Machine Learning to Intrusion Detection: A Systematic Mapping Study and Review. *ACM Computing Surveys*, 2024.

[58] Wei Hao, Zixi Wang, Lauren Hong, Lingxiao Li, Nader Karayanni, Chengzhi Mao, Junfeng Yang, and Asaf Cidon. Monitoring and Adapting ML Models on Mobile Devices. *arXiv preprint arXiv:2305.07772*, 2023.

[59] Bruno Miranda Henrique, Vinicius Amorim Sobreiro, and Herbert Kimura. Literature Review: Machine Learning Techniques Applied to Financial Market Prediction. *Expert Systems with Applications*, 124:226–251, 2019.

[60] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New Directions in Automated Traffic Analysis. In *ACM CCS*, 2021.

[61] Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large Language Models Can Self-Improve. *arXiv preprint arXiv:2210.11610*, 2022.

[62] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. AI/ML for Network Security: The Emperor Has No Clothes. In *ACM CCS*, 2022.

[63] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *ICML*, 2019.

[64] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

[65] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Paul Schmitt, Francesco Bronzino, and Nick Feamster. Generative, High-Fidelity Network Traces. In *ACM HotNets*, 2023.

[66] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In *ACM SOSR*, 2016.

[67] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. RECL: Responsive Resource-Efficient Continuous Learning for Video Analytics. In *USENIX NSDI*, 2023.

[68] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. In-Band Network Telemetry via Programmable Dataplanes. In *ACM SIGCOMM*, 2015.

[69] Jared Knofczynski, Ramakrishnan Durairajan, and Walter Willinger. ARISE: A Multitask Weak Supervision Framework for Network Measurements. *IEEE Journal on Selected Areas in Communications*, 40(8):2456–2473, 2022.

[70] David Koeplinger, Matthew Feldman, Raghu Prabhakar, Yaqi Zhang, Stefan Hadjis, Ruben Fiszel, Tian Zhao, Luigi Nardi, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. Spatial: A Language and Compiler for Application Accelerators. In *ACM PLDI*, 2018.

[71] Franck Le, Mudhakar Srivatsa, Raghu Ganti, and Vyas Sekar. Rethinking Data-Driven Networking with Foundation Models: Challenges and Opportunities. In *ACM HotNets*, 2022.

[72] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*, 2020.

[73] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. HPCC: High Precision Congestion Control. In *ACM SIGCOMM*, 2019.

[74] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is Your Code Generated by ChatGPT Really Correct?

Rigorous Evaluation of Large Language Models for Code Generation. In *NeurIPS*, 2023.

[75] Shinan Liu, Francesco Bronzino, Paul Schmitt, Arjun Nitin Bhagoji, Nick Feamster, Hector Garcia Crespo, Timothy Coyle, and Brian Ward. LEAF: Navigating Concept Drift in Cellular Networks. *Proceedings of the ACM on Networking*, 1(CoNEXT2):1–24, 2023.

[76] Shinan Liu, Ted Shaowang, Gerry Wan, Jeewon Chae, Jonatas Marques, Sanjay Krishnan, and Nick Feamster. ServeFlow: A Fast-Slow Model Architecture for Network Traffic Analysis. *arXiv preprint arXiv:2402.03694*, 2024.

[77] Yingqiu Liu, Wei Li, and Yunchun Li. Network Traffic Classification Using K-Means Clustering. In *IMSCCS*, 2007.

[78] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep Packet: A Novel Approach for Encrypted Traffic Classification Using Deep Learning. *Soft Computing*, 24(3):1999–2012, 2020.

[79] Inbal Magar and Roy Schwartz. Data Contamination: From Memorization to Exploitation. *arXiv preprint arXiv:2203.08242*, 2022.

[80] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource Management with Deep Reinforcement Learning. In *ACM HotNets*, 2016.

[81] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*, 2017.

[82] Tahir Mehmood and Helmi B Md Rais. SVM for Network Anomaly Detection Using ACO Feature Subset. In *IEEE iSMSC*, 2015.

[83] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, Giovani Estrada, Khaldun Ma'ruf, Florin Coras, Vina Ermagan, Hugo Latapie, Chris Cassar, John Evans, Fabio Maino, Jean Walrand, and Albert Cabellos. Knowledge-Defined Networking. *ACM SIGCOMM CCR*, 47(3):2–10, 2017.

[84] Nour Moustafa and Jill Slay. UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[85] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online Model Distillation for Efficient Video Inference. In *ICCV*, 2019.

[86] Anirudh Muthukumar and Ramakrishnan Durairajan. Denoising Internet Delay Measurements Using Weak Supervision. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 479–484. IEEE, 2019.

[87] OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

[88] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019.

[89] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language Models as Knowledge Bases? *arXiv preprint arXiv:1909.01066*, 2019.

[90] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models Are Unsupervised Multitask Learners. *OpenAI Blog*, 1(8):9, 2019.

[91] Alex Ratner, Braden Hancock, Jared Dunnmon, Roger Goldman, and Christopher Ré. Snorkel Metal: Weak Supervision for Multi-Task Learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, pages 1–4, 2018.

[92] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid Training Data Creation with Weak Supervision. *The VLDB Journal*, 29(2-3):709–730, 2020.

[93] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data Programming: Creating Large Training Sets, Quickly. In *NeurIPS*, 2016.

[94] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental Classifier and Representation Learning. In *CVPR*, 2017.

[95] Adam Roberts, Colin Raffel, and Noam Shazeer. How Much Knowledge Can You Pack into the Parameters of a Language Model? *arXiv preprint arXiv:2002.08910*, 2020.

[96] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the Social Network's (Datacenter) Network. In *ACM SIGCOMM*, 2015.

[97] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. Can the Network Be the AI Accelerator? In *ACM NetCompute*, 2018.

[98] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *International Conference on Information Systems Security and Privacy (ICISSP)*, 1:108–116, 2018.

[99] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy. In *International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019.

[100] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental Learning of Object Detectors without Catastrophic Forgetting. In *ICCV*, 2017.

[101] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. Re-Architecting Traffic Analysis with Neural Network Interface Cards. In *USENIX NSDI*, 2022.

[102] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, 2018.

[103] Linqi Song, Cem Tekin, and Mihaela Van Der Schaar. Online Learning in Large-Scale Contextual Recommender Systems. *IEEE Transactions on Services Computing*, 9(3):433–445, 2014.

[104] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. Taurus: A Data Plane Architecture for Per-Packet ML. In *ACM ASPLOS*, 2022.

[105] Tuan A. Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. In *IEEE WINCOM*, 2016.

[106] Gemini Team. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint arXiv:2312.11805*, 2024.

[107] Saar Tochner, Giulia Fanti, and Vyas Sekar. Gen-T: Reduce Distributed Tracing Operational Costs Using Generative Models. In *Temporal Graph Learning Workshop@ NeurIPS 2023*, 2023.

[108] Gerry Wan, Fengchen Gong, Tom Barbette, and Zakir Durumeric. Retina: Analyzing 100GbE Traffic on Commodity Hardware. In *ACM SIGCOMM*, 2022.

[109] Minxiao Wang, Ning Yang, Nicolas J Forcade-Perkins, and Ning Weng. ProGen: Projection-Based Adversarial Attack Generation against Network Intrusion Detection. *IEEE Transactions on Information Forensics and Security*, 2024.

[110] Qineng Wang, Chen Qian, Xiaochang Li, Ziyu Yao, and Huajie Shao. Lens: A Foundation Model for Network Traffic in Cybersecurity. *arXiv preprint arXiv:2402.03646*, 2024.

[111] Keith Winstein and Hari Balakrishnan. TCP ex Machina: Computer-Generated Congestion Control. In *ACM SIGCOMM*, 2013.

[112] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. NetLLM: Adapting Large Language Models for Networking. *arXiv preprint arXiv:2402.02338*, 2024.

[113] Jun Xiao, Minjuan Wang, Bingqian Jiang, and Junli Li. A Personalized Recommendation System with Combinational Algorithm for Online Learning. *Journal of Ambient Intelligence and Humanized Computing*, 9:667–677, 2018.

[114] Zhaoqi Xiong and Noa Zilberman. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *ACM HotNets*, 2019.

[115] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Alexander Levis, and Keith Winstein. Learning in situ: A Randomized Experiment in Video Streaming. In *USENIX NSDI*, 2020.

[116] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Alexander Levis, and Keith Winstein. Pantheon: The Training Ground for Internet Congestion-Control Research. In *USENIX ATC*, 2018.

[117] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *USENIX Security*, 2021.

[118] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical GAN-Based Synthetic IP Header Trace Generation Using NetShare. In *ACM SIGCOMM*, 2022.

[119] Liangcheng Yu, John Sonchack, and Vincent Liu. Mantis: Reactive Programmable Switches. In *ACM SIGCOMM*, 2020.

[120] Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation. *arXiv preprint arXiv:2310.02304*, 2024.

[121] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-Resolution Measurement of Data Center Microbursts. In *ACM IMC*, 2017.

[122] Changgang Zheng, Zhaoqi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. IIsy: Hybrid In-Network Classification Using Programmable Switches. *IEEE/ACM Transactions on Networking*, 2024.

[123] Gan Zheng, Ioannis Krikidis, Christos Masouros, Stelios Timotheou, Dimitris-Alexandros Toumpakaris, and Zhiguo Ding. Rethinking the Role of Interference in Wireless Networks. *IEEE Communications Magazine*, 52(11):152–158, 2014.

[124] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *NeurIPS*, 2024.

[125] Zhizhen Zhong, Mingran Yang, Jay Lang, Christian Williams, Liam Kronman, Alexander Sludds, Homa Esfahanizadeh, Dirk Englund, and Manya Ghobadi. Lightning: A Reconfigurable Photonic-Electronic SmartNIC for Fast and Energy-Efficient Inference. In *ACM SIGCOMM*, 2023.

A Details on LLM-based Knowledge Source

A.1 Model Choice & Reproducibility

We use the `gpt-4-1106-preview` model snapshot from the OpenAI API service as the LLM—the latest model available at the time of implementation and evaluation of Caravan. We anticipate that future model snapshots released by OpenAI (such as `gpt-4-turbo` and `gpt-4o`) or Google (such as `Gemini Ultra` and `Gemini Flash`) could be adapted for data labeling using similar prompts, as long as the model supports a sufficiently large context window. The same would hold true for emerging open-source LLMs, such as those from Meta (e.g., Llama 3 series [16]) and Mistral AI (e.g., Mixtral 7B [64]).

The behavior of commercial LLM APIs may evolve over time, even using the same model snapshot and prompts [39]. To ensure reproducibility, one strategy would involve leveraging open-source LLMs instead of third-party APIs. However, these LLMs necessitate high-end GPUs or aggressive compression before deployment; we do not use these in our paper. Another approach is to decrease the temperature [22] during the generation process to minimize variability across different runs when utilizing third-party APIs.

A.2 System Prompts

> **a. System Prompt (UNSW-NB15):** You are an expert in network security. The user is now labeling a network intrusion detection dataset, and he/she wants to assign a binary label (0 for benign or 1 for malicious) to each traffic flow in the dataset based on each flow's input features. He/She will give you a few labeled flows for reference, and you will then help him/her label another few unlabeled flows. Feel free to use your own expertise and any information the user gives you. These are the features of the input flows and meanings of the features: dur (record total duration), proto (transaction protocol, which will be categorized), sbytes (source to destination transaction bytes), dbytes (destination to source transaction bytes), sttl (source to destination time to live value), dttl (destination to source time to live value), sload (source bits per second), dload (destination bits per second), spkts (source to destination packet count), dpkts (destination to source packet count), smean (mean of the packet size transmitted by the src), dmean (mean of the packet size transmitted by the dst), sinpkt (source interpacket arrival time (mSec)), dinpkt (destination interpacket arrival time (mSec)), tcprtt (TCP connection setup round-trip time), synack (TCP connection setup time, the time between the SYN and the SYN_ACK packets), ackdat (TCP connection setup time, the time between the SYN_ACK and the ACK packets), ct_src_ltm (no. of connections of the same source address in 100 connections according to the last time), ct_dst_ltm (no. of connections of the same destination address in 100 connections according to the last time), ct_dst_src_ltm (no. of connections of the same source and the destination address in 100 connections according to the last time).

> **b. System Prompt (CIC-IDS2017):** You are an expert in network security. The user is now labeling a network intrusion detection dataset, and he/she wants to assign a binary label (0 for benign or 1 for malicious) to each traffic flow in the dataset based on each flow's input features. He/She will give you a few labeled flows for reference, and you will then help him/her label another few unlabeled flows. Feel free to use your own expertise and any information the user gives you. These are the features of the input flows and meanings of the features: flow IAT min (minimum packet inter-arrival time in microseconds), flow IAT max (maximum packet inter-arrival time in microseconds), flow IAT mean(average packet inter-arrival time in microseconds), packet length min (minimum packet length), packet length max (maximum packet length), packet length mean (average packet length), total packet length (total packet length), number of packets (total number of packets in the flow), SYN flag count (number of TCP SYN flags), ACK flag count(number of TCP ACK flags), PSH flag count (number of TCP PSH flags), FIN flag count (number of TCP FIN flags), RST flag count (number of TCP RST flags), ECE flag count (number of TCP ECE flags), flow duration (duration of flow in microseconds), and DST port (destination port).

A.3 A Rule Extraction Output by LLM

In Figure 14, we show an example heuristics-based labeling function generated by the LLM, based on 500 unlabeled flows from the UNSW-NB15 dataset. The model generates all comments along with the function.

A.4 Concerns Regarding Data Contamination

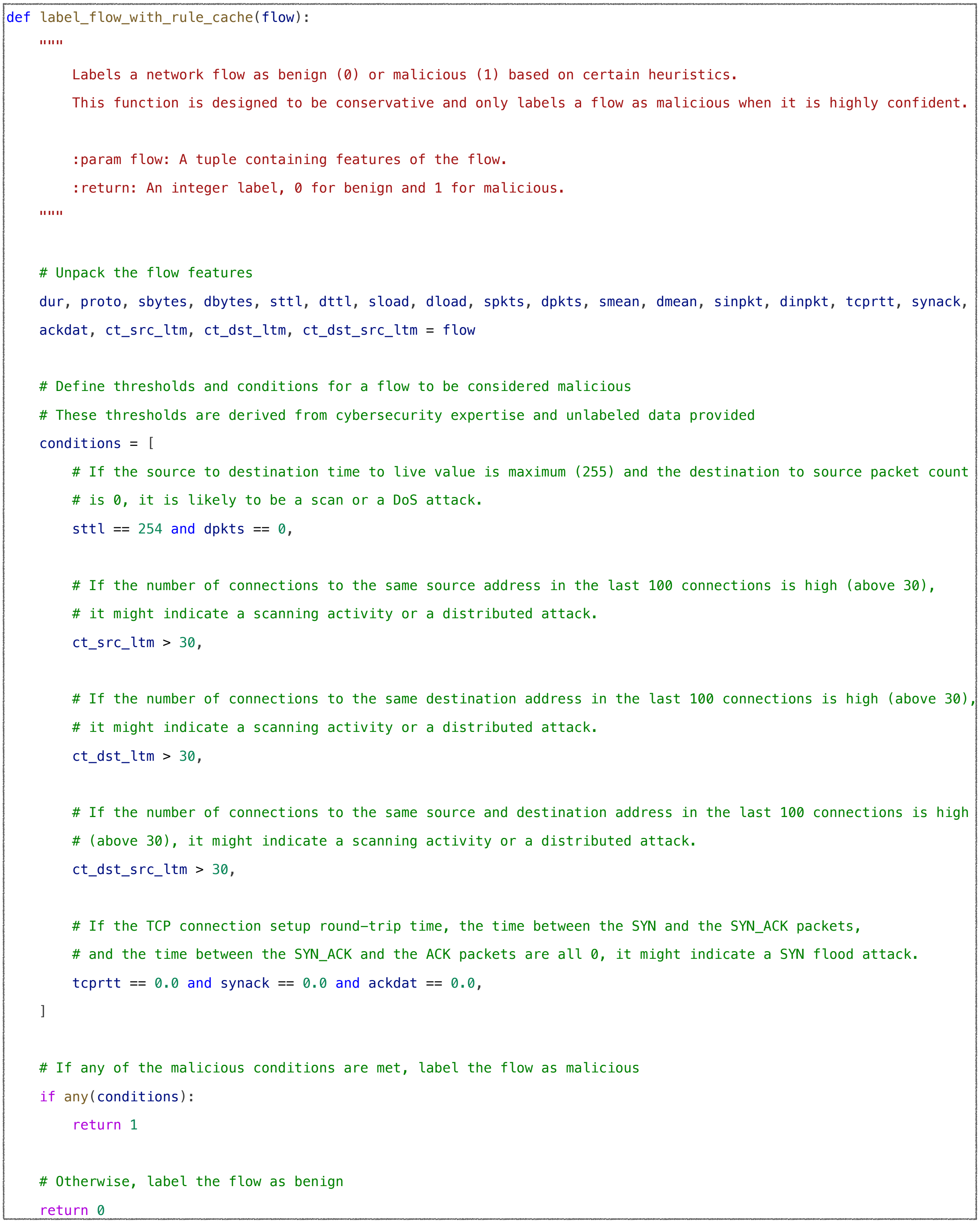
In the domain of machine learning, data contamination [79] occurs when pre-trained language models, trained on extensive web-based datasets, inadvertently incorporate downstream test sets into their pretraining corpus. This phenomenon can lead to biases in evaluation metrics and diminish the generalization capacity of large language models. In Caravan, the issue of data contamination is also pertinent, as we adhere to previous methodologies by utilizing public datasets for evaluation.

While it is not feasible to entirely eradicate the issue of data contamination through retraining a new model from scratch, we implement the following measures to mitigate its potential impact on our evaluation: (a) We ensure that the LLM never generates *trivial* labels, meaning labels that precisely match the ground truth labels. (b) We carefully examine the labeling rule functions produced by the LLM to ensure they do not generate labels through *memorization* based on features, such as IP address.

B Artifact Appendix

B.1 Abstract

The artifact includes the software simulation and the FPGA/P4 testbed we built for implementing and evaluating Caravan. We also include a detailed guide with step-by-step



```python
def label_flow_with_rule_cache(flow):
    """
        Labels a network flow as benign (0) or malicious (1) based on certain heuristics.
        This function is designed to be conservative and only labels a flow as malicious when it is highly confident.

        :param flow: A tuple containing features of the flow.
        :return: An integer label, 0 for benign and 1 for malicious.
    """

    # Unpack the flow features
    dur, proto, sbytes, dbytes, sttl, dttl, sload, dload, spkts, dpkts, smean, dmean, sinpkt, dinpkt, tcprtt, synack,
    ackdat, ct_src_ltm, ct_dst_ltm, ct_dst_src_ltm = flow

    # Define thresholds and conditions for a flow to be considered malicious
    # These thresholds are derived from cybersecurity expertise and unlabeled data provided
    conditions = [
        # If the source to destination time to live value is maximum (255) and the destination to source packet count
        # is 0, it is likely to be a scan or a DoS attack.
        sttl == 254 and dpkts == 0,

        # If the number of connections to the same source address in the last 100 connections is high (above 30),
        # it might indicate a scanning activity or a distributed attack.
        ct_src_ltm > 30,

        # If the number of connections to the same destination address in the last 100 connections is high (above 30),
        # it might indicate a scanning activity or a distributed attack.
        ct_dst_ltm > 30,

        # If the number of connections to the same source and destination address in the last 100 connections is high
        # (above 30), it might indicate a scanning activity or a distributed attack.
        ct_dst_src_ltm > 30,

        # If the TCP connection setup round-trip time, the time between the SYN and the SYN_ACK packets,
        # and the time between the SYN_ACK and the ACK packets are all 0, it might indicate a SYN flood attack.
        tcprtt == 0.0 and synack == 0.0 and ackdat == 0.0,
    ]

    # If any of the malicious conditions are met, label the flow as malicious
    if any(conditions):
        return 1

    # Otherwise, label the flow as benign
    return 0
```

Figure 14: A heuristics-based labeling function generated by the LLM.

instructions for automatically running the key experiments and plotting the figures presented in the paper.

B.2 Scope

The `simulation/` folder contains the source code to automatically run key experiments from the paper and reproduce the corresponding figures (i.e., Figures 4–10, 12). The `testbed/` folder contains the new code changes and the instructions to set up and run the FPGA/P4-based evaluations for CARAVAN.

B.3 Contents

The artifact is provided as a self-contained repository available at https://github.com/Per-Packet-AI/Caravan-Artifact-OSDI24.

- **`simulation/`** contains the software code for reproducing evaluated figures, with automation scripts for generating data and producing figures located at `simulation/scripts/experiments.sh` and `simulation/scripts/plots.sh`, respectively.
- **`testbed/`** contains a modified version of the Taurus FPGA testbed [104] for testing CARAVAN's use cases.

B.4 Hosting

CARAVAN is hosted on GitHub: https://github.com/Per-Packet-AI/Caravan-Artifact-OSDI24.

B.5 Requirements

Hardware. CARAVAN requires at least an 8-core server with 16 GiB of RAM, one CUDA 12.1-compatible GPU (e.g., Nvidia V100), along with Internet connectivity to access OpenAI API endpoints. We recommend using a Google Compute Engine (g2-standard-8) instance.

Software. CARAVAN runs with Python version 3.10 or later with CUDA support. The complete list of dependencies is available in `simulation/pyproject.toml` and gets installed automatically using `pip install -e .` from the `simulation/` directory.