RESEARCH ARTICLE

PAC-GPT: A Novel Approach to Generating Synthetic Network Traffic With GPT-3

DANIAL KHOSH KHOLGH[ID] **AND PANOS KOSTAKOS**[ID]

Center for Ubiquitous Computing, University of Oulu, 90570 Oulu, Finland

Corresponding author: Panos Kostakos (panos.kostakos@oulu.fi)

ABSTRACT The application of machine learning models, particularly in cybersecurity, has surged significantly in the past few years. However, the effectiveness of these models is predominantly tethered to the quality and breadth of the training data they ingest. The scarcity of realistic datasets within the cybersecurity field constitutes a considerable challenge to the development of industry-grade tools intended for real-world application scenarios. Specifically, current datasets are either significantly outdated or fall short on both qualitative and quantitative fronts, primarily because many organizations exhibit reluctance in data sharing, stemming from privacy concerns or the potential threat to trade secrets. To address this challenge, the paper introduces PAC-GPT, a novel framework to generate reliable synthetic data for machine learning methods based on Open AI's Generative Pre-trained Transformer 3 (GPT-3). The core components of this framework are two modules, namely a Flow Generator, which is responsible for capturing and regenerating patterns in a series of network packets, and Packet Generator, which can generate individual network packets given the network flow. We also propose a packet generator based on LLM chaining and then proceed to assess, compare, and evaluate its performance using metrics such as loss, accuracy and success rate, concluding that transformers are a suitable approach for synthetic packet generation with minimal fine-tuning performed. Lastly, a streamlined command line interface (CLI) tool has been devised to facilitate the seamless access of this innovative data generation strategy by professionals from various disciplines.

I. INTRODUCTION

The global surge in cyber attacks and their significant financial implications have intensified the importance of cybersecurity for public and private entities alike. To mitigate these threats, organizations are required to process vast quantities of data, but the scale of the task makes manual analysis highly impractical. Machine learning (ML) tools, especially Deep Learning (DL) models, have been successfully used for cybersecurity applications thanks to the ever-increasing availability of data. Malware detection [1], [2], [3], spam filters [4], [5], [6], and phishing detection [7], [8] are just a number of applications to mention. However, the efficacy of machine learning approaches in the field

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam[ID].

of cybersecurity is hampered by the significant shortfall in the quality of the current training datasets, a critical requirement for these methods. The majority of currently accessible cybersecurity datasets suffer from either outdated information or insufficient quantity and quality, rendering them inadequate for meeting the demanding requirements of machine learning (ML) algorithms [9]. Conversely, high-quality realistic datasets from real network environments hold personal and sensitive data, which can compromise privacy, thus presenting obstacles for data scientists when acquiring appropriate data for model training. Various solutions have been suggested to address data privacy concerns. The three most notable methods include non-cryptographic anonymization, cryptographic anonymization, and differential privacy, a type of perturbation technique. Nevertheless, these methods have drawbacks like the exposure of private information,

unsuitability for large datasets, and deterioration of data quality, rendering them less ideal for machine learning tasks [10].

Based on our previous work which focused on generative modeling architectures like GANs and Transformers [11], we propose here an innovative Large Language Model (LLM) Chaining technique which enables us to produce synthetic network traffic closely resembling real data and thereby providing a suitable alternative for machine learning tasks within the cybersecurity domain. Specifically, our main aim is to create high-quality synthetic network traffic that can be applied to various tasks, such as training intrusion detection systems (IDS).

To accomplish our objectives, we initially developed a robust 'packet generator', designed to produce network packets on demand. The primary aim of our framework was to achieve this through the exploration of various transformer-based models, which were evaluated using a diverse range of metrics. Subsequently, we employed a collection of Python scripts to establish a 'flow generator' capable of injecting a series of custom packets into the network. These two integral components were amalgamated into a single command-line interface (CLI) tool, designed to seamlessly manage the data generation pipeline from start to finish. Lastly, we conducted a thorough investigation into the viability of this synthetic alternative for various cybersecurity tasks. Our primary goal was to assess whether this methodology could effectively substitute real data.

To summarize, the aim is to answer the following research questions:

- **1:** How do transformers perform while tasked with generating different types of network packets?
- **2:** How useful is the tool for generating network packets, when considering various threat modeling frameworks?
- **3:** What impacts does synthetic network traffic generation have on the industry? Or more precisely, which type of end users could benefit (or suffer) from this tool?

We introduce in this work an end-to-end tool called *PAC-GPT*, which is capable of generating high quality synthetic network traffic at a low cost. We propose novel framework consisting of multiple GPT-3 large language models chained together. To the best of our knowledge, this technique has not been utilized before for generating synthetic network packets. This data in turn can be used in various cybersecurity tasks as an alternative to authentic data, which is very costly to acquire especially in large quantities. All of the implementations and data that we used and generated in this work are publicly available.[1]

II. RELATED WORKS

Over the past ten years, academic research in computer networking has experienced significant expansion. Conducting thorough research experiments with network topologies that mirror real-world networks in terms of node and link quantity

[1] https://github.com/dark-0ne/NetworkPacketGenerator

and variety has become significantly easier. As a result, there is a growing need for realistic traffic workloads in testing environments. However, since actual production-level traffic data is typically inaccessible to academic researchers due to privacy concerns [12], traffic generators have become widely used in network science and engineering research. Even when privacy is not a concern, the challenges of scaling a production traffic capture for a test-bed can be overwhelming [13], [14]. Additionally, replaying existing traffic traces is only useful for specific research experiments where the local topology matches the network where the packets were initially captured. In other words, the ability to replay these traces is considerably limited when the network topologies differ greatly from the original capture network [9]. Consequently, researchers often turn to alternative methods to create traffic workloads for their experiments, with traffic generators being a popular choice. In this context, the remaining section presents a chronological overview of various traffic generation techniques.

A. CHRONOLOGICAL SURVEY OF NETWORK TRAFFIC GENERATORS

Network traffic generators and simulators have historically served as an alternative data procurement strategy in the realm of academic research. The timeline of synthetic network traffic creation can be roughly divided into three generations, with each being explored briefly in the following:

1) FIRST GENERATION: NETWORK TRAFFIC SIMULATORS

A packet simulator or packet builder is a program that generates random packets or enables users to create customized, detailed packets. This software produces packets that allow users to examine and evaluate network setups, protocols, and system designs without requiring actual network traffic. Users can either use a predefined set of random packets or create their own, specifying the individual components of the packet, such as header information and payload data. As a result, this software becomes an essential tool for network engineers and administrators who need to test and fine-tune their networks on a large scale with consistent repeatability. Some traffic generators are designed to stress test network devices and software, while others are intended for creating packets to assess performance and verify the accuracy of behavior [15].

Throughout the years, a considerable amount of research has emerged in the domain of network traffic generation [16], [17]. The primary focus of these investigations is to explore the applications of network traffic workloads for validating different aspects, such as the precision of algorithms, protocols, and network functions. These workloads also serve to evaluate the influence of novel and evolving technologies on network performance.

Multiple studies, especially in the early phases, concentrated on performing benchmark assessments of various tools and technologies. Kolahi et al. [16] compared the

performance of four network traffic generators (Iperf, Net-perf, D-ITG, and Internet Protocol (IP) Traffic) in a lab setting, and found that these tools can yield significantly different results, even when deployed on Windows operating systems over a 100 Mbps (Megabits per second) link with differences in TCP (Transmission Control Protocol) connection speed reaching up to 16.5 Mbps among the mentioned monitoring tools. Mishra et al. [18] performed a similar analysis of six traffic generators (D-ITG, PackETH, Ostinato, Iperf, Netperf, and IP Traffic) in terms of performance under different scenarios. The results of their study also revealed that each generator outperformed others in distinct metrics under diverse conditions. Therefore, the authors deduced that no single traffic generator is appropriate for every network type, and that different generators are suitable for addressing the unique requirements and features of a specific network and application.

In a similar fashion, Emmerich et al. [19] explored the potential issues that could emerge when a particular traffic pattern must be generated reliably at high speeds. This is a crucial factor in evaluating a packet generator, as the generated pattern can impact the performance of a system being tested. As a result, the authors categorized software packet generators into two groups: i) traditional software packet generators that rely on the kernel's interface for packet IO (e.g., 1 Gbit (Gigabit) platforms), and ii) modern software packet generators that utilize specialized frameworks to bypass an OS's entire network stack (e.g., 10 Gbit platforms).

Molnár et al. [17] examined numerous traffic generators and classified them into five groups based on the metrics used for validation purposes. These categories consist of:

1) **Replay Engines**, which capture and replay network traffic to reproduce real-world patterns.
2) **Maximum Throughput Generators**, created to optimize network throughput.
3) **Model-Based Generators**, which produce network traffic using mathematical models to emulate various traffic types.
4) **High-Level and Auto-Configurable Generators**, which enable user-defined parameters to generate tailored network traffic.
5) **Special Scenario Generators**, utilized to generate traffic for specific testing situations.

In a more recent review, Adeleke et al. [9] present an extensive survey of 92 frequently cited generators over a 13-year period from 2006 to 2018. This survey emphasizes an in-depth classification of functional behaviors. Expanding on the classification offered by Molnár et al. [17], the study proposes various categories of traffic generators:

1) **Constant or maximum throughput generators:** This category of traffic generators sends packets at a constant rate or the maximum possible rate (in bits per second (bps) or packets per second). Examples include iperf2 and netperf. These generators are often the simplest to use and are suitable for rapid network throughput stress testing.

2) **Application-level generators:** Specialized generators (e.g., httperf) generate network packet traffic for a specific type of application or higher-layer protocol. Researchers can run actual applications and execute a specified set of workloads, using their data exchanges to generate traffic. This workload generation method can produce more realistic packet variations for a specific application or protocol.

3) **Trace file replay tools:** These traffic generators reproduce network traces based on recorded data (e.g., pcap files, a common file format for storing and analyzing network packets). This allows users to simulate conditions from previous experiments or real-life cyber-attacks and can be used for testing purposes such as performance evaluation or Intrusion Detection Systems (IDS) debugging.

4) **Model-based traffic generators:** These generators create realistic network traffic by sending packets that adhere to random distributions in terms of time intervals, packet sizes, and other parameters. By selecting the appropriate random distributions, the generator can produce traffic that closely resembles specific real-world traffic workloads.

5) **Trace Driven Model-Based Traffic Generators:** In addition to the model-based approach, some traffic generators use trace file input or log files from actual production networks as a source. This involves analyzing the input files to fit traffic parameters to random distributions, which are then used to generate packets that are statistically similar to the real packets seen in the corresponding production network trace or log file.

6) **Script-Based Traffic Generators:** These generators provide users with the flexibility to adjust the entire range of packet header and content data using complex coding logic. They enable users to create any packet type with customizable header values and the capability to modify packets during runtime.

2) SECOND GENERATION: TRAFFIC GENERATION AND CLASSIFICATION USING ML

Traffic generators play a vital role in evaluating network systems' performance and are used to test communication protocol designs, detect malware traffic, and assess network infrastructure capabilities (e.g., IDS). However, conventional traffic generators have their limitations. They either emulate traffic patterns using probability distributions, which may not accurately represent real-world packet traffic, or they require an extensive setup of targeted applications with user modeling, which can be a challenging and time-consuming process. Wu et al. [20] point out that while some traffic generators offer advanced distributions that account for bursty traffic and anomalies, the actual bottleneck is that packet lengths in real-life network traffic are categorical (i.e., limited to specific values or ranges like IP addresses and port numbers), not continuous. Categorical data refers

to data that can be divided into distinct categories or bins, while continuous values are those that can take any value within a given range. Consequently, these distributions cannot precisely model the multiple peaks in packet length probabilities that occur in real-life networks.

The challenges of generating datasets for modern networking applications have been intensified by the success of machine learning approaches in surpassing traditional methods for various cybersecurity tasks. This makes machine learning a promising solution for addressing communication and networking issues, including anomaly and intrusion detection. Several frameworks have been introduced to generate datasets (and classify network flows for IDS applications [21]) suitable for a wide range of applications, facilitating the implementation of machine learning advancements in real-world network environments. These datasets aim to significantly impact production networks, overcoming the limitations of previous methods. The proposed methods differ from traditional protocol-based data generation/classification, which relies on rules, heuristics, or information about IP, port numbers, and signature protocols. Several methods, including Bayesian methods (specifically neural networks), modified Association algorithms, Support Vector Machines, Venn Probability Machines, k-Nearest Neighbor, and k-means clustering algorithms, have been employed to produce class probability distributions [22]. Also, neural networks, particularly deep neural networks, have demonstrated tremendous potential in the area of generation [21], [22].

Machine learning models are often developed to recognize and categorize various forms of network traffic, such as regular traffic, malicious traffic, and suspicious activities. These models are generally benchmarked against a dataset of network traffic to assess their accuracy and effectiveness. In the following list, we outline some key datasets commonly used for benchmarking machine learning models in cybersecurity:

- The DARPA dataset [23] is a frequently used benchmark dataset for machine learning models in cybersecurity. It is based on both audit logs and network traffic and includes seven weeks of training data and two weeks of testing data. These were gathered in two parts: an offline evaluation using network traffic and audit logs collected on a simulation network, and a real-time evaluation through Air Force Research Laboratory (AFRL).
- The KDD'99 dataset [24] remains the widely accepted standard for machine learning in networking, despite its limitations. Composed of Ethernet transmissions captured for identifying known malware activity, it was released for use in developing intrusion detection systems and has been used in network research, either in its original form or the improved NSL-KDD version. Despite its shortcomings, KDD'99 is still frequently cited in current research papers [25], [26].
- The TON_IoT datasets [27] are a new generation of Industry 4.0/Internet of Things (IoT) and Industrial IoT (IIoT) datasets designed for evaluating the accuracy and efficiency of various cybersecurity applications based on Artificial Intelligence (AI), specifically Machine/Deep Learning algorithms. These datasets can be utilized for validation and testing of a wide range of AI-based cybersecurity applications, including intrusion detection systems, threat intelligence, malware detection, fraud detection, privacy preservation, digital forensics, adversarial machine learning, and threat hunting.

3) THIRD GENERATION: SYNTHETIC TRAFFIC GENERATION

The limited availability of cybersecurity data has led to the emergence of various unsupervised machine learning techniques. Recently, deep generative models like GANs [28], Variational Autoencoders (VAEs) [29], and Transformers [30] have gained considerable popularity due to their capability to generate high-quality synthetic images, audio, and text. Moreover, deep generative models have made substantial contributions to the cybersecurity field through the application of machine learning. They have been employed in tasks such as traffic classification and network packet generation, proving to be valuable assets in the development and testing of cyber defense tools. We will have a look at two prominent architectures GANs and Transformers, which have both gained significant attention in cybersecurity domain recently.

a: GENERATIVE ADVERSARIAL NETWORKS

GANs are a class of deep learning models introduced by Ian Goodfellow et al. [28] in 2014. GANs have found numerous applications across various domains such as image synthesis [31], [32], super-resolution [33], anomaly detection [34], [35], and even natural language processing [36], [37], [38]. GANs have also found a more prominent role in the cybersecurity field. GAN-based architectures have become capable of comprehending intricate data distributions at the packet level and producing comparable outcomes with minor differences throughout the recent years. This has been achieved through certain modifications made to the model, as described in [21] and [22]. GAN implementations commonly generate data in three different formats: packet, flow, or tabular. When generating data in the packet format, individual packets are created, while the flow format produces connected packets transmitted between hosts. In the tabular format, data is presented in a structured table, with each row representing a data point and each column indicating a characteristic or attribute of that data point. We will mention some of the most recent works in each category in the following.

Packet-level data pertains to the unprocessed information gathered from individual network packets, which may contain data such as source and destination IP addresses, packet size, and protocol type. Cheng's PAC-GAN, as explained in [39], proposes a method of using GANs to create authentic network traffic data at the IP packet layer for the application in cyber and network security tasks. With the help of GANs, the author

was able to demonstrate the possibility of generating real traffic flows, including Internet Control Message Protocol (ICMP) Pings, Domain Name Service (DNS) queries, and Hyper Text Transfer Protocol (HTTP) web requests. The paper suggests an alternative encoding of network traffic data into a Convolutional Neural Network (CNN) GAN model, where experiments indicate that the generated traffic can be transmitted successfully through the internet and generate the desired network responses. This novel application of GANs in the cybersecurity domain employs CNN GANs and an encoding scheme for network traffic data into image-based matrix representations.

The PcapGAN architecture, as mentioned in [40], shares similarities with the CNN GAN model discussed earlier and comprises an encoder, a generator, and a decoder, working together in a hybrid structure to produce realistic pcap data. The encoder's job is to extract information from pcap data format and convert it into appropriate formats, such as graphs, images, or sequences. PcapGAN also introduces the concept of "style" as a vector value to efficiently express host2host relationships, such as the C&C server-botnet connections. The generator is designed based on the edge style, and advanced GAN models generate new data for each data format extracted by the encoder. Finally, the decoder combines the generated information to reconstruct a valid pcap file, making PcapGAN an effective method for generating authentic pcap data with high levels of customization.

On the other hand, network flow data is a collective representation of network traffic over a specific duration, including data such as the number of packets, bytes, connections, and statistical features like mean and variance. In Ring et al. [41], WGAN-GP, a deep learning technique incorporating a Two Time-Scale Update Rule, was used to transform and regenerate the continuous attributes obtained from flow-based network traffic. This involved treating network attributes as numerical values, creating binary attributes from categorical attributes, and using IP2Vec [42] similarity measures to learn vector representations from categorical attributes. Furthermore, the researchers extracted features such as IP addresses, Destination Ports, and Transport Protocols from the flow-based network traffic and converted them into one-hot vectors. The neural network employed in this process had an input and output layer, each with the same number of neurons as the vocabulary size, and a hidden layer with fewer neurons than the input layer. Ultimately, the output layer leveraged a Softmax Classifier to predict the probability of each value of the vocabulary in the same flow as the input value, with the sum of all output neurons normalized to 1.

Lastly, tabular data refers to data that is organized in tables or spreadsheets, with rows representing individual records and columns representing features or attributes. In Bourou et al. [21], the use of GANs is discussed in generating synthetic IDS data for network security. The paper highlights popular GAN architectures such as VanillaGAN

[28] and concentrates on tabular data generation models like CTGAN [43], CopulaGAN [44], and TableGAN [45].

b: TRANSFORMERS

Lately, transformer models have emerged as a groundbreaking development in the field of natural language processing (NLP) and deep learning. Introduced by Vaswani et al. [30], transformers are an innovative architecture that departs from the traditional recurrent neural networks (RNNs) and CNNs previously used for sequence modeling tasks. Since their introduction, transformers have rapidly become the de facto standard for various NLP tasks, including machine translation [46], sentiment analysis [47], text summarization [48], and question-answering systems.

Transformers have also become very prominent in the network and cybersecurity domain over recent years. Bikmukhamedov et al. [49], [50] introduced a new generative transformer-based model for network traffic that serves the purpose of generating and classifying network data. The model uses packet size and inter-packet time sequences as flow features to simplify inputs and can be trained in two ways: for generating network traffic and as a network flow classifier. To design the model, the researchers utilized the well-known GPT-2 [51] architecture, optimizing its configuration by making numerous changes including altering layers in the architecture, modifying input dimension, quantizing packets, and using a specific dataset for training. The results of the study showed that the model's generated traffic quality was comparable to that of a first-order Markov chain trained independently on each traffic class. Additionally, enriching the dataset with external traffic from various domains improved the quality of the generated traffic on target classes. The researchers also found that generative pre-training had a positive impact on the traffic classification task's quality, with the classifier outperforming the ensemble by an average of 4% according to the F1-macro metric when all model parameters were trained.

Wass [52] studied the need for improved resource utilization in mobile networks, given the high costs and limitations of network resources, as well as the growing demand for mobile data. Lin et al. [53] have introduced a new model called ET-BERT for classifying encrypted communication. The proposed model can pre-train deep contextual datagram-level traffic representations from vast quantities of unlabeled data and effectively categorize encrypted traffic for numerous situations using only a modest quantity of task-specific labeled data.

Various studies have proposed pre-trained transformer methods to address various cybersecurity topics, such as Intrusion Detection System [54], Honeypot Log analysis [55], Distributed Denial of Service (DDoS) [54], anomaly detection classifier [55] and encrypted traffic classification [56], [57]. These methods are designed to analyze and classify data related to cyber-attacks and security breaches, enabling the identification of potential threats and improving network

security. By leveraging the power of pre-training, these models can achieve high levels of accuracy and performance with minimal fine-tuning on task-specific datasets. However, the potential of using these models for generating network data has not been fully explored.

B. EVALUATING SYNTHETIC NETWORK TRAFFIC DATA

Just as the data generation methods themselves, evaluation techniques for synthetic network data has also gone through a number of advancements throughout the years. Ensuring the quality and applicability of the generated data is vital when dealing with synthetic data. Therefore, a great number of prior works have focused on this aspect as well. In this part we will have a look at some of the most common evaluation methods for synthetic network traffic data.

Emmerich et al. [19] assessed the rate control capabilities of the examined packet generators based on three criteria (bandwidth, accuracy, and precision) to regulate the generated traffic to adhere to specific characteristics. These criteria are briefly defined as the following:

1) **Bandwidth**: Indicates the maximum transmission capacity achieved during the generation process, i.e. the speed of packet transmission measured in packets per second.
2) **Accuracy**: Refers to systematic errors, representing statistical bias, and gauges the proximity of the observed average rate to the set rate.
3) **Precision**: Refers to random errors, which measure statistical variability, and determines the deviation of individual inter-packet gaps from the established value.

Molnár et al. [17] study devised a classification system for the measurements employed to validate network traffic sources. The four categories include:

1) **Packet-Level Metrics:** These metrics assess individual packet characteristics, such as packet size, packet inter-arrival time, and packet loss rate.
2) **Flow-Level Metrics:** These metrics evaluate flow characteristics, including flow size, flow duration, and flow inter-arrival time.
3) **Scaling Characteristics:** These metrics gauge a traffic generator's ability to scale up or down concerning the number of flows, the number of packets, or the packet generation rate.
4) **QoS/QoE Related Metrics:** These metrics determine a traffic generator's quality of service (QoS) or quality of experience (QoE), considering factors like packet delay, jitter, and throughput.

Ring et al. [41] use several approaches to evaluate the quality of their generated datasets, including:

- Visualizing attributes of both the generated data and real data and comparing them.
- Calculating Euclidean distances between generated and real flow-based network data attributes to evaluate the diversity and distribution of the generated data.
- Designing domain knowledge checks, a novel method to evaluate the quality of the content and relationships between attributes within a flow. This is done through automated test procedures on the basis of experts' domain knowledge.

Cheng [39] introduces the following two novel metrics for evaluating the data generated by their GAN model, as traditional GAN evaluation schemes can not be applied to synthetic network data:

- **Success Rate:** The success rate of a traffic generator is defined as the ratio of the number of packets successfully sent to the total number of packets generated by the generator. When a packet is sent on the Internet and an acceptable network reply is sent back, such as a ping response, a DNS query answer, or the HTTP server's webpage, then it is safe to assume a packet has been correctly generated. In the context of synthetic network data generation, the success rate is one of the most important evaluation measurements. Generators that fail to generate data that can be transmitted over the network are not practical and hardly have any utility.
- **Byte Error:** This metric measures how erroneous a packet is when compared to a properly formed packet by averaging the number of byte errors in every packet. In other words, this metric checks whether or not packet headers are according to well-defined network standards.

III. METHODOLOGY

The problem statement and the proposed research questions outline the main purpose of this work: To find out a suitable approach for generating synthetic network traffic data to be used for various tasks such as training ML-based intrusion detection/prevention systems. In this section, details about different methods and techniques utilized to achieve this end will be discussed. We start by defining our novel synthetic network traffic generation framework and then delve into the details of PAC-GPT, our proposed transformer-based packet generator, and discuss how it was trained. After that, the methods and metrics used in order to evaluate the quality of generated data will be examined. Lastly, a CLI tool is introduced that is based on the mentioned models and can be utilized for end-to-end network traffic data generation.

A. NETWORK TRAFFIC GENERATION

In this subsection, we will delve deeper into the functioning of our traffic generation method. We initially lay down the foundation with the description of our framework for network traffic generation. This is succeeded by the introduction of our novel packet generator, termed as PAC-GPT, which is fundamentally based on the GPT-3 model.

1) SYNTHETIC NETWORK TRAFFIC GENERATION FRAMEWORK

In response to the objectives and research inquiries articulated in section I, as well as the definitions outlined in section II, the problem of network traffic generation is addressed at both
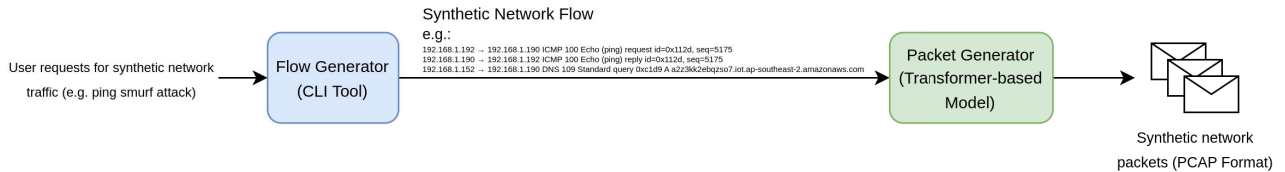
FIGURE 1. An overview of synthetic network traffic generation process.

the flow and packet levels. The main pipeline of this process is as the following:

1) User requests network traffic either using a specific scenario (e.g., normal traffic data, a Ping-of-Death attack, etc.) or by specifying a set of network protocols (e.g., ICMP, DNS, etc.)

2) A *Flow Generator*, which is a Python script complete with a Command Line Interface in our case, produces network flow in text format. That is, it generates the sequence and textual summary of packets that make up the network traffic, although not creating the actual packets themselves.

3) Lastly, the flow summary is passed to the *Packet Generator*, which is a transformer-based model capable of creating packets using the network flow. The packet generator then creates the specified packets and writes them in PCAP format to a file.

Figure 1 outlines this process thoroughly. The upcoming subsection will delve into the specifics of the packet generator, while Section III-C will elucidate the workings of the Flow Generator.

2) PAC-GPT

As previously discussed, transformers are one of the more recent architectures in the generative modeling field that have received a great deal of publicity thanks to the success of models such as GPT-3. While most of their applications remain in the text generation domain like chatbots, they can be utilized for other purposes including synthetic network traffic generation. Some of these approaches were mentioned in section II but here a novel method of using transformers for generating network packets is explained.

The transformer-based model is based on GPT-3 [58] introduced by OpenAI. GPT-3 is a state-of-the-art language model pre-trained on a massive corpus of text data, learning language patterns, grammar, syntax, semantics, and even some factual knowledge during this unsupervised pre-training phase. It has gained significant attention due to its performance on zero-shot or few-shot learning tasks, where the model can quickly adapt to new tasks without extensive fine-tuning on task-specific labeled data. This has made GPT-3 a versatile language model capable of addressing a wide array of challenges including network packet generation. Its flexibility to adapt to solving new tasks (i.e., few-shot learning) is the main reason it was

chosen for this research but the model has the capability to be fine-tuned on specific tasks as well, which was also taken advantage of as part of our work. This fine-tuning is facilitated through OpenAI API,[2] which gives access to a flavor GPT-3 models each with different parameter sizes, speeds, and performances.

The pipeline for constructing the transformer-based generator is illustrated in Figure 2 and its details are as follows:

1) Network packets (i.e., training data) are converted into a text representation. For this purpose, *tcpdump*[3]— a powerful command-line packet analyzer— was used to extract the summary of each packet in the dataset. Some data pre-processing and cleaning were also carried out, with the final output of this step shown in Figure 3. For this part, a total number of 10000 sample packets were randomly selected from ToN IoT Dataset [27], [59], [60], [61], [62], [63], [64], equally consisting of ICMP and DNS packets.

2) Extracted packet summaries are first fed to GPT-3 *"DaVinci"* model, the largest and most capable variation offered by OpenAI API at the time this work was carried out. DaVinci is then used to generate Python code for creating the packets given their text summary, through a technique known as *prompt engineering*. Prompt engineering is a method used to guide the language models to generate desired outputs for specific tasks and is used here to enhance the few-shot learning process. An example of this process can be seen in Figure 4. As a result of this step, 800 training samples are acquired, each consisting of a packet summary and the equivalent code to generate that packet in Python.

3) The training samples compiled in the previous step are in turn now used to fine-tune *"Babbage"*, another variation of GPT-3 which is much smaller and consequently faster and cheaper compared to DaVinci. Utilizing this fine-tuned model allows us to skip many tokens required by DaVinci as prompt engineering is no longer needed, and therefore generate packets more efficiently and inexpensively. Figure 5 demonstrates an input/output pair of Babbage. The model was fine-tuned on the 800 samples generated in the last step using OpenAI APIs.
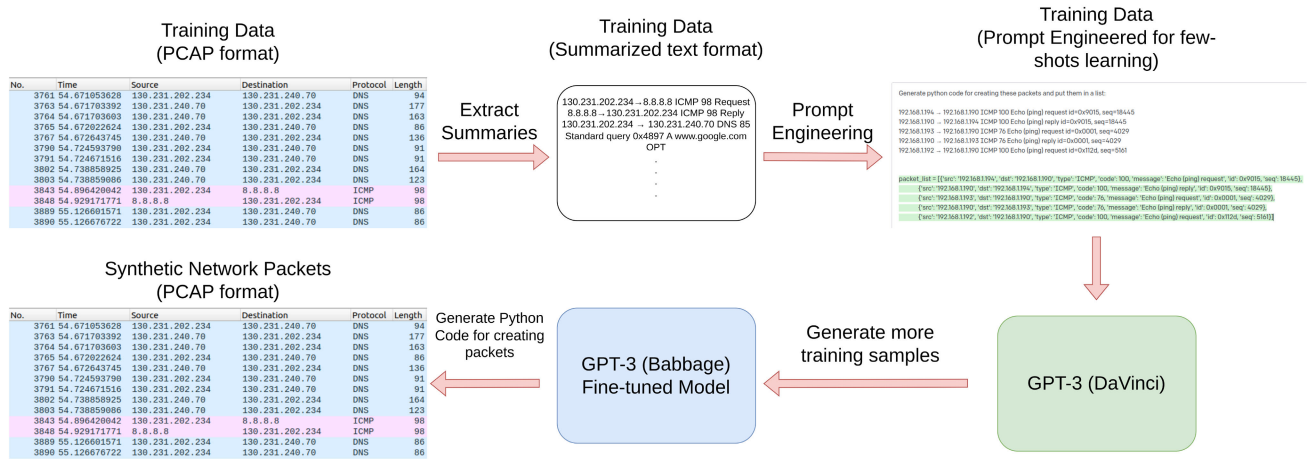
---

[2]https://platform.openai.com/docs/api-reference
[3]https://www.tcpdump.org/

FIGURE 2. Packet generation process using GPT-3.

FIGURE 3. Sample output of packet summarization step after pre-processing.

FIGURE 4. Sample input and output of DaVinci model while used for generating more training data. The first part is the prompt/input, while the second highlighted part is generated text/output.

FIGURE 5. Sample input and output of Babbage fine-tuned model while used for generating the packets. The top part is the input without any prompts, whereas the lower part is generated code capable of creating the packets with Python and Scapy.

4) Finally, the code generated by the fine-tuned Babbage model is executed to create the concrete packets using Python and Scapy.[4] Scapy, a Python-written framework for packet manipulation, is extensively used in networking scripts. In our case, it is employed not only for packet creation, but also for crucial post-processing actions such as checksum correction, saving the packets to a file, and, if needed, replaying them on the network.
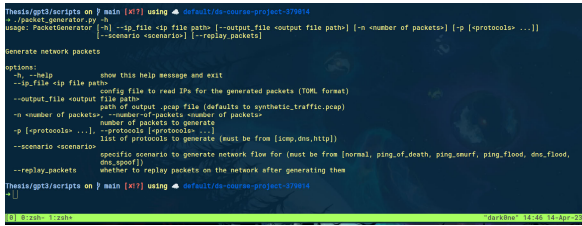
B. EVALUATION METRICS
A number of methods and metrics were used for evaluating the models, which can be divided into two groups:

[4]https://scapy.readthedocs.io/en/latest/api/scapy.html

- **Intrinsic metrics:** As the core of our proposed generators are deep learning models, they come with a set of metrics to evaluate the performance of the model and sometimes even control the learning process. These metrics include training loss, training sequence accuracy, and training token accuracy.
- **Extrinsic metrics:** Since the ultimate goal of a packet generator is to create authentic high-quality packets, it is vital to also assess whether or not the generated packets are legitimate network packets resembling real data. Intrinsic metrics fail to indicate this as they are only designed to reflect the performance of underlying machine learning models. Therefore, a method similar to [39] was used in order to evaluate the quality of the generated packets. This metric is *"success rate"*, which as mentioned before is the ratio of packets that are sent and have received a response in return proportionally to the total number of packets generated by the models. In other words, to measure the quality of generated packets we replay them in a live network and observe how many of them are transmitted correctly and receive a proper response in return. The valid response would be a ping reply received from destination of ping request, and a hostname resolved for a DNS query. This way, we can be sure that generated packets are of the same quality as authentic ones.

FIGURE 6. An overview of CLI tool for generating network traffic.

C. CLI TOOL

A network flow is a series of packets that comprise the network traffic. The described packet generators only generate individual, separate packets and are unable to create this flow on their own. Therefore, a Python script was implemented to generate this network flow required by the packet generators. The script is developed as a CLI tool and is not only capable of generating the flow but can also handle necessary calls to packet generator models and is essentially a wrapper around them. In addition, it offers some utility features such as writing generated packets to PCAP file format and transmitting the generated packets using a network interface. Figure 6 is a screenshot of this CLI tool and some of its options, which are explained below:

- *ip_file:* Path to a configuration file utilized by the flow generator while setting sender and receiver IP fields in a packet. The IPs can either be specified individually or by a subnet. The file can also include the IPs of attackers and victims in case a specific malicious scenario is used.
- *output_file:* Path to save the output PCAP file containing generated synthetic packets.
- *n:* Number of packets to generate. This number is used as an approximate, as the flow generator takes into account also other factors such as the scenario and also because some of the packets created by the packet generator are erroneous and need to be dismissed.
- *protocols:* Network protocols which packets should be generated for.
- *scenario:* Scenario which packets should be generated for. Options include Normal network traffic, ping smurf, ping flood, and DNS flooding.
- *replay_packets:* Boolean option to indicate whether generated packets should also be replayed on the network or not.

As described above, the CLI tool has a number of options for creating the flow. Each of these *"scenarios"* can be briefly defined as the following:

- **Normal:** Indicates a normal flow of traffic in the network with no particular cyber attack present. Specifically, in this mode flow is generated randomly between all the nodes in the network based on indicated protocols.
- **Ping of Death Attack:** A *Ping of Death* is a form of cyber attack that exploits a specific characteristic of the Internet Control Message Protocol (ICMP), which is typically used to send error messages across network

devices and to help troubleshoot network connectivity issues. ICMP operates at the Network layer of the Open Systems Interconnection (OSI) model, where IP packets are processed. The ICMP protocol includes a request-response mechanism used for diagnostic purposes, commonly known as the 'ping' operation. Normally, the sender transmits an echo request to a target, which in turn sends an echo reply back. This operation is beneficial in ensuring that a network host is reachable and in determining round-trip packet times. A Ping of Death attack manipulates these innocuous operations by sending IP packets that exceed the maximum allowable size of 65,535 bytes. This is accomplished by fragmenting the large packet into smaller segments that individually comply with the Internet Protocol's maximum transmission unit (MTU) but exceed it when reassembled at the target machine. When a system receives such an oversized packet, it may result in buffer overflow conditions. Depending on the system's vulnerability to such conditions and how it handles them, this can cause system crashes or induce undefined behavior, thereby achieving the objective of the DoS (Denial of Service) attack. The way this scenario is handled in our CLI tool is that in addition to a number of normal traffic packets generated for the whole network (similar to the normal scenario), there are also some malicious packets created with the discussed specifications originating from a specified attacker node in the network targeted at a victim node in the network.

- **Ping Flood:** In a standard operation ping packets are used to check the availability of a network device by sending an ICMP Echo Request and waiting for its ICMP Echo Reply. However, in a *Ping Flood* attack, the attacker sends an overwhelming number of these packets to a target device without waiting for the ICMP Echo Reply. This flood of incoming packets saturates the target's network bandwidth and consumes its resources to process the incoming traffic, potentially leading to service disruption. For this scenario, the flow was generated in such a way that in addition to normal network traffic, there are also bursts of ICMP Echo Requests sent from the attacker's IP to the victim's IP.
- **Ping Smurf:** Similar to ping flood attack, *Ping Smurf* attack attempts to overload the victim's system through a large number of ICMP packets, though it achieves through a different method. In a Smurf attack, the attacker uses a forged ICMP Echo Request packet with a spoofed source IP address - typically, the IP address of the intended victim. This packet is sent to an IP broadcast network. An IP broadcast network consists of a range of IP addresses wherein any packet sent to the broadcast address is automatically forwarded to all hosts within that network. The hosts, behaving as expected, respond to the Echo Request by sending an ICMP Echo Reply. However, since the source IP address in the original ICMP packet has been spoofed to that

of the victim, all replies are sent to the victim's IP address. This can result in an overwhelming flood of traffic that consumes network bandwidth and potentially disrupts the target's services. This attack is imitated in the CLI flow generator by generating a large number of ICMP Echo Request packets with the source IP address spoofed to the victim's, and the target address set to nodes in the network. Additionally, ICMP Echo Reply packets are generated in response to the requests.

- **DNS Flood:** Domain Name System (DNS) servers are integral components of the Internet's infrastructure, responsible for translating human-readable domain names into machine-readable IP addresses. In a DNS Flood attack, the attacker aims to overwhelm a DNS server with a barrage of requests, with the intention of exhausting the server's resources, such as processing capacity and network bandwidth. In executing a DNS Flood attack, the attacker typically uses multiple devices to send a high volume of DNS request packets to the target server. These requests can be legitimate or malformed; in either case, the goal is to inundate the server and consume its resources. This flood of requests leads to slower responses to legitimate requests, or in severe cases, the DNS server can become entirely unresponsive, disrupting the target network's access to the Internet. This type of attack is simulated through the flow generator by making a great number of DNS query requests from the attacker's nodes to the victim's system in addition to the normal traffic data.

IV. RESULTS

In the previous section, a proposed framework for generating synthetic network traffic data was discussed, following with an examination on how different architectures can be utilized for the task at hand. Also, some of the details of the training processes were mentioned. In this part, we will go over the results of these experiments in terms of the defined evaluation metrics. The section starts off with the intrinsic metrics of each model and then we proceed to compare all of them using the extrinsic metrics.

A. INTRINSIC METRICS

As defined before, intrinsic metrics refer to those that are defined and employed by the underlying machine learning models used in this work. One of the most common evaluation metrics in machine learning is the loss function value, which is primarily used to optimize the models but it can also be seen as an evaluation metric to gauge models' training progress. In addition to loss value, transformer-based models also have other metrics that are discussed in the following.

For this part of our experiments, we fine-tuned three variations of GPT-3 using the OpenAI API: DaVinci, Curie, and Babbage. Among these, DaVinci represents the largest model, while Babbage is the smallest and fastest. Curie occupies a middle ground between these two extremes.
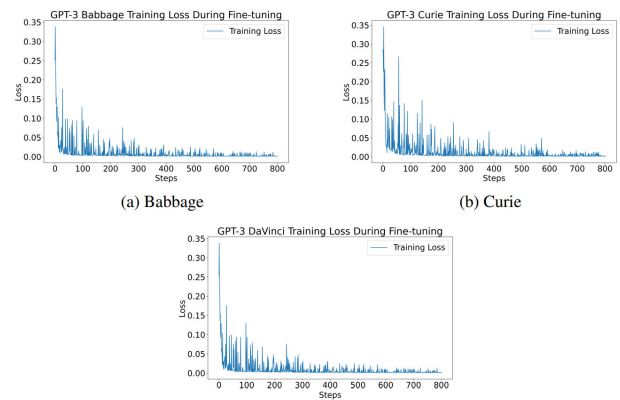
FIGURE 7. Loss values for GPT-3 models during fine-tuning process.
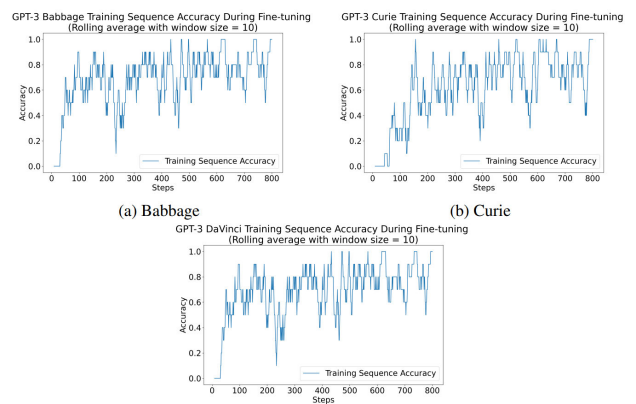
FIGURE 8. Sequence accuracy for GPT-3 models during fine-tuning process.

Figure 7 shows the loss function value for these models over the course of their training.

Other than the loss function value, OpenAI API [65] provides two other evaluation metrics, namely Training Sequence Accuracy and Training Token Accuracy, which are defined below:

- **Training Sequence Accuracy:** The percentage of completions in the training batch for which the model's predicted tokens matched the true completion tokens exactly for all tokens. For instance, if the batch size is 3 and the completions are ([1, 2], [0, 5], [4, 2]), while the model's predictions are ([1, 1], [0, 5], [4, 2]), the accuracy would be calculated as 2/3, resulting in 0.67.
- **Training Token Accuracy:** The percentage of tokens in the training batch that were correctly predicted by the model. For example, with a batch size of 3, if the data contains the completions ([1, 2], [0, 5], [4, 2]) and the model predicted ([1, 1], [0, 5], [4, 2]), this accuracy will be 5/6 = 0.83.

Figures 8 and 9 show training sequence accuracy and training token accuracy respectively for all GPT-3 fine-tuned models.

B. EXTRINSIC METRICS

An additional performance metric was used to evaluate the proposed packet generation methods from an end-to-end
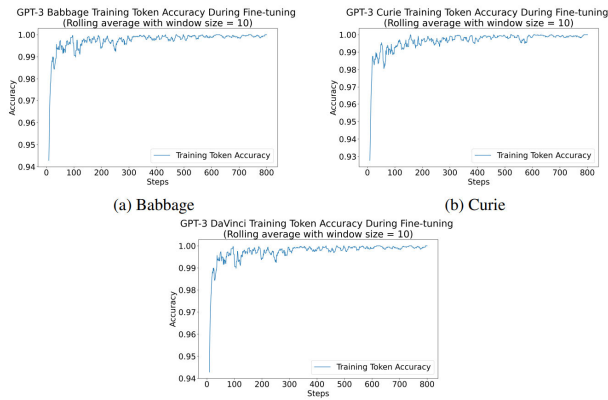
FIGURE 9. Token accuracy for GPT-3 models during fine-tuning process.

TABLE 1. List of Ping IPs and DNS Hostnames used for evaluation.

Ping IP (Organization)	DNS Hostname
8.8.8.8 (Google)	www.google.com
208.67.222.222 (OpenDNS)	www.opendns.com
9.9.9.9 (Quad9)	www.facebook.com
176.103.130.130 (AdGuard)	www.atlassian.com
58.96.3.34 (Exetel)	www.bing.com

perspective. This extrinsic metric is the success rate, which was defined earlier. Here, the results of all models are presented on this metric.

Models were trained with three sets of data based on the network packet types: ICMP packets which were ping requests and replies, DNS packets which included DNS query requests and responses, and ICMP/DNS which is the combination of both. Each dataset consisted of 200 pairs of prompt-completion, with each pair containing five packet summary-code rows, resulting in 1000 training samples. All previously discussed models were trained on each dataset, resulting in three GPT-3 fine-tuned variations. Additionally, a *pre-trained DaVinci* model was also included in these experiments. This pre-trained model is the base model offered by OpenAI and unlike the other GPT-3 variations experimented with in our work, does not benefit from any fine-tuning which means it was necessary to use prompt engineering methods mentioned in section III-A2 to have it generate the specified packets.

For the experiments, each model had to generate 100 packets corresponding to the type of data they were trained on (i.e., if a model was trained on only ICMP packets, it generated only ICMP packets during evaluation). Furthermore, to reduce the effect of random noise, each experiment was run 5 times, and the best and worst-case results were included in the following. To effectively test the capabilities of the packet generation models, the models generated packets to and from a list of public IPs and hostnames (for ICMP and DNS packets, respectively) on the internet. The list of these IPs and hostnames is indicated in Table 1.

Table 2 reports the success rate metric for models on each dataset while generating normal traffic flow. The lower

TABLE 2. Success Rate for packet generation models for normal network flow.

Model	ICMP	DNS	ICMP/DNS
GPT-3 *"Pre-trained DaVinci"*	90-100%	3-10%	48-61%
GPT-3 *"Fine-tuned Babbage"*	75-100%	0-7%	41-58%
GPT-3 *"Fine-tuned Curie"*	0-23%	0-2%	1-16%
GPT-3 *"Fine-tuned DaVinci"*	91-100%	5-8%	46-59%

TABLE 3. Success Rate for packet generation models for malicious network flow (Ping flood scenario).

Model	ICMP	DNS	ICMP/DNS
GPT-3 *"Pre-trained DaVinci"*	87-100%	N/A[1]	44-66%
GPT-3 *"Fine-tuned Babbage"*	80-100%	N/A[1]	39-60%
GPT-3 *"Fine-tuned Curie"*	4-25%	N/A[1]	0-23%
GPT-3 *"Fine-tuned DaVinci"*	95-100%	N/A[1]	44-57%

number is the minimum value of the metric for five tries, and the higher number is the maximum value for the experiments. Similarly, Table 3 reports this metric while generating packets for a malicious flow scenario, specifically, a ping flood attack.

V. DISCUSSION
The key outcomes of our experimental work using diverse metrics were laid out in the preceding section. In this segment, we will scrutinize these results to address our research questions, along with an analysis of the implications and limitations associated with our research.

A. INTERPRETING THE RESULTS
We begin by addressing the first proposed research question: How do the implemented generative models perform when tasked with network packet generation? To gauge this, we will examine the intrinsic metrics of each model as indicators of their training efficiency. Subsequently, these models will be compared from an end-to-end perspective using an extrinsic metric.

Our GPT-based models seem to be performing somewhat similarly to one another, with some minor differences in their training process. Figure 7 shows each model's loss during training, and as can be seen, all models eventually reach stability. However, an important observation here is that, in terms of the loss values, Curie seems to face a shallower initial learning curve compared to Babbage and DaVinci. This pattern is also evident in Figure 8, reinforcing that Curie's training progression is relatively slower than the other two models. Babbage, although needing more training time than DaVinci, is in line with our expectations considering DaVinci's larger capacity among the three models. Lastly, Figure 9 reports token accuracy, and accordingly DaVinci is slightly outperforming the other two again with Curie this time having an easier time getting fine-tuned compared to Babbage. Mostly, these results align with our expectations, with the notable exception of Curie's performance being inferior to Babbage in some instances. This is somewhat unexpected, considering that Curie is theoretically a larger

TABLE 4. Summary of core papers for synthetic network traffic generation using generative modeling.

Citation	Data Source	Method	Evaluation
PAC-GAN [39]	Real network traffic captured using tcpdump	Use CNN GANs to generate network packets, using a novel encoding scheme for representing packets as images.	Evaluated using two metrics, success rate and packet byte error, which measure the quality of individual packets
Pcap-GAN [40]	MACCDC 2012	Use CNN GANs to generate network capture files using an styled encoding scheme	Trained IDS to classify generated and original data. Also visualized generated data and compare features with original data.
Flow-GAN [41]	CIDDS-001 [66]	Use GANs to capture and generate flow data (i.e, features about the network flow) using 3 different encoding schemes	Evaluate using a combination of visualizations, euclidean distance between distributions and custom-defined domain knowledge checks to verify their work
CopulaGAN, CTGAN, and Table-GAN [21]	NSL-KDD [67]	Authors utilize some variations of GAN to generate network data to train different IDS	Evaluate using a combination of distribution visualizations, statistical metrics between distributions and also a number of custom-defined machine learning-based metrics
Generative Transformer... [49]	Combination of collected packet on a local network, ISCXIDS2012 dataset [68], and UNSW-NB15 dataset [69]	Use GPT-2 and a tokenization method to both generate and classify network traffic	Statistical metrics for measuring the distance between original and synthetic distributions (Kolmogorov-Smirnov metric), machine learning-based metrics for classification tasks (F1 micro and accuracy)
PAC-GPT (Our work)	TON_IoT dataset [27]	Use GPT-3 and LLM chaining to generate both the network packets and the network flow	Evaluated using a set of intrinsic metrics (i.e, loss and accuracy) and extrinsic metrics (i.e, success rate).

and thus more potent model than Babbage. Furthermore, both models were trained on identical data for an equivalent duration, making this discrepancy quite unusual.

Beyond the intrinsic metrics covered so far, there are also extrinsic metrics that warrant our attention and discussion. Starting with the success rate for normal and malicious network flows (Tables 2 and 3 respectively), the models mostly perform well when faced with the task of ICMP packet generation. Babbage and DaVinci perform adequately when fine-tuned, sometimes producing packets that are 100% transmittable and all receive responses. Impressively, DaVinci manages to achieve similar results even without the fine-tuning process. This model is capable of generating equally high-quality packets solely through pre-training and prompt-engineering techniques, albeit at a marginally higher cost per packet, as previously explained. This also holds true for experiments involving malicious scenarios, such as a ping flood attack. This is the anticipated outcome given that the malicious scenario is managed by the flow generation component, and the packet generation models operate entirely independent of the flow generator module.

Unfortunately, the same level of performance does not translate to the DNS packet generation process. When tasked with creating code for generating DNS packets, all models significantly underperform, with the Pre-trained DaVinci managing the best outcome, but still at a rather low success rate of only 10%. From our observations, this underperformance appears to be linked to Scapy, the underlying Python framework that takes care of packet generation post the model's code generation. The syntax

Scapy uses to handle DNS packet creation is considerably more extensive (and more complex) compared to that for ICMP packets. This is reasonable considering that DNS is a more intricate network protocol than ICMP.

Finally, the integration of the two protocols appears to yield slightly better results, with models receiving responses for nearly half of the packets in general. However, this doesn't necessarily imply that the models have improved their understanding of DNS header fields. As half of the dataset in the final experiment (i.e., the ICMP/DNS dataset) consists of ICMP packets, it's possible that the models are simply more proficient at generating ICMP packets.

We have also provided a brief summary of our model and other related works in Table 4. It should be noted though that Table 4 is not meant as a comparison of PAC-GPT with these other models since measuring and comparing the performance of synthetic network traffic generators is not an straightforward task and is still an open area of research in the field. Instead, Table 4 is intended to give the reader an overview of existing works related to PAC-GPT, their methodology and their evaluation strategies.

B. IMPLICATIONS

So far, we have analyzed the experimental results from a technical standpoint to address our first research question. We will now turn our attention to the remaining two research questions, exploring them from a more interpretive perspective. Specifically, this section will delve into how the results obtained can be integrated into existing cybersecurity

frameworks and the potential applications they may have within the cybersecurity domain.

Firstly, the principal utility of the tool, particularly the packet generator, is the generation of synthetic network traffic. This synthetic traffic can be used for testing existing network architectures and security systems. For instance, a red team could use the model to generate a large number of packets to evaluate an IDS running on the network or to stress-test certain parts of the network using specific packets. Many tools exist to evaluate a network from different aspects, ranging from performance tests (throughput, latency, packet loss, etc.) and network traffic analysis to penetration tests and vulnerability scanning. The way most of these tools work is by generating specific types of packets either through preset scripts or built-in network functions in the system; but as an alternative, the mentioned traffic generation tool can be used instead to generate the necessary packets for this purpose.

Additionally, the developed CLI tool can help a security architect/engineer develop custom-fit security measures by generating customized network traffic. This is especially helpful when implementing a machine-learning-based IDS/IPS (Intrusion Detection System or Intrusion Prevention System), as these systems require a large amount of training data to be developed and the CLI tool can easily provide users with an arbitrary amount of network data. The implemented *"malicious scenarios"* were specifically designed for this purpose, as training an IDS/IPS requires both positive and negative sample data (i.e. normal and malicious traffic). In the sense of NIST threat framework [70], this puts the tool at the detection layer where the purpose is to detect anomalies in the network, where usually an IDS resides to monitor the network for malicious activities. Additionally, it could also be useful in the protection layer, where an IPS could be trained not only to detect malicious activity but also takes proactive measures to thwart any such attempt.

Lastly, another potential use case for the tool is training and educating security personnel in the cybersecurity domain. It enables security professionals and students to gain hands-on experience in analyzing, detecting, and responding to different types of network traffic and security incidents, without exposing real systems to risks. Training personnel is always a costly endeavour for companies not only because of the required time and money needed for instructing new recruits but also due to the fact that allowing untrained personnel to probe around production environments and infrastructure has the potential to result in major disruptions for the company infrastructure. This challenge is further complicated by the fact that the data traversing through a company's network is often highly sensitive and confidential. Exposing such information to new personnel poses an unacceptable risk for most organizations. Conversely, the availability of synthetic data circumvents these concerns. The risk of personnel inadvertently (or perhaps even intentionally) meddling with the existing infrastructure is eliminated, and the potential for private data breaches becomes insignificant when synthetic network traffic is utilized instead.

It should also be mentioned that just like any other cybersecurity tool set, there are some possible *"misuse"* cases in addition to the mentioned potential use cases. In the simplest case, a malicious user could use the CLI tool to generate a massive amount of traffic to transmit on a network. This would render some parts (or even the whole) network inoperable by overloading bottleneck if necessary precautions are not put into place.

Another, more sophisticated malicious use case of the CLI tool is to reverse engineer AI-based IDS/IPS. Very much like a security analyst, an attacker could also develop their own ML-based IDS/IPS using the synthetic data to test. Then, they could try to find vulnerabilities to circumvent these systems in a production environment. They could also gain a great deal of insight into the strengths and weaknesses of these systems as they can explore and analyze the data they are trained with.

C. LIMITATIONS

Thus far only the results and implications of our research have been discussed, but as with any work, there are also a number of limitations and constraints on what has been achieved, especially in practice. Some of these will now be discussed.

The first limitation of the paper is that only 2 basic network protocols, namely ICMP and DNS, were considered. This was done due to the fact that these protocols are easier to generate as they do not have as many header fields as other TCP or User Datagram Protocol (UDP) sub-protocols. But on the other hand, this severely limits the usefulness of the tool as most network traffic consists of protocols in addition to the two considered in our work. Future endeavors could try to improve on this by including protocols other than ICMP and DNS.

The next limitation is the amount of training data and training duration for the models used. The GPT-3 models were only trained on 200 pairs of prompt-completions, which is barely more than the minimum recommended number of training samples indicated by OpenAI. Most of the low results and achieved performance could also be attributed to this fault in the authors' opinion, as like any other machine learning task more data could benefit some (if not all) of the models. Currently, this high cost severely limits the potential of the tool in practical settings.

Another constraint is the number of scenarios defined using the CLI tool. Those implemented so far were only done so to indicate the potential use cases of such flow generators but by no means is an exhaustive list of malicious network scenarios that can be imitated by an AI-based flow generator. In fact, the malicious scenarios we have considered here are of the most basic ones and hardly ever considered a threat any more. For the tool to be potentially useful, future works may have heavily expand upon malicious scenarios.

There is also the matter of evaluation of course. A number of methods and metrics (both intrinsic and extrinsic) were used to measure the performance and potential utility of the models and tools implemented; but even so, there is no consensus among the academia at the time of writing as to

how to evaluate synthetic network data, with every other work proposing a novel approach or metric to measure this performance. Therefore, the ultimate practicality of the work remains to be determined by the experts in the field at a later time.

Lastly, it must be noted that accurately reproducing certain aspects of the experiments we conducted may prove to be extremely challenging or even unfeasible due to the intricate nature of Large Language Models (LLMs) and the inherent randomness in their text generation capabilities. For example, as previously mentioned, we employed DaVinci to generate some training samples, utilizing prompt-engineering techniques for subsequent stages. Achieving identical training samples using the same prompt we utilized would be highly improbable. This variance would significantly impact subsequent phases of our proposed framework.

VI. CONCLUSION

A. SUMMARY

This study explored synthetic network traffic generation and proposed a framework for end-to-end traffic generation using LLMs. Additionally, several methods capable of achieving this were implemented within the proposed framework. To summarize the results of our research, the following outcomes were derived:

- A number of transformer-based models were trained and tested to generate network packets for ICMP and DNS protocols, and their performance was compared using different evaluation metrics. The main conclusion was that transformers perform adequately well when generating simple network packets such as ICMP protocol packets even without fine-tuning, but struggle to do the same for more complex protocols like DNS.
- A new end-to-end network traffic generator framework was proposed and implemented. The framework consists of two components: a packet generator and a flow generator. This compartmentalization was done not only to help with the technical aspects of implementing the research but also to make it possible to individually exchange and improve each of these tasks in the larger scheme of generating synthetic network data.
- A CLI tool was implemented using Python libraries to take care of the flow generation part in the framework and to facilitate the packet generation pipeline for end users. Furthermore, several malicious scenarios were included in the CLI tool that provides alternatives in addition to the normal traffic generation.

B. FUTURE WORK

Finally, in this section, we will have a brief look at the possible directions for the research that can be expanded in the future.

To begin with, one of the main areas the whole data generation process can be improved in is the network protocols supported. As mentioned in section V-C, only two

protocols ICMP and DNS were focused on for their ease of implementation, and this heavily limits the usefulness of the tool for many practical purposes. ICMP and DNS packets only make up a very low percentage of network traffic in most scenarios. Adding support for TCP/UDP protocols could drastically improve the diversity of the data generated and consequently affect the utility of the synthetic data especially for production environments. This will not be an easy task to achieve though, as more sophisticated protocols may pose new complications that our proposed models might not be capable of handling for now.

Just like the packet generator, the flow generator's functionality could also be extended by incorporating more scenarios. As of now, only a handful of straightforward 'attack scenarios' have been considered, but the inclusion of additional, more complex scenarios could further enhance the diversity and quality of the generated data, especially if more network protocols are supported by the packet generators.

Another direction that future research might focus on is the flow generation component of the data generation pipeline. The CLI tool takes care of this part for now by generating packet summaries through a set of customizable scripts. However, these scripts are fairly static (in other words, hard-coded) and hardly benefit from any AI-based logic. By replacing this module with a smarter, more dynamic flow generator that takes advantage of the recent advancements in the field of text generation, the synthetic data could incorporate a more meaningful flow of packets that closely resembles authentic network data. Specifically, the same transformer-based LLMs that were used in our work can be utilized to generate this flow in addition to the packet generation task they have been performing so far in this research. This could result in a hierarchical architecture of LLMs to facilitate synthetic network traffic generation.

Lastly, an area that could greatly benefit from further research is the validation/evaluation of synthetic network generation. Synthetic network data suffers heavily from a lack of consensus on an evaluation method acceptable among the community. A couple of techniques were utilized to broadly measure the performance of the models and methods from different aspects, but there is still room for improvement in this regard.

REFERENCES

[1] R. Canzanese, M. Kam, and S. Mancoridis, "Toward an automatic, online behavioral malware classification system," in *Proc. IEEE 7th Int. Conf. Self-Adaptive Self-Organizing Syst.*, Sep. 2013, pp. 111–120.

[2] S. Sheen, R. Anitha, and V. Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," *Neurocomputing*, vol. 151, pp. 905–912, Mar. 2015.

[3] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: High-fidelity, behavior-based automated malware analysis and classification," *Comput. Secur.*, vol. 52, pp. 251–266, Jul. 2015.

[4] L. Zhu, A. Sun, and B. Choi, "Detecting spam blogs from blog search results," *Inf. Process. Manage.*, vol. 47, no. 2, pp. 246–262, Mar. 2011.

[5] M. Luckner, M. Gad, and P. Sobkowiak, "Stable web spam detection using features based on lexical items," *Comput. Secur.*, vol. 46, pp. 79–93, Oct. 2014.

[6] V. M. Prieto, M. Álvarez, and F. Cacheda, "SAAD, a content based web spam analyzer and detector," *J. Syst. Softw.*, vol. 86, no. 11, pp. 2906–2918, Nov. 2013.

[7] X. Gu, H. Wang, and T. Ni, "An efficient approach to detecting phishing web," *J. Comput. Inf. Syst.*, vol. 9, pp. 5553–5560, Jul. 2013.

[8] J. Cao, D. Dong, B. Mao, and T. Wang, "Phishing detection method based on URL features," *J. Southeast Univ.-Engl. Ed.*, vol. 29, no. 2, pp. 134–138, Jun. 2013.

[9] O. A. Adeleke, N. Bastin, and D. Gurkan, "Network traffic generation: A survey and methodology," *ACM Comput. Surveys*, vol. 55, no. 2, pp. 1–23, Jan. 2022.

[10] A. Majeed and S. Lee, "Anonymization techniques for privacy preserving data publishing: A comprehensive survey," *IEEE Access*, vol. 9, pp. 8512–8545, 2021.

[11] D. K. Kholgh, "Synthetic network traffic generation using generative modeling," M.S. thesis, Univ. Oulu, Oulu, Finland, 2023.

[12] D. C. Sicker, P. Ohm, and D. Grunwald, "Legal issues surrounding monitoring during network research," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas.*, Oct. 2007, pp. 141–148.

[13] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. 6th Int. Conf.*, Nov. 2010, pp. 1–12.

[14] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015.

[15] V. Reddy, M. Safwan, D. Nn, G. Shobha, and S. Premkumar, "Network traffic simulator from real time captured packets," *Int. J. Appl. Eng. Res.*, vol. 12, pp. 10134–10137, Jan. 2017.

[16] S. Kolahi, S. Narayan, D. Nguyen, and Y. Sunarto, "Performance monitoring of various network traffic generators," in *Proc. UkSim 13th Int. Conf. Comput. Modelling Simulation*, Mar. 2011, pp. 501–506.

[17] S. Molnár, P. Megyesi, and G. Szabó, "How to validate traffic generators?" in *Proc. IEEE Int. Conf. Commun. Workshops (ICC)*, Jun. 2013, pp. 1340–1344.

[18] S. Mishra, S. Sonavane, and A. Gupta, "Study of traffic generation tools," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 4, no. 6, pp. 4–7, 2015.

[19] P. Emmerich, S. Gallenmüller, G. Antichi, A. W. Moore, and G. Carle, "Mind the gap—A comparison of software packet generators," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, May 2017, pp. 191–203.

[20] C. Wu, Y. Chen, P. Chou, and C. Wang, "Synthetic traffic generation with Wasserstein generative adversarial networks," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, Dec. 2022, pp. 1503–1508.

[21] S. Bourou, A. El Saer, T.-H. Velivassaki, A. Voulkidis, and T. Zahariadis, "A review of tabular data synthesis using GANs on an IDS dataset," *Information*, vol. 12, no. 9, p. 375, Sep. 2021.

[22] T. J. Anande and M. S. Leeson, "Generative adversarial networks (GANs): A survey of network traffic generation," *Int. J. Mach. Learn. Comput.*, vol. 12, pp. 333–343, Nov. 2022.

[23] C. Thomas, V. Sharma, and N. Balakrishnan, "Usefulness of DARPA dataset for intrusion detection system evaluation," in *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008*, vol. 6973. Bellingham, WA, USA: SPIE, 2008, pp. 164–171.

[24] A. A. Olusola, A. S. Oladele, and D. O. Abosede, "Analysis of KDD'99 intrusion detection dataset for selection of relevance features," in *Proc. World Congr. Eng. Comput. Sci. (WCECS)*, vol. 1. San Francisco, CA, USA, Oct. 2010, pp. 20–22.

[25] M. Katzef, A. C. Cullen, T. Alpcan, C. Leckie, and J. Kopacz, "Wireless network simulation to create machine learning benchmark data," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, Dec. 2022, pp. 6378–6383.

[26] G. Meena and R. R. Choudhary, "A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA," in *Proc. Int. Conf. Comput., Commun. Electron. (Comptelix)*, Jul. 2017, pp. 553–558.

[27] N. Moustafa, "A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets," *Sustain. Cities Soc.*, vol. 72, Sep. 2021, Art. no. 102994.

[28] I. Goodfellow, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[29] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[31] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," 2018, *arXiv:1809.11096*.

[32] Z. Qin, Z. Liu, P. Zhu, and Y. Xue, "A GAN-based image synthesis method for skin lesion classification," *Comput. Methods Programs Biomed.*, vol. 195, Oct. 2020, Art. no. 105568.

[33] X. Zhu, L. Zhang, L. Zhang, X. Liu, Y. Shen, and S. Zhao, "GAN-based image super-resolution with a novel quality loss," *Math. Problems Eng.*, vol. 2020, pp. 1–12, Feb. 2020.

[34] X. Xia, X. Pan, N. Li, X. He, L. Ma, X. Zhang, and N. Ding, "GAN-based anomaly detection: A review," *Neurocomputing*, vol. 493, pp. 497–535, Jul. 2022.

[35] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient GAN-based anomaly detection," 2018, *arXiv:1802.06222*.

[36] Y. Huang, W. Zhang, and Y. Zhou, "TD-GAN for chatbot text generation," in *Proc. CS Secur. Privacy Workshops*, 2017, pp. 1–9.

[37] Z. Yang, W. Chen, F. Wang, and B. Xu, "Unsupervised neural machine translation with weight sharing," 2018, *arXiv:1804.09057*.

[38] N. Dang, A. Khanna, and V. R. Allugunti, "TS-GAN with policy gradient for text summarization," in *Data Analytics Management*. Cham, Switzerland: Springer, 2021, pp. 843–851.

[39] A. Cheng, "PAC-GAN: Packet generation of network traffic using generative adversarial networks," in *Proc. IEEE 10th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Oct. 2019, pp. 728–734.

[40] B. Dowoo, Y. Jung, and C. Choi, "PcapGAN: Packet capture file generator by style-based generative adversarial networks," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 1149–1154.

[41] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Comput. Secur.*, vol. 82, pp. 156–172, May 2019.

[42] M. Ring, A. Dallmann, D. Landes, and A. Hotho, "IP2 Vec: Learning similarities between IP addresses," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2017, pp. 657–666.

[43] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional GAN," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.

[44] S. Kamthe, S. Assefa, and M. Deisenroth, "Copula flows for synthetic data generation," 2021, *arXiv:2101.00598*.

[45] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks," 2018, *arXiv:1806.03384*.

[46] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," 2019, *arXiv:1906.01787*.

[47] U. Naseem, I. Razzak, K. Musial, and M. Imran, "Transformer based deep intelligent contextual embedding for Twitter sentiment analysis," *Future Gener. Comput. Syst.*, vol. 113, pp. 58–69, Dec. 2020.

[48] U. Khandelwal, K. Clark, D. Jurafsky, and L. Kaiser, "Sample efficient text summarization using a single pre-trained transformer," 2019, *arXiv:1905.08836*.

[49] R. F. Bikmukhamedov and A. F. Nadeev, "Generative transformer framework for network traffic generation and classification," *T-Comm*, vol. 14, no. 11, pp. 64–71, 2020.

[50] R. F. Bikmukhamedov and A. F. Nadeev, "Multi-class network traffic generators and classifiers based on neural networks," in *Proc. Syst. Signals Generating Process. Field Board Commun.*, Mar. 2021, pp. 1–7.

[51] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[52] D. Wass, "Transformer learning for traffic prediction in mobile networks," KTH, School Elect. Eng. Comput. Sci. (EECS), TRITA-EECS-EX, Tech. Rep. 2021:644, 2021, p. 47.

[53] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proc. ACM Web Conf.*, Apr. 2022, pp. 633–642.

[54] Z. Wu, H. Zhang, P. Wang, and Z. Sun, "RTIDS: A robust transformer-based approach for intrusion detection system," *IEEE Access*, vol. 10, pp. 64375–64387, 2022.

[55] F. Setianto, E. Tsani, F. Sadiq, G. Domalis, D. Tsakalidis, and P. Kostakos, "GPT-2C: A parser for honeypot logs using large pre-trained language models," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, Nov. 2021, pp. 649–653.

[56] H. Wang and W. Li, "DDosTC: A transformer-based network attack detection hybrid mechanism in SDN," *Sensors*, vol. 21, no. 15, p. 5047, Jul. 2021.

[57] R. Kozik, M. Pawlicki, and M. Choraundefined, "A new method of hybrid time window embedding with transformer-based traffic data classification in IoT-networked environment," *Pattern Anal. Appl.*, vol. 24, no. 4, pp. 1441–1449, Nov. 2021.

[58] T. B. Brown, "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.

[59] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. H. D. Hartog, "ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion data sets," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 485–496, Jan. 2022.

[60] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems," *IEEE Access*, vol. 8, pp. 165130–165150, 2020.

[61] N. Moustafa, M. Keshky, E. Debiez, and H. Janicke, "Federated TON_IoT windows datasets for evaluating AI-based security applications," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 2020, pp. 848–855.

[62] N. Moustafa, M. Ahmed, and S. Ahmed, "Data analytics-enabled intrusion detection: Evaluations of ToN_IoT Linux datasets," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 2020, pp. 727–735.

[63] N. Moustafa, "A systemic IoT–fog–cloud architecture for big-data analytics and cyber security systems: A review of fog computing," in *Secure Edge Computing*. Boca Raton, FL, USA: CRC Press, 2021, pp. 41–50.

[64] J. Ashraf, M. Keshk, N. Moustafa, M. Abdel-Basset, H. Khurshid, A. D. Bakhshi, and R. R. Mostafa, "IoTBoT-IDS: A novel statistical learning-enabled botnet detection framework for protecting networks of smart cities," *Sustain. Cities Soc.*, vol. 72, Sep. 2021, Art. no. 103041.

[65] *Openai Api Docs*. Accessed: Jun. 13, 2023. [Online]. Available: https://platform.openai.com/docs/api-reference

[66] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection," in *Proc. 16th Eur. Conf. Cyber Warfare Secur. (ACPI)*, 2017, pp. 361–369.

[67] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 1–6.

[68] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012.

[69] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Nov. 2015, pp. 1–6.

[70] *Framework for Improving Critical Infrastructure Cybersecurity*, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2018.

DANIAL KHOSH KHOLGH received the B.S. degree in computer engineering from the Khajeh Nasir University of Technology, Tehran, Iran, in 2020, and the M.Sc. degree in computer science and engineering from the University of Oulu, Finland, in 2023. He is currently a Researcher with the Center for Ubiquitous Computing, University of Oulu. His research interests include natural language processing, recommender systems, and cyber threat intelligence.

PANOS KOSTAKOS is currently a Senior Research Fellow with the Center for Ubiquitous Computing, University of Oulu. He leads the Research Group Cyber Security Informatics (CSI). His research interests include the intersection of AI, information security, and security orchestration, focusing on autonomous, mutable, and cognitive cyber defence mechanisms.

• • •