Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 00 (2025) 1–19

GPT on the wire: towards realistic network traffic conversations generated with large language models

Javier Aday Delgado-Soto[a], Jorge E. López de Vergara[a,*], Iván González[a], Daniel Perdices[a], Luis de Pedro[a]

[a]*Department of Electronic and Communication Technologies, School of Engineering, Universidad Autónoma de Madrid, Spain*

Abstract

Realistic network traffic generation is essential for evaluating the performance, security, and scalability of modern communication systems. Traditional methods, such as traffic replay systems and statistical models, while useful, often fall short in capturing the complexity and variability of real-world network scenarios. Recent advancements in Artificial Intelligence (AI), especially Large Language Models (LLMs) like ChatGPT, have introduced new approaches to synthetic traffic generation. This paper presents a novel architecture using OpenAI's GPT-3.5 Turbo to generate synthetic network traffic, with a focus on creating multi-protocol conversations that are indistinguishable from real-world interactions. Through fine-tuning and prompt engineering, the proposed system successfully generates packet- and conversation-level network traffic for ICMP, ARP, DNS, TCP and HTTP protocols. Additionally, by integrating a Mixture of Experts (MoE) architecture, this model simulates real-world network conversations with high accuracy, being able to generate a conversation combining ARP, DNS, TCP and HTTP without packet or protocol errors. The results show how the application of LLMs in network traffic generation improves realism and adaptability, establishing this approach as a valuable tool for future security testing and network performance evaluation. In addition, the proposed methodology is easily adaptable to other LLMs available both through APIs and to be downloaded and executed on your own computer.

Keywords: Network traffic generation, Generative AI, GPT, Prompt engineering, Fine-tuning, Mixture of Experts (MoE).

1. Introduction

Today, the rise of Machine Learning (ML) in network and service management is paving the way for cybersecurity and operations. These new approaches provide advantages over traditional systems, but it is well known that they require vast datasets, especially Deep Learning (DL) models. In this sense, data quality is key to improve the models, but the amount of data must be enough to extrapolate the patterns. However, high-quality datasets available for training are small and scarce, threatening the ability of the models to generalize.

In this light, generating realistic traffic that complies with existing communication protocols arises as an optimal solution to increase the available data for training, evaluation, and testing, while keeping the quality to a certain level. In particular, the ability to generate network traffic simulating different scenarios can improve the evaluation of network elements and equipment, preparing them for unknown events. Additionally, this can also be useful for data augmentation tasks. Besides, this helps to cope with many limitations of existing network traffic datasets, such as class imbalance, low quality of the features, or the age of the captures [1, 2]. Increasing the amount of available network traffic and making it indistinguishable from real traffic enables new test scenarios to evaluate and improve network management strategies. Realistic packet generation techniques can improve many aspects of network security, such as packet classification [3] models for traffic policing [4, 5] or cybersecurity [6, 7]. Without proper cohesion of elements such as those mentioned above, the new paradigm of AI-based cybersecurity will not be achieved.

Recently, a revolution in the generation of content has erupted as a consequence of new models with exceptional generative capabilities. In particular, Transformers [8] and Large Language Models (LLMs) are changing the approach to text and Natural Language Processing (NLP) problems. In the con-

*Corresponding author

Email addresses: `javieraday.delgado@gmail.com` (Javier Aday Delgado-Soto), `jorge.lopez_vergara@uam.es` (Jorge E. López de Vergara), `ivan.gonzalez@uam.es` (Iván González), `daniel.perdices@uam.es` (Daniel Perdices), `luis.depedro@uam.es` (Luis de Pedro)

Received: November 29th, 2024. Revised: March 6th, 2025. Accepted: 14th April 2025.

text of network management, NLP is receiving attention and changing the way to solve many problems such as web browsing analytics [9], packet or flow classification [10, 11] or log analytics [12, 13]. Since LLMs excel in text and code generation, the topic of network traffic generation might be another convenient topic. It is true that the throughput of these LLMs is not good enough to withstand modern transmission rates, but they might be the best tool to achieve realism.

Therefore, the objective of this work is to bridge the gap between realistic traffic generation and LLMs, designing a set of tools that learn and generalize patterns. Consequently, we must focus not only on the methodology for generating network packets, but also on the strategies to evaluate its performance. The challenge of the evaluation is particularly complicated because the types of errors that can appear are both at the packet-level and at the protocol-level (i.e., the conversation between two nodes). This requires a deep analysis of the generated traffic to ensure that there are no mistakes in stateful protocols such as TCP or HTTP.

This paper contributes by presenting a novel framework that uses LLMs like GPT-3.5 to produce highly realistic network traffic, extending current methods of traffic generation, offering a flexible and scalable alternative to existing solutions. The primary focus is on creating multi-protocol network conversations, aiming to generate traffic that is indistinguishable from real-world interactions. By multi-protocol we mean the different protocols that can intervene in a network interaction, such as an ARP request and response of a router MAC address, followed by a DNS request and response of a web server IP address, and concluding with the full TCP connection that includes the HTTP request to that web server and its response. By leveraging a Mixture of Experts (MoE) architecture and innovative prompt engineering techniques, the proposed approach demonstrates improvements in traffic realism and generation efficiency across multiple protocols. Additionally, apart from detecting the correct generation of packets, we also leverage the information provided by a well-known tool such as Wireshark [14] to determine if there is any problem in the protocol conversations obtained with our system, to better assess the quality of the generated packets. Moreover, we also test our methodology with some use cases and using several different LLMs, both from OpenAI and its competitors, comparing the results among different models. Our approach presents a novel methodology that can be easily adapted to any other protocol stack, advancing network traffic generation towards general environments with minimal pre-training.

The rest of the document follows this outline: section 2 provides a summary of the current state of the art, focusing on the novelty of this work. Next, section 3 explains the decisions made to create the generator, including data, architecture, and the model used. After this, section 4 aims to give insight to the process of creating packets and conversations for each protocol using different approximations. Then, section 5 presents the testing methodology, which is later used in section 6 to benchmark the performance of the model with several facts in mind, such as deep down packet evaluation and *Success Rate* in sending the packets. Next, section 7 applies the methodology to other LLMs. After this, section 8 comments on the findings and outcomes of this work and finally, section 9 concludes the document summarizing the main results of this work.

2. State of the art

The ability to generate realistic network traffic has long been a subject of interest to both researchers and practitioners in the fields of network security and performance evaluation. Traditional methods for network traffic generation have commonly used traffic replay systems and statistical models to simulate network behaviors, such as packet arrival rates and user interactions. Traffic replay systems are capable of replicating large-scale network flows, ensuring accurate reproduction of traffic patterns across diverse environments [15], while statistical models offer an analytical approach to predict and simulate interaction dynamics based on known distributions and empirical data.

Synthetic network traffic generation can be categorized into several types, depending on the level of granularity required:

- **Metadata-based Traffic Generation**: This type of generation focuses on creating network traffic that replicates metadata, such as packet size, timestamps, and other global variables [16]. While useful for testing basic network throughput, metadata-based traffic generation lacks the complexity required to simulate realistic communication flows or attack scenarios.
- **Flow-based Traffic Generation**: Flow-based traffic generation simulates entire communication flows between devices, including protocol behavior and user interaction patterns. This approach is more realistic than metadata generation, as it captures the higher-level interactions within a network, such as TCP/UDP flow dynamics and session persistence. However, it often falls short in replicating specific network protocol behaviors at a packet level like temporal dependencies, which is a significant limitation [17].
- **Packet-based Traffic Generation**: Packet-based generation focuses on generating individual packets that mimic specific behaviors, such as those seen in DNS queries, ICMP traffic, or HTTP requests. This method is necessary for testing fine-grained behaviors, such as packet-level attacks or the interaction between protocols at different layers of the OSI model [18].

Despite these advancements, the primary challenge with early synthetic traffic generation techniques was their inability to simulate network traffic that adapted to complex, real-world scenarios. Most models were static and could not account for shifts in traffic patterns due to unexpected user behaviors, protocol changes, or malicious activity.

In recent years, machine learning, particularly deep learning models, has revolutionized network traffic generation [19, 20]. The introduction of generative models, especially Generative Adversarial Networks (GANs) and LLMs such as GPT, has brought a new dimension to synthetic traffic generation by addressing the limitations of traditional statistical and replay-based methods [21]. AI models brought adaptability and learning capabilities that were absent from previous approaches. For metadata-based traffic, deep learning models have been employed to learn the distribution of network packet features (e.g., size, time intervals), enabling the generation of synthetic traffic that closely mirrors real-world patterns.

For flow-based traffic generation, deep learning models such as Recurrent Neural Networks (RNNs) like Long Short-Term Memory (LSTM) networks, have been utilized to capture the temporal dependencies between packets within a flow. These models learn from historical traffic data to produce sequences that replicate actual communication patterns between devices. The work by Meslet-Millet et al. [22] shows how RNNs were successfully applied to generate entire flows of synthetic traffic, improving the realism of generated traffic for network simulation and security testing.

For packet-based traffic, the same RNNs as in flow-based generation could be applied, as *NetCSTGen* demonstrates [22]. We could also make a case for Convolutional Neural Networks (CNNs) and other deep architectures that have been applied to generate synthetic packets that mimic specific protocol behaviors. These models can generate realistic traffic that emulates attack scenarios or detailed protocol exchanges, significantly improving the accuracy of network simulations.

Research into GAN-based models for traffic generation gained attention a few years ago. They revolutionized network traffic generation, offering new capabilities thanks to their Generator-Discriminator architecture. One of the best examples of this is *PAC-GAN* [23], where the generation of diverse and realistic synthetic data has been explored by modifying the traditional GAN framework to improve stability and performance in high-dimensional data such as network traffic. Previously, Ring et al. [17] proposed a GAN-based framework for generating labeled synthetic data to improve intrusion detection systems using *Wasserstein GANs with Gradient Penalty (WGAN-GP)*. By simulating realistic traffic patterns, the model significantly improved detection accuracy. Similarly, Nukavarapu et al. [18] applied *Wasserstein GAN (WGAN)* to generate synthetic UDP packets. The WGAN model addressed stability issues typically found in standard GANs, producing high-quality network traffic that improved the performance of anomaly detection algorithms.

Another work that uses GAN-based models is Netshare [24]. This work focuses on understanding how far the distribution of the generated traffic is from real traffic, measuring if the IP addresses or ports of the generated packet headers resemble real traces with metrics such as the Jensen-Shannon divergence. Note, however, that apart from IP addresses and ports, it is also important to check that the rest of the packet content is consistent. Other works that also use a statistical approach to measure their results are [25] and [26]. Instead of GAN, they use diffusion models and state space models, respectively, proving they have lower diverge when compared to prior work. These works can be useful to generate traffic that resembles statistical features of a certain dataset. However, a low divergence value does not assure that the generated packets follow correctly every protocol. In this case, other techniques have to be used, like inspecting every packet consistency, both internally and within a flow.

Regarding previous approaches, recently, transformer-based models like GPT (Generative Pre-training Transformer) have emerged as powerful tools for generating realistic, complex network traffic. These models, initially developed for language processing tasks, have been adapted for traffic generation due to their ability to produce sequences that resemble real-world patterns. These models excel in creating context-aware traffic that adapts to various protocols and network conditions, surpassing the limitations of both metadata-based and flow-based traffic generation.

The *PAC-GPT* framework proposed by Kholgh and Kostakos [27] leverages GPT-3 to generate synthetic network traffic, capturing the dynamic interactions between network devices. However, they only generate ICMP and DNS packets with low accuracy, and without generating matched request-reply conversations. Anyway, this work shows the flexibility of transformer-based models allows for context-aware traffic generation, providing a more adaptive solution compared to traditional replay systems. The ability of LLMs to generate text-like sequences allows them to model not just the structure of the traffic but also the interactions between different communication protocols, making them ideal for generating multi-protocol traffic conversations. Thus, we take this reference as a starting point for our contribution.

Related to that approach, a notable work in this topic is the *TrafficGPT* framework [28], a generative pre-trained transformer model designed to overcome limitations of tokenization and sequence length in traffic generation tasks. By leveraging a linear attention mechanism, *TrafficGPT* extends the capacity of previous models to handle sequences up to 12 032 tokens, enabling realistic simulation of network traffic flows. Its main drawback is that a whole new model has to be trained, instead of using prompts with existing GPT models in a a few-shot learning approach. LLMs have proven to be a great fit for few-shot learning strategies [29], being this approach therefore very interesting for generation optimization and cost reduction.

3. Our proposed architecture

As already mentioned, this investigation takes the work of Kholgh and Kostakos [27, 30] as a starting point. Therefore, the proposed architecture, depicted in Figure 1, grounds on the one presented there, but expanded to support new protocols and features, such as conversation generation, a per-protocol Mixture of Experts (MoE) approach or better data sources for training.

The first component of the proposed architecture is *Training data*. We consider the best format for this data to be the definition of each packet summary provided by *Tshark* (the command-line version of Wireshark) output. It must be noted that this only applies to individual packets' description, as *Tshark* does not give back a simple output for protocol conversation description, therefore needing a new summary description format for this case.

Initially, single packet data is obtained from three sources. The first one is **Ton-IoT** [31], which is the *PAC-GPT* dataset, so the data from this research is taken as the foundation. The second source is the packet summaries obtained from other

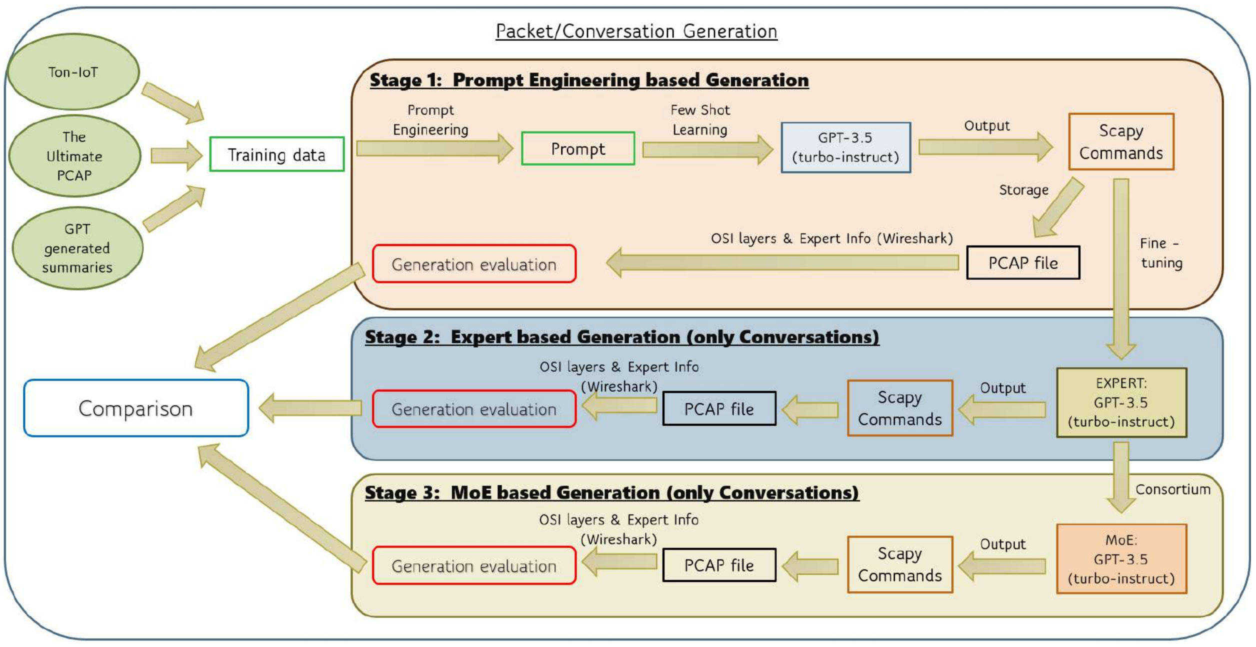

Packet/Conversation Generation
Ton-IoT
The Ultimate PCAP
GPT generated summaries
Training data
Stage 1: Prompt Engineering based Generation
Prompt Engineering
Prompt
Few Shot Learning
GPT-3.5 (turbo-instruct)
Output
Scapy Commands
Storage
PCAP file
OSI layers & Expert Info (Wireshark)
Generation evaluation
Fine - tuning
Stage 2: Expert based Generation (only Conversations)
EXPERT: GPT-3.5 (turbo-instruct)
Output
Scapy Commands
PCAP file
OSI layers & Expert Info (Wireshark)
Generation evaluation
Comparison
Consortium
Stage 3: MoE based Generation (only Conversations)
MoE: GPT-3.5 (turbo-instruct)
Output
Scapy Commands
PCAP file
OSI layers & Expert Info (Wireshark)
Generation evaluation

Fig. 1. Proposed architecture.

datasets, such as *The Ultimate PCAP* [32], which is a collection of network traffic traces by Johannes Weber. It includes traffic from a wide variety of protocols and from different versions of each over the years. It is used as an addition to the data extracted from the previous work, since for some protocols it is insufficient, and for others, such as ARP or HTTP, there is no previous data. Even with these datasets, we consider the available data insufficient, as we need larger quantities of samples to fine-tune and test the model on single packet generation. For this reason, the third source of data are new summaries with randomized variables generated by Wireshark and GPT-3.5. Based on the packet description summaries generated by Wireshark/Tshark, we use prompt engineering to teach GPT-3.5 how to replicate this structure and generate new data by modifying variables such as IP and MAC addresses, and other options according to the explanations provided to the model. This approach is necessary to get a balanced number of packet summaries among protocols.

Table 1 shows the difference between the base dataset and the one created for the investigation. The first group of columns present the base dataset, where column *base dataset characteristics* presents the characteristics of the initial dataset, while column *#base dataset packets* shows the total number of packet summaries in this dataset. The second group of columns describes the additions done to this initial data. *New Sources* specifies where the additional data is extracted from, specifying the exact number of packets from each source as the number in parentheses. Column *#packets* shows the final number of packets in the current dataset. Lastly, column *new dataset characteristics* specifies the differences between the base and proposed dataset.

Since data from other sources is not available for the generation of conversations, and there is no standard summary, a new base data type was defined for each protocol, and thanks to GPT models, a series of summaries were generated that are used as a basis for the model to learn. These base summaries will be shown later when discussing the creation of conversations.

The next component of the architecture is *Prompt*. Using prompt engineering, a query is generated so that the model learns based on a *few-shot learning* approach. The query includes examples of Scapy commands (a packet manipulation library written in Python) [33] for generating packets or conversations for the target protocol, and the model is asked to generate the commands for a new packet or conversation, using the input data. The model initially used in this research is *GPT 3.5 turbo-instruct*, and it returns the Scapy code, which is then used for two purposes. First, the code is executed to create the requested packets or conversations, and save them in a PCAP file. Various levels of checks are applied on this PCAP file, and thus we extract whether the model has generated the network packets correctly or not. The second use given to the generated code is as a basis for training the fined-tuned model. It should be noted that for this purpose, previous control filters are also applied to the code to know if it is well generated, since, if it was not, it could present serious damage to the results of the fined-tuned model when trained with erroneous data.

Subsequently, the code generated by the fined-tuned model (the *GPT 3.5 turbo-instruct* model will also be used for this purpose) are executed and passed through the same test filters. On a third stage, the same process is repeated for the MoE creation, having its results tested using the same filters as those generated by the prompt engineering and fine-tuning so that the three approaches can be compared.

Protocol	**Base dataset**	**# base dataset packets**	**New sources (# summaries)**	**# new dataset packets**	**new dataset characteristics**
ICMP	*echo request* *echo reply*	1271	ToN-IoT (1271)	1271	*Echo request* *Echo reply* *Timestamp* *Time-to-live exceeded* *Destination unreachable*
ARP	Non-Existent	0	The Ultimate PCAP (92) GPT 3.5 (918)	1000	*Directed* *Broadcast*
DNS	*Type A*	526	ToN-IoT (526) The Ultimate PCAP (189) GPT-3.5 (285)	1000	*Types A, PTR, MX, CAA, SOA, HTTPS and RP*
HTTP	Non-Existent	0	The Ultimate PCAP (46) GPT 3.5 (954)	1000	*GET* *HEAD* *RESPONSE* *SUBSCRIBE* *UNSUBSCRIBE* *CONNECT*

Table 1
Base dataset and extended dataset comparison

4. Methodology

In this section, the general structure used for the development of the model is presented. As a central and key element for its operation, it is necessary to define a format to be used, trying to make it as replicable as possible. As commented in section 3, *Tshark* summaries are only available for individual packets. Thus, a new summary description format is needed for conversation explanation. These new summaries are based on the necessary characteristics of each type of conversation, and they are presented later.

Regarding the creation of prompts, a structure with the following sections is established:

1. Brief explanation of needs: protocol, packet or conversation, and input data.
2. Presentation of one or more examples.
3. Additional explanations. They are added to better guide the model, improve its performance, or to correct detected errors.

As mentioned earlier, we intend to generate both individual packets and conversations. The generation of packets is proposed as an initial test environment to discover the feasibility of creating more complex elements, such as conversations, based on the correct generation and relationship of individual packets.

This iterative approach has allowed us to detect errors common to all types of conversation. Such as the initial inability of the models to handle changing directions and delays, causing the conversations to lose all coherence in the eyes of a machine and of a person familiar with the basic structure of network conversations. These errors, and many others, have been corrected thanks to the explanations added at the end of each prompt. It is important to highlight that, throughout the tests, the model understands better and seems to remember more vividly when generating packets instructions at the end than those at the beginning. When carrying out these tests, input data was divided into splits composed of 10 summaries each and evaluated independently.

All commented prompts and datasets can be found in the Git repository included in the Resources after the conclusions.

4.1 Packet generation

PAC-GPT [27] generated only ICMP and DNS protocols. It is undeniable that these two protocols represent a significant percentage of the daily network traffic, but it is unrealistic to consider a traffic generator with these capabilities alone. One of the highlights in the future work of the baseline research is the incorporation of new protocols. Therefore, we want to take a step forward by adding new protocols to evaluate the feasibility of the approach under different conditions. For this purpose, we intend to introduce ARP and HTTP, as well as to extend the ICMP and DNS generation capabilities.

The reason behind the introduction of ARP is the evaluation of the model generation capabilities with a protocol, which, although simple as ICMP, presents a different challenge since it does not share many characteristics with this protocol.

Meanwhile, the introduction of HTTP is because, although the DNS protocol also belongs to the application layer, the HTTP protocol is supported by TCP, and for its correct operation it is necessary to establish a TCP conversation, which is not necessary for DNS since it usually works over UDP. An HTTP conversation is considered a complex test of traffic generation feasibility, which is why it is introduced in this paper.

It must be noted, that due to economic reasons, not all the packets are generated, as it would highly increase the cost of the investigation.

We will now go on to explain the particularities and creation of the different packets.

4.1.1. ICMP

ICMP is the first protocol tested, as it is one of the easiest, commonly used protocols. As seen in Table 1, original data is quite limited. Although it is true that it represents a large amount of the packets sent using this protocol, it does not represent the whole functionality of ICMP. Furthermore, using only these two easily created ICMP packet types, and such a high presence in the datasets of this protocol, might be biasing the evaluation of the generators.

As an example, the Scapy commands required for the generation of two random ICMP packets are:

```
scapy.IP(src= "8.8.8.8", dst= "130.231.202.234")
     / scapy.ICMP(type=0, id=0x0002, seq=4)
scapy.IP(src= "130.231.202.234", dst= "8.8.8.8")
     / scapy.ICMP(type=8, id=0x0002, seq=5)
```

It should be noted that the `ttl` parameter does not appear in the commands, as it is not generated in the basic research, and due to its importance, especially in the ICMP protocol, it is considered a necessary addition. We do include this parameter in our dataset, as it appears very useful in case in the future we want to teach the model to simulate tools such as `traceroute`, where this parameter is necessary.

To solve these problems, we decided to generate a new dataset, including new types of ICMP packets, which, although less present in daily network traffic, are very useful and showcase more challenging situations. The selection of the packets to be generated is based on their capacity to establish bilateral conversations, since this is the real purpose of this study, to demonstrate that this type of conversations can really be generated. Additionally, supporting all of these ICMP types requires a query several times more complex. The query begins by detailing to the model what its functionality will be, telling it that it must create ICMP packets (it seems obvious, but it is not), and then explaining how to create these packets with examples. After several iterations, where the model failed several times, we added additional explanations to the prompt, some of them completely dependent on the protocol and others on the model, to solve the errors.

By improving ICMP generation capabilities, we aim to demonstrate how newer LLMs and highly refined prompts can improve packet creation.

4.1.2. ARP

The ARP packet generation process is similar to ICMP, since it is also a query-response connectionless-oriented protocol. Therefore, we can use the ICMP prompt as the basis for the ARP prompt. Due to the simplicity of the creation of these packets, it is taken as a design decision to explain the different types that are intended to be generated, not giving the model the option of applying previous knowledge in the generation of each packet.

The lack of short prompts concatenated to the definition of each type of packet is also noteworthy, mainly because, thanks to the low complexity of the protocol, it is not necessary to add these explanations to warn the system of the different options it offers, thus reducing the number of tokens used in the prompt.

4.1.3. DNS

DNS is an application layer protocol, which makes it more complicated to generate because it works over lower layer protocols such as UDP and IP, having, therefore, more possible points of failure. At the same time, it is a protocol rich in options, designed to have many of these and be useful in various scenarios. Packets of this protocol were generated in the research base; therefore, we start from its initial design to understand and improve the generation.

As shown in Table 1, initial data is not varied enough. For this reason, it was decided to expand the input. With these new inclusions, we assess the real capabilities of the model of generating any kind of DNS traffic.

One of the main objectives of this work is to demonstrate that the state-of-the-art approach to DNS packet generation can be improved. To this end, we intend to substantially enhance the explanation given to the model to achieve correct packet generation.

From our prompt, it can be extracted that a large part of the errors in previous DNS generation may be due to the little prior information given to the model. This left the model free to make random choices which are not reproducible. By giving more precise prompts, we reduce the randomness of the generating, reducing the probability of incorrect generation.

It should be noted that the other researchers attributed this failure to Scapy, given the large number of parameters and options, especially for the response and additional resource fields. It is true that this is a critical factor, and that it is also largely responsible for the low quality of the generated DNS traffic.

At first glance, the difference in the size of the prompts stands out. Of the 3 protocols discussed so far, DNS has the longest prompt. This is mainly due to the number of protocol options and errors detected in the generation.

As in the previous cases, the beginning of the prompt is the same, to prevent typical errors such as the generation of multiple packets for a single summary. In this prompt, the same structure as in the previous ones can be observed, a sample of a summary and the command that generates the specified packet and a subsequent series of comments and clarifications of the variables of these commands, specifying the value or reasoning necessary to give one value or another in different cases.

This collection of clarifications is based on the different errors that have been found, each explanation in the prompt being the solution found to an error. As mentioned above, it is preferable to make clarifications at the end because it seems that the model retains them better and consequently applies them.

We will not go into the details of the different prompts, since these are based on technical aspects of the protocol and its parameters, as well as intrinsic aspects of the creation of these packets with *Scapy*.

4.1.4. HTTP

HTTP packet generation is still an open topic in LLM-assisted packet generation. We decided to add this protocol because we wanted to evaluate the performance of a higher-complexity protocol in the generation of conversations. A connection-oriented protocol is quite challenging, since it must automatically manage attributes of the connection, such as the TCP SYN and ACK numbers. This is interesting in comparison with DNS, since the latter is based on UDP and does not require the establishment of sessions; therefore, HTTP will be used to see if the model is capable of generating more complex sequences.

As a first feasibility test, it is decided to evaluate whether it is possible to generate simple HTTP packets, without the need of connection establishment. The next section will cover the full TCP&HTTP client-server dialogue.

As with ARP, we do not possess any example of HTTP creation initially; thus, we cannot modify a base prompt in this case either. Therefore, we create the prompt from scratch.

It should be noted that this prompt is much shorter than the previous ones, explaining only two types of packets. It is decided to use this approach (completing it with a textual explanation of how to change the necessary parameters in the case of another type of packet) given the similarity in the creation of the different types of traffic, being only necessary to change certain variables both in the request and in the response.

There are not many prompts. This is because the generation of these packets is simple, thanks to the use of the Scapy classes `HTTPRequest()` and `HTTPResponse()`, which greatly facilitate the creation of these packets and the definition of their parameters in a way that is understandable for the model.

It is true that, since HTTP is a textual protocol, it could have been created without using these classes, simply by generating the text string. This approach was also tried, but was discarded because it was more complicated for the model to understand, having to change specific parts of a string.

Special emphasis has been placed on ports. This has its origin in the model's treatment of them, since it seems that they are difficult to understand, and they are always structured in the same way for the model.

4.2 Conversations generation

This section is considered the core of our research. It is intended to demonstrate something that, although it may seem logical, has not been previously shown in the studied literature: that it is possible to generate synthetic conversations hardly distinguishable from real ones with already available LLMs. This work is a first step on this path, serving as a demonstration of the conversation creation capabilities of the following protocols. In this sense, we approached individual packet generation as a way to ensure the generation limits of the model, then working on these limits with conversation generation to reach the edge of traffic generation without errors. For this purpose, we select only the best-case scenario for each protocol, willing to assure a base advance on this field to continue working from it on the future. Table 2 shows the number of summaries gathered for each type of conversation, using from each protocol a minimum of 200 conversations for testing.

	ICMP	**ARP**	**DNS**	**HTTP**
# summaries	280	305	220	220

Table 2
Number of conversations in dataset

Before starting to describe the process of generating these conversations, it is important to emphasize another essential element of the protocol conversations. The communication delay between the different ends, derived from the processing and forwarding of the packets by the intermediate elements and the other end of the communication. If one intends to design a system capable of simulating conversations, it is imperative to model this delay, since it is inherent to the conversation, and without its presence one cannot think of a realistic conversation.

This delay is usually modeled with a Gaussian distribution. Therefore, we intend to teach the model to introduce a random waiting time between each pair of packets (regardless of the type of conversation) so that this condition is met.

4.2.1. ICMP

ICMP conversations are one of the simplest that can be found on the network. These, from the point of view of the sender and receiver, have only 2 packets.

4.2.1.1 Input data

As discussed above, there is no input data; therefore, it is necessary to define a format to summarize these conversations so that the model can easily understand the different parameters. An example of the chosen format is:

```
    Source:  IP="172.16.0.5" // Destination:
IP="172.16.0.15" // Others:  id=0x3532 seq= 147
                   type=Echo
    Source:  IP="10.10.10.1" // Destination:
 IP="10.10.10.2" // Others:  id=0x8234 seq= 223
                 type=Timestamp
```

These summaries, although simple, gather the minimum parameters to establish an ICMP conversation. In them, we can find data related to the endpoints of the conversation, as well as internal variables of the conversation such as the ID or the initial sequence number. In turn, the type of conversation to be generated is defined.

For the generation of these summaries, a base structure is defined, and for the replication of this structure, completed with different values in each summary generated, the *GPT 3.5 turbo-instruct* model is used.

4.2.1.2 Generation of the conversation

Since this part of the work was developed after the generation of packets, the previously acquired knowledge is used to generate the prompts. This allows a more efficient prompt engineering, achieving an efficient reduction of the number of tokens needed for the prompts.

Studying the prompt, it is evident that the generation of these conversations is simple, since we do not find prompts that resolve errors in it. In the prompt, two different conversations can be seen, defining step by step how the packets should be and the generation of the random time between them. These packets are created in the same way as if they were individual packets. More importantly, it has been proven that eliminating one of the two conversations does not significantly reduce the quality of the generated code.

4.2.2. ARP

Hand in hand with ICMP conversations, ARP conversations are among the simplest that can be found on the network, normally used to obtain MAC addresses and update the routing tables of each of the devices. They also have a request-response structure, so for the endpoints they are no more than 2 packets.

4.2.2.1 Input data

As in the other cases, an initial data type has to be defined. This data type is based on the one generated and tested during the generation of ICMP conversations. An example of such summaries is:

```
Source: MAC="e9:f0:a1:b2:c3:d4",
IP="10.10.20.1" // Destination: MAC=
"d4:c3:b2:a1:f0:e9" // Wanted: IP= "10.10.20.2"
Source: MAC="f1:a2:b3:c4:d5:e6",
IP="172.16.1.10" // Destination: MAC=
"FF:FF:FF:FF:FF:FF" // Wanted: IP=
"172.16.1.20"
```

Only 2 types of ARP conversations are generated, those starting from a query to *Broadcast* and those starting from a query to a specific destination. The dataset has an even representation of these two types of conversations, although in the real world, it is the first ones that are more used.

For the generation of these summaries, as in ICMP, the *GPT 3.5 turbo-instruct* model has also been used, asking it to generate both IP addresses and MACs.

4.2.2.2 Generation of the conversations

These conversations consist of the same two-packet scheme, with a Gaussian random time between them. In this case, it has been decided to apply another approach, instead of generating a single query combining the prompts of both cases, 2 different queries are created, and we select which one is sent to the model in each case depending on the input summary we have.

An important insight extracted from the generation of packets is that the model works better when it does not have to generate variables defined by text. That is, logically, it is easier for the model to replicate *"var=randint(1,5)"* than to understand *"you must generate a random integer between 1 and 5, and assign it to the variable var"*. Therefore, this new approach will be used in cases where it is necessary to generate random variables.

As in the case of ICMP, there are no auxiliary prompts for error correction, this is also due to the good generation of results from the beginning, with no results to correct.

4.2.3. DNS

DNS conversations have a similar structure to the previous ones, i.e., they are based on a request-response architecture, given that, although it is an application layer protocol, thanks to its encapsulation over UDP it is not necessary to establish connections.

However, in this case it is necessary to pay special attention to the individual generation of the packets, since, as mentioned in subsection 4.1.3, they present greater difficulties in their creation.

For reasons that will be detailed in the following sections on tests and results, it was decided to create only DNS type A conversations, taking these as a basis for future improvement and the inclusion of new types of conversations.

4.2.4. Input data

For DNS, the generated data has the following form:

```
Source: IP= "172.16.0.5" // Destination: IP=
"172.16.0.15" // Others: id=0x3532 ,
resource= "www.google.com" , response =
"172.217.7.196"
Source: IP= "10.10.10.1" // Destination: IP=
"10.10.10.2" // Others: id=0x8234 ,
resource= www.youtube.com , response =
"172.217.10.206"
```

A continuist structure with the previous cases is maintained, where the addition of new parameters such as *resource* and *response* stands out. In these two fields, we find the domain name of the resource from which we want to obtain the IP address, and the response with the IP address in which this resource is located.

As in the previous cases, these IP addresses and resources have been generated with the *GPT 3.5 turbo-instruct* model; therefore, they do not correspond to the real world, they are simply test data. If we wanted them to correspond to the real world, it would be enough to simply modify the input data so that they are in accordance with the real ones, in this way, the model would generate traffic with real addresses.

4.2.4.1 Generation of conversations

Having decided to generate only conversations of DNS type A, the process of generating them is greatly simplified, since the internal structure of these conversations is relatively simple. This structure has a simple query in the request packet, and a response field with a single IP in the response packet, with no *Additional Resources*.

Contrary to what might be expected, the definition of a DNS conversation is slightly more complex than that of an ICMP or ARP conversation. This is only because of the number of parameters that must be handled when creating the conversation, but by reducing the number of conversation types to be generated, the complexity of managing these parameters is considerably reduced.

Thanks to the new approach to the generation of variables presented in DNS, a major point of failure in the generation has been corrected: the management of ports to simulate the use of unregistered or ephemeral ports.

4.2.5. HTTP

The feasibility of this research lies largely in the correct generation of HTTP conversations, since its main purpose is to demonstrate that LLMs are capable of generating complex sessions, as well as managing variables such as the sequence number or the ACK throughout a conversation. These two variables are critical for the operation of the conversation; therefore, special attention is paid to them throughout the development.

4.2.5.1 Input data

HTTP conversations are the most difficult to generate, but this is intended to show that they can be generated of different types, with different codes and *Reason Phrases*. It must be noted that we do not use real IP addresses, a random IP address is generated and then written in the summary. This decision does not affect the model functioning because, if the real IP address was to be used, the model would be trained to generate an output with that exact IP address. To do this, a dataset containing records such as the following is generated:

```
Source: IP="192.168.1.10", port=1241, Window:
8192 // Destination: IP="192.168.1.20",
port=80, Window: 1634 // Others:
Host="http://ip.webernetz.net/", Path="",
Code=200, Reason_Phrase= "OK"
Source: IP="58.29.214.109", port=1123, Window:
48392 // Destination: IP="204.79.197.200",
port=80, Window: 24196 // Others:
Host="http://www.pinterest.com/", Path="",
Code=307, Reason_Phrase= "Temporary Redirect"
Source: IP="50.62.160.105", port=5020, Window:
12345 // Destination: IP="104.244.42.129",
port=80, Window: 6172 // Others:
Host="http://www.twitter.com/", Path="",
Code=403, Reason_Phrase= "Forbidden"
```

Clearly, these are the most complex conversation summaries, given the large number of IP, TCP and HTTP parameters that the model must handle. Initially, a design decision is made to endow the model with data such as source port or windows at both ends to force the model to focus on correct packet construction, and not so much on variable generation. It is true that in real conversations the window size can be variable, but this is considered future work for this project.

As in the rest of the protocols, there are two similar datsets, one used for training and the other for testing. In all datasets, there is HTTP traffic mixed with the following codes:

- **200**: OK
- **307**: Temporary Redirect
- **403**: Forbidden
- **404**: Not Found

4.2.5.2 Conversation generation

A conversation structure of 11 packets has been defined, being 3 for the *TCP Handshake*, 2 for HTTP messages and another 2 for their respective *ACKs*, and finally, 4 for the *TCP Teardown*. This makes the generation of these conversations particularly delicate, having to pay close attention to the management of sequence and acknowledgment numbers, as discussed above.

This prompt is the longest of all, counting with 32 commands. Among these commands, we can find the definition of the different packets, as well as a continuous update of the variables previously commented, to ensure their correct management. This variable control solution was reached after several iterations, in which we tried to explain to the model how to manage these variables by means of text. With the previous strategy, the model was unable to generate meaningful outputs, so it was decided to force the model to follow stricter rules so that its "originality" would not be impaired.

However, we add a brief explanation at the end of the prompt to solve a recurring error in the model. Strangely, even with so many example packets, the model did not correctly create the name of the HTTP variables, which meant that the classes `HTTPRequest()` and `HTTPResponse()` could not be executed.

4.3 Experts creation

From the beginning, the creation of experts is proposed as a mean of optimizing the operation of the model, reducing the required input tokens, and improving the efficiency of the model. This has already been discussed, but there is another major motive behind this decision, the reduction of errors. It is proposed that if the model is trained with multiple generated and revised data, without having the typical model generation errors, this will prevent the model from committing this type of error again, thus ensuring its correct operation.

For the creation of these experts, the *GPT 3.5 turbo-instruct* model is used, allowing direct comparison with the generation by prompt engineering. As stated in the OpenAI model fine-tuning guide [34], the first step required for fine-tuning is the preparation of the data. These data must follow the dictionary-based format, so pre-processing of the data is necessary.

Two datasets are generated, one to carry out the generation of conversations and training of the fine-tuned model, and another one with data not seen by the fine-tuned model for its later evaluation.

As input data, we have the conversations previously generated by prompt engineering. We have stored all responses returned by the model in separate files, which are then treated independently to filter only the returned code (model responses usually include additional information such as tokens used, temperature, model used, date and time, etc.). Additionally, it validates this data by searching for errors in the generation of the data, and studies the distribution of the messages, and finally, it returns an estimate of the training cost of each of the models based on the generated data.

For this step, the conversation datasets of the different protocols are divided in two, one part will be used for training and the other for validation, the ratio between the parts is 70-30.

In this input data generation, it is necessary to define a *"role": "system", "content"*, so that the model understands what is wanted from it. For all cases, this will be:

```
You are a new generation traffic generator.
You are specialized in the {} protocol and
conversations generation using python and scapy.
You are especially attentive to variables and
different types of traffic.
```

As can be seen, it is a relatively simple prompt. In it, it is only intended to make clear what is expected of the model and how it should behave. It appears {}, this is nothing more than a space reserved for the name of the protocol in each of the cases in which it is necessary to generate an expert.

With this input data, the different experts can be generated. As one of the purposes of this investigation is the comparison with the existing best models in equal conditions, we use the same number of epochs (4) as the base investigation [27], as the number of epoch can clearly influence the model fitting and affect testing. We also took into account OpenAI's community recommended and preestablished number of epochs (which is also 4) [35].

Different iterations of this generation of experts have been carried out while improving the generation of conversations. However, at the end of the project, there are 4 experts (one per protocol) capable of generating conversations of similar quality to those obtained by prompt engineering.

4.4 Mixture of Experts

The concept of expert mixing is not new, it originated in 1991, when Robert Jacobs and Geoffrey Hinton proposed in their paper *Adaptive Mixture of Local Experts* [36], a new alternative architecture to the traditional single-network architecture. In this new architecture, a consortium of expert networks was used in subtasks of a larger problem, thus improving the accuracy and flexibility of the model. In this article, the concept of a *Gating network*, or routing network, was also presented, which was used to decide which expert was best suited to solve a given input. This routing network had to be trained to adjust its weights, representing the network's confidence in each of the experts.

This architecture was associated with different Artificial Intelligence systems such as neural networks or Support Vector Machines (SVMs), having good results but facing problems such as the computational demand generated by increasing the number of models and the difficulty of training the routing network for effective operation.

For this reason, in the years following its presentation, the architecture was gradually lost, but with the advance of hardware, thanks to new GPUs and greater computational capacity, the importance of this architecture grew. Anyway, the real breakthrough came in 2020, when it was demonstrated that the union of a MoE architecture with LLMs offered unexpectedly good results in natural language processing tasks, and similar. This has led many to theorize about whether OpenAI's new model,

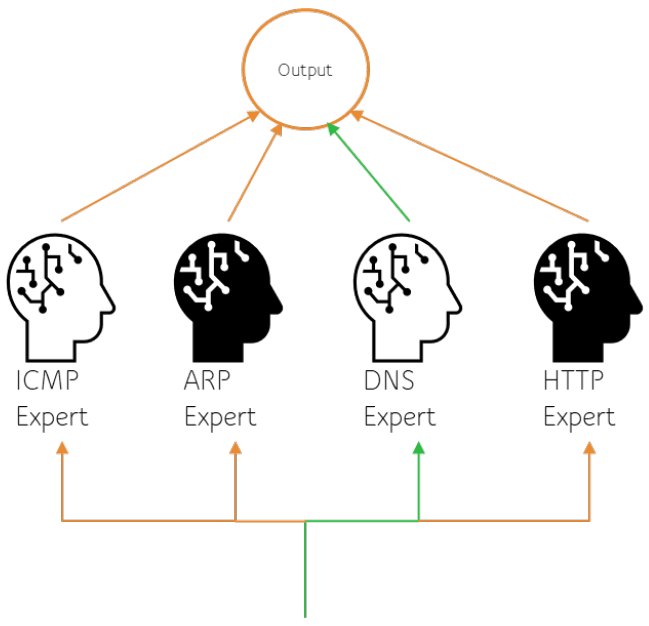


Fig. 2. MoE operation for a DNS summary

GPT-4, is built on this architecture. There is no completely accurate information on this, but what is clear is that Google has used this architecture for the creation of its most advanced model, *Gemini*.

In our *MoE* there is no need to introduce a routing network, since it is known in advance what type each summary is, and the ARP expert would never be asked to generate an HTTP conversation. In this way, it is possible to reduce the computational cost, eliminating the cost derived from the training and execution of the network and the management of weights in each iteration, and the economic cost of sending the query to all the models, sending only one query to the selected model. Therefore, the proposed architecture will be the one shown in Figure 2.

4.5 Network Conversations Generator

To use the *MoE*, a script is created to generate traffic. This script has the particularity that it does not have input data, it is the script itself that randomly generates summaries of the different protocols of the experts. Although initially the traffic generation is random, in the future, with a higher number of protocols, it is intended to make this randomness depend on the representation of each protocol in the real traffic.

After generating these summaries, when sending them to one of the models, the script detects which protocol the summary belongs to and redirects to the corresponding model. Finally, the received response is processed and the returned commands are executed so that they can be stored in a PCAP file and be the basis for future network traffic datasets.

The generator inherits some features of the packet generator proposed in the initial research, but it is more focused on the generation of conversations.

It is important to note that for a conversation to be truthful, the ends must be truthful. No network expert will be sure that

Protocol	# packets	Correctly sent	Malformed	Non-existent	Success rate	Presence in total
IP	683	683	0	0	100%	75.138%
ICMP	200	200	0	0	100%	20.100%
DNS	217	192	0	25	88.48%	21.122%
UDP	217	217	0	0	100%	23.872%
ARP	226	226	0	0	100%	24.862%
HTTP	352	351	0	1	99.72%	38.614%
TCP	352	352	0	0	100%	38.724%

Table 3
Packet generation (GPT-3.5) results divided by protocol

	ICMP	ARP	DNS	HTTP
Total	51W-0E (0W-0E)	98W-0E (0W-0E)	0W-25E (0W-25E)	43W-1E (0W-1E)

Table 4
Warnings (W) or errors (E) detected by Wireshark in the packet generation PCAP files. In parentheses, the results after inspecting the cases.

a DNS request to Google's DNS servers will point to the IP 192.168.10.1, that is why variables such as IPs and MACs are extracted from a document where you have a control of what they are, and are not generated randomly.

5. Testing methodology

Regarding the evaluation of the packets generated, as mentioned earlier in the article, we will follow the same line as previous research, evaluating the success rate of sending all the packets. However, first of all, we must be able to detect errors that the packets may have in the different protocol stack layers. For this, the following checks are made:

- Errors and/or packet malformations detection at OSI data link, network, transport, and application layers.
- Errors detection when sending packets.

At the same time, an in-depth study of the PCAP files contained in the packets generated is proposed, to detect errors that may go unnoticed by the first level of testing. In this step, special consideration is given to Wireshark's own study of the packets and conversations, since this program has specific tools for detecting anomalies and errors in the protocols.

Therefore, as there is a high confidence in the detection of this type of failure by Wireshark, a new metric is proposed for measuring the quality of the traffic generated. This new metric is the number of warnings and errors detected by Wireshark's *Expert Info* in a capture, which can be of different severity.

Once the evaluation environment is defined, the tests are carried out. In these, multiple iterations of packet generation are carried out, handing the model different data splits (10 summaries each) and evaluating each one.

Subsequently, the results of the tests on each of the protocols in the different scenarios (packets, conversations, experts, etc.) are compared with the rest of the protocols to evaluate the results as a whole, taking into account de mean value of the metric in the different splits.

6. Results evaluation

6.1 Packet generation

Table 3 shows the results of packet generation using GPT-3.5. As an initial impression, if compared with the base study results, a substantial improvement in the results can be seen. It is true that these are not perfect, but they are significantly better. Above all, the improvement in DNS stands out, which goes from a poor 10% (the best result for this protocol in PAC-GPT [27]) to an acceptable 88.48%, having 25 packets in which Python and Scapy cannot detect a DNS packet when loading the DNS packet PCAP file. Further on, with the PCAP file study, the reason for this error is detailed.

On the other hand, the results for the rest of the protocols seem encouraging, with ICMP remaining in the upper margin of the results shown in the base investigation. Considering that in this study the number of different types of ICMP to be generated has been extended, it confirms that these models are capable of generating this traffic at the highest level. The same conclusion is drawn for ARP. As a note, we would like to highlight the good abilities of the model to generate "support" protocols such as IP, UDP and TCP, having no failures in their generation.

Evaluating then the protocols according to the number of failures that Wireshark's *Expert Info* shows in the different PCAP files, we have the values shown in Table 4. Although they may look like alarming numbers, the real result is the one shown in parentheses. For packet generation, these warnings are not considered relevant, since they are mainly due to non-detection of responses or duplicate IP or MAC addresses between packets. The metric is kept because of its interest for the study of conversations. Thus, in this case, we focus only on the study of Wireshark detected errors.

It is not a coincidence that for DNS, we find the same number of errors in the Wireshark warnings/errors as in the script designed to detect failures. Anyway, this allows us to easily identify the reason for these errors.

These errors are generated by packet malformation, this is the message returned by Wireshark. Further investigation re-

Protocol	# conversations	# packets	Correctly sent	Malformed	Non-existent	Success rate	Presence in total
IP	534	2274	2274	0	0	100%	85.041%
ICMP	200	400	400	0	0	100%	14.959%
DNS	200	400	400	0	0	100%	14.959%
UDP	200	400	400	0	0	100%	14.959%
ARP	200	400	400	0	0	100%	14.959%
HTTP	134	268	268	0	0	100%	10.022%
TCP	134	1474	1474	0	0	100%	55.123%

Table 5
Conversations generation (GPT-3.5) results divided by protocol

	ICMP	ARP	DNS	HTTP
Total	0W-0E (0W-0E)	0W-0E (0W-0E)	0W-0E (0W-0E)	0W-0E (0W-0E)

Table 6
Warnings (W) or errors (E) detected by Wireshark in the conversation generation (GPT-3.5) PCAP files. In parentheses, the results after inspecting the cases.

veals that all of these errors occur in DNS responses, specifically when they have a poorly declared *Additional Resource* in the input summary. An example of a summary that causes this error is:

```
130.231.240.70 130.231.202.234 DNS 153 Standard
      query response 0x23f8 No such name SOA
            usage.fdown.net.oulu.fi SOA
```

In the above summary, it can be seen how at the end of the packet definition, it is specified that the packet has a type SOA *Additional Resource*. Reviewing the generated commands, it is found that the model cannot generate this record correctly. It usually gives this field malformed, or adds more records to the *Additional Resources* count in the protocol header. This causes Wireshark to mismatch the numbers and generate the error. In the same way, in the script created, we can find this failure in the form of a packet that cannot be read because Scapy is not able to handle a packet in which the data does not match.

From this error, it can be seen that even if the model can correctly generate a large part of the packet, if the summary is not clear enough, it fails, since it is not able to reconstruct entire elements based on a single word. This error is attributed more to the input data than to the capabilities of the model.

In the case of HTTP, in Table 3, we can observe that the script detects one malformed packet, whereas, Wireshark detects 43 warnings and one error, as shown in Table 4. This difference in warnings is mainly because Wireshark does not know that all packets are HTTP, but the script does. Wireshark detects some packets as TCP retransmissions, as they might have slightly different ACK or SEQ numbers. The error detected is due to a slight Python coding failure, not related to the protocol comprehension. These results seem extremely promising for future generation of complex HTTP-based structures.

6.2 Conversations generation

Table 5 shows the results of conversation generation using GPT-3.5.

In this case, the number of packets that have been created stands out. This is due to the number of packets generated per summary. For ICMP, DNS and ARP, 2 packets are created per summary, while for HTTP, 11 are created. Regarding the results shown, the Success Rate percentages obtained stand out, being 100% in all cases. This is mainly due to the simplicity of certain conversations, as well as the prompt engineering approach more oriented to the generation of variables by commands and the reduction of explanations, as can be seen in the Git repository.

The results obtained for ICMP and ARP follow the line of those obtained for packet generation. This result is considered logical, since the packet generation demonstrated the system's ability to generate both query and response packets, teaching the model in this step nothing more than the coherent union of these.

The high *Success rate* found in DNS may be surprising, but it is due to the results obtained in the packet generation. In this phase, it was shown that the model was unable to generate correct responses when they had a final *Additional Resources* field vaguely explained in the input data, and this affected the perception of the generation capabilities of this protocol. In this case, we decided to test the ability to generate conversations where these final fields did not exist, taking as input the most common DNS requests in the network, the IP query type A requests. These are relatively simple, so this result is to be expected. Progress should be made in this direction to improve the generation skills for the *Additional Resources*.

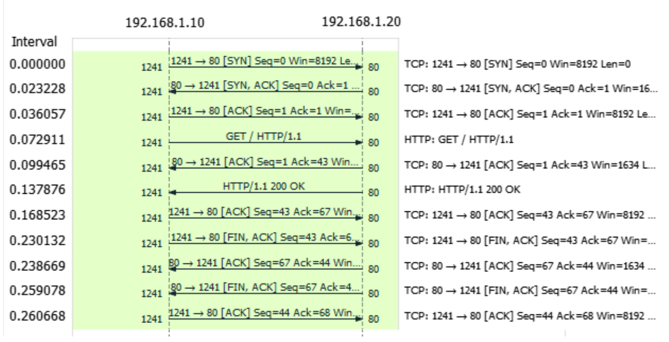
As for the results obtained in HTTP, these are particularly remarkable, given the complexity of the generated conversation. The conversation shown in Figure 3 shows the packet breakdown of the conversation. As a comment, we would like to highlight the ease of the model to manage sequence and acknowledgment numbers. Initially, a textual prompt was tested, returning poor results, but, when switched to a prompt based on variables and commands, the results changed drastically. These results showcase that these models are still relatively far from fully understanding and replicating human reasoning,

Protocol	# conversations	# packets	Correctly sent	Malformed	Non-existent	Success rate	Presence in total
IP	516	2076	2076	0	0	100%	83.845%
ICMP	200	400	400	0	0	100%	16.155%
DNS	200	400	400	0	0	100%	16.155%
UDP	200	400	400	0	0	100%	16.155%
ARP	200	400	400	0	0	100%	16.155%
HTTP	116	232	232	0	0	100%	9.370%
TCP	116	1276	1276	0	0	100%	51.535%

Table 7
Conversations generation (Experts) results divided by protocol

	ICMP	ARP	DNS	HTTP
Total	0W-0E (0W-0E)	22W-0E (0W-0E)	0W-0E (0W-0E)	0W-0E (0W-0E)

Table 8
Warnings (W) or errors (E) detected by Wireshark in the conversation generation (Experts) PCAP files.



```
Interval
0.000000  1241 → 80 [SYN] Seq=0 Win=8192 Le...        TCP: 1241 → 80 [SYN] Seq=0 Win=8192 Len=0
0.023228  80 → 1241 [SYN, ACK] Seq=0 Ack=1 ...        TCP: 80 → 1241 [SYN, ACK] Seq=0 Ack=1 Win=16...
0.036057  1241 → 80 [ACK] Seq=1 Ack=1 Win=...         TCP: 1241 → 80 [ACK] Seq=1 Ack=1 Win=8192 Le...
0.072911  GET / HTTP/1.1                              HTTP: GET / HTTP/1.1
0.099465  80 → 1241 [ACK] Seq=1 Ack=43 Win...         TCP: 80 → 1241 [ACK] Seq=1 Ack=43 Win=1634 L...
0.137876  HTTP/1.1 200 OK                             HTTP: HTTP/1.1 200 OK
0.168523  1241 → 80 [ACK] Seq=43 Ack=67 Win...        TCP: 1241 → 80 [ACK] Seq=43 Ack=67 Win=8192 ...
0.230132  1241 → 80 [FIN, ACK] Seq=43 Ack=6...        TCP: 1241 → 80 [FIN, ACK] Seq=43 Ack=67 Win=...
0.238669  80 → 1241 [ACK] Seq=67 Ack=44 Win...        TCP: 80 → 1241 [ACK] Seq=67 Ack=44 Win=1634 ...
0.259078  80 → 1241 [FIN, ACK] Seq=67 Ack=4...        TCP: 80 → 1241 [FIN, ACK] Seq=67 Ack=44 Win=...
0.260668  1241 → 80 [ACK] Seq=44 Ack=68 Win...        TCP: 1241 → 80 [ACK] Seq=44 Ack=68 Win=8192 ...
```

Fig. 3. HTTP presented conversation example

while they are more than capable of performing any kind of task that can be explained logically.

By re-evaluating this conversation generation according to the number and type of warnings/errors returned by Wireshark, we obtain Table 6. In this table, the results returned by the packet validation script can be corroborated, considering that Wireshark also evaluates the completeness and perfection of the conversations. Therefore, the fact that this program does not return any warnings or errors for the different protocols reaffirms the theory that the generated traffic is indistinguishable from the real one.

6.3 Experts creation

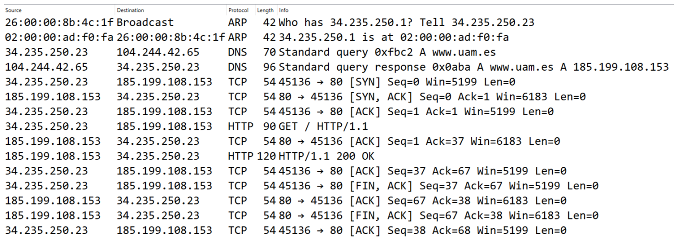
Table 7 shows the results of the generated conversations using the experts created with GPT-3.5. The ideal result would be for these experts to match the results obtained in conversation generation.

To complete this assessment, Table 8 shows the number of warnings and errors returned by Wireshark. There are 22 warnings in the case of ARP, but these are due to the MAC addresses, since 13 of them are group addresses and not individual ones. This is not a problem with the model, but with the input data. Specifically, the input data generated with generative AI.

Comparing tables 5 and 7, in addition to tables 6 and 8, it is observed that both approaches obtain the same results in terms of *Success Rate* and errors. Therefore, it is considered that the goal pursued with the creation of these experts has been achieved, given that, obviating the training cost, it has been demonstrated that the same results can be obtained without the need to repeatedly explain to the model how it should generate traffic, thus efficiently reducing costs and increasing efficiency.

6.4 MoE feasability

Since the correct functioning of the experts (fine-tuned models) has already been demonstrated, it is also intended to check if the Mixture of Experts architecture works correctly. For this purpose, 2 tests are proposed. The first test is more of a theoretical level test and integration of the models' evaluation, it is intended to demonstrate that with this architecture it is possible to simulate the process that a computer must follow to obtain a resource hosted on a web page. To obtain this resource, a first ARP request to your router is necessary to obtain its MAC address and thus be able to route traffic at layer 2 in its direction. Subsequently, a query must be made to the DNS server to obtain the IP address associated with the device on which the resource is hosted. Finally, when the IP address is available, a TCP connection is established to obtain the resource via HTTP. This process is particularly well explained in chapter 6 of the book "Computer networking: a top-down approach" [37].



```
Source            Destination       Protocol Length Info
26:00:00:8b:4c:1f Broadcast         ARP     42 Who has 34.235.250.1? Tell 34.235.250.23
02:00:00:ad:f0:fa 26:00:00:8b:4c:1f ARP     42 34.235.250.1 is at 02:00:00:ad:f0:fa
34.235.250.23     104.244.42.65     DNS     70 Standard query 0xfbc2 A www.uam.es
104.244.42.65     34.235.250.23     DNS     96 Standard query response 0x0aba A www.uam.es A 185.199.108.153
34.235.250.23     185.199.108.153   TCP     54 45136 → 80 [SYN] Seq=0 Win=5199 Len=0
185.199.108.153   34.235.250.23     TCP     54 80 → 45136 [SYN, ACK] Seq=0 Ack=1 Win=6183 Len=0
34.235.250.23     185.199.108.153   TCP     54 45136 → 80 [ACK] Seq=1 Ack=1 Win=5199 Len=0
34.235.250.23     185.199.108.153   HTTP    90 GET / HTTP/1.1
185.199.108.153   34.235.250.23     TCP     54 80 → 45136 [ACK] Seq=1 Ack=37 Win=6183 Len=0
185.199.108.153   34.235.250.23     HTTP   120 HTTP/1.1 200 OK
34.235.250.23     185.199.108.153   TCP     54 45136 → 80 [ACK] Seq=37 Ack=67 Win=5199 Len=0
34.235.250.23     185.199.108.153   TCP     54 45136 → 80 [FIN, ACK] Seq=37 Ack=67 Win=5199 Len=0
185.199.108.153   34.235.250.23     TCP     54 80 → 45136 [ACK] Seq=67 Ack=38 Win=6183 Len=0
185.199.108.153   34.235.250.23     TCP     54 80 → 45136 [FIN, ACK] Seq=67 Ack=38 Win=6183 Len=0
34.235.250.23     185.199.108.153   TCP     54 45136 → 80 [ACK] Seq=38 Ack=68 Win=5199 Len=0
```

Fig. 4. A day in the life of an HTTP connection. Wireshark view of the packets generated for ARP, DNS, TCP and HTTP protocols.

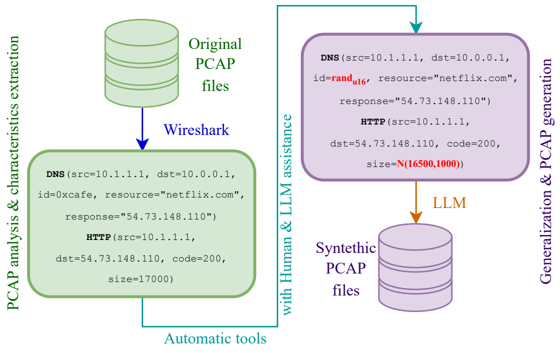
The results obtained from this test have been satisfactory (shown in Figure 4), fully simulating a real situation. Demonstrating that it is possible to logically combine conversations using LLMs. Reviewing the PCAP file, it is observed that, for example, the MAC address of the router obtained in the ARP response is later the one used in the frames that encapsulate the datagrams with an external IP address. In the same way, the IP address obtained in the DNS response is the one used for the establishment of the TCP connection and subsequent HTTP query and response.

Additionally, the random mix in the generation of conversations is tested to evaluate the quality of the datasets that can be generated by the model. The results obtained are very positive (shown in Table 9), obtaining the same results as in the case of the experts, shown in Figure 7, this being logical, given that they are the same models.

Protocol	# conversations	# packets	% Success Rate
IP	467	2131	100%
ICMP	168	336	100%
DNS	166	332	100%
UDP	166	332	100%
ARP	148	296	100%
HTTP	133	266	100%
TCP	133	1463	100%

Table 9
MoE dataset generation summary

6.5 Towards realistic traffic generation


PCAP analysis & characteristics extraction
Original PCAP files
Wireshark
Automatic tools
with Human & LLM assistance
LLM
Synthethic PCAP files
Generalization & PCAP generation

```
DNS(src=10.1.1.1, dst=10.0.0.1,
id=0xcafe, resource="netflix.com",
response="54.73.148.110")
HTTP(src=10.1.1.1,
dst=54.73.148.110, code=200,
size=17000)
```

```
DNS(src=10.1.1.1, dst=10.0.0.1,
id=rand_u16, resource="netflix.com",
response="54.73.148.110")
HTTP(src=10.1.1.1,
dst=54.73.148.110, code=200,
size=N(16500,1000))
```

Fig. 5. Simplified example of realistic network traffic generation

Through extensive testing and evaluation, we have shown that generating PCAP files from text descriptions is possible. This eases the path towards realistic packet generation, as we cover in this section. Our methodology, based on a Chain-of-Thought [38] methodology, enables researchers to expand existing datasets with minimal supervision.

First, the original data should be analyzed to extract both the connections and some metadata of the connections. For that sake, standard tools such as Wireshark/tshark can be employed. This first stage is mostly automatic, with close to no human intervention required. The expected output of each original PCAP file is a description of each connection and some metrics, such as inter-arrival times, packet sizes, etc.

Second, the processed data should be generalized. In that sense, the experiments should be aggregated to characterize the traffic. For example, if a TCP connection from the client to a server with server port 22 is always observed, then it should be generated and have performance metrics akin to the ones of the original PCAP files. For that, the original connection metadata is employed to analyze the distributions of most of the metrics. While this second stage can be mostly unsupervised by following common conventions such as randomly changing the TCP/UDP client ports and using normal distributions to characterize everything, it is true that the attention to the detail makes a difference. Human supervision, or multi-agent LLM environments where an LLM supervises the generation, might have a deeper understanding of the underlying phenomena that affect traffic, such as caching mechanisms, outages, heavy-tailed distributions, etc.

Third, the previous generalization allows us to generate random text descriptions of the traffic that can be fed into our model. As we mentioned before, the better the previous process, the more realistic the traffic would be. This process is completely automatic, with no human intervention. Each of the text descriptions will generate the connections of an artificially generated PCAP file. Lastly, the descriptions are fed into the model to generate scapy/Python code. Then, the code runs to generate the PCAP files.

This whole step-by-step methodology is based on a Chain-of-Thought to overcome the limitations of LLMs. Basically, given the end-problem to the model (i.e., "Generate a PCAP compatible with Netflix video playback") is almost an impossible task for the model directly, you must guide it to smaller and more guided sub-tasks. To summarize the whole process, Figure 5 shows a simplified example of the pipeline to generate realistic data from some PCAP repository.

7. Comparison with other LLMs

The objective of this section is to check if the prompts generated for the GPT-3.5 model can be used with other models that have appeared during the development of this research. In this comparison, it should be considered that as new models are released, they have been trained with greater amounts of data, making them more capable. Therefore, it is logical that the results improve, even without modifying the inputs sent to the models.

The models are selected based on their results on Python coding benchmarks like HumanEval, ClassEval and BigCodeBench among others [39, 40, 41, 42, 43]. The selected models are the pre-deployed models GPT-4o, GPT-4o mini and GPT-o1 from OpenAI, Gemini Pro from Google (all of them accessible through their vendors' API [44, 45]) and Codestral and Mixtral models from Mistral, available on the Huggingface platform (Codestral-22B-v0.1 and Mixtral-8x7B-Instruct-v0.1) to download, customize and run on our own hardware [46].

Protocol	GPT-3.5	GPT-4o	GPT-4o mini	GPT-o1	Codestral	MoE Mixtral
IP	100%	100%	100%	100%	94.89%	76.16%
ICMP	100%	100%	100%	100%	99.21%	98.97%
DNS	81.48%	97.5%	87.21%	84.29%	81.1%	79.5%
UDP	100%	100%	100%	100%	90.9%	89.8%
ARP	100%	100%	100%	100%	99.6%	95.8%
HTTP	99.72%	100%	100%	100%	95.2%	7.4%
TCP	100%	100%	100%	100%	96.42%	35.67%

Table 10
Comparison of *Success Rates* obtained in the packet generation

Protocol	GPT-3.5	MoE GPT-3.5	GPT-4o	GPT-4o mini	GPT-o1	Codestral	MoE Mixtral
ICMP	100%	100%	100%	100%	100%	100%	97.8%
DNS	100%	100%	100%	100%	100%	100%	100%
ARP	100%	100%	100%	100%	100%	100%	93.1%
HTTP	100%	100%	100%	100%	100%	99.3%	21.6%
TCP	100%	100%	100%	100%	100%	96%	32%

Table 11
Comparison of *Success Rates* obtained in the conversation generation

We intend to compare the traffic generation against more of the most advanced models right now. For this purpose, we plan to evaluate the generation of packets and conversations for all the selected models. This test was carried out under the previously established conditions, using the same prompt explanations and making the minimum modifications necessary to adapt the prompt to the different LLMs. In the particular case of the Mistral models, they have been run on a 40 GB Nvidia A100 GPU using 4-bit quantization so that they can fit into the available memory. Quantization is used in LLMs to reduce memory usage and improve processing speed, making them more efficient and cost-effective to run on hardware with limited resources. Despite the reduced precision, advanced techniques help maintain the model's performance, ensuring it can still perform tasks effectively.

Before showing the results, it is necessary to comment that it was impossible to obtain results from these tests on Google's Gemini model. When we tried to make it generate traffic, it was unable to understand the prompts that the OpenAI models could, generating in most iterations non-executable code. Among all the errors, curiously, it stood out the non-completeness of commands, mainly given in DNS commands, since these were the longest and the ones with the largest number of variables. In these, it usually forgot to define variables, left the parentheses unclosed or did not give value to some variables.

The comparison of the results obtained from the different models is shown in Table 10 for packet generation, and Table 11 for conversation generation. The results obtained seem similar to those obtained by GPT-3.5, since these were difficult to overcome. But, diving a little deeper into the generation process, differences in the generation process are observed. The most notable difference observed is that GPT-4o, GPT-o1, Codestral and Mixtral seemed to be trained to return ```` ```python ```` at the start of each code, even if it was specified otherwise. This rendered the code non-executable, so it was necessary to make a modification in the processing of the returned code.

Apart from this problem in code generation, focusing on packet generation, Mixtral obtains the worst results, particularly with HTTP and TCP, where most of the packets have generated incorrectly due to the fact that the model escapes all the underscores (e.g., `Http\_Version` instead of `Http_Version`), which is especially critical in the parameter names of the related Scapy functions. In general, Mistral's models behave worse than OpenAI models for these tasks, being the most common faults that some parameters of the functions are invented.

In the case of DNS single packet generation, where the GPT-3.5 model failed, the newer OpenAI models are better, with GPT-4o substantially better and slightly better in the case of GPT-4o mini and GPT-o1. Specifically, in the DNS protocol, the generation of error packets has been considerably improved. GPT-4o can generate standard Additional Resources based on a word in the summary. On the other hand, HTTP single packet generation capabilities are maintained with OpenAI's newer models.

From these results, we draw two main conclusions: that GPT-4o can react better to situations with little information and that, at the same time, it can better understand human reasoning explained by text. On the other hand, by analyzing the GPT-4o mini results, we can observe that the absolute worsening of these results is not much in comparison with the GPT-4o results. In most of these cases, the smaller model is able to maintain the level of the larger one, being only reduced in the case of DNS. In this case, the reduction is pronounced. As discussed above, this protocol presents the most complex instructions, and the model is not able to correctly process the details related to the different variables in it. In particular, it fails more in the man-

Protocol	Codestral (s)	MoE Mixtral (s)
ICMP	10.21 ± 1.20	18.50 ± 1.50
DNS	26.16 ± 1.39	44.27 ± 2.61
ARP	23.61 ± 1.66	39.63 ± 3.62
HTTP	81.12 ± 3.75	148.24 ± 1.88
TCP	73.37 ± 2.30	112.36 ± 1.96

Table 12
Average generation time and standard deviation of conversations per protocol on GPU

agement of Resource Records, being unable to generate them or to count them correctly.

Contrary to our expectations, it seems that the use of reasoning models such as GPT-o1 does not substantially improve the resolution of the problems attributed to the rest of the models in the generation of DNS traffic. Particularly promising are the results obtained for conversations generation, specially by GPT-4o mini. It can be observed that there is no reduction in conversation generation skills when reducing the size of the model. This result opens the door to the minimization of traffic generation models while maintaining acceptable generation levels.

Regarding the models executed on a local GPU, it is important to note that no significant changes have been made to the prompts from GPT-3.5, so the results are very good, especially for Codestral, taking into account that these models are relatively small compared to those from OpenAI (and they were also quantized). The most interesting thing about this comparison is that the Codestral model, trained to generate code, is much better in most cases than the Mixtral model, which, despite being an MoE and being much larger, it is not as good at code generation. Apart from correct generation of the packets, when checking with Wireshark, certain errors similar to those obtained with the OpenAI models are detected. For example, malformed packets appear in ARP due to the generated MAC, which does not match with the Ethernet source MAC. In DNS results for Mixtral, the request and response transaction numbers do not match in most cases. In HTTP, poorly generated TCP segments appear: segments out of order, lack of acknowledgements, etc. It seems to be a problem more related to TCP generation, particularly for Mixtral.

Finally, regarding conversation generation times, it has been found that generation is faster with GPT-4o models than GPT-3.5 model, being GPT-4o mini the fastest among all of API-based models. As these models are used over the Internet, it is difficult to obtain an average generation time, since the response time varies depending on the server load, etc. However, for models run locally on the GPU, that measurement is easier to calculate using a progress bar (Python's `tqdm` library), which, in addition to showing the progress in the generation of the packets, also shows the average time per iteration. Table 12 shows the average execution time and the standard deviation of the conversations per protocol. It can be seen that the greater the complexity of the protocol, the longer the generation time. The standard deviation values allow us to conclude that the generation time is related to the number of tokens generated, which is not deterministic. The size of the model also affects the generation time. The Mixtral model, even quantized, is larger than the Codestral model, hence the time difference between models. These conclusions can be easily extrapolated to OpenAI models.

8. Discussion

The achievements of this research underscore the potential of LLMs to revolutionize synthetic network traffic generation. Through fine-tuning and prompt engineering, the system demonstrated significant improvements in traffic realism across multiple protocols. A key objective of this research was to simulate multi-protocol network conversations that closely resemble real-world interactions. This was successfully achieved, as indicated by the high success rates in generating realistic traffic for ICMP, ARP, DNS, and HTTP protocols. Notably, the system's ability to handle both simple and complex protocols marks a major step forward in traffic generation technologies.

One of the primary accomplishments is the introduction of a Mixture of Experts (MoE) architecture, which improved the model's accuracy in generating protocol-specific traffic. By partitioning the traffic generation process into distinct expert models, the system was able to create more precise and context-aware network traffic. This architecture also facilitated the generation of more complex network interactions, such as those involving ARP, DNS and TCP connections. Additionally, the use of fine-tuning allowed for greater flexibility in handling different traffic generation tasks.

The results also highlight the effectiveness of LLMs in addressing limitations found in previous traffic generation models, particularly in handling dynamic network conditions. This study confirms that transformer-based models can not only generate traffic at the packet level but also sustain conversations across different layers of the OSI model, enhancing their utility in both performance evaluation and security testing environments.

In conclusion, this research demonstrates that the integration of LLMs in network traffic generation represents a promising direction for future work. We showcased our methodology with popular network protocols, but same procedure can be extended to less common protocols in many areas depending on the domain of the data. The ability to generate realistic and adaptable network traffic provides significant benefits for network simulation, cybersecurity, and testing. Further research could explore additional protocols and refine the prompt engineering techniques used, while the inclusion of more diverse data sources could enhance the system's generalizability to different network environments.

9. Conclusions

The potential of AI and LLMs in network traffic engineering is vast, with applications ranging from network and service management to traffic analysis and generation. These models have already proven their effectiveness in areas such as protocol prediction, network data interpretation, and packet creation;

showcasing their adaptability to address complex engineering issues. Ongoing research and development are likely to reveal even more capabilities, amplifying the impact of LLMs on computer networks and communications. By improving model designs, expanding training datasets, and exploring explicitly tailored models for network applications, researchers can fully leverage these advanced tools to enhance network engineering and accelerate scientific advancements.

In addition, incorporating LLMs into network analysis processes has the potential to revolutionize communications research. These models provide researchers with extensive access to network data, enabling them to uncover hidden trends, generate new insights, and make more precise, data-driven decisions. As a result, LLMs are poised to transform network research, helping to advance solutions to existing challenges in the field.

However, there are some issues that should be taken into account when working with these tremendously powerful and promising technologies. Ethical considerations must also be properly considered, addressing issues concerning misuse of the tool in order to commit cybercrime. Safeguarding network security and upholding ethical data collection, storage, and analysis standards are paramount to ensuring the responsible and ethical use of LLMs in network engineering.

Moreover, on top of all the model challenges lies the interpretation of the tool. Interpretability remains a key concern, as the internal mechanisms of large language models are often opaque and challenging to understand [47]. Gaining insight into how these models make predictions is essential for ensuring the reliability and trustworthiness of their outputs.

Resources

All presented resources, datasets, and prompts are available at: `https://github.com/javieradelgado/GPT-on-the-wire`.

Acknowledgement

This work was supported in part by the Spanish Research Agency under the project AgileMon (AEI PID2019-104451RB-C21), by the Comunidad de Madrid under the project RAMONES-CM (TEC-2024/COM-504) and by the Comunidad de Madrid and the Universidad Autónoma de Madrid under the project LINO (SI4/PJI/2024-00221).

References

[1] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, A. Hotho, A survey of network-based intrusion detection data sets, Computers & Security 86 (2019) 147–167.

[2] L. A. H. Ahmed, Y. A. M. Hamad, A. A. M. A. Abdalla, Network-based intrusion detection datasets: A survey, in: 2022 International Arab Conference on Information Technology (ACIT), 2022, pp. 1–7. doi:10.1109/ACIT57182.2022.9994201.

[3] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, A. Pescapé, AI-powered internet traffic classification: Past, present, and future, IEEE Communications Magazine 62 (9) (2024) 168–175. doi:10.1109/MCOM.001.2300361.

[4] G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapé, Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges, IEEE Transactions on Network and Service Management 16 (2) (2019) 445–458. doi:10.1109/TNSM.2019.2899085.

[5] G. Bovenzi, F. Cerasuolo, A. Montieri, A. Nascita, V. Persico, A. Pescapé, A comparison of machine and deep learning models for detection and classification of android malware traffic, in: 2022 IEEE Symposium on Computers and Communications (ISCC), 2022, pp. 1–6.

[6] G. Bovenzi, D. Di Monda, A. Montieri, V. Persico, A. Pescapè, Classifying attack traffic in iot environments via few-shot learning, Journal of Information Security and Applications 83 (2024) 103762. doi:10.1016/j.jisa.2024.103762.

[7] G. Agrawal, A. Kaur, S. Myneni, A review of generative models in generating synthetic attack data for cybersecurity, Electronics 13 (2). doi:10.3390/electronics13020322.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017, pp. 6000 – 6010.
URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[9] D. Perdices, J. Ramos, J. L. García-Dorado, I. González, J. E. López de Vergara, Natural language processing for web browsing analytics: Challenges, lessons learned, and opportunities, Computer Networks 198 (2021) 108357. doi:10.1016/j.comnet.2021.108357.

[10] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, J. Yu, ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification, in: Proceedings of the ACM Web Conference 2022, WWW '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 633–642. doi:10.1145/3485447.3512217.

[11] J. Yang, X. Jiang, Y. Lei, W. Liang, Z. Ma, S. Li, MTSecurity: Privacy-preserving malicious traffic classification using graph neural network and transformer, IEEE Transactions on Network and Service Management 21 (3) (2024) 3583–3597. doi:10.1109/TNSM.2024.3383851.

[12] M. Boffa, G. Milan, L. Vassio, I. Drago, M. Mellia, Z. Ben Houidi, Towards NLP-based processing of honeypot logs, in: 2022 IEEE European Symposium on Security and Privacy Workshops, 2022, pp. 314–321.

[13] M. Boffa, L. Vassio, M. Mellia, I. Drago, G. Milan, Z. B. Houidi, D. Rossi, On using pretext tasks to learn representations from network logs, in: Proceedings of the 1st International Workshop on Native Network Intelligence, NativeNi '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 21–26. doi:10.1145/3565009.3569522.

[14] Wireshark Foundation, Wireshark, https://www.wireshark.org/ (2024).

[15] W. Chu, X. Guan, Z. Cai, L. Gao, Real-time volume control for interactive network traffic replay, Computer Networks 57 (7) (2013) 1611–1629. doi:10.1016/j.comnet.2013.02.012.

[16] M. Ruiz, M. Ruiz, F. Tabatabaeimehr, L. Gifre, S. López-Buedo, J. E. López de Vergara, Ó. González, L. Velasco, Modeling and assessing connectivity services performance in a sandbox domain, Journal of Lightwave Technology 38 (12) (2020) 3180–3189. doi:10.1109/JLT.2020.2975641.

[17] M. Ring, D. Schlör, D. Landes, A. Hotho, Flow-based network traffic generation using generative adversarial networks, Computers & Security 82 (2019) 156–172. doi:10.1016/j.cose.2018.12.012.

[18] S. K. Nukavarapu, M. Ayyat, T. Nadeem, Miragenet - towards a gan-based framework for synthetic network traffic generation, in: GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 2022, pp. 3089–3095. doi:10.1109/GLOBECOM48099.2022.10001494.

[19] O. A. Adeleke, N. Bastin, D. Gurkan, Network traffic generation: A survey and methodology, ACM Comput. Surv. 55 (2). doi:10.1145/3488375.

[20] Y. W. Xuying Meng, Chungang Lin, Y. Zhang, NetGPT: Generative pretrained transformer for network traffic, arXiv.
URL https://arxiv.org/pdf/2304.09513

[21] G. Bovenzi, F. Cerasuolo, D. Ciuonzo, D. Di Monda, I. Guarino, A. Montieri, V. Persico, A. Pescapè, Mapping the landscape of generative ai in network monitoring and management, arXiv preprint arXiv:2502.08576.

[22] F. Meslet-Millet, S. Mouysset, E. Chaput, Necstgen: An approach for realistic network traffic generation using deep learning, in: GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 2022, pp. 3108–3113. doi:10.1109/GLOBECOM48099.2022.10000731.

[23] A. Cheng, PAC-GAN: Packet generation of network traffic using generative adversarial networks, in: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2019, pp. 0728–0734. doi:10.1109/IEMCON.2019.8936224.

[24] Y. Yin, Z. Lin, M. Jin, G. Fanti, V. Sekar, Practical gan-based synthetic ip header trace generation using netshare, in: Proceedings of the ACM SIGCOMM 2022 Conference, 2022, pp. 458–472.

[25] X. Jiang, S. Liu, A. Gember-Jacobson, A. N. Bhagoji, P. Schmitt, F. Bronzino, N. Feamster, Netdiffusion: Network data augmentation through protocol-constrained traffic generation, Proceedings of the ACM on Measurement and Analysis of Computing Systems 8 (1) (2024) 1–32.

[26] A. Chu, X. Jiang, S. Liu, A. Bhagoji, F. Bronzino, P. Schmitt, N. Feamster, Feasibility of state space models for network traffic generation, in: Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing, 2024, pp. 9–17.

[27] D. K. Kholgh, P. Kostakos, PAC-GPT: A novel approach to generating synthetic network traffic with gpt-3, IEEE Access 11 (2023) 114936–114951. doi:10.1109/ACCESS.2023.3325727.

[28] J. Qu, X. Ma, J. Li, TrafficGPT: Breaking the token barrier for efficient long traffic analysis and generation (2024). arXiv:2403.05822.

[29] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.
URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[30] dark 0ne, Networkpacketgenerator - gpt-3 commands notebook, https://github.com/dark-0ne/NetworkPacketGenerator/blob/main/gpt3/notebooks/davinci_commands_gen.ipynb (2023).

[31] University of New South Wales, The TON_IoT datasets, https://research.unsw.edu.au/projects/toniot-datasets (2021).

[32] J. Weber, The ultimate PCAP, https://weberblog.net/the-ultimate-pcap/ (2020).

[33] P. Biondi, Scapy: A python library for packet manipulation, https://scapy.net/ (2024).

[34] OpenAI, Fine-tuning guide, https://platform.openai.com/docs/guides/fine-tuning (2023).

[35] O. D. Community, How many epochs for fine-tunes?, https://community.openai.com/t/how-many-epochs-for-fine-tunes/7027 (2021).

[36] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton, Adaptive mixtures of local experts, Neural Computation 3 (1) (1991) 79–87. doi:10.1162/neco.1991.3.1.79.

[37] J. F. Kurose, Computer networking : a top-down approach, Pearson Addison Wesley, Boston, 2010.

[38] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22, Curran Associates Inc., Red Hook, NY, USA, 2022, pp. 1–14.

[39] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, W. Zaremba, Evaluating large language models trained on code (2021). arXiv:2107.03374.
URL https://arxiv.org/abs/2107.03374

[40] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, Y. Lou, Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation (2023). arXiv:2308.01861.
URL https://arxiv.org/abs/2308.01861

[41] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, S. Brunner, C. Gong, T. Hoang, A. R. Zebaze, X. Hong, W.-D. Li, J. Kaddour, M. Xu, Z. Zhang, P. Yadav, N. Jain, A. Gu, Z. Cheng, J. Liu, Q. Liu, Z. Wang, D. Lo, B. Hui, N. Muennighoff, D. Fried, X. Du, H. de Vries, L. V. Werra, Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions (2024). arXiv:2406.15877.
URL https://arxiv.org/abs/2406.15877

[42] M. Ravkine, Can AI code? - results, https://huggingface.co/spaces/mike-ravkine/can-ai-code-results (2023).

[43] State-of-the-art code generation on humaneval, https://paperswithcode.com/sota/code-generation-on-humaneval (2024).

[44] OpenAI, Api platform (2024).
URL https://openai.com/api/

[45] Google, Gemini (2024).
URL https://gemini.google.com/?hl=es-ES

[46] Hugging Face, Hugging face – the ai community building the future., https://huggingface.co/ (2024).

[47] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, M. Du, Explainability for large language models: A survey, ACM Trans. Intell. Syst. Technol. 15 (2). doi:10.1145/3639372.

Javier Aday DELGADO-SOTO is a public servant at the Secretary of State for Digitalization and Artificial Intelligence, Ministry for Digital Transformation and Public Administration. He received his B.Sc. and M.Sc. degrees in Telecommunication Engineering from Universidad Autónoma de Madrid (Spain) in 2021 and 2024, respectively. He is currently enrolled in the AI Ph.D. program at Universidad Politécnica de Madrid (Spain). His research interests include machine learning applied to communication networks, cybersecurity and bio-engineering.

Jorge E. LÓPEZ DE VERGARA is associate professor at Universidad Autónoma de Madrid (Spain) since 2007 and accredited as full professor since 2023. He was a founding partner from 2009 to 2023 of Naudit HPCN, a spin-off company devoted to high-performance traffic monitoring and analysis. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1998 and 2003, respectively, where he also held an FPU-MEC research grant. During his Ph.D., he stayed for 6 months in 2000 at HP Labs in Bristol, U.K. He studies network and service management and monitoring, and has coauthored more than 100 scientific papers on topics related to this field.

Iván GONZÁLEZ received his M.Sc. degree in Computer Engineering in 2000 and his Ph.D. in Computer Engineering in 2006, both from UAM, Spain. From October 2002 to October 2006, he was a teaching assistant at the Computer Engineering Department of UAM. From November 2006 to January 2008, he was a postdoctoral research scientist at the High-Performance Computing Laboratory (HPCL), Electrical & Computer Engineering Department, George Washington University (Washington, DC). He was a faculty member of the NSF Center of High-Performance Reconfigurable Computing (CHREC) at George Washington University. He is currently associate professor at UAM, where he is teaching computer-architecture-related courses. He was also partner of Naudit HPCN. His main research interests are heterogeneous computing (with GPUs, FPGAs, etc.), parallel algorithms, and performance tuning. Other interests include big data, machine learning and data analytics.

Daniel PERDICES is an assistant professor at Universidad Autónoma de Madrid (Spain) since 2023. He previously held an FPU research scholarship by the Spanish Ministry of Science, Innovation, and Universities. Previously, he was an R&D engineer at Naudit HPCN. He received the B.Sc. (Honors) degrees in Mathematics and in Computer Science (2018), the M.Sc. in Mathematics (2019), the M.Sc. in Information and Communications Technologies (2020), and Ph.D. in Computer Science and Telecommunication Engineering (2023), all at Universidad Autónoma de Madrid (Spain). He was a visiting scholar in 2022 for three months at SmartData@PoliTO, Politecnico di Torino (Italy). He researches on statistics, mathematical modeling, machine learning, and artificial intelligence applied but not limited to network traffic analysis.

Luis DE PEDRO is associate professor at Universidad Autónoma de Madrid (Spain) since 1994, and has been president and founding partner from 2009 to 2023 of Naudit HPCN, a spin-off of the UAM, devoted to high performance traffic monitoring and analysis. He received his M.Sc. (Honors) and Ph.D. (Honors) degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1987 and 1992, respectively. He has been a Hewlett-Packard Executive from 1986 to 2019. He currently researches on statistical models for network traffic.