Unveiling Cyber Threat Actors: A Hybrid Deep Learning Approach for Behavior-Based Attribution

EMIRHAN BÖGE, Sabanci University, Istanbul, Türkiye
MURAT BILGEHAN ERTAN, Vrije Universiteit Amsterdam, Amsterdam, Netherlands
HALIT ALPTEKIN, PRODAFT, Yverdon-les-Bains, Switzerland
ORÇUN ÇETIN, Sabanci University, Istanbul, Türkiye

In this article, we leverage natural language processing and machine learning algorithms to profile threat actors based on their behavioral signatures to establish identification for soft attribution. Our unique dataset comprises various actors and the commands they have executed, with a significant proportion using the Cobalt Strike framework in August 2020–October 2022. We implemented a hybrid deep learning structure combining transformers and convolutional neural networks to benefit global and local contextual information within the sequence of commands, which provides a detailed view of the behavioral patterns of threat actors. We evaluated our hybrid architecture against pre-trained transformer-based models such as BERT, RoBERTa, SecureBERT, and DarkBERT with our high-count, medium-count, and low-count datasets. Hybrid architecture has achieved F1-score of 95.11% and an accuracy score of 95.13% on the high-count dataset, F1-score of 93.60% and accuracy score of 93.77% on the medium-count dataset, and F1-score of 88.95% and accuracy score of 89.25% on the low-count dataset. Our approach has the potential to substantially reduce the workload of incident response experts who are processing the collected cybersecurity data to identify patterns.

CCS Concepts: • **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies** → **Machine learning**; **Neural networks**; **Natural language processing**; **Artificial intelligence**;

Additional Key Words and Phrases: Digital forensics, threat intelligence, machine learning, threat actor attribution, deep learning, natural language processing

ACM Reference format:
Emirhan Böge, Murat Bilgehan Ertan, Halit Alptekin, and Orçun Çetin. 2025. Unveiling Cyber Threat Actors: A Hybrid Deep Learning Approach for Behavior-Based Attribution. *Digit. Threat. Res. Pract.* 6, 1, Article 2 (February 2025), 20 pages.
https://doi.org/10.1145/3676284

1 Introduction

Cyber threat actor (CTA) attribution, a critical aspect of **cyber threat intelligence (CTI)** and digital forensics, thus incident response research, is the process of identifying the responsible party or actor behind a particular cyber-attack. The significance of attribution lies in determining the origin of past attacks and enabling predictive

Emirhan Böge and Murat Bilgehan Ertan contributed equally to this research.

Authors' Contact Information: Emirhan Böge, Sabanci University, Istanbul, Türkiye; e-mail: emirhanboge@sabanciuniv.edu; Murat Bilgehan Ertan (corresponding author), Vrije Universiteit Amsterdam, Amsterdam, Netherlands; e-mail: m.b.ertan@student.vu.nl; Halit Alptekin, PRODAFT, Yverdon-les-Bains, Switzerland; e-mail: halit@prodaft.com; Orçun Çetin, Sabanci University, Istanbul, Türkiye; e-mail: orcun.cetin@sabanciuniv.edu.

ACM 2576-5337/2025/2-ART2
https://doi.org/10.1145/3676284

analyses for potential future attacks. One dimension of CTA attribution that enhances its accuracy and relevance is the incorporation of behavioral signatures and threat actor profiling.

Behavioral signatures refer to distinct patterns of activities that can be associated with a specific threat actor. On the other hand, threat actor profiling involves creating comprehensive profiles of cybercriminals based on a range of attributes, including but not limited to their behavioral signatures, tools used, **tactics, techniques, and procedures (TTPs)**. Profiling not only identifies the actors but also establishes definitive guidelines for attribution. Thus, a combination of behavioral signatures and threat actor profiling makes attribution more precise and actionable.

One of the primary problems with cybercriminal attribution and digital forensics is the quality of the threat intelligence data. Some analysts perceive many threat intelligence feeds to be of varying quality. They may sometimes lack richness, meaning they may not contain enough detailed information to establish reliable behavioral signatures or construct comprehensive profiles. Consequently, analysts must put in substantial effort to process and make sense of the collected intelligence often combing through sparse and noisy data to identify key indicators of attribution. Hence, integrating behavioral signatures and threat actor profiling could enrich the attribution process and make it more efficient and effective.

A crucial challenge in CTI research is the need for a professional and automated approach to assigning a threat to a specific actor or group, using the commands carried out by that threat actor as a basis. Digital forensics experts can significantly benefit from attributing threats to specific actors based on command analysis. By studying the unique commands and sequences a threat actor uses, experts can discern distinct behavioral patterns that act like digital fingerprints. This facilitates quicker and more accurate investigations by offering insights into the attacker identification. Previously, threat actor attribution was done by various methods, and none of this research utilized the sequence of commands executed by the threat actors. For instance, a study uses high-level indicators of compromise, such as TTPs and malware [17]. At the same time, some studies have also used CTI reports to attribute cybercriminals [10]. In this research, we refrain from being restrained by TTPs or high-level indicators as CTAs have developed new strategies to make it hard for analysts to detect them [25].

Attributing CTAs using the sequence of commands they have executed can provide valuable insight into their behavioral processes, which can ease the attribution process. However, detecting CTAs based on the commands they have executed is not a feasible task to be accomplished by most analysts. This process needs not only experience but also lots of workload and time. Therefore, there needs to be a study on CTA attribution based on the actions and commands of the threat actors. We aim to fill this gap by providing CTA attribution based on the sequence of commands they have executed. Our methodology emphasizes the importance of soft attribution, as it allows us to profile and categorize threat actors based on observed behaviors without needing the concrete evidence required for hard attribution. This approach is particularly effective in a landscape where direct evidence can be elusive, and attackers often use sophisticated methods to conceal their identities. Moreover, an article from Virus Bulletin draws attention to the method of *attack replication*, which is an advanced defense technique that consists of mimicking the attacker completely, including the command that the attacker has executed, and stating that this is one of the most powerful yet least adopted tool [5]. In our study, even though it is not based entirely on attack replication, we are leveraging the original commands that the attacker executes not the replica ones. Our approach empowers a more informed approach to decision-making, providing a valuable starting point and a strategic direction for the CTA attribution process. Organizations can adapt this research into their infrastructure to attribute the CTAs that they have encountered.

The dataset used in this study comprises unique identifiers of threat actors. Threat actors and CTA groups use various frameworks, commands, and malware to facilitate their activities. In our case, a high portion of the data consists of commands executed using the Cobalt Strike. The sequences of commands executed by CTAs vary significantly, with the volume of commands ranging from as many as 1,000 to as few as 15 per threat actor. This imbalance in dataset sizes across different CTAs may negatively affect the performance of the trained deep learning models due to the potential for both overfitting in data-rich scenarios and underfitting in data-sparse scenarios. To address this challenge and overcome its impact on model robustness and generalization, we partitioned the data into three distinct groups: high-count, medium-count, and low-count datasets. This categorization enables targeted training and validation approaches that accommodate the diverse data volumes, ensuring that our models are not biased towards the characteristics of more extensively represented data. The decision to create separate datasets instead of a single bucketed dataset, which is the low-count dataset, was strategic, aiming to ensure that our evaluation metrics genuinely reflect the model's ability to perform well under varied data conditions. By adopting this approach, we aim to enhance model performance across varied operational contexts, which reflects more realistic conditions that organizations might encounter. Further details on the methodology and its justification are provided in Section 5.3, emphasizing our choices to maintain integrity and reliability in model assessment.

Additionally, we have developed a standardization method to simplify the unprocessed commands into a **standardized command language (SCL)** for deep learning models, which significantly improved the performance of the model. This standardization process involves converting variations of commands, which might differ due to syntactic discrepancies or the unique ways in which different threat actors format their commands, into a consistent structure. Moreover, this standardization helps in mitigating issues like overfitting, where a model trained on highly specific command formats might perform poorly when exposed to slightly different commands in practice. By abstracting the commands to a more generalized form, the models can better generalize from the training data to real-world scenarios, where the exact wording of commands might vary, but the underlying intent or action remains the same.

In this study, we leverage **natural language processing (NLP)** architectures such as transformer [29], which learns from a sequence of tokens to train a machine learning model. These techniques learn from the positions of the tokens and can understand the underlying relationship between these tokens. Understanding the relationship between the sequence of commands and their relations benefits the attribution process as it uncovers their hidden features. Additionally, we have further included different deep learning [12] methods, merging the strengths of the transformer architecture with **convolutional neural networks (CNN)** [13]. This article presents a uniquely tailored hybrid deep learning architecture that integrates various methods specifically designed for the complex task of attributing threat actors. We compared our hybrid deep learning architecture against notable pre-trained models. Namely, **Bidirectional Encoder Representations from Transformers (BERT)** [4], RoBERTa [14], SecureBERT [1], and DarkBERT [11].

Our main contributions in this article are:

— We present the first study for analyzing and attributing the malicious commands executed by threat actors in compromised machines, which is usually an inefficient and time-consuming process that digital forensics or incident response specialists do.
— We have created a **Standardized Command Language Converter (SCLC)** that converts unprocessed commands into a standardized language to improve the efficiency and accuracy of NLP architectures and reduce overfitting.
— We develop a hybrid deep learning architecture for threat actor attribution, which outperforms existing pre-trained transformer-based models like BERT, RoBERTa, SecureBERT, and DarkBERT. The hybrid architecture achieved F1-score of 95.11% and accuracy score of 95.13% on the high-count dataset, F1-score of 93.60% and an accuracy score of 93.77% on the medium-count dataset, and F1-score of 88.95% and an accuracy score of 89.25% on the low-count dataset.[1]

This article is structured as follows: in Section 2, we mention the previous studies about the attribution process. Later in Section 3, we give background information about the techniques we have used. Section 4 explains the dataset, including its source and content. Section 5 explains the implementation methods for this study, which are the SCL component that processes our dataset, the proposed hybrid deep learning architecture, and the data

[1]https://github.com/bogertaNET/Unveiling-CTAs

preparation step for experimentation. Lastly, in Sections 6 and 7, we first show the results of our experiments, explain, discuss them, and point out some limitations.

2 Related Works

Traditionally, manual analysis is the most common approach for attribution [19]. As far as we are aware, our study is the first to propose a CTA attribution based on the *sequence of commands* that these actors have executed using various NLP and machine learning techniques. Nonetheless, there have been several attempts at threat actor attribution using machine learning and deep learning techniques based on the actor's behavior.

Recent studies proposed the idea of finding similarities between malware, hence, de-anonymizing the CTA based on the source code and the behavior of the malware [2, 20]. Moreover, Rosenblum et al. [23] unfold the idea of authorship attribution of program binaries using stylistic similarities of authors between programs using machine learning techniques.

Noor et al. [17] conducted a study utilizing NLP and machine learning techniques to analyze CTI reports. Their main goal was to identify and profile CTAs targeting FinTech based on the specific patterns of their attacks. To accomplish this, they applied an NLP technique known as distributional semantics. They trained and evaluated a variety of machine learning and deep learning models using these publicly available CTI reports. Remarkably, the deep learning model they developed achieved an accuracy of 94%. Irshad and Siddiqui [10] have also focused on attributing cyber-threat actors using NLP and machine learning techniques to analyze unstructured CTI reports. With cyber-attackers using obfuscation and deception to hide their identities, the researchers aimed to develop an automated system for extracting features from these reports to profile and attribute the attacks. These features include tactics, techniques, tools, malware, and target information. They use an embedding model called "Attack2vec," trained on domain-specific embeddings. Various machine learning algorithms, such as decision tree, random forest, and support vector machine, are utilized for classification. The model achieved an accuracy of 96%. Perry et al. [19] proposed a method for attack attribution that involves the textual analysis of threat intelligence reports, employing NLP and machine learning techniques. The researchers developed a unique text representation algorithm capable of capturing contextual information. Their approach utilizes vector space representation of incident reports obtained from a blend of labeled reports and an extensive corpus of security literature. Upon the previous study, a new study by Puzis and Angappan [24] emerged to attribute threat actors based on similar CTI reports. Unlike the traditional machine learning methods that focus on analyzing malware samples, this research proposes a deep learning architecture for the task of attribution [24]. The authors use the same dataset as Perry et al. used in their research [19].

Han et al. [6] developed a method to detect the unique characteristics of **Advanced Persistent Threats (APTs)**. These threats are notorious for their *low-and-slow* attack patterns. To identify these anomalies, they leveraged provenance graphs, which offer rich contextual and historical information [6]. Regarding APT detection, Milajerdi et al. [16] introduced HOLMES, yet another system for APT detection, by leveraging the correlation of suspicious information flows that arise during an attack. Moreover, while Han et al. and Milajerdi et al. focus primarily on detecting APT groups, our approach can detect any threat actor present within our dataset. Notably, we harness the power of NLP for our detection process, a technique not used by these studies.

There have also been various studies that mentioned the importance of replication of the attack. Guerrero-Saade presented the idea that attack replication enables extreme opportunities in defense and offense [5]. However, it is essential to mention that this study does not directly relate to the replication of the attack; instead, it focuses on the actual attack logs. For instance, the dataset used only consists of the executed commands in an attack, nothing more; because of that, our model can only learn which commands the actors execute and their style of execution. Guerrero-Saade also mentions some profiling characteristics that are helpful for actor attribution and actor profiling. While there is no strict guideline for profiling, and it consists of various aspects including but not limited to technical aspects and social aspects of the actor, it indeed includes the commands that are executed and the execution style of the commands [5] which is the backbone of this research, meaning, our model is already paying attention to the traits that are important to the profiling.

3 Background

This section of the article is dedicated to providing a foundation of the core concepts and a more comprehensive understanding of the context of our research while underlining the specific challenges and considerations within CTA attribution research.

3.1 Transformer Architecture

The transformer architecture was first proposed by Vaswani et al. [29]. They differ from previous NLP techniques as they are much more capable of handling long-range dependencies. Transformer uses an attention mechanism that helps the model create global dependencies between the input and output. This architecture is composed of two main components. Namely, an encoder and a decoder and both of these components have self-attention and position-wise feed-forward networks. The self-attention mechanism replaces the idea of recurrence, which was used primarily in the previous NLP architectures such as long short-term memory [9]. This mechanism of the transformer architecture allows the developed model to address the problem of long-range dependencies by weighing the importance of different inputs for generating each output in the sequence. In addition, a positional encoding mechanism was used in the transformer architecture because, unlike previous NLP architectures, transformers do not have a recurrent mechanism that enables the architecture to take the order of the sequence into account, which is a crucial aspect of NLP tasks. The positional encoding provides the sequence information to the transformer architecture by adding a unique signal to each input token, representing its position in the sequence. This allows the architecture to learn and generalize about positional relationships between tokens.

In this study, transformers can help detect CTAs based on their commands used by attackers. Command sequences that actors have executed often contain long dependencies where the meaning of a command might be influenced by a command executed long ago. Therefore, transformer architecture is used in our hybrid deep learning architecture. The transformer architecture has also been used for training larger language models such as BERT [4], RoBERTa [14], SecureBERT [1], and DarkBERT [11]. These models have been used as a baseline for specific tasks as they have been trained on a large corpus of texts. Using these pre-trained transformer-based models in a specific task requires fine-tuning the model with the new dataset for that specific task's domain. In our case, this dataset is the CTA's command sequences.

3.2 CNNs

CNNs [13] are a deep learning algorithm primarily used for processing structured grid data. CNNs have been most influential in image and video recognition applications. The core component of this architecture is the convolution operation, which allows the model to learn spatial hierarchies of features. For instance, features in the early layers can be edges and textures, and in late layers, features can be parts of objects.

To analyze CTA command sequences, we employ **one-dimensional CNNs (Conv1D)**. This approach captures local and contextual features from command sequences by sliding filters across the text. Utilizing varying filter sizes, which correspond to different n-gram analyses, the network becomes more able to extract diverse semantic and syntactic features that are indicative of threat actor behaviors. In our case, a CTA's commands can be considered text. Therefore, these sequences may have patterns that can give insights into detecting characteristics of the behavior of a threat actor, and 1D CNNs can extract these patterns.

3.3 Residual Connections

Residual connections, introduced by He et al. [7], are implemented by allowing the output of an earlier layer to skip some layers and be added to the output of a later layer. This is valuable because it mitigates the "vanishing gradient" problem that usually arises while training complex models. The vanishing gradient problem refers to gradients becoming extremely small in deeper layers of the trained model, which can slow down the training process. Residual connections alleviate this problem by enabling gradients to flow through from earlier layers to later layers of the model, improving the effectiveness of training in even very deep networks.

3.4 Regularization Techniques

Overfitting is a common problem that must be addressed, which happens when a model learns the training data too well and performs poorly on the unseen data. This can be mitigated with regularization techniques such as dropout [26] and weight decay [15].

Dropout is a regularization technique that randomly sets a fraction of input units to zero, which helps prevent overfitting, leading to a general and robust model. Moreover, spatial dropout [27] is a specific type of dropout used in CNNs that preserves the spatial coherency of the input data, which allows the model to learn the spatially local features. In our case, spatial dropout can be beneficial as it is much more capable of helping the architecture capture meaningful localized features. For example, the pixels in an image may form meaningful patterns in a local context, which can be edges, textures, and shapes. Similarly, local patterns may have significant meaning in command sequences, such as command flags, arguments, and their sequential order. Additionally, weight decay is another technique to mitigate the overfitting problem. This method prevents weights of the model from reaching large values that may lead to overfitting by adding a penalty term to the loss function.

3.5 Hyperparameter Tuning

Hyperparameters are parameters set before training the model that alter how the model works. For instance, they include dropout fraction, weight decay, learning rate, number of epochs that the model has to be trained, and many others. The process of deciding hyperparameters is crucial as it affects the model significantly.

3.6 Cross Validation

Cross validation technique is used for assessing the effectiveness and robustness of a model. First, we partition the dataset into k subsets. One of these subsets is used for validation, and the rest $k-1$ subsets are used for training the model. This process is repeated k times, each time updating the subset used as the validation set, and the training set as the rest. Finally, k results are averaged to produce a more robust metric. In addition, the standard deviation of the performance across the k folds is calculated, where a low standard deviation indicates that the model is robust and consistent across different subsets of data.

3.7 Metrics for Evaluation

We leverage two metrics for evaluating our model: accuracy and weighted F1-score. Accuracy simply measures the proportion of correctly classified instances.

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}. \tag{1}$$

F1-score is a useful metric when dealing with imbalanced classes. It is the harmonic mean of precision and recall, which are calculated based on true positives, false positives, and false negatives.

$$F1\ score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{2}$$

For multi-class problems like ours with more than 2 CTAs, we calculate weighted average of F1-scores.

$$Weighted\ F1\ score = \frac{1}{N} \sum_{i=1}^{N} w_i \cdot F1\ score_i. \tag{3}$$

Table 1. Command Data

Field	**Type**	**Example**
data	string	execute_cmd: net group
timestamp	UNIX Epoch in seconds	1646158272

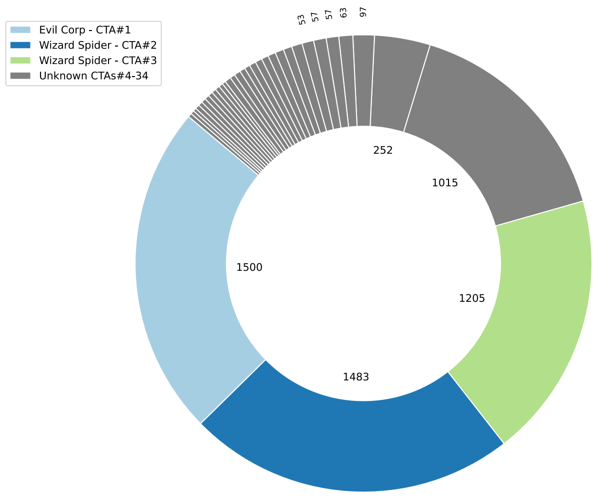

Fig. 1. Commands per CTA.

4 Dataset

European Threat Intelligence company ***Proactive Defense Against Future Threats (PRODAFT)*** provided private threat intelligence data collected from August 2020 to October 2022. Commands executed by the attackers were captured by *PRODAFT* using various private honeypot servers located in China, Europe, and the United States. The aim is to lessen the workload of cybersecurity analysts and facilitate progress in the field of CTI.

Dataset contains several attack frameworks, such as the Cobalt Strike. The raw form of the data consists of lists of commands executed by CTAs along with the exact time that is executed. The data is already clustered with CTA's unique identifications, such that the data are labeled with CTA's aliases. For example, Table 1 shows a sample command structure and its content. The command execution JavaScript Object Notation data consists of two fields, namely *data* and *timestamp*. The *data* field contains the information and commands executed by CTAs. On the other hand, the *timestamp* field is designed to hold UNIX epoch timestamp data.

The dataset mainly contains 34 CTAs with over 5,000 commands. The number of command sequences that the dataset contains for each CTA is not uniform, meaning while CTA#1 has 1,500 commands, CTA#25 only has 21 command sequences, as Figure 1 displays. Only one uses a custom malware, called *SocGholish*, and the rest of the threat actors use the Cobalt Strike to execute commands. The variety of malware is intended since we would like to demonstrate that our architecture can predict threat actors that use different malware and can also predict distinct actors even though they use the same malware. The top three CTAs in our dataset are attributed to various known threat actor groups. Their details can be seen in Table 2; affiliation of the rest of the CTAs is unknown, yet it is known that they are distinct.

More importantly, *PRODAFT* stated with strong confidence that one of the threat actors is attributed to the *SilverFish* group. This group has strong ties with the notorious SolarWinds incident and the globally recognized *Evil Corp* group [22]. Moreover, as the Health Sector Cybersecurity Coordination Center (HC3) of the U.S. claims,

Table 2. Details of the Important CTAs

Threat Actor	**# of Command Sequences**	**Malware**
Evil Corp—CTA#1	1500	SocGholish
Wizard Spider—CTA#2	1483	Cobalt Strike
Wizard Spider—CTA#3	1205	Cobalt Strike

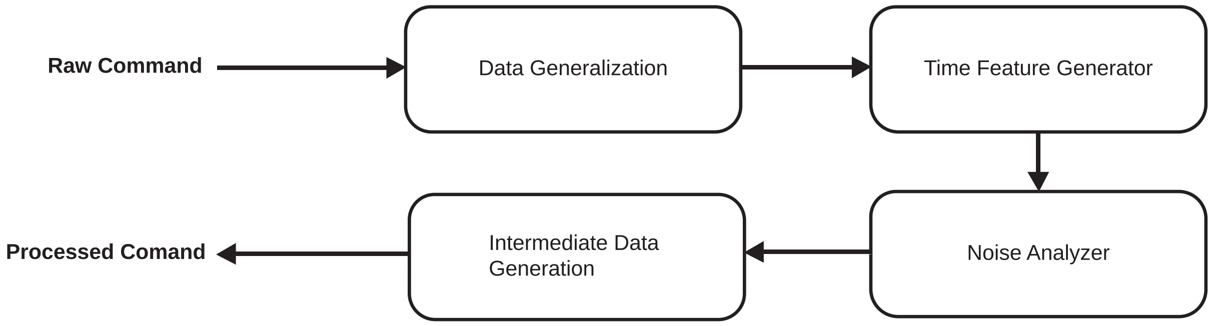

Fig. 2. SCLC schema.

Evil Corp is one of the most sophisticated cybercriminal syndicates in which they developed and operated several critical malware and ransomware variants, such as the *Dridex* malware that caused more than $100 million in theft [8]. In addition, two of the threat actors are also a subgroup of a threat actor group called *Wizard Spider* [21], and one of them is affiliated with group *Royal Ransomware* [3].

5 Methodology

5.1 SCL

Overfitting is a phenomenon used to describe when a developed machine learning model performs well on the dataset used to train the model but performs poorly on a dataset that the model has not seen before. For example, cybercriminals may use extremely specific file names, IP addresses, URLs, and dates. This can cause our machine learning model to learn those particular features and not learn the features that represent the behavioral and decision-making processes of the threat actor. The commands must be transformed into a more generic form to prevent this. Therefore, in this study, an intermediate component is created for changing each command into a more generic form to prevent the problem of lack of generality.

The SCLC aims to simplify unprocessed commands into an SCL tailored explicitly to standardize the command data. The SCLC entails transforming unstructured and diverse command data into a universal format that adheres to specific syntax and semantics defined by SCL, for instance, converting different file path syntax into a universal one. Creating such a language provides a more accurate, scalable, and uniform environment for language processing, increasing our NLP architecture's overall efficiency and accuracy. As shown in Figure 2, a raw command must be passed through four steps before it becomes a processed command.

5.1.1 SCLC—Data Generalization Step. The first place that SCLC is processing raw data. The motivation behind this step is to create a generic command mapping since different malware that CTAs use work in different environments and in nearly completely distinct manners. Therefore, we have manually inspected common and similar commands to create a mapping for similar commands into a single command. Moreover, all file paths, domains, usernames, and indications for a CTA are being obfuscated by SCLC.

Similar commands such as *execute_cmd*, *run*, and *start*, which have the same meaning, are mapped into the command *execute_cmd*. By doing so, our NLP architecture becomes more aware of the context, becoming

Table 3. Before SCL Generation

Field	Command
data	list jobs
when	1663713006
data	shell wmic \node:192.168.0.254 process call create "C:\ProgramData\sys.exe"
when	1663713268

Table 4. After Standardized Command Generation

Field	Command
data	list jobs
data	wait: zero hours four minutes twenty-one seconds
data	execute_cmd wmic \node:IPADDRESS process call create FPATH

independent of the malware that CTA uses. For instance, two threat actors who use different malware to execute the same command with different syntax, not necessarily, but probably. However, our NLP architecture should not distinguish CTAs by the malware they use but by their unique operating style.

5.1.2 SCLC—Time Feature Generation Step. The data includes the timestamp a command has executed, so one can easily deduce the seconds between two consecutive commands. Then, we calculate the elapsed seconds for each consecutive command and insert it as a command between them. Hence, we are imitating the wait time between words an author writes in their book, or our case, the thought process of a threat actor before executing any command.

5.1.3 SCLC—Noise Analyze Step. In this step, any command execution series with over 32 commands are separated into sequences with 32 commands. The main reason behind the split is to prepare a uniform execution series structure, in other words, to handle the unbalanced command distribution over the command execution series. In addition, without any splitting, the model's training becomes unfeasible, and the consequences of splitting are negligible.

5.1.4 SCLC—Standardized Data Generation Step. In the final step, nearly processed data is prepared; one can find more detailed information in Step 5.3. The main objective of this step is to double-check the data for duplicates and noises. For instance, even though a command execution series has only 32 commands, if it has more than 256 sequences, which is mentioned in Step 5.3, the data point is still dropped from the dataset to prevent noise. As a result, the intermediate data has been generated and is ready to be prepared for the model.

Sample input and output for the execution of the SCLC can be seen in Tables 3 and 4. As mentioned in *Data Generalization Step*, to create a uniform structure, all file paths are obfuscated into the text "FPATH," and all IP addresses are obfuscated into "IPADDRESS." Finally, in *Time Feature Generation Step*, a new command is inserted between two consecutive commands that specify the wait time between commands. Hence, unprocessed data has successfully inherited a new syntax and semantics.

5.2 Architecture

In this study, we use a hybrid deep learning structure that leverages the power of transformers and CNNs to attribute CTAs. Our architecture is designed to capture global and local contextual information within the sequence of commands. This is crucial because the significance of specific terms in the commands can drastically change depending on the broader, global context. For instance, a particular command used in one context may not help us to attribute the CTA. Still, when the relation between other commands is examined, this specific command can indicate the unique operational patterns of a particular CTA. Such understanding requires a hybrid

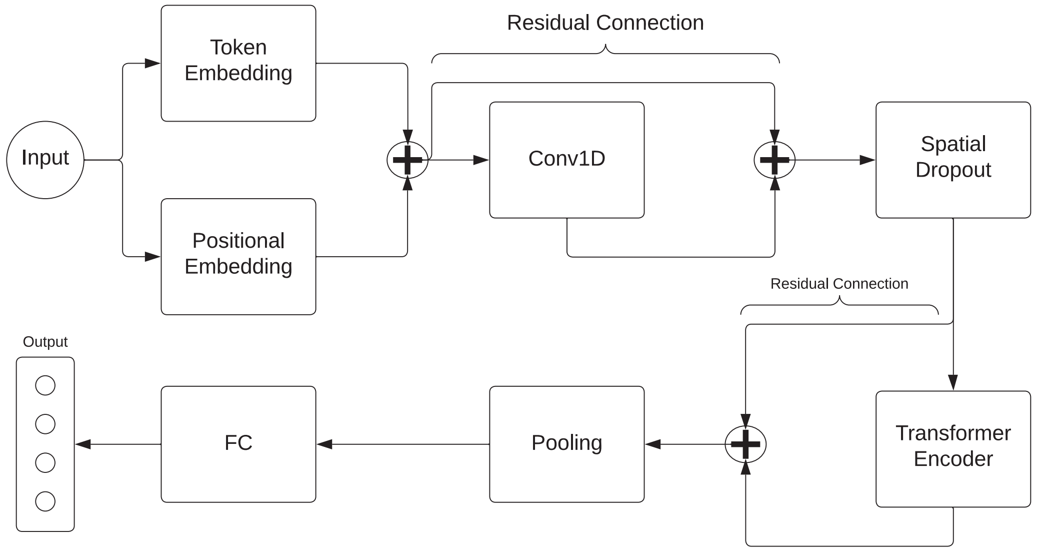

Fig. 3. Hybrid Architecture.

architecture that can comprehend both the immediate surroundings of a term, local context, and the broader sequence of commands, global context.

Our hybrid architecture consists of several key components:

— *Token and Positional Embedding*: These two separate embedding layers are used for encoding the sequence of input tokens and their positions in the sequence. These embedding layers enable our model to capture both the semantic meaning of the tokens and the order in which these tokens are represented in the sequence. The size of the embeddings is controlled by a parameter named hidden size, which is tuned to its best optimal size in the hyperparameter tuning experiments.

— *Conv1D*: This layer helps our model to capture local features from the input sequence. The 1D convolutional layer is used because our input data is in a sequential form, represented in 1D vectors. It uses a kernel that moves across the sequence to learn and extract meaningful patterns. The size of the kernel is determined by a parameter named kernel size, and the number of output filters is controlled by a parameter named number of filters. Again, these parameters are tuned to their best size in the hyperparameter tuning experiments.

— *Spatial Dropout*: This layer prevents overfitting. We have used spatial dropout instead of regular dropout as it preserves the spatial coherence of the input data and, as a result, allows a more efficient way of learning spatially local features. The dropout rate is determined in the hyperparameter tuning experiments.

— *Transformer Encoder*: This component captures the global dependencies in the command sequence. It consists of multiple transformer encoder layers with self-attention mechanisms. The parameter number of layers determines the number of layers, and the number of attention heads is controlled by the number of heads, as done in the other components; these parameters are tuned in the hyperparameter tuning experiments.

— *Residual Connections*: These connections are used after the Conv1D and transformer encoder layers. This component of our hybrid architecture enables the retention of important information throughout the deeper layers.

As seen in Figure 3, the input command sequence first goes through the token and positional embedding layers. Later, for local feature extraction, the output of these embedding layers is passed through a Conv1D layer with a residual connection. After applying spatial dropout, the sequence is fed into the transformer encoder, providing

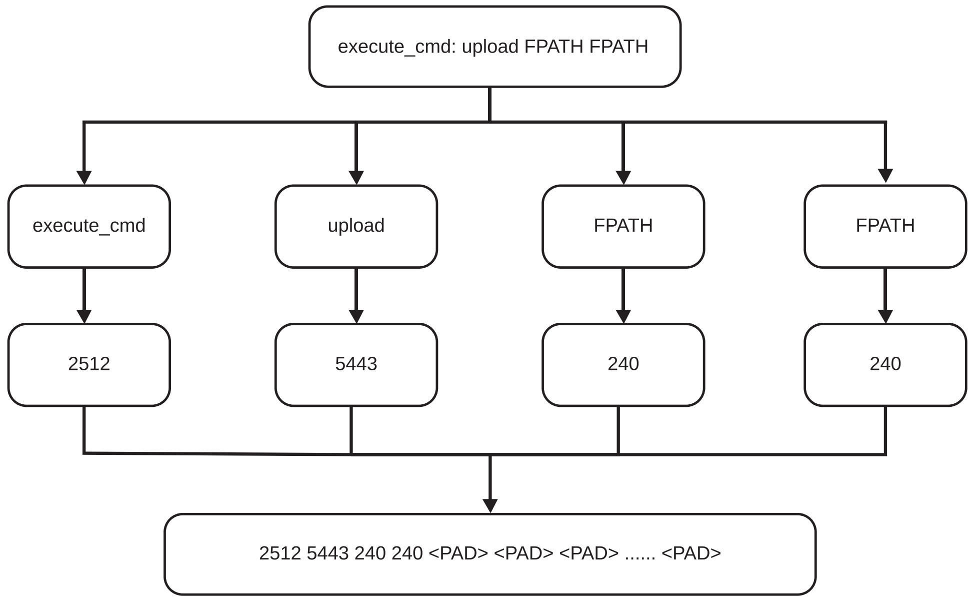
execute_cmd: upload FPATH FPATH
execute_cmd
upload
FPATH
FPATH
2512
5443
240
240
2512 5443 240 240 <PAD> <PAD> <PAD> ...... <PAD>

Fig. 4. Tokenization of commands.

an enriched new sequence with contextual information from all other positions in the sequence. Another residual connection is used at this stage. Subsequently, a mean pooling across the output is used for reducing the size of the output tensor, which concentrates the sequence's most crucial information. Finally, the pooled output from this layer is fed into a linear fully connected layer, which performs the final transformation of the data, giving us the final predictions.

5.3 Data Preparation

We transform the given sequence of commands into numerical tokens to enable our NLP architectures to understand and learn from the commands. As one can see from Figure 4, every unique word in commands is extracted and mapped into a number, and then we replace every word in the processed data with the corresponding number. For example, the command in Figure 4 shows that the command *execute_cmd* is interpreted as the number 2512. In addition, to make all of our sequences the same length, we are fixing the maximum length of a sequence into an optimal number of tokens, in our case, 256, and pad the shorter sequences to the maximum length by appending arbitrary tokens. After the tokenization is completed, our data is wholly represented numerically.

The chosen sequence length of 256 was found to offer the best balance between capturing sufficient contextual information and maintaining feasible computation times and memory usage. Moreover, choosing a shorter sequence length than 256 leads to insufficient context for making accurate predictions, as this may cause vital command elements to be trimmed. Additionally, longer sequence lengths require more padding to more CTA command data, which thins out the information present in the training data.

To reduce the impact of dataset size variability among the 34 threat actor datasets on model performance, we partitioned the data into three categories: high-count, medium-count, and low-count. This categorization serves several purposes. Firstly, it allows a more in-depth analysis of model performance across datasets of different sizes, which is critical in understanding the model's robustness and generalizability. High-count CTAs, having over 1,000 sequences each, provide a dense data environment where models can learn complex patterns without overfitting. Medium-count CTAs with more than 50 sequences, which includes also the CTAs in the high-count dataset as they have more than 50 sequences too, represent a more typical real-world scenario where there is enough data to train on but not as much as CTAs present in high-count datasets. Low-count CTAs include all 34 datasets, again including both high-count and medium-count CTAs, to examine the model's behavior in the most challenging scenario where data scarcity might disrupt the model's ability to learn effectively.

The decision to include all CTAs in the low-count category and to allow for overlapping between the low-count, medium-count, and high-count datasets was made to ensure comprehensive coverage and robust evaluation of our models across the entire spectrum of data availability. Additionally, the choice to combine CTAs with varying numbers of commands reflects the challenging environments that organizations often encounter, where they must manage a few datasets with large volumes of data alongside many smaller, sparser datasets. Thus, we have integrated low-count with medium-count and high-count and medium-count with high-count datasets. By evaluating models across these overlapping categories, we can more accurately assess their performance and understand the effects of data volume on learning outcomes. This approach ensures that our analysis extends beyond ideal conditions with plentiful data, which is crucial for developing universally robust and effective models.

— High-count: CTAs with more than 1,000 sequences (4 CTAs)
— Medium-count: CTAs with more than 50 sequences (10 CTAs)
— Low-count: All CTAs (34 CTAs)

Lastly, we split each dataset into training, validation, and test sets. The training set, which comprises 70% of the data, is used for training the model. The validation set makes up 15% of the data and is used to determine the best set of hyperparameters. Lastly, the remaining 15% of the data is the test set, which is used to evaluate the model's performance.

6 Experiments

6.1 Experimental Environment

To construct our experimental environment, we leveraged the deep learning library PyTorch [18]. In addition, our study utilized several pre-trained transformer-based models, specifically BERT, RoBERTa, SecureBERT, and DarkBERT. These models are implemented using the Hugging Face's[2] Transformers library.

Our dataset consists of command sequences of 34 threat actors. However, the CTAs are not uniformly sized. To deal with the imbalance in dataset sizes, we partitioned the CTAs into three categories: high-count, medium-count, and low-count. Experiments are conducted across these differently sized datasets using all of the aforementioned deep learning models to ensure a comprehensive evaluation. Later, as explained before, the dataset is divided into a training set, constituting 70% of the data, a validation set comprising 15%, and a test set, making up the remaining 15%.

6.2 Hyperparameter Tuning

For pre-trained transformer models, the tuned hyperparameters are the learning rate, the number of epochs the model is trained, and weight decay. Moreover, for our hybrid architecture, dropout rate, the number of epochs, hidden size, kernel size, **learning rate (LR)**, filters, heads, and layers are the hyperparameters tuned in the hyperparameter tuning phase.

We employed an extensive hyperparameter tuning process to optimize the performance of our models, and the best combination of hyperparameters was chosen considering the validation F1-score. The best set of hyperparameters and its metrics for the pre-trained transformer-based models can be seen in Table 5. Again, the best hyperparameters and their metrics for the hybrid architecture can be seen in Table 6. The change of performance across different dataset settings that can be observed in the aforementioned tables, which hints that as the number

[2]https://huggingface.co/

Table 5. Hyperparameter Tuning Results—Pre-Trained Transformers

AQ15

Model	Dataset	LR	Epochs	Weight Decay	Val. Accuracy	Val. F1-score
BERT	High-count	3e-05	6	0.05	0.9243	0.9241
BERT	Medium-count	3e-05	4	0.01	0.9042	0.9018
BERT	Low-count	3e-05	6	0.05	0.8602	0.8456
DarkBERT	High-count	3e-05	6	0.05	0.9307	0.9303
DarkBERT	Medium-count	3e-05	6	0.05	0.8881	0.886
DarkBERT	Low-count	2e-05	6	0.01	0.8143	0.7914
RoBERTa	High-count	3e-05	6	0.01	0.9256	0.9253
RoBERTa	Medium-count	3e-05	6	0.05	0.8731	0.8713
RoBERTa	Low-count	3e-05	6	0.05	0.8060	0.7818
SecureBERT	High-count	3e-05	6	0.05	0.9269	0.9268
SecureBERT	Medium-count	3e-05	6	0.05	0.8892	0.8880
SecureBERT	Low-count	3e-05	6	0.01	0.7987	0.7739

Table 6. Hyperparameter Tuning Results—Hybrid Architecture

Dataset	Dropout	Epochs	Hidden Size	Kernel Size	LR	Filters	Heads	Layers	Val. Acc.	Val. F1
High-count	0.1	100	128	3	0.002	128	4	4	0.9628	0.9626
Medium-count	0.1	100	128	5	0.02	128	16	2	0.9365	0.9350
Low-count	0.1	150	128	5	0.02	128	16	3	0.9103	0.9074

of commands per CTA in the available dataset decreases for training, the models generally show a decline in validation accuracy and F1-score. Compared to pre-trained models, hybrid architecture with the best set of hyperparameters performs generally well across all dataset settings.

6.3 Cross Validation

After the hyperparameter tuning process, the optimal hyperparameters for each model with each dataset setting are found. With the best hyperparameters set, each model is cross validated using a 10-fold cross validation method, which is explained in the background Section 3. Moreover, in this phase of the experimentation, training, and validation datasets are merged, leading to a dataset for cross validation constituting 85% of the dataset. The training and validation of the deep learning models in the cross validation step is done with this merged dataset.

Employing 10-fold cross validation presents us a more reliable performance estimation and shows the robustness of each model. This process minimizes the risk of the training process becoming skewed toward a particular division of the dataset. Additionally, by providing mean and standard deviation metrics, we give a comprehensive view of each model's strengths and weaknesses. The mean and standard deviation of the performance metrics, computed across the 10 folds, are summarized in Table 7.

When the results of the 10-fold cross validation are inspected, the hybrid architecture in all dataset settings consistently outperforms each pre-trained model. Additionally, the standard deviation of the F1 scores are generally low on all datasets for the hybrid architecture, which again showcases the robustness of the architecture. Moreover, it should be noted that SecureBERT performed comparably well on the medium-count dataset. Generally, the standard deviations are observed to be low, suggesting that the hybrid architecture, and the fine-tuned pre-trained models are relatively stable across different folds of the dataset. This stability is especially important in practical applications where the trained deep learning model has to generalize and perform well in unseen data.

Table 7. Mean and Standard Deviation Results in 10-Fold Cross Validation

Model	Dataset	Mean Accuracy	Standard Deviation Accuracy	Mean F1	Standard Deviation F1
BERT	High-count	0.9197	0.0141	0.9195	0.0142
BERT	Medium-count	0.8811	0.0167	0.8763	0.0175
BERT	Low-count	0.8501	0.0107	0.8318	0.0119
DarkBERT	High-count	0.9179	0.0143	0.9177	0.0143
DarkBERT	Medium-count	0.8994	0.0153	0.8975	0.0150
DarkBERT	Low-count	0.8096	0.0189	0.7834	0.0207
RoBERTa	High-count	0.9151	0.0145	0.9147	0.0144
RoBERTa	Medium-count	0.8964	0.0153	0.8943	0.0160
RoBERTa	Low-count	0.8264	0.0188	0.8049	0.0199
SecureBERT	High-count	0.9133	0.0148	0.9132	0.0151
SecureBERT	Medium-count	0.9000	0.0098	0.8990	0.0100
SecureBERT	Low-count	0.8071	0.0246	0.7799	0.0286
HybridArchitecture	High-count	0.9418	0.0087	0.9416	0.0090
HybridArchitecture	Medium-count	0.9161	0.0129	0.9157	0.0129
HybridArchitecture	Low-count	0.8855	0.0118	0.8814	0.0131

Table 8. Experiment Results

Model	Dataset	Test Accuracy	Test F1
Hybrid	High-count	**0.9513**	**0.9511**
Hybrid	Medium-count	**0.9377**	**0.9360**
Hybrid	Low-count	**0.8925**	**0.8895**
BERT	High-count	0.9129	0.9126
BERT	Medium-count	0.8824	0.8802
BERT	Low-count	0.8696	0.8569
DarkBERT	High-count	0.9167	0.9165
DarkBERT	Medium-count	0.8986	0.8975
DarkBERT	Low-count	0.8237	0.7998
RoBERTa	High-count	0.9180	0.9178
RoBERTa	Medium-count	0.9135	0.9127
RoBERTa	Low-count	0.8613	0.8487
SecureBERT	High-count	0.9206	0.9204
SecureBERT	Medium-count	0.9228	0.9220
SecureBERT	Low-count	0.8206	0.7972

The highest accuracy and F1 values are highlighted in bold.

7 Results

In this section, we delve into the outcomes obtained from the comprehensive evaluation of our hybrid deep learning architecture. The performance is compared against several established pre-trained transformer-based models, namely BERT, RoBERTa, SecureBERT, and DarkBERT. This juxtaposition intends to fully encapsulate the strengths and potential areas of improvement for each model and, importantly, highlight the performance of the hybrid architecture. In the final testing step, for training, the merged train and validation datasets are leveraged, which is 85% of the dataset, and for testing the test set is used, which consists of 15% of the dataset.

The results in Table 8 depict the success of the hybrid architecture's performance on the test set over all dataset settings after various experimental iterations. The extensive experimentation, including numerous runs of hyperparameter tuning and 10-fold cross validation, which ensures the robustness of each model, resulted

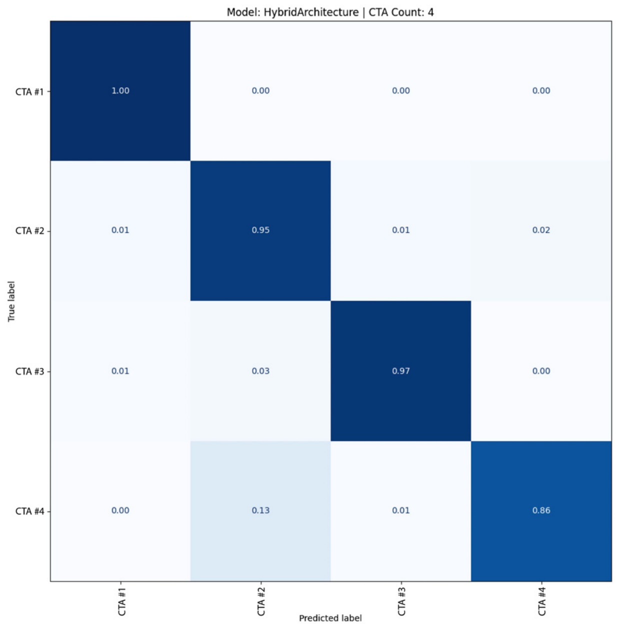

Fig. 5. High-count dataset.

in optimal performance settings for each model. Later, with the test set, the models were evaluated using test accuracy and F1 metrics, providing a holistic view of their performance across all classes of our multi-class classification problem. These results suggest that our hybrid deep learning architecture is the most effective model among the other pre-trained models that have been tested in terms of both accuracy and F1-score.

The confusion matrices for different dataset settings offer valuable insights into the model's performance by displaying the detection accuracy of the architecture for each CTA. Confusion matrices for hybrid architecture are depicted in Figures 5, 6, and 7 for high-count, medium-count, and low-count datasets, respectively. It should be noted that, the number of command sequences for each CTA is in sorted order. To explain, CTAs with higher indexes have much less sequence of commands compared to the lower index ones, for example CTA#1 has the most number of commands.

As shown in Figure 5 for the high-count dataset, the hybrid architecture demonstrates good performance across almost all CTAs. The high values along the diagonal indicate that the model is effectively classifying instances for each class. In Figure 6 for the medium-count dataset, a relatively balanced performance is observed. While the diagonal elements are strong, indicating good classification, there is room for improvement, for example, with the CTA #6. The low-count dataset, depicted in Figure 7 showcases a more challenging scenario. Although the model performs well for the majority of CTAs, it is observed that the model struggles with CTAs with lesser sequences of commands.

As can be seen from the confusion matrices, conducted experiments on different sizes of datasets indicate a correlation between the number of command sequences in each CTA and the performance of the deep learning models. Specifically, we observe:

— *High-count dataset*: The models achieve significantly better results, with higher accuracy and F1-scores. The large dataset size allows the models to learn complex patterns and generalize effectively.
— *Medium-count dataset*: A moderate decrease in performance metrics is observed, the models suffer from some limitations in capturing the behavior of the CTAs with lower examples of command sequences.

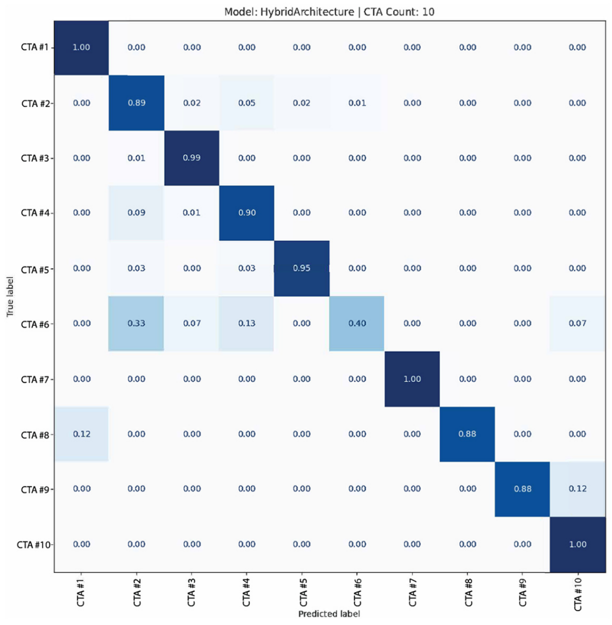

Fig. 6. Medium-count dataset.

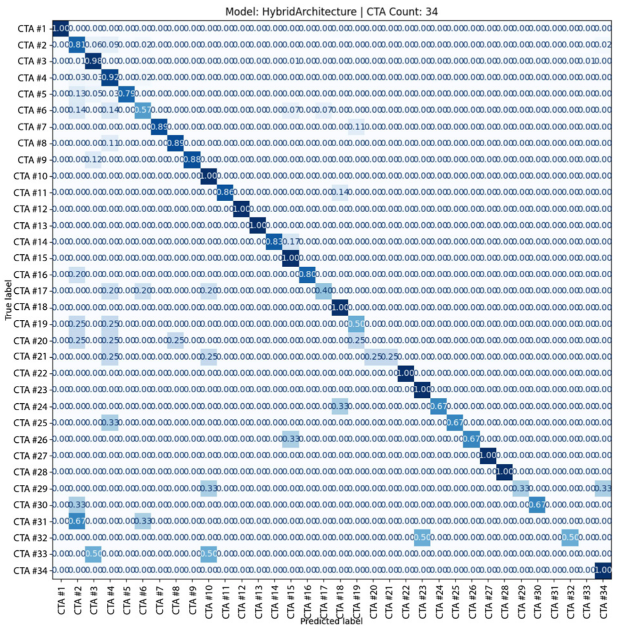

Fig. 7. Low-count dataset.

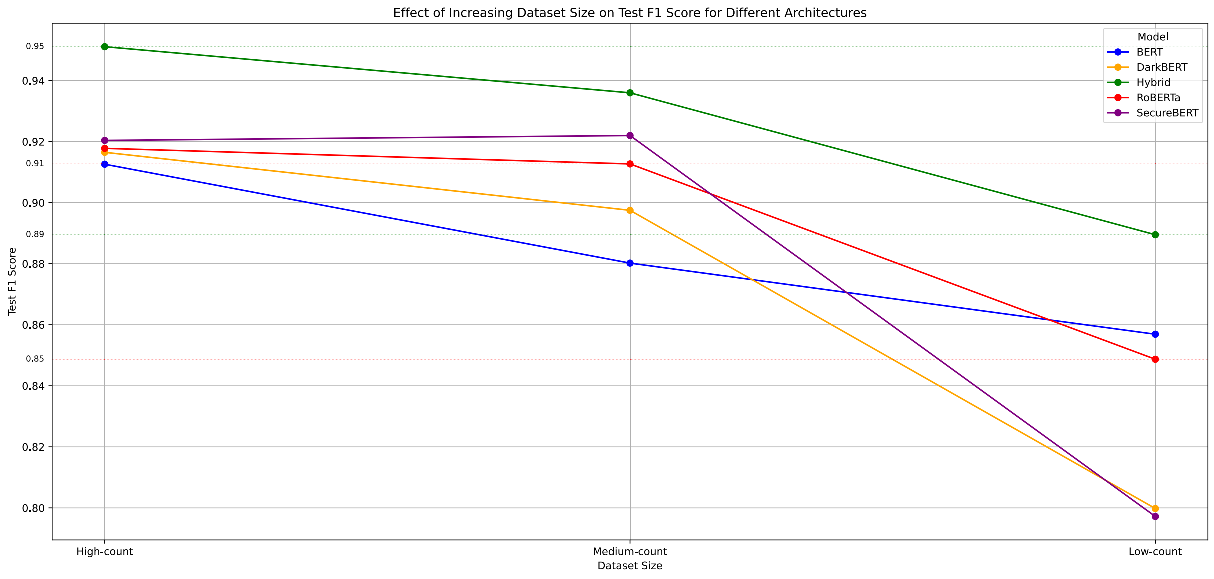

Fig. 8. Performance metrics for different dataset sizes.

— *Low-count dataset*: The models result in poorer performance, including lower accuracy and F1-scores. This is likely due to insufficient data, which prevents the model from learning the patterns in the CTAs with lower examples of command sequences well.

Figure 8 displays the relationship between the dataset sizes and the performance metrics. These findings exemplify the importance of dataset size for an effective model that detects CTAs. Nevertheless, it should be noted that the hybrid architecture proves to be relatively well across all dataset settings. This shows the architecture's ability to learn essential features of CTAs from their sequences of commands and generalize better in data-scarce environments. This resilience to data sparsity makes the hybrid architecture much more important as labeled datasets are often difficult to obtain in real-world applications.

8 Limitations and Future Work

However, it is important to acknowledge that, in cyberspace, actors continuously change their style to hide their signatures and characteristics, or they even borrow open-source frameworks to *hide in the noise* [5]. It is still unclear if this model can predict with high accuracy if *Evil Corp* switches to another malware, but there is still much to delve into in this study area. It is also essential to know that the CTA attribution process is not solely dependent on the sequence of the commands that the actor executes. There are numerous traits and methods that should be considered during the attribution, meaning that our model should not be the only source for the process but a guideline that is to be considered.

In addition, due to the nature of our experiments and the fact that we did not have access to real-life victims but mostly honeypots, our dataset only consists of the commands that the attacker executed. This selective data collection method inherently introduces a selection bias, as it does not capture commands from legitimate users typically present in real-world datasets [28]. Consequently, our model has a bias toward attacker commands, raising concerns about its effectiveness in real-world scenarios. To mitigate this bias and enhance the model's applicability to real-world scenarios, further research is needed. This could involve simulating more authentic datasets by incorporating legitimate user commands or utilizing actual user and attacker command datasets should such become publicly available.

We suggest exploring other influencing factors that dictate the behavior and tactics of threat actors. For example, one promising future research direction could be investigating the correlation between specific command sequences and the broader strategic goals of CTAs. It would be interesting to identify if a particular type of attack is more prevalent in the presence of a particular CTA. Additionally, it is beneficial to note that this study will contribute to coping with evolving threats; by analyzing the CTAs and their unique style, it is also possible to further research the emergent threats and their relations. Furthermore, it is important to gain more insights into how deep learning models determine the attribution of a CTA. As future work, we aim to explore the decision-making processes of these models, which could reveal the distinct patterns and behaviors they associate with specific CTAs. By interpreting the trained models, researchers and analysts could gain valuable insights that may not be immediately apparent from manual analysis alone.

9 Conclusion

In our study, we presented a novel approach to attributing CTA based on the sequence of commands they execute. This research was grounded in a unique intersection of NLP techniques and deep learning architectures, effectively distinguishing between various CTAs with high performance. We have developed a SCLC for CTA command sequence data, which effectively mitigates the issue of overfitting by transforming syntactic differences in the commands into a uniform format. This enhancement significantly improves the performance of our deep learning architecture.

Our hybrid deep learning architecture has proved to be a powerful solution for capturing local and global contextual information within the sequences of commands as it outperformed the notable pre-trained transformer models such as BERT, RoBERTa, SecureBERT, and DarkBERT. Hybrid architecture achieves an F1-score of 95.11% and an accuracy score of 95.13% on the high-count dataset, an F1-score of 93.60% and an accuracy score of 93.77% on the medium-count dataset, and an F1-score of 88.95% and accuracy score of 89.25% on the low-count dataset. These results indicate the potential of integrating CNNs and transformer mechanisms in complex NLP tasks, specifically within cybersecurity.

Our study resulted in several key findings that provide considerable insights into cybersecurity. First, we discovered that our approach could significantly reduce the workload of cybersecurity analysts by providing a systematic and automated means of attributing CTAs. Through the application of our method, the analysts are spared from manually tracing the origin of the threats. This allows them to concentrate their expertise and effort on more pressing or strategic tasks. Additionally, this automation not only enhances productivity but also reduces the risk of human error by providing an initial foothold for the threat actor attribution process.

With this knowledge, creating more effective countermeasures and defense strategies becomes much more feasible. Unique defense strategies can be developed based on the commands of the specific CTA. For instance, understanding a CTA's penchant for certain command sequences or attack vectors could guide the design of specific intrusion detection rules or firewall configurations. Similarly, knowing a CTA's pattern of behavior can also inform incident response plans, enabling swift and effective action in the unfortunate event of a security breach.

Furthermore, our research facilitates a deeper understanding of the modus operandi of specific CTAs. This nuanced understanding equips cybersecurity professionals with valuable knowledge to predict potential targets and anticipate the type of attack, thus enabling proactive rather than reactive defense strategies. Moreover, our work opens up multiple exciting avenues for future research within CTI and digital forensics. This includes investigating a deeper understanding of CTA behavior and the relationship between the sequence of commands and the strategic intent of the CTAs. These explorations could result in a more nuanced and thorough understanding of CTAs and their motivations.

Our research highlights the power and potential of NLP techniques and deep learning architectures in addressing complex digital forensic challenges. Our work represents a pivotal stepping stone towards an era where automated, intelligent systems play a central role in fortifying our cybersecurity infrastructure and simplifying digital forensics, as well as incident response challenges, by introducing a new method for CTA attribution. The pathway forward is filled with promising opportunities for innovative and groundbreaking research in cyber defense and threat intelligence research.

References

[1] Ehsan Aghaei, Xi Niu, Waseem Shadid, and Ehab Al-Shaer. 2022. SecureBERT: A domain-specific language model for cybersecurity. In *Proceedings of the International Conference on Security and Privacy in Communication Systems*. Springer, 39–56.

[2] Aylin Caliskan, Fabian Yamaguchi, Edwin Dauber, Richard Harang, Konrad Rieck, Rachel Greenstadt, and Arvind Narayanan. 2015. When coding style survives compilation: De-anonymizing programmers from executable binaries. arXiv:1512.08546. Retrieved from https://doi.org/10.14722/ndss.2018.23304

[3] Cybersecurity and Infrastructure Security Agency (CISA). 2023. #StopRansomware: Royal ransomware. Retrieved from https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-061a

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. DOI: https://doi.org/10.18653/v1/N19-1423

[5] Juan Andres Guerrero-Saade. 2018. Draw me like one of your french APTs—Expanding our descriptive palette for cyber threat actors. In *Proceedings of the Virus Bulletin Conference*. 1–20.

[6] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. UNICORN: Runtime provenance-based detector for advanced persistent threats. In *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium (NDSS '20)*. DOI: https://doi.org/10.14722/ndss.2020.24046

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. arXiv:1512.03385. Retrieved from https://doi.org/10.48550/arXiv.1512.03385

[8] Health Sector Cybersecurity Coordination Center. 2022. HC3 Threat Profile: Evil Corp. U.S. Department of Health and Human Services. Retrieved from https://www.hhs.gov/sites/default/files/evil-corp-threat-profile.pdf

[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735

[10] Ehtsham Irshad and Abdul Basit Siddiqui. 2023. Cyber threat attribution using unstructured reports in cyber threat intelligence. *Egyptian Informatics Journal* 24, 1 (2023), 43–59.

[11] Youngjin Jin, Eugene Jang, Jian Cui, Jin-Woo Chung, Yongjae Lee, and Seungwon Shin. 2023. DarkBERT: A language model for the dark side of the internet. arXiv:2305.08596. Retrieved from https://doi.org/10.48550/arXiv.2305.08596

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[13] Yann LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1, 4 (1989), 541–551. DOI: https://doi.org/10.1162/neco.1989.1.4.541

[14] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. arXiv:1907.11692. Retrieved from https://doi.org/10.48550/arXiv.1907.11692

[15] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. arXiv:1711.05101. Retrieved from https://doi.org/10.48550/arXiv.1711.05101

[16] Sadegh M. Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V. N. Venkatakrishnan. 2019. HOLMES: Real-time apt detection through correlation of suspicious information flows. arXiv:1810.01594. Retrieved from https://doi.org/10.48550/arXiv.1810.01594

[17] Umara Noor, Zahid Anwar, Tehmina Amjad, and Kim-Kwang Raymond Choo. 2019. A machine learning-based FinTech cyber threat attribution framework using high-level indicators of compromise. *Future Generation Computer Systems* 96 (2019), 227–242.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. arXiv:1912.01703. Retrieved from https://doi.org/10.48550/arXiv.1912.01703

[19] Lior Perry, Bracha Shapira, and Rami Puzis. 2019. NO-DOUBT: Attack attribution based on threat intelligence reports. In *Proceedings of the 2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 80–85. DOI: https://doi.org/10.1109/ISI.2019.8823152

[20] Avi Pfeffer, C. Call, John Chamberlain, L. Kellogg, Jacob Ouellette, T. Patten, Greg Zacharias, Arun Lakhotia, Suresh Golconda, J. Bay, Robert Hall, and Daniel Scofield. 2012. Malware analysis and attribution using genetic information. In *Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software*. 39–45. DOI: https://doi.org/10.1109/MALWARE.2012.6461006.

[21] PRODAFT. 2022. [ws] Wizard Spider Group In-Depth Analysis. Retrieved from https://www.prodaft.com/resource/detail/ws-wizard-spider-group-depth-analysis

[22] PRODAFT. 2021. Silverfish: Global Cyber Espionage Campaign Case Report. Retrieved from https://www.prodaft.com/resource/detail/silverfish-global-cyber-espionage-campaign-case-report
[23] Nathan Rosenblum, Xiaojin Zhu, and Barton P. Miller. 2011. Who wrote this code? Identifying the authors of program binaries. In *Proceedings of the 16th European Conference on Research in Computer Security (ESORICS '11)*. Vijay Atluri and Claudia Diaz (Eds.), Springer, Berlin, 172–189.
[24] S. Naveen, Rami Puzis, and Kumaresan Angappan. 2020. Deep learning for threat actor attribution from threat reports. In *Proceedings of the 2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP)*. 1–6. DOI: https://doi.org/10.1109/ICCCSP49186.2020.9315219.
[25] Md Sahrom, S. Rahayu, Aswami Ariffin, and Y. Robiah. 2018. Cyber threat intelligence – Issue and challenges. *Indonesian Journal of Electrical Engineering and Computer Science* 10, 1 (2018), 371–379. DOI: https://doi.org/10.11591/ijeecs.v10.i1.pp371-379
[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[27] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. 2015. Efficient object localization using convolutional networks. arXiv:1411.4280. Retrieved from https://doi.org/10.48550/arXiv.1411.4280
[28] Antonio Torralba and Alexei A. Efros. 2011. Unbiased look at dataset bias. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11)*. IEEE, 1521–1528.
[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 6000–6010.

Received 30 December 2023; revised 11 May 2024; accepted 27 June 2024