Article Type: Description (see below for more detail)

# Reliable Cloud Operations using Transformers

Jorge Cardoso, *Huawei Cloud, 80992 Munich, Germany, and University of Coimbra, CISUC/IS, DEI, 3000 Coimbra, Portugal*

*Abstract—Managing large-scale public clouds presents significant challenges, particularly in analyzing the risk of commands executed by operators to guarantee they do not cause damages to the infrastructure. Traditional rule-based systems have been used but fall short in scalability when managing operations at a global scale. While machine learning techniques have been proposed as alternatives, they have typically been applied in small-scale environments. This article presents a novel approach to address the complexities of global-scale command risk analysis and standard operation procedure (SOP) verification using Large Language Models (LLMs). Through an extensive evaluation in production, the technique shows to be more suitable than existing methods.*

*Keywords: LLM, deep learning, security, reliability, IT operations, cloud computing,*

While the precise count of servers owned by public cloud providers such as Amazon Web Services, Microsoft Azure, Google Cloud, and Huawei Cloud is not publicly disclosed, estimates suggest they collectively manage between 1 million and 8 million servers [1].

Managing large-scale cloud infrastructure poses interesting challenges for operations and maintenance (O&M) due to the scale, complexity, and dynamic nature of the infrastructure. Daily, operators and automated systems access remote servers to configure and repair services. Given that thousands of operations are executed per day, mechanisms are needed to identify incorrect, dangerous, or malicious commands that exploit security vulnerabilities that can cause infrastructure failures. According to a Google study, configuration errors are the $2^{nd}$ major cause of service failures and human errors are responsible for >24% of outages (Uptime Institute).

During high-severity incidents (e.g., service outage, security breach), an operator may act under pressure and execute commands such as:

```
$ iptables -F
$ cat $FLIST | xargs -0 rm
$ echo `kill 7890`
```

Without proper validation, these commands could unintentionally remove critical infrastructure components. For instance, while the command `iptables -L` is safe and displays all firewall rules, the command `iptables -F` is dangerous, as it deletes all rules in the current table. In a rushed situation or due to inexperience, an operator might restart the wrong service, kill essential processes, or scale down production workloads, aggravating the issue.

Google Cloud utilizes a form of Role-Based Access Control (RBAC) [1] to restrict the commands operators can execute based on their roles and permissions. Prior to the development of the approach described in this article, Huawei Cloud relied solely on rules and regular expressions. Other cloud providers have not publicly disclosed the specific techniques they employ. Nonetheless, the typical solution is to mediate the access to production services through a bastion host or proxy, and use some kind of rule-based engine to evaluate the risk of a command using a blacklist of prohibited commands. Unfortunately, rule-based systems have limited adaptability, as they are often static and do not adapt well to rapidly changing environments. Managing and maintaining a large set of rules is overwhelming in large-scale cloud infrastructures.

Evaluating the risk of a command can also be achieved using machine learning. The goal is to predict the categorical class label of commands. This task is still an open and challenging problem due to the high number of dynamically changing cloud

XXXX-XXX © 2024 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

[1]As of December 2023, AWS operates 166 data centers worldwide, with each data center capable of accommodating up to 50,000 servers. A rough estimate suggests AWS may host approximately 8.3 million servers in total.

services, middleware, and infrastructure components; the heterogeneity of commands that can be executed, their context and semantics, and the dimensionality of arguments, flags, and environment variables.

In this article, we describe how a Large Language Model (LLM), transfer learning, and a transformer-based method can extend a rule-based system to classify live commands and analyze operating procedures. We validate the approach with two use cases: command interception and Standard Operating Procedure verification. We explore the dual learning procedure of LLMs in the following way: 1) we pre-train the model on a large-scale, domain-specific corpus of CLI commands data generated by Huawei Cloud [2] and 2) we fine-tune the model with supervised data generated by the rule-based engine.

Results show that the approach provides a desired level of generalization and is able to pinpoint dangerous commands more effectively (F1-score=+22%). When verifying Standard Procedure Operations, the results of the approach meet O&M requirements: precision=83%, recall=69%.

## BACKGROUND

### Secure and Reliable Operations

O&M operators typically access remote servers via a bastion host, which reviews, approves, and executes commands without allowing a direct connection to remote servers [1]. Often, a sequence of a subset of commands to be executed is described in a formal document called a Standard Operating Procedure (SOP). Bastion hosts implement an interception system, which captures *nix CLI commands to be executed, and analyzes them before enabling or blocking their remote execution.

### Rule-based Systems

Rule-based classifiers define rules over regular expressions and rely on *whitelists*/*blacklists* to allow/block commands. While simple, rule-based systems have two major limitations:

- High dimensionality. Defining rules for every possible combination of programs, flags, and arguments is impractical for large, complex cloud infrastructures.
- Search patterns. The complexity of command-line syntax, which is not uniform across *nix programs, requires specialized languages beyond regular expressions.

In addition, rules should be manually adjusted when new commands emerge or are no longer observed. Commercial systems such as Elastic Stack and Splunk allow users to ingest, search, analyze, and visualize data in real-time, including CLI command logs. Users can define custom rules to analyze commands and detect potential threats.

### Machine Learning

The academic literature proposes to use machine learning models, both traditional NLP (e.g., n-gram, BoW) and deep neural network models (e.g., CNN [12], [11] and LSTM [6]). Nonetheless, the research field remains relatively untreated by the academic community. Most work has addressed the simpler problem of detecting malicious commands within small computing environments (e.g., Microsoft PowerShell commands [5]). Unfortunately, the use of character-level one-hot encoding with a closed vocabulary (as in [5]) does not allow to effectively model the semantics and interrelationships between input tokens. Thus, we believe that Byte-pair Encoding [8] and Wordpiece embeddings [10], which enable the model to learn the most frequent tokens directly from the training data and co-occurrence, and meaning-related tokens in the embedding space are more suitable. Furthermore, recent neural network models (e.g., BERT [3]), which can capture a corpus and, afterwards, fine-tune the model for context-specific tasks, are promising.

## RULE-BASED APPROACH

Figure 1 (a) illustrates the architecture of a rule-based interception system. First, commands executed over a remote terminal are intercepted by an access control system (i.e. the bastion host). Then, the risk of intercepted commands is evaluated using a rule-based classifier.

The rule language is simple and intuitive. It defines a pattern to match simple Bash commands, complex Bash programs, SQL statements, and arbitrary strings. The input is parsed into an AST which is used to extract atomic fragments, such as simple commands, redirect statements, and variable definitions. The matcher handles POSIX- and GNU-style arguments. A wildcard token is used to generalize from the specific command to a rule. Rules are assigned with one of 5 risk levels (Priority 1/High Risk to Priority 5/Low Risk). Internally, rules are processed according to their priority. If a rule with higher priority matches the input, then the

[2]https://www.huaweicloud.com/intl/en-us/

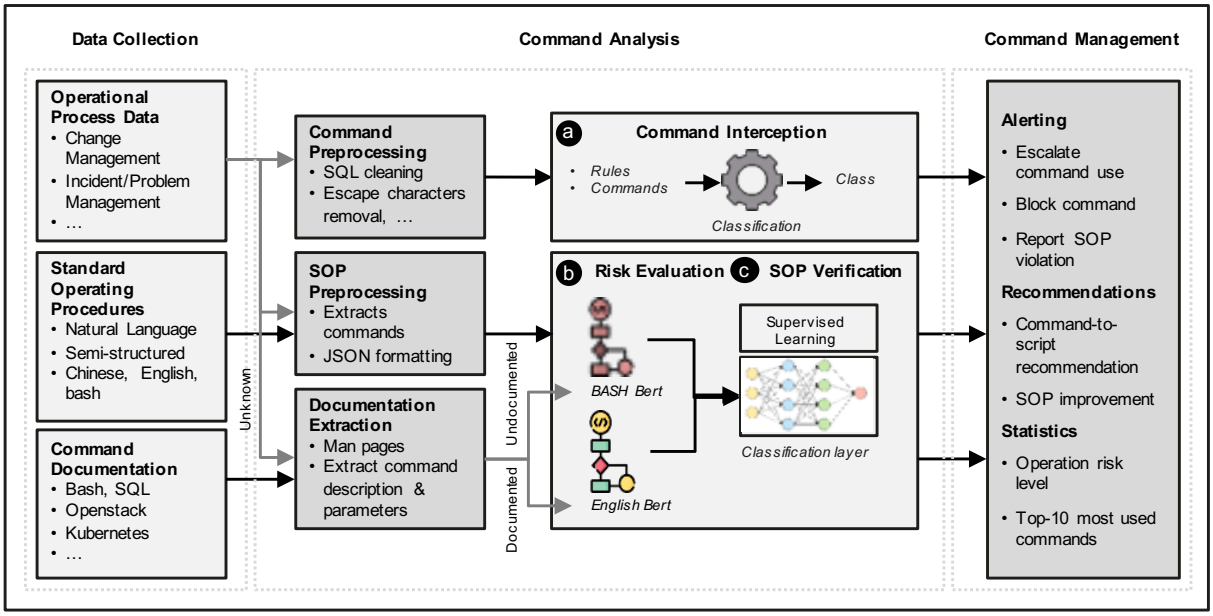


**FIGURE 1.** Architecture of a command interception system driven by a command risk classification component. Commands executed by operators are intercepted and evaluated by the bastion using a set of rules and/or an ML model. If a command is evaluated as safe, it is forwarded to the target infrastructure; otherwise an error is reported.

matching pipeline stops, and the corresponding rule is returned.

Bash commands correspond to 94% of the commands analyzed. Nonetheless, the classifier also handles other types of commands: SQL (5% of commands) and MongoDB (<1%). An example of a command is:

```
AllowUserFXP=`cat /etc/ftpd/ftpd.conf | grep
-v "^#" | grep -i "AllowUserFXP" | grep -i
"no" | wc -l`
```

Interception based on risk classifies commands as (Table 1):

- SAFE: Commands are always allowed since they do not significantly alter the infrastructure.
- BLOCKED: Dangerous, forbidden commands that can cause damage.
- RISKY: Unclassified commands that follow a handling procedure, e.g., permission escalation.

When a command does not match any SAFE or BLOCKED rule, it is labeled RISKY. This usually happens because the command and its parameters have not been seen before. These cases account for 0.3% of all commands processed.

The rule-based system has a high accuracy. An evaluation conducted on a dataset with 100k+ entries, showed an accuracy of >99.9%. One of the biggest benefits of this type of system is that the results are explainable. On the other hand, the system has difficulties maintaining a good accuracy when new, unseen commands are presented to the system.

## TRANSFORMER-BASED APPROACH

We integrate a machine learning model with an existing rule-based engine to leverage the strengths of both paradigms. Figures 1 b and c show the new component. Rules cover deterministic decision making based on explicit conditions and known patterns. Rules are explicit, specialized, and created manually. However, the ML model handles scenarios where new and unseen command patterns emerge. The ML model is implicit, generalized, and created automatically.

### Model

LSTMs have shown to be effective for sequential data analysis that requires capturing long-term dependencies. CNNs might also be used for text-related tasks (e.g., text classification with 1D convolutions). Thus, their use was explored in previous work to build command classification systems since commands can be seen as sequences of tokens (e.g., command name, flags, and parameters) with dependencies.

However, newer models, particularly transformer-

| Command | Example | Risk Class | Comment |
|---|---|---|---|
| iptables | iptables -v -n -L | SAFE | lists the rules in iptables |
| | iptables -F | BLOCKED | flushes (deletes) all the rules |
| fdisk | fdisk -l /dev/sda | SAFE | lists the partition table of the disk device |
| | fdisk /dev/sda | BLOCKED | manages partitions (create, delete, modify) |
| rm | rm /tmp/file1.txt | SAFE | deletes a temporary file |
| | rm -rf /bin/* | BLOCKED | recursively deletes executable files |
| | cat $FLIST \| xargs -0 rm | RISKY | deletes a set of unknown files |
| kill | echo 'kill 7890' | SAFE | prints a string |
| | echo `kill 7890` | BLOCKED | backticks execute the `kill` command |
| | time kill -9 12345 | RISKY | command `kill` hidden by `time` call |

**TABLE 1.** Examples of *nix CLI commands with different complexity and risk. Small changes to parameters, flags, and command-line syntax significantly influence the risk of executed commands.

based architectures like BERT (Bidirectional Encoder Representations from Transformers) [3] and GPT (Generative Pre-trained Transformer), have demonstrated remarkable performance across various NLP tasks, especially those requiring an understanding of relationships within language. BERT uses a bidirectional transformer encoder, processing text in both left-to-right and right-to-left directions simultaneously. The model learns to predict masked words based on both preceding and following context, making it particularly effective for natural language understanding (NLU) tasks such as text classification and sentiment analysis [3], [7], [9]. GPT (3, 4, or 4.5), in contrast, uses a unidirectional transformer decoder, processing text from left to right. It predicts the next word in a sequence based solely on prior context, making it better suited for natural language generation (NLG) tasks such as text completion and code generation. While the various versions of GPT have not changed the directionality of the transformer, they have been optimized to provide faster responses and consider a larger number of tokens.

Thus, BERT's bidirectional context understanding makes it particularly effective at recognizing the intent and structure of commands—an essential requirement for command classification systems, where parameters, options, and flags must be analyzed in relation to their position and context. Figure 2 shows the various phases involved when constructing a BERT model for command classification. Our procedure is composed of three major steps: BPE training, BERT training (pre-training and BERT finetuning), and BERT Inference

## Dataset Collection

We programmatically collected a corpus of Bash files with commands from public repositories. We searched for GitHub repositories with the Bash language tag. For each repository, we selected only Bash-related files by matching specific criteria (first line contains a shebang or the file has a *.sh* extension). This resulted in ∼7100 scripts (∼500 MB).

We collected one million labeled commands from Huawei Cloud for training and evaluation. The public infrastructure has 30 regions in Europe, Asia Pacific, Latin America, and Africa, and 80+ availability zones. It has over 220+ cloud services and 210+ solutions.

## Pre-processing

The corpus of commands was used to learn the tokens and patterns of the Bash CLI commands. The raw command strings were split into a sequence of tokens using the Byte-Pair Encoding (BPE) algorithm [8], [4]. BPE is an unsupervised tokenization method, in which the most frequently occurring pair of characters is recursively replaced with a character that does not occur in the vocabulary.

## Pre-training

BERT pre-training requires to train the transformer network on self-supervised contextual tasks. Therefore, we first annotate each sample from our Bash corpus with labels. The transformer model is pre-trained using the procedure described in [3]: bidirectional masked language modelling, by masking 15% of sequence tokens at random; and next sentence prediction, i.e. by predicting the next command in the sequence of commands inside a Bash script. After pre-training, only the network backbone, which outputs, a $h_i$-long representation of the input command is retained, while the contextual output layers are dropped. After this step, the total dataset size is ∼15GB.

We selected the *4/256 BERT mini* model as our architecture with the following configuration: sequence length=256; hidden layer size=256; number of hidden layers=4; activation=GELU; attention heads=4, optimizer=Adam. We pre-trained our transformer model for

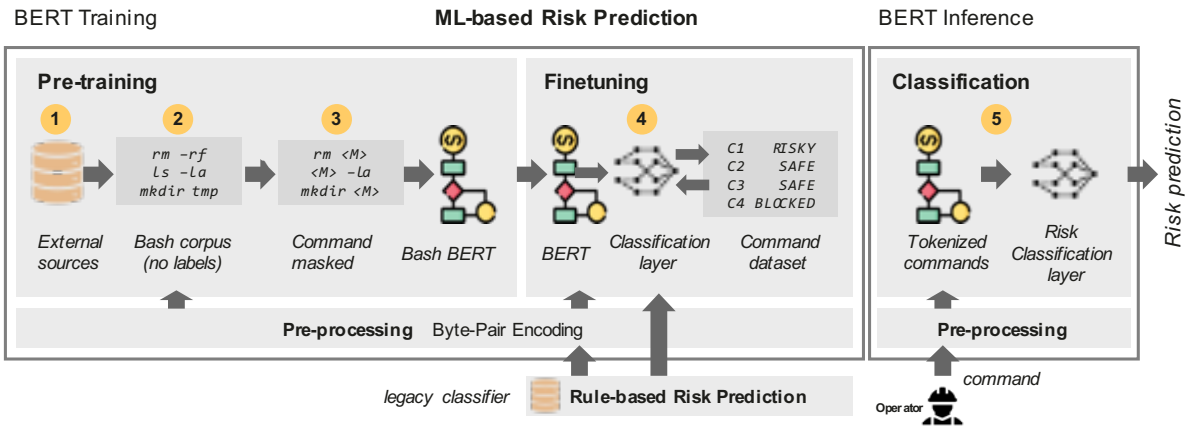


**FIGURE 2.** Three main phases of model construction: pre-training, fine-tunning, and inference. During pre-training, a large corpus of Bash commands is used to learn the language and context relationships. During fine-tuning, a dataset of labeled commands is used to specialize the model for risk classification. Input commands are pre-processed via Byte-Pair encoding.

350,000 iterations (or 160 epochs). The training time was 10 days on a single VM GPU instance with NVIDIA Tesla V100 PCIe 32GB.

### Fine-tuning

The pre-trained transformer backbone was extended with a classification layer to enable command risk classification (application 1) and SOP Verification (application 2). The layer was composed of a fully connected linear layer preceded by a dropout layer and followed by a softmax normalization layer. The network is trained using the softmax loss to maximize the log likelihood of the training dataset. The network weights are updated using gradient descent via back-propagation.

For command risk classification, the dataset of commands was labeled with a risk class: {SAFE, RISKY, BLOCKED}. For SOP Verification, the dataset was labeled with set {COMMAND, GARBAGE}. The final output was a list of class probabilities associated with the different labels. The class with the highest associated probability was the predicted class.

## RISK CLASSIFICATION

### Command Risk Classification

We applied the Bash/BERT model to command risk classification. In addition to the pre-training described in the previous section, we collected a second dataset of commands labeled with a class risk. This dataset was used for training and evaluation. It contained commands used by O&M operators.

Since the labeled commands originated from Huawei Cloud, invalid samples resulting from errors in the interception system, such as password prompts, non-Bash commands, and Bash terminal output have been automatically filtered out and manually verified. Misspelled and unknown commands have been preserved, as they still represent executable commands and may cause damage. Multi-program commands (resulting from pipelining, `xargs`, ...) and script file calls have also been preserved.

The risk labels (i.e., SAFE, RISKY, BLOCKED) originated from our rule-based system. They were manually verified by experts and corrected in case of discrepancies with true command risk (see Table 1).

### Fine-tuning

To ensure the labeled dataset was representative of a real command distribution and O&M workload, we studied the distribution of commands. After removing invalid commands, we observed an approximate 80%, 20% split between SAFE and RISKY commands, with an additional 0.3% component of (extremely rare) BLOCKED commands.

Therefore, after expert evaluation, we down-sampled the dataset to replicate the class ratios observed in our analysis. In the end, we collect 47158 high-quality commands. We divided our fine-tuning dataset into *train*, *dev*, and *test* splits with 70%, 20%, 10% ratios, while preserving the class distribution.

| Algorithm | Precision | Recall | F1-score |
|---|---|---|---|
| Word2Vec+RF | 0.944 | 0.664 | 0.780 |
| BoW | 0.705 | 0.323 | 0.443 |
| 3-gram | 0.933 | 0.858 | 0.894 |
| 4-cnn | 0.942 | 0.889 | 0.915 |
| ***ours*** | ***0.972*** | ***0.916*** | ***0.943*** |

**TABLE 2.** Evaluation results of the classification task.

### Results

The classification model was evaluated in terms of precision, recall, and F1-score (Table 2). The baseline model was a Word2Vec embedding [2] with a Random Forest algorithm. To compare our approach with NLP and deep learning-based solutions, we selected a BoW model, a 3-gram model, and a one-dimensional CNN of 4 layers. We implemented these algorithms and used the original hyper-parameters when known (c.f. [5]).

Our approach achieved the highest absolute score. If we consider the ratio of dangerous commands of 20% and we assume an average number of 3 million commands executed per month, which corresponds to our production system, the increase of recall enables to intercept ∼60k additional dangerous commands.

## SOP VERIFICATION

### Standard Operating Procedures

We also applied the Bash/BERT model to SOP Verification. SOPs are detailed, written instructions that provide a step-by-step guide on how to operate a particular service within Huawei Cloud. SOPs are crucial for maintaining consistency, ensuring quality, and promoting the efficiency of cloud infrastructures. The challenges of SOP verification include:

- Multi-language: SOPs contain several languages: English, Chinese, and Bash. Most available NER models are trained only on a single language.
- Tokenization: Because English uses space-based tokenization while Chinese (a logographic language), the use of traditional character-based tokenization yields poor results.
- Structure: SOPs are a program written in a mixture of natural and Bash languages. Dealing with structured logic is difficult.

The following instructions show a simplified example of an SOP (Chinese was translated to English; system names have been replaced by symbols S2, S2, and S3):

```
1. etcd/kafka ip:port: Login to each node to
be changed and run cat /opt/elb/l4.conf to
query information
2. Bonding scheme: Login to each node to be
changed and run cat /opt/elb/cvs.ini
3. bw_enable: Check LB instances at the site
and check whether there are instances whose
bandwidth exceeds the upper limit
4. Collect dynamic and static parameters
of system S1 required by system S2
(rm -rf /var/log)
5. Check whether the bandwidth of instances
with role S3 (list --user test --project
test) exceeds the upper limit
```

### Verification Task

The SOP Verification task is defined as: having an SOP (acting as a workflow) and a list of commands executed by operators (acting as tasks), verify that the Bash commands executed followed the workflow described by the SOP.

SOP verification has two main sub-systems: command extraction and command comparison. The first sub-system extracts the potential commands from the text-based SOPs. It trims UTF-8 tokens (Chinese characters) and replaces them with a special character. It then uses the trained language model to check if strings in-between trimmed sections are valid commands. Command comparison verifies that all commands executed by an operator were prescribed by the supporting SOP. Violations are reported.

### Fine-tuning

We fine-tuned the pre-trained Bash/BERT model on a dataset which contains valid and invalid commands, numbers, passwords, and garbage captured by the bastion. The garbage dataset had ∼7000 samples and Bash command recognition had ∼4000 samples. Since we used 7000 input samples and 200 training epochs results, 1,400,000 passes are needed in the network. At a processing speed of 900 samples/sec, fine-tuning takes ∼25 minutes. Processing an SOP will take between 1-2 seconds depending on how many commands there are.

### Results

Approximately ∼600 commands were verified. Commands were identified within an SOP with a precision of 83% and a recall of 69% (Table 3). While the results satisfy the initial requirements, there is still room for improvement. Identifying commands within an SOP written in Chinese is difficult due to:

| Task | Precision | Recall | F1-score |
|---|---|---|---|
| Verification | 0.832 | 0.697 | 0.751 |

**TABLE 3.** Evaluation results of the verification task.

- *Parameters*: In some cases, parameters are not present in the commands themselves but are explained in Chinese within the SOP.
- *Context*: SOPs often include a detailed natural language description of the required state of the target system prior to command execution.
- *Jargon*: SOPs may contain technical terms — some related to commands and others specific to the business domain, which can be conflated.

To improve the approach, several strategies can be considered: create a glossary of technical terms and use Named Entity Recognition (NER), generate synthetic data by describing commands in plain text, and use bilingual language models like mBERT to handle mixed Chinese-English text.

## CONCLUSIONS

In this article, we described how to build a language model for the Bash command-line language and demonstrated how it can be used to support two tasks: command risk classification and SOP verification. The approach exploits the contextual knowledge learned during pre-training to achieve a higher classification accuracy when pinpointing dangerous and valid commands. The results show that BERT models outperform previous technologies for command analysis.

We believe that LLMs can find applicability in other areas which we will explore in the future, such as:

- System comprehension: LLMs can consolidate knowledge from data sources such as design documents, manuals, and incident reports.
- Log analysis: LLMs can be trained to understand complex log data from various systems, and identify anomalies and security threats.
- Investigations: LLMs can understand users' questions in plain text, and generate SQL to query observability databases to guide troubleshooting.

## ACKNOWLEDGMENT

I would like to acknowledge Soroush Haeri, Paolo Notaro, and Ilya Shakhat for the implementation of the main systems described in this article, and all the colleagues from the evaluation and productization teams.

## REFERENCES

1. Heather Adkins, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea, and Adam Stubblefield. *Building secure and reliable systems*. O'Reilly, 2020.
2. Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.
3. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
4. Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, feb 1994.
5. Danny Hendler, Shay Kels, and Amir Rubin. Detecting malicious powershell commands using deep neural networks. *CoRR*, abs/1804.04177, 2018.
6. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
7. M. V. Koroteev. BERT: A review of applications in natural language processing and understanding. *CoRR*, abs/2103.11943, 2021.
8. Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725, August 2016.
9. Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Chinese Computational Linguistics*, pages 194–206. Springer, 2019.
10. Yonghui Wu *et al.* Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.
11. Muhammd Mudassar Yamin and Basel Katt. Detecting malicious windows commands using natural language processing techniques. In *Innovative Security Solutions for Information Technology and Communications*, pages 157–169. Springer, 2019.
12. Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification, 2016.

**Dr. Jorge Cardoso** is a leading expert in AI, Cloud Operations, Reliability, and Observability technologies. He is the founder and director of the Large-scale AIOps Lab at Huawei Cloud in Munich, Germany, and an invited professor at the University of Coimbra, Portugal. Contact him at https://jorge-cardoso.github.io/.