applied sciences

MDPI

Article

Semantic Hierarchical Classification Applied to Anomaly Detection Using System Logs with a BERT Model

Clara Corbelle *, Victor Carneiro and Fidel Cacheda

Department of Computer Science and Information Technologies, University of A Coruña, 15071 A Coruña, Spain; victor.carneiro@udc.es (V.C.); fidel.cacheda@udc.es (F.C.)
* Correspondence: c.corbelle@udc.es

Abstract: The compaction and structuring of system logs facilitate and expedite anomaly and cyberattack detection processes using machine-learning techniques, while simultaneously reducing alert fatigue caused by false positives. In this work, we implemented an innovative algorithm that employs hierarchical codes based on the semantics of natural language, enabling the generation of a significantly reduced log that preserves the semantics of the original. This method uses codes that reflect the specificity of the topic and its position within a higher hierarchical structure. By applying this catalog to the analysis of logs from the Hadoop Distributed File System (HDFS), we achieved a concise summary with non-repetitive themes, significantly speeding up log analysis and resulting in a substantial reduction in log size while maintaining high semantic similarity. The resulting log has been validated for anomaly detection using the "bert-base-uncased" model and compared with six other methods: PCA, IM, LogCluster, SVM, DeepLog, and LogRobust. The reduced log achieved very similar values in precision, recall, and F1-score metrics, but drastically reduced processing time.

Keywords: system logs; anomaly detection; BERT model; hierarchical codes; semantic similarity

Citation: Corbelle, C.; Carneiro, V.; Cacheda, F. Semantic Hierarchical Classification Applied to Anomaly Detection Using System Logs with a BERT Model. *Appl. Sci.* **2024**, *14*, 5388. https://doi.org/10.3390/app14135388

Academic Editors: Sohag Kabir and Ibrahim Ghafir

Received: 16 May 2024
Revised: 7 June 2024
Accepted: 18 June 2024
Published: 21 June 2024

1. Introduction

The growth of communication networks and computer systems has led to an increase in cyberattacks and consequently the number of incidents recorded through the information residing in their logs [1]. Peripheral nodes where events occur send their logs to central nodes responsible for analyzing them and detecting anomalies that may cause unexpected system behavior. The exponential increase in log size slows the process, increases the number of false positives, and consequently generates alert fatigue when cybersecurity professionals are overwhelmed by a high volume of security alerts, reducing their ability to react and detect anomalies [2]. Logs generated by a system often have a high number of repeated or similar messages.

Modern anomaly detection systems based on logs utilize various artificial intelligence techniques, the most commonly used being those based on machine-learning algorithms [3] or natural language processing like Large Language Models [4]. These techniques consist of three phases: log parsing, feature extraction, and anomaly detection using a trained model. In log parsing, the log is converted from its semi-structured format into a format that can be analyzed by the machine. With feature extraction, the goal is to extract the most important information from the data. The third phase, model execution, has resource needs and performance proportional to the volume or number of tokens processed.

The literature includes various works that have addressed the task of log compaction, and, as in our case, have encountered various challenges. Logs can have different origins, which implies a certain diversity of words in the messages, complicating their analysis [5]. There are no dictionaries that associate all the words in a log's messages with their natural language meanings so that this log's semantics can be automatically obtained and used in its analysis [6]. The hierarchies of themes currently used refer to very restricted thematic

areas and have few levels in their hierarchical tree [7,8]. Nor is there a numerical encoding of the themes of the words that simply stores complete semantic information of each word of the log and its messages [9].

Our work processes a system log and converts it into a much smaller log with the same semantics but without repeated messages. In this way, we significantly reduce the log size, automatically decreasing the time and resources needed for its processing with an AI model, as well as reducing the number of false positives.

To carry out this task, we implemented an algorithm that develops a hierarchical classification of the themes of the words in a log and assigns each theme a numerical code, called a hierarchical code. Each of these codes represents the theme of a node in the tree and the themes of its superior nodes. A hierarchical code is formed by the concatenation of digits that identify each of the nodes in the branch of the tree that descends from the root node to the node in question. Thus, the coding performed allows a code to reflect the specificity of its theme in the hierarchy. The constructed tree has more than 15 levels and contains all names in English, which facilitates its applicability to various logs. From the themes of the hierarchical tree and their codes, we generate a catalog of words ordered alphabetically, each associated with its possible hierarchical codes, also considering the existence of synonyms. We automatically use the catalog to obtain the themes of all the messages from a system log and then compare and analyze the themes of different messages of the log among themselves to obtain a concise summary with non-repeated themes.

The rest of this article is organized as follows. Section 2 introduces previous work related to our proposal. The algorithm we developed is shown in Section 3, where we explain the thematic classifier tree, the themes and codes catalog we created, and the log analysis using hierarchical codes. Next, in Section 4, we present the experimentation performed, describing the logs used, their evaluation, and the analysis of results. Finally, Section 5 summarizes the conclusions and possible future work.

2. Related Work

Hierarchical thematic classification in system logs offers a structured and efficient methodology for reducing data size and improving anomaly detection by considering the hierarchical structure of themes and the underlying semantic relationships in system logs. Hierarchical thematic classification in a system log can be an effective strategy for reducing its size and preparing it for fine-tuning Large Language Models (LLM) in anomaly detection, which are starting to yield promising results [4]. This technique involves organizing themes in a hierarchical structure that reflects the relationship between them, which can facilitate the identification of patterns and more precise anomaly detection [10,11].

LLMs have emerged as powerful tools in the field of cybersecurity, particularly for anomaly detection. By leveraging extensive training on diverse datasets, LLMs can understand and generate human-like text, enabling them to identify unusual patterns and potential threats within network traffic and system logs [12].

By applying hierarchical classification to system logs, thematic trees can be used to analyze and categorize log messages more efficiently [13]. This methodology can be especially useful when dealing with large datasets as it allows for a more compact and structured representation of the information contained in system logs [14].

Moreover, the use of hierarchical classification models can improve the accuracy of anomaly detection by considering the latent structure of semantic relationships in the data [15,16]. These approaches allow for a more effective capture of contextual and thematic information in system logs, which can be crucial for accurately identifying anomalous behavior [17,18].

Log analysis for anomaly detection has seen significant advances with the integration of BERT (Bidirectional Encoder Representations from Transformers) models. BERT is a cutting-edge natural language processing model developed by Google using a bidirectional transformer architecture to understand the context of words from both directions, enabling deeper language comprehension [19]. Pre-trained on a large corpus, it can be

fine-tuned for specific tasks. Wang et al. [20] introduced a method that combines BERT and Variational AutoEncoders (VAE) to extract semantic and statistical features from log sequences for anomaly detection. Similarly, Almodovar et al. [21] proposed LogFiT, which fine-tunes a BERT-based language model to recognize linguistic patterns in normal log data. Guo et al. [22] further improved anomaly detection with LogBERT, a self-supervised framework based on BERT, addressing the limitations of RNN-based models. Additionally, recent studies like LogEvent2vec by Wang et al. [23] and LogEncoder by Qi et al. [24] have explored contrastive representation learning based on logs and vector transformations for anomaly detection, respectively. These approaches leverage the power of BERT to enhance the accuracy of anomaly detection. Furthermore, Lv et al. [25] proposed ConAnomaly, a model that uses log sequence encoders and LSTM networks for anomaly detection, demonstrating the versatility of log analysis techniques.

Fält et al.'s research [3] emphasizes the importance of learning normal system behavior for accurate anomaly detection, aligning with the methodology of leveraging BERT models to better understand log sequences. Moreover, Ott et al. [26] highlighted the robustness and transferability of anomaly detection models using pre-trained language models, which is crucial for real-world applications.

Our work can be considered complementary to previous studies, as it can be applied to any log analysis technique to enhance performance, particularly in terms of the response time as it will significantly reduce the log size. To validate the semantic similarity of the log records generated by our algorithm in anomaly detection, we focus on the use of Large Language Models (LLMs) because the introduction of the Transformers model [27], which laid the groundwork for LLMs, provides superior performance compared to other models, such as those based on neural networks like LSTM (Long Short-Term Memory) or autoencoders.

3. Hierarchical Classification

This work uses a thematic hierarchical classifier tree and a thematic catalog in which each theme is associated with one or more numeric codes deduced from the hierarchical tree. The themes in the catalog were represented by words that can exist in the messages; specifically, in this work, we focused on English nouns appearing in WordNet [28]. For the system log analysis process, the words of each message were read, searched for in the catalog, and their thematic codes were extracted. The analysis of these hierarchical codes allowed obtaining a semantically equivalent but much more compact log.

In the initial phase, a thematic classification tree is constructed, and a catalog of hierarchical codes is created with the themes and their relationships. Subsequently, an algorithm is developed that processes the original log to identify similar or duplicated themes, thereby compacting the input records while maintaining high semantic similarity.

The steps of this algorithm are detailed below, whose novelty lies in the use of a catalog of hierarchical codes that reflect both their thematic specificity and their position within the higher hierarchy. Unlike traditional methods that simply reduce redundancy, our algorithm preserves the semantics of the logs using a hierarchical coding system based on natural language semantics. This ensures that essential information is not lost during the compaction process. Structuring and compacting logs more effectively significantly reduces fatigue from false alerts, enhancing the efficiency of the cyberattack and anomaly detection process.

3.1. Thematic Classifier Tree

The first step was to build a thematic classifier tree in which each node represents a specialization of its parent node, forming a structure where descendant nodes encompass subtopics of their predecessors. Each node is labeled with a code that increases in length as it advances toward more specific nodes.

To define the themes of the hierarchy, we used the WordNet database, specifically the index.pos and data.pos files (where "pos" indicates the grammatical category: noun,

verb, adjective, or adverb). For example, index.noun contains an alphabetical list of nouns, each followed by offset codes representing different meanings. In data.noun, these codes, ordered incrementally, identify a set of synonyms for each meaning with a description at the end. Each entry also indicates the code of its hypernym (@) and hyponyms (~), facilitating the deduction of the thematic hierarchy. In our prototype, each word or group of words was treated as a unique theme where the offset code at the beginning of each entry in data.noun indicates the byte offset from the beginning of the file to the corresponding set of synonyms.

3.2. Catalog of Words from a Message

The catalog we created is a dictionary of words to which we associated the codes of the most specific themes they belong to, as shown in Figure 1.

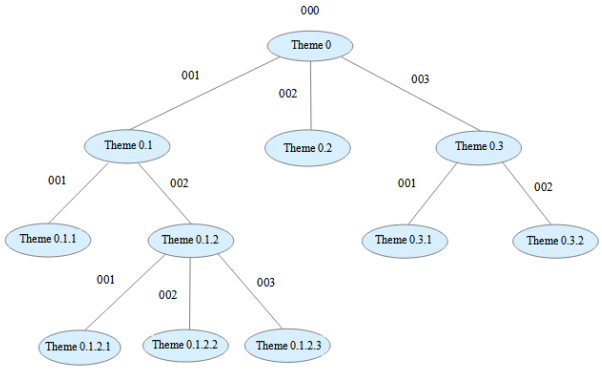

Figure 1. Generation of hierarchical codes.

To generate the hierarchical codes, our algorithm reads the data.noun file starting from the root node and descending to each child or hyponym successively, creating in this process the hierarchical codes assigned to each node. For each level descended, more digits are added to the hierarchical code being created. As a result, a file is obtained that associates each hierarchical code with an offset code.

Figure 2 shows a fragment of the generated "codes" file where the offset code appears in the first column and the hierarchical code in the second column.

```
06201838 000002001004015001001
06201959 0000020010040150010001001
06202151 000002001003002011005010002002001
06202496 0000020010030020110050100020020001001
06202620 00000200100300201100501000200200100 2
06202850 000002001003002011005010002002001002001
06203001 000002001003002011005010002002001002002
06203161 0000020010030020110050100020020001003
06203291 000002001005006022011002001
06203456 00000200100401500100 2
06204681 0000020010040150010020001
```

Figure 2. Fragment of the codes file.

The algorithm creates the hierarchical codes using character strings grouped into blocks of three digits (from 000 to 999), allowing up to 1000 possibilities per node, reflecting a theoretical maximum of 1000 children per theme. For example, Node Theme 0.1.2.2 has the hierarchical code 000001002002, indicating its position within Theme 0.1.2, which in turn has the code 00000002. The root node is identified with the code 000, simplifying the representation of the thematic tree in data_noun and facilitating future expansions.

The hierarchical codes we create are not a number of bytes, like the offset codes, but rather codes that represent the meaning of a word and all its hypernyms in the thematic hierarchy, which will be useful in thematic analysis. By looking at Figure 2, we can deduce that the more digits two hierarchical codes have in common starting from the left, the

more similar the themes they represent are, although the length of both codes must also be considered.

In the process illustrated in Figure 3, the "generatewords" algorithm transforms the index.noun file into a new file called namesjer. In this new file, the original offset codes of index.noun are replaced by hierarchical codes. Thus, each line of the namesjer file begins with a word followed by hierarchical codes representing the themes associated with that word, forming a structured catalog of themes and their corresponding hierarchical codes.

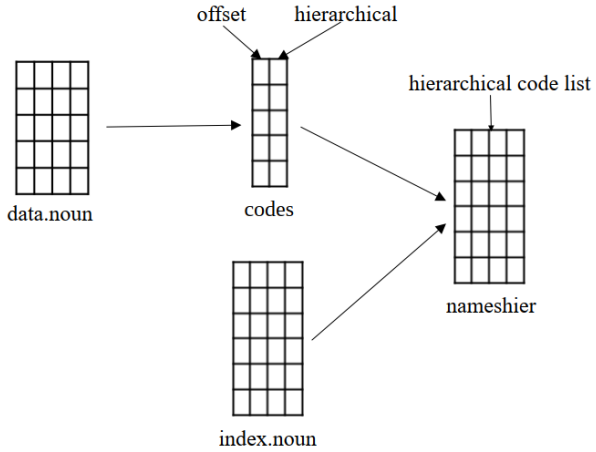

Figure 3. Creation of the world catalog and hierarchical codes.

3.3. Log Analysis Using Thematic Hierarchical Codes

The next phase involved analyzing log messages to identify common themes and eliminate duplicates. The process consisted of searching the words of each message in the catalog, extracting their hierarchical codes, and comparing them between messages. Codes that shared a common prefix up to a certain threshold indicated a similar theme. This threshold was defined as a percentage of matching digits of the shortest code, allowing the determination of thematic specificity and relationship between messages.

If we consider L as the set of all lines in the log file, each line l_i is a vector of words or tokens $w_i = w_{i1}w_{i2}\ldots w_{in}$, and w_{ij} is the j-th word in the i-th line, and n_i is the number of words in line i. We have a dictionary D that associates words w with codes c represented as a mapping $D : W \rightarrow C$, where W is the set of all possible words and C is the set of all possible codes. The transformation of each line l_i is performed using the substitution function f, which is applied to each word w_{ij} in w_i.

The function $f(w)$ is defined as:

$$f(w) = \begin{cases} D(w) & \text{if } w \in \text{dom}(D) \\ \emptyset & \text{if } w \notin \text{dom}(D) \end{cases}$$

where $\text{dom}(D)$ represents the domain of definition of the dictionary D, i.e., the set of words or tokens for which D has an associated code.

We apply f to each word in w_i to obtain a new vector of codes $c_i = f(w_{i1})f(w_{i2})\ldots f(w_{in})$, and we apply g to each line l_i of the original file, consisting of the vector w_i, to obtain a new output file:

$$g(w_i) = w_i\, c_i = v_i$$

The transformed log file is represented by a new set of vectors v_i, where each line contains the vector w_i from the original file followed by the vector c_i resulting from applying the transformation to each line l_i of the original file.

We can describe the complete transformation process of the log file as follows:

$$v_i = w_i 1 w_i 2 \ldots w_i n f(w_i 1) f(w_i 2) \ldots f(w_i n)\, \forall\, w_i \in L$$

To compensate for differences between the base dictionary and the words used in the logs, we adapted the words in the messages to identify themes rather than exact matches. The adaptations included treating the words separated by dots or that start with uppercase letters as different, even if they are joined. All uppercase letters were transformed to lowercase, and substrings such as URLs, domains, and multimedia file extensions were removed. Digits, control characters, punctuation marks (except the apostrophe that appears in the base dictionary), and functional words stored in a file called 'paldescart' were also discarded.

Additionally, accents were removed to align with the base dictionary, which lacks accented words. If the hierarchical codes of a word with "-ing" or "-ed" endings were not found, they were replaced by "-ion", and if still not found, alternatives were tried by removing the last, penultimate, or antepenultimate character. If no matches were found, it was checked if the word was in plural, consulting the 'plurirreg' file for irregular plurals and adjusting the ending as necessary to search for the singular corresponding hierarchical code.

Obtaining the Summary Log

The 'datacod' file contains rows that include both the original log message and the hierarchical codes that reflect its meaning. We implemented an algorithm that processes this file to obtain a much more compact log than the original, representing each template of the original log with a single message. The process is divided into three main modules:

First, the log entry is stripped of rows with repeated hierarchical codes. Each log row includes at the end the hierarchical codes of the message that encapsulate the themes of that message. By removing messages with duplicate themes and keeping only one per theme, templates and their original semantics are preserved, significantly reducing the log size and increasing its efficiency.

Let L be the set of all lines in the 'datacod' file, where each line l_i is an element of L, i.e., $l_i \in L$. Each line l_i is a vector $v_i = (w_i\, c_i)$, where $w_i = (w_{i1}, w_{i2}, \ldots, w_{in})$ and $c_i = (c_{i1}, c_{i2}, \ldots, c_{im})$, with w_{ix} being a token or word and c_{ix} a hierarchical code. Let n_i be the number of lines with identical hierarchical codes up to line i, i.e.:

$$n_i = \sum_{j=1}^{i} 1 \,|\, c_i = c_j$$

The transformation of the 'datacod' file to one without lines with identical codes is performed using the function f, which is applied to each vector v_i such that:

$$f(v_i) = \begin{cases} w_i\, c_i & \text{if } n_i = 1 \\ \emptyset & \text{if } n_i > 1 \end{cases}$$

The resulting log file is reduced to a set of vectors $\{v_i\}$, where each v_i is the result of applying the function f to each line l_i of the original file. We can describe the complete process on the 'datacod' file as: $\{v_i | v_i = f(v_i)\, \forall\, v_i \in L\}$.

In the second step, rows with similar meaning codes are removed from the resulting log. To determine which rows are similar, the themes of all messages are compared using a specificity threshold, which is a percentage of the shortest code's length among the two codes to be compared. Another threshold is set to determine the percentage of codes of the same theme needed to consider that two messages belong to similar themes.

Finally, the last step starts from the log resulting from the previous module and removes all hierarchical codes, leaving only the messages. The final log presents one message per existing template in the original log, simplifying the representation and improving the system's efficiency.

4. Experimental Analysis

In this phase, we evaluated the performance of the original log files and those processed with the algorithms described in the previous section. To this end, we used the BERT model (Bidirectional Encoder Representations from Transformers) [29]. Bidirectional Encoder Representations from Transformers (BERT) is based on a deep bidirectional architecture to understand the context of a word based on its surrounding words, leading to superior performance in various NLP tasks. This model pre-trains on vast amounts of text data to capture linguistic nuances, which are then fine-tuned for specific applications. BERT, with its advanced contextual understanding and ability to process large volumes of textual data, presents a promising solution for enhancing anomaly detection using system logs.

The specific checkpoint called 'bert-base-uncased' refers to a model configuration pre-trained using a vast corpus of English text without distinguishing between uppercase and lowercase letters.

For the experimental evaluation of our system, we implemented a set of tests using distributed file system logs (HDFS). For the analysis, we selected two log files: one containing a smaller set of 2000 messages and the other significantly larger with 149,477 messages and an approximate size of 20.4 MB. These data volumes were chosen to provide a clear perspective on the system's performance under different processing loads.

The format of the HDFS logs we handled follows a standardized schema that facilitates automatic analysis. Each log entry includes several comma-delimited fields: the date and time of the event (including minutes and seconds), a unique identifier for the process that generated the message, the message's severity level, the system component the message refers to, and the textual content of the message itself. This structured format is essential for the proper functioning of our program, as it allows efficient segmentation and categorization of information for subsequent processing and analysis.

4.1. Log Evaluation

For the project experimentation, we implemented software that calculated certain statistical parameters of the original HDFS logs, the 2 K messages one, and the 20 MB one, as well as the two logs processed with the semantic compaction algorithms developed in this work.

The results for the 2 K log are shown in Table 1. Analyzing the original 2 K log (HDFS_2k) and its corresponding processed log (HDFS_2k_HC), we can state that the total number of lines decreases by 99.20%. Similarly, the total number of tokens in the entire log decreases by 99.25%. The total number of hierarchical codes representing the themes of each log also decreases by 99.29%. However, the number of different hierarchical codes in the log remains the same in both the original and the processed logs (147). This allows us to conclude that the reduction in the file size and its elements has not affected its semantics and that the processed log retains the same themes as the original log.

Additionally, we focused on the distribution of rows in the original log that repeat in themes and correspond to a single row in the processed log. In the original log, there are repeated rows in 1984 cases, but there are no repeated rows in the processed log. This demonstrates that the processed log retains all the meaning with the minimum size.

Table 1. Statistics of 2 K Log.

Log	Total Lines	Total Num Tokens	Total Codes	Total Codes Dif	Total Repetition
HDFS_2k	2000	14,890	64,065	147	1984
HDFS_2k_HC	16	111	456	147	0

We performed the same statistical tests again for the original 20 MB log (HDFS_20M) and its processed log (HDFS_20M_HC) in Table 2, and the conclusions were similar. In this case, the compaction is greater, as both the number of lines of the original log, the tokens,

and the total number of hierarchical codes are reduced by 99.99%, maintaining the number of different hierarchical codes. The total number of repeated rows in the processed file is also zero, as was the case with the original 2 K log.

Table 2. Statistics of 20 MB log.

Log	Total Lines	Total Num Tokens	Total Codes	Total Codes Dif	Total Repetition
HDFS_20M	149,477	1,147,745	4,894,020	213	149,455
HDFS_20M_HC	22	188	732	213	0

Figure 4 shows a comparison that considers the number of tokens per line and the number of hierarchical codes per line for the original 2 K log. It is observed that for each line, the number of codes is higher than the number of tokens. This happens because each token can have one or several hierarchical codes since its meaning responds to more than one theme in the classifier tree due to the possibility of synonyms. It can be seen in the figure that generally, the more tokens a line has, the more hierarchical codes it will have.

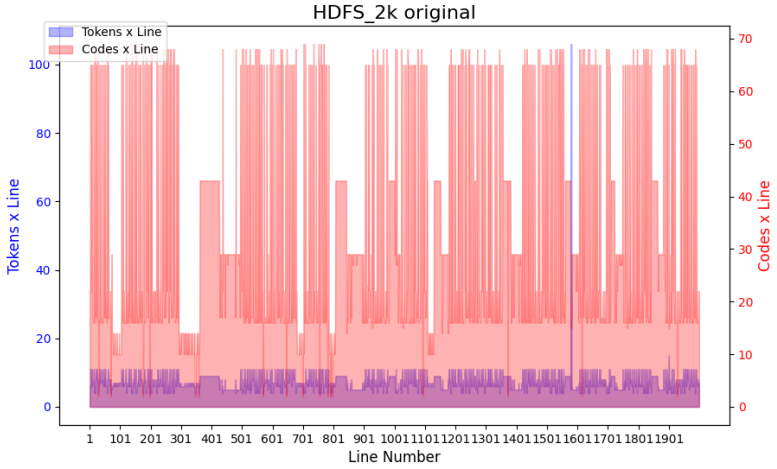

Figure 4. Statistics of original 2 K log.

A similar comparison was made with the original 20 MB log, as shown in Figure 5, and the results are similar to those of the smaller original log, i.e., the number of codes per line is higher than the number of tokens, and this is maintained with some uniformity throughout the log.

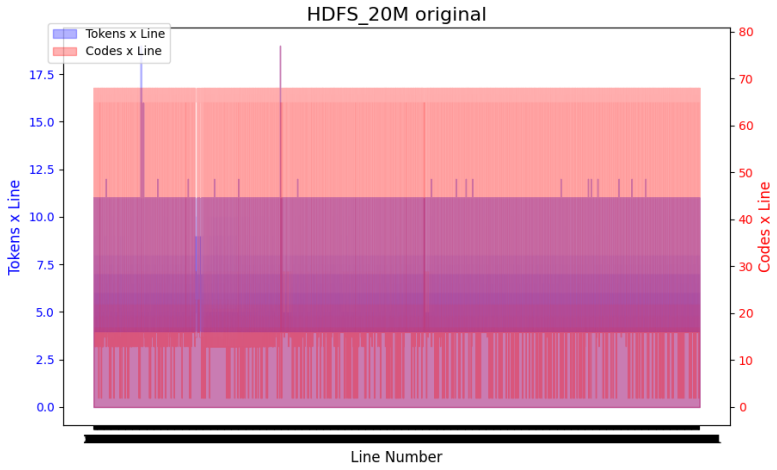

Figure 5. Statistics of original 20 MB log.

It should be noted that some tokens might not have an associated hierarchical code in the catalog. We say that tokens are categorized if they have associated hierarchical codes. Therefore, we also counted the number of categorized tokens per line, as shown in Figure 6 for the processed 2 K log and in Figure 7 for the processed 20 MB log. The proportion of categorized tokens to all tokens is significant, allowing the processed file to retain the necessary semantics of the log. With fewer rows, the processed logs show less uniformity in the relationship between tokens per line and codes per line, which would relate to greater thematic diversity in the lines, as there are no repeated rows in the processed logs.

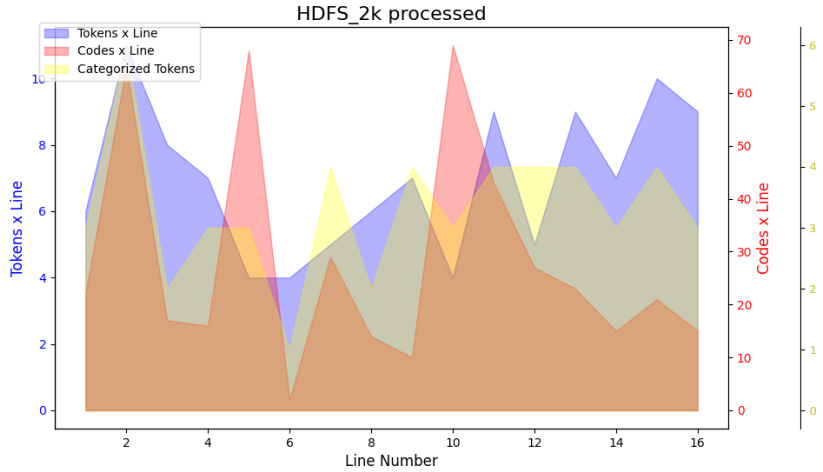

Figure 6. Statistics of processed 2 K log.

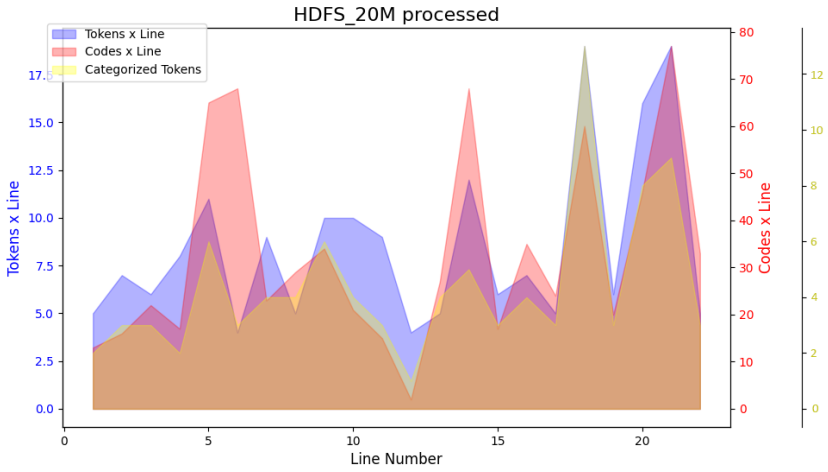

Figure 7. Statistics of processed 20 MB log.

For log processing, we used the 'transformers' and 'datasets' libraries available in Python, using the PyTorch library for optimization and efficiency, a widely used open-source machine learning library for deep learning applications.

First, with a batch size of 8, we iterate through the log file lines, obtaining the following tokenization statistics:

```
data_metrics = {

    'dataset'        :  Dataset name
    'lineCount'      :  Number of lines in the dataset
    'batchSize'      :  Number of lines in each batch
    'tokensCount'    :  Total number of tokens in the dataset
    'tokensMeanBatch':  Mean number of tokens in each batch
    'tokensStdBatch' :  Standard deviation of tokens between batches
```

```
    'tokenizedTime'  :  Total tokenization time of the dataset
    'modelizedTime'  :  Total modeling time of the dataset
    'meanBatchTime'  :  Mean execution time of a batch
    'tokensBatchList':  List with the number of tokens in each batch
    'timeBatchList'  :  List with the processing time per batch

}
```

For each log line, we store the processed tokens per line and the processing time for each line.

Table 3 presents a quantitative analysis of four datasets derived from the Hadoop Distributed File System (HDFS), differentiated by their data volume. The suffix "HC" refers to the log version processed with the hierarchical classification method described earlier. The "totalLines" attribute refers to the total number of lines per dataset, with "HDFS_2k" and "HDFS_2k_HC" containing 2000 and 16 lines, respectively, and "HDFS_20M" and "HDFS_20M_HC" containing 14,947,770 and 22 lines, respectively. The "tokensCount" indicates the total number of tokens, being substantially higher in the larger datasets. The mean ("tokensMeanBatch") and standard deviation ("tokensStdBatch") of tokens per batch provide a measure of centrality and dispersion, respectively, in the distribution of the number of tokens per batch within each dataset. Notably, the "HC" subsets show a higher mean of tokens per batch, indicating higher lexical density or information concentration.

Table 3. Tokens per line.

Log	Total Lines	Tokens Count	Tokens Mean Batch	Tokens Std Batch
HDFS_2k	2000	152,960	611	234
HDFS_2k_HC	16	1344	672	16
HDFS_20M	149,477	11,043,195	591	54
HDFS_20M_HC	22	1944	648	28

Table 4 compares the performance metrics of two main datasets "HDFS_2k" and "HDFS_20M" along with their corresponding "HC" subsets. The "lineCount" metric indicates the number of lines in each dataset, providing context for interpreting the subsequent metrics. "Tokenized Time" indicates the time in seconds required to tokenize the datasets, with the "HC" subsets requiring less time due to their smaller size. "Modelized Time" denotes the time needed to process the 'bert-base-uncased' model, with the "HDFS_20M" set requiring significantly more time, consistent with its larger size. Finally, "Mean Batch Time" represents the average time taken per batch, which remains relatively constant across all cases, suggesting that the time per batch is independent of the dataset's total size and that the "HC" subsets have characteristics that require more processing time per batch, such as greater data complexity.

Table 4. Processing times per line.

Log	Total Lines	Tokenized Time	Modelized Time	Mean Batch Time
HDFS_2k	2000	0.8816	54.0701	0.2156
HDFS_2k_HC	16	0.2109	0.5704	0.2842
HDFS_20M	149,477	50.0624	4039.0378	0.2155
HDFS_20M_HC	22	0.2086	0.8397	0.2791

4.2. Analysis of Results

As seen in the previous section and as might be evident, reducing the log size significantly improves its processing. When evaluating how similar the original and processed logs are using hierarchical classification, we focus on creating a similarity matrix between the lines of each analyzed set, both original and processed, and between them.

Figure 8 shows two heatmaps of similarity matrices for HDFS log records. In the graph on the left, the matrix for the "HDFS_2k vs. HDFS_2k" dataset shows uniformity in color, suggesting high similarity between the log lines, implying little variation between them, as expected in unprocessed log records where repetitive messages are common. On the other hand, the graph on the right shows the matrix "HDFS_2k_HC vs. HDFS_2k_HC" corresponding to a processed dataset of high cardinality. The more varied colors in this matrix indicate more significant differences in similarity between the log lines, suggesting that processing has highlighted or preserved the most unique lines or anomalies, leading to greater diversity in the log content. The presence of dark red blocks indicates pairs of lines with high similarity, while lighter colors represent lower similarity. This reflects the typical purpose of a high-cardinality process, which is to identify and isolate the less frequent or more distinct events within a large dataset.

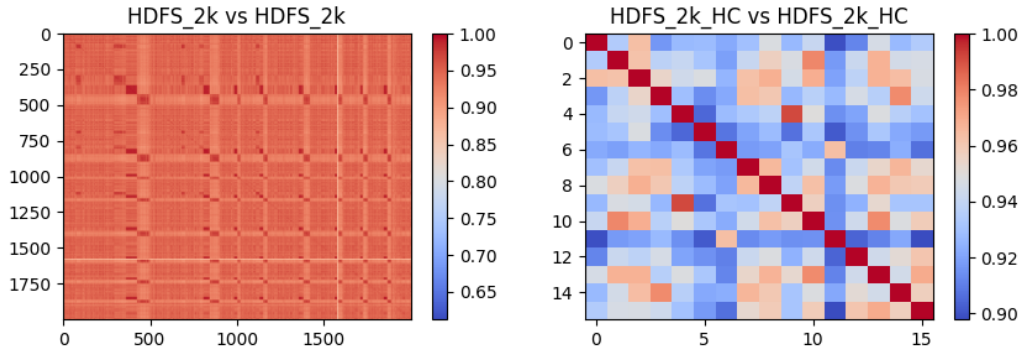

Figure 8. Internal similarity of original and processed log.

Figure 9 presents a heatmap showing a range of similarities between the lines of the original "HDFS_2k" dataset and its processed version "HDFS_2k_HC". The dominant red tones indicate high similarity, while blue represents lower similarity.

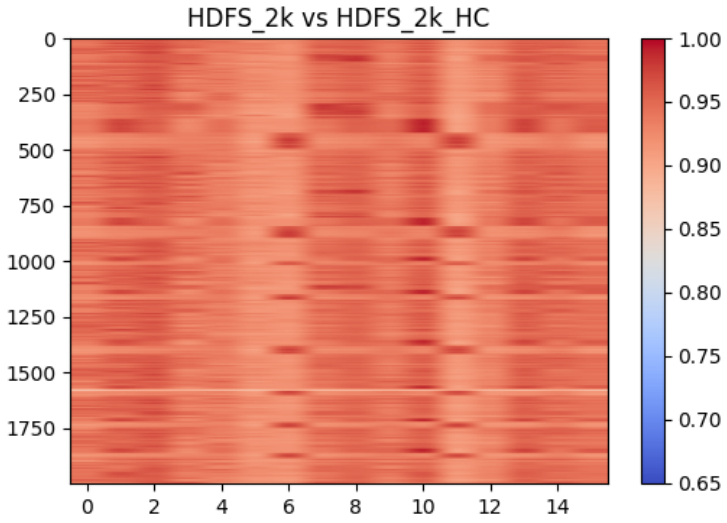

Figure 9. Similarity between the original and processed log.

The predominance of red throughout the map suggests that despite the processing, there is notable consistency or significant similarity between the corresponding lines of both logs. The lack of distinct blue blocks also indicates that there are no lines between the datasets that are entirely different. The processing aimed to preserve essential information while reducing redundancy; this heatmap suggests that a considerable degree of the original information has been retained in the processed version.

To further compare the similarity between the original and processed log, we used the cosine similarity function from the PyTorch deep learning library to calculate the similarity between two sets of embeddings. The torch.nn.functional .cosine_similarity function calculates the cosine similarity between two tensors, which in this context are embeddings corresponding to the two tokenized log files. Cosine similarity measures the cosine of the angle between two vectors in a multidimensional space, providing an indicator of their relative orientation but not their magnitude.

Cosine similarity between two vectors is calculated using the following formula:

$$\text{cosine similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$$

In the case of vectors with only positive components, as is common in text representations and many other applications, the cosine similarity will be between 0 and 1.

Using cosine similarity between the two datasets, we obtain a value of 0.9724, indicating that the vectors represented in these tensors are very similar in terms of their orientation in vector space. In other words, the embeddings of both tensors have very similar directions, even if their magnitudes (i.e., the length of the vectors) might be different.

Since cosine similarity focuses on orientation and not magnitude, this high value suggests that the data represented by these embeddings share many features or patterns in the context being modeled by the vector space. Given that these embeddings result from running the 'bert-base-uncased' model on both the original and processed log files, it can be inferred that they share a similar context or semantic meaning.

In the previous section, we verified the significant reduction in processing time of the 2 K and 20 M logs, achieving reductions of 760.8% and 995.8%, respectively, in tokenization time, and 989.5% and 999.8%, respectively, in BERT model training time.

Finally, we analyzed the results of applying the BERT model to the processed 20 M log (HDFS_20M_HC) for anomaly detection, comparing it with other methods present in the literature. These methods are briefly described below:

- PCA [13]: this work converts logs into vectors and then uses the Principal Component Analysis (PCA) algorithm to divide them into normal space and anomaly space.
- Invariant Mining [30]: this paper proposes an Invariant Mining (IM) method to extract invariants from log vectors, which are labeled as anomalous logs.
- LogCluster [31]: LogCluster proposes an approach that clusters logs for easier identification of issues based on logs and uses a knowledge base to reduce redundancy by pre-examining log sequences.
- SVM [32]: this work represents log sequences as vectors and uses a Support Vector Machine (SVM) as its classification algorithm.
- DeepLog [33]: DeepLog is a deep neural network model that uses LSTM to model a system log as a natural language sequence.
- LogRobust [34]: LogRobust extracts semantic information from log events and represents them as semantic vectors. It uses a Bi-LSTM attention-based model to detect anomalous log sequences.

To carry out this comparison, we used traditional evaluation metrics: precision, recall, and F1-Score.

As shown in Table 5, our model (BERT_20M) has a precision of 88%, a recall of 86%, and an F1-Score of 83%. Comparing these results with those obtained by other methods, we observe that BERT_20M has a precision of 88%, which is lower than the best methods such as PCA, IM, LogCluster, and DeepLog, which reach 100% precision. However, it outperforms SVM, which has the worst precision with 62%.

Table 5. Evaluation of HDFS_20M_HC log.

Model	Precision	Recall	F1-Score
PCA	1.00	0.67	0.70
IM	1.00	0.83	0.80
LogCluster	1.00	0.96	0.96
SVM	0.62	0.73	0.73
DeepLog	1.00	0.78	0.98
LogRobust	0.81	0.91	0.91
BERT_20M	0.88	0.86	0.83

In terms of recall, BERT_20M obtains 86%, which is lower than the recall of LogCluster (96%) and LogRobust (91%), but significantly better than PCA, which has the worst recall with 67%. Regarding the F1-Score, BERT_20M achieves a value of 0.83. Although it is lower than LogCluster (0.96) and DeepLog (0.98), it surpasses PCA (0.70), which has the lowest F1-Score.

Although the BERT_20M model does not achieve the best metrics compared to LogCluster and DeepLog, it offers considerably better performance than PCA and SVM.

5. Conclusions and Future Research

In this work, we initially transcribed the thematic contents inherent to system logs using a numerical coding scheme that assigns unique identifiers to each topic, modeling thematic specificity and its contextual interconnections. Subsequently, we performed an extraction and analytical synthesis operation on the thematic vectors corresponding to the entries in an HDFS system log, allowing the obtainment of a much-condensed log, reducing the number of lines, tokens, and thematic codes by 99%, and maintaining semantic similarity.

The flexibility of the methodological approach suggests a cross-sectional adaptability for its application in HDFS log files and other computer system architectures. The expansion of this prototype to a thematic plurality could be carried out by integrating Wordnet World instead of WordNet, which would expand the lexical index's linguistic spectrum to multiple languages.

A comparative study of the pre- and post-processing datasets was conducted using hierarchical thematic classification models. The intrinsic similarity preservation of the generated embeddings was evidenced, demonstrating a high cosine similarity coefficient (0.9724), thus corroborating the retention of a shared core of meaning and semantic context between the two data instances.

This methodology was subjected to an empirical analysis of its effectiveness and efficiency using a 'bert-base-uncased' model. The quantitative evaluation focused on traditional precision, recall, and F1-score metrics, demonstrating that the application of this compaction technique not only maintains semantic similarity but also allows anomaly detection similar to other methods used in the literature.

In conclusion, the implementation of hierarchical thematic classification in system log collection systems is revealed as an efficient vector for data compression without compromising semantic integrity. This compression proves to be a key component in monitoring infrastructure, optimizing both alert management in Security Information and Event Management (SIEM) systems and anomaly detection through deep learning language models (LLMs), mitigating operational overload and alert fatigue.

Author Contributions: Conceptualization, V.C. and C.C.; methodology, V.C.; software, V.C. and C.C.; validation, V.C.; formal analysis, V.C. and C.C.; investigation, V.C. and C.C.; resources, V.C. and C.C.; data curation, V.C. and C.C.; writing—original draft preparation, V.C. and C.C.; writing—review and editing, F.C., V.C. and F.C.; visualization, V.C. and F.C.; supervision, V.C. and F.C.; project administration, V.C. and F.C.; funding acquisition, F.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ustun, T.S.; Hussain, S.M.S.; Ulutas, A.; Onen, A.; Roomi, M.M.; Mashima, D. Machine Learning-Based Intrusion Detection for Achieving Cybersecurity in Smart Grids Using IEC 61850 GOOSE Messages. *Symmetry* **2021**, *13*, 826. [CrossRef]
2. Ban, T.; Takahashi, T.; Ndichu, S.; Inoue, D. Breaking Alert Fatigue: AI-Assisted SIEM Framework for Effective Incident Response. *Appl. Sci.* **2023**, *13*, 6610. [CrossRef]
3. Fält, M.; Forsström, S.; Zhang, T. Machine Learning Based Anomaly Detection of Log Files Using Ensemble Learning and Self-Attention. In Proceedings of the 2021 5th International Conference on System Reliability and Safety (ICSRS), Palermo, Italy, 24–26 November 2021; pp. 209–215. [CrossRef]
4. Balasubramanian, P.; Seby, J.; Kostakos, P. Transformer-based LLMs in Cybersecurity: An in-depth Study on Log Anomaly Detection and Conversational Defense Mechanisms. In Proceedings of the 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 15–18 December 2023; pp. 3590–3599. [CrossRef]
5. Liu, Y.; Zhang, X.; He, S.; Zhang, H.; Li, L.; Kang, Y.; Xu, Y.; Ma, M.; Lin, Q.; Dang, Y.; et al. UniParser: A Unified Log Parser for Heterogeneous Log Data. In Proceedings of the ACM Web Conference, Lyon, France, 25–29 April 2022; pp. 1893–1901. [CrossRef]
6. Ekelhart, A.; Kiesling, E.; Kurniawan, K. Taming the logs—Vocabularies for semantic security analysis. *Procedia Comput. Sci.* **2018**, *137*, 109–119. [CrossRef]
7. Gu, X.; Ding, W. A hierarchical prototype-based approach for classification. *Inf. Sci. Int. J.* **2019**, *505*, 325–351. [CrossRef]
8. Silla, C.N.; Freitas, A.A. A survey of hierarchical classification across different application domains. *Data Min. Knowl. Discov.* **2011**, *22*, 31–72. [CrossRef]
9. Altaf, A.; Mohamed, D.; Soumia, Z.; Badia, E.; Jamal, Z. Model for data codification in hierarchical classifications with application to the biodiversity domain. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 1426–1433. [CrossRef]
10. Chen, C.-M.; Ho, S.-Y.; Chang, C. A hierarchical topic analysis tool to facilitate digital humanities research. *Aslib J. Inf. Manag.* **2022**, *75*, 1–19. [CrossRef]
11. Chen, P.; Zhang, N.L.; Liu, T.; Poon, L.K.M.; Chen, Z.; Khawar, F. Latent tree models for hierarchical topic detection. *Artif. Intell.* **2017**, *250*, 105–124. [CrossRef]
12. Zhang, J.; Bu, H.; Wen, H.; Chen, Y.; Li, L.; Zhu, H. When llms meet cybersecurity: A systematic literature review. *arXiv* **2024**, arXiv:2405.03644. [CrossRef]
13. Xu, W.; Huang, L.; Fox, A.; Patterson, D.; Jordan, M.I. Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd Symposium on OPERATING Systems Principles, Big Sky, MT, USA, 11–14 October 2009; pp. 117–132. [CrossRef]
14. Beitzel, S.M.; Jensen, E.C.; Chowdhury, A.; Frieder, O.; Grossman, D. Temporal analysis of a very large topically categorized Web query log. *J. Am. Soc. Inf. Sci. Technol.* **2007**, *58*, 166–178. [CrossRef]
15. Li, R.; Lin, C.; Collinson, M.; Li, X.; Chen, G. A Dual-Attention Hierarchical Recurrent Neural Network for Dialogue Act Classification. In Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL), Hong Kong, China, 3–4 November 2019; pp. 383–392. [CrossRef]
16. Rizun, N.; Taranenko, Y.; Waloszek, W. Improving the Accuracy in Sentiment Classification in the Light of Modelling the Latent Semantic Relations. *Information* **2018**, *9*, 307. [CrossRef]
17. Fang, Y.; Zhao, Z.; Xu, Y.; Liu, Z. Log Anomaly Detection Based on Hierarchical Graph Neural Network and Label Contrastive Coding. *Comput. Mater. Contin.* **2022**, *74*, 4099–4118. [CrossRef]
18. Wang, J.; Tang, Y.; He, S.; Zhao, C.; Sharma, P.K.; Alfarraj, O.; Tolba, A. LogEvent2vec: LogEvent-to-Vector Based Anomaly Detection for Large-Scale Logs in Internet of Things. *Sensors* **2020**, *20*, 2451. [CrossRef] [PubMed]
19. Rogers, A.; Kovaleva, O.; Rumshisky, A. A primer in BERTology: What we know about how BERT works. *Trans. Assoc. Comput. Linguist.* **2021**, *8*, 842–866. [CrossRef]
20. Wang, Q.; Zhang, X.; Wang, X.; Cao, Z. Log Sequence Anomaly Detection Method Based on Contrastive Adversarial Training and Dual Feature Extraction. *Entropy* **2022**, *24*, 69. [CrossRef]
21. Almodovar, C.; Sabrina, F.; Karimi, S.; Azad, S. LogFiT: Log Anomaly Detection using Fine-Tuned Language Models. *IEEE Trans. Netw. Serv. Manag.* **2024**, *21*, 1715–1723. [CrossRef]
22. Guo, H.; Yuan, S.; Wu, X. LogBERT: Log Anomaly Detection via BERT. *arXiv* **2021**, arXiv:2103.04475. [CrossRef]
23. Wang, R.; Nie, K.; Wang, T.; Yang, Y.; Long, B. Deep Learning for Anomaly Detection. In Proceedings of the 13th International Conference on Web Search and Data Mining, Houston, TX, USA, 3–7 February 2020; pp. 894–896. [CrossRef]

24. Qi, J.; Luan, Z.; Huang, S.; Fung, C.; Yang, H.; Li, H.; Zhu, D.; Qian, D. LogEncoder: Log-Based Contrastive Representation Learning for Anomaly Detection. *IEEE Trans. Netw. Serv. Manag.* **2023**, *20*, 1378–1391. [CrossRef]
25. Lv, D.; Luktarhan, N.; Chen, Y. ConAnomaly: Content-Based Anomaly Detection for System Logs. *Sensors* **2021**, *21*, 6125. [CrossRef]
26. Ott, H.; Bogatinovski, J.; Acker, A.; Nedelkoski, S.; Kao, O. Robust and Transferable Anomaly Detection in Log Data using Pre-Trained Language Models. In Proceedings of the 2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence), Madrid, Spain, 29 May 2021; pp. 19–24. [CrossRef]
27. Parmar, N.; Vaswani, A.; Uszkoreit, J.; Kaiser, L.; Shazeer, N.; Ku, A.; Tran, D. Image Transformer. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 4055–4064. Available online: https://proceedings.mlr.press/v80/parmar18a.html (accessed on 1 September 2023).
28. Miller, G.A.; Beckwith, R.; Fellbaum, C.; Gross, D.; Miller, K.J. Introduction to WordNet: An On-line Lexical Database*. *Int. J. Lexicogr.* **1990**, *3*, 235–244. [CrossRef]
29. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805. [CrossRef]
30. Lou, J.-G.; Fu, Q.; Yang, S.; Xu, Y.; Li, J. Mining invariants from console logs for system problem detection. In Proceedings of the 2010 USENIX Annual Technical Conference (USENIX ATC 10), Boston, MA, USA, 23–25 June 2010. Available online: https://www.usenix.org/event/atc10/tech/full_papers/Lou.pdf (accessed on 12 October 2023).
31. Lin, Q.; Zhang, H.; Lou, J.-G.; Zhang, Y.; Chen, X. Log clustering based problem identification for online service systems. In Proceedings of the 38th International Conference on Software Engineering Companion, Austin, TX, USA, 14–22 May 2016; pp. 102–111. [CrossRef]
32. He, P.; Zhu, J.; He, S.; Li, J.; Lyu, M.R. Towards Automated Log Parsing for Large-Scale Log Data Analysis. *IEEE Trans. Dependable Secur. Comput.* **2018**, *15*, 931–944. [CrossRef]
33. Du, M.; Li, F.; Zheng, G.; Srikumar, V. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298. [CrossRef]
34. Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; et al. Robust log-based anomaly detection on unstable log data. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Tallinn, Estonia, 26–30 August 2019; pp. 807–817. [CrossRef]