No Features Needed: Using BPE Sequence Embeddings for Web Log Anomaly Detection

Nitesh Suresh Sehwani
sehwan@amazon.com
Amazon Web Services, Inc.
Seattle, Washington, USA

ABSTRACT

Problem: Manual data analysis for extracting useful features in web log anomaly detection can be costly and time-consuming. Automated techniques on the other hand (e.g. Auto-Encoders and CNNs based) usually require supplemental network trainings for feature extractions. Often the systems trained on these features suffer from high False Positive Rates (FPRs) and rectifying them can negatively impact accuracies and add training/tuning delays. Thus manual analysis delays, mandatory supplementary trainings and inferior detection outcomes are the limitations in contemporary web log anomaly detection systems.

Proposal: Byte Pair Encoding (BPE) is an automated data representation scheme which requires no training, and only needs a single parsing run for tokenizing available data. Models trained using BPE-based vectors have shown to outperform models trained on similar representations, in tasks such as machine translation (NMT) and language generation (NLG). We therefore propose to use BPE tokens obtained from web log data and consequently vectorized by a *pre-trained* sequence embedding model for performing web log anomaly detection. Our experiments using two public data sets show that ML models trained on BPE sequence vectors, achieve better results compared to training on both manually and automatically extracted features. Moreover our technique of obtaining log representations is fully automated (requiring only a single hyper-parameter), needs no additional network training and provides representations that give consistent performance across different ML algorithms (a facet absent from feature-based techniques). The only trade-off with our method is an increased upper limit in system memory consumption, as compared to using manual features. This is due to the higher dimensions of the utilized pre-trained embeddings, and reducing it, is our motivation for future work.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation, Systems security, Network security, Software and application security**.

KEYWORDS

Anomaly detection, byte-pair encoding, sequence embedding

IWSPA '22, April 24–27, 2022, Baltimore, MD, USA

ACM ISBN 978-1-4503-9230-3/22/04.
https://doi.org/10.1145/3510548.3519375

ACM Reference Format:
Nitesh Suresh Sehwani. 2022. No Features Needed: Using BPE Sequence Embeddings for Web Log Anomaly Detection. In *Proceedings of the 2022 ACM International Workshop on Security and Privacy Analytics (IWSPA '22), April 24–27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3510548.3519375

1 INTRODUCTION

Manually extracting features from HTTP traffic logs involves feature constructions based on expert experiences and heuristics. These include analyzing logs for presence/absence of special characters, appearance frequencies of unusual elements, characterization of request paths, entropy of request parameters, etc. [2, 12, 13]. A common practice is to initially collect a large number of such features, and then identify the most discriminative ones using feature selection algorithms such as SVDD [3] and Evolutionary Algorithms [4]. Such selection procedures increase complexities, as well as the training times of anomaly detection systems. Additionally, with an exploding amount of logs in modern day systems, manual analysis becomes increasingly infeasible. Moreover if the data changes considerably over time, periodic re-analysis becomes necessary, which can become tedious and expensive with manual examination [1].

Automated feature extraction techniques on the other hand, mainly involve training supplementary networks (e.g. Auto-Encoders [5-6] and CNNs [14]) on parsed log sequences [8]. These training modules compound the execution overhead of the system, as they themselves need to be tuned with suitable hyper-parameters. Additionally such feature learning networks can frequently over-train, thus needing variant-inducing techniques such as Gaussian noise induction [23] for error correction. Without such corrections, the systems can often suffer from lower precisions.

Byte Pair Encodings (BPE) are widely used in NLP to encode data sequences with most frequent character n-grams obtained from the training data [11]. Essentially, it iteratively replaces the most common pairs of consecutive characters by their combination until either the vocabulary limit or the merge limit is reached. They're similar in nature to the existing automated feature extraction techniques; but importantly, the BPE extraction process does not involve any network training steps and only requires a single hyper-parameter (number of token merges) for tokenizing raw data sequences. Secondly, a key concern in anomaly detection systems, is their performance on test data unseen during training [40]. Such sequences when un-accounted for, contribute towards higher FPRs. But this requirement can be easily handled by BPE tokens, given their evident effectiveness against the *unseen token* problem [41]. Thus BPE extractions are relatively quick, their representations perform well on unseen data and they add minimum overhead to

systems while extracting representations. This makes them well suited as a log vectorization scheme in anomaly detection systems. We thus explore using BPE tokens for tokenizing log data sequences in web log anomaly detection.

Post extraction, BPE tokens can be converted to suitable vectors either by training ML models to extract these vectors, or by using a pre-trained model. Pre-trained embeddings obtained from language data, have been successfully used to obtain suitable vectors in diverse tasks, such as code retrieval/function-name prediction [42] and health informatics [24]. In the latter study, the embeddings were even found to outperform specifically trained vectors for the task-at-hand. Utilizing pre-trained embeddings excises the need of training to obtain text vectors, which aligns with our tenet of having minimum-to-no training for obtaining data representations. It also reduces system complexities by eliminating additional training modules. Thus we choose this approach in our system design to extract data vectors. Furthermore, as opposed to word embedding models which generate per-word vectors (such as Word2vec [43] and FastText [7]), sequence embedding models (such as Universal Sentence Encoder (USE) [10]) encode individual sequences into vectors using linguistic characteristics such as context. Recent works have shown that adding such sequence semantics to vector representations, can help in improving detection accuracies of downstream anomaly detectors [9]. To the best of our knowledge, utilizing BPE tokens with pre-trained sequence embeddings for obtaining log data representations, has not been previously explored towards anomaly detection. This is what we thus explore in this work and our technique is as follows:

- Raw web log data is fed to a BPE tokenizer which identifies the most frequent character n-grams in the data. After the tokenizer is learnt, each log sequence is then tokenized using corresponding BPE token IDs.
- The tokenized log sequences are fed to a publicly available, pre-trained sequence embedding model. In this work we use the USE model, which encodes each sequence into a 512 dimensional vector. USE was deigned to be *universally* applicable across tasks in multiple domains via a *transfer learning* approach [10] which makes the case for promising results.
- The encoded sequence vectors are then converted to an appropriate ingestion format (arrays, frames, etc), before being fed to anomaly detection ML algorithms.

Our objectives of obtaining data representations using the above approach are to:

- Have an automated technique for obtaining log data representations which requires no supplementary network training, and provides better accuracies and FPRs compared to existing automated techniques.
- Have data representations which provide similar or better anomaly detection outcomes compared to using manual features.
- Eliminate the long time and efforts spent by experts in extracting features for training anomaly detection systems.

The remaining work is organized as follows: In Section 2 we catalog the relevant work in data representations for web log anomaly detection, in Section 3 we explain our approach, in Section 4 we brief about the data used and in Section 5 we list the different techniques compared and algorithms used for the analysis. The results are analyzed in Section 6 and Section 7 concludes our work. Finally Section 8 is our acknowledgement.

2 RELATED WORK

Manually extracted features are often sifted through to improve detection outcomes. This requires supplemental expert knowledge and testing to obtain a suitable combination of features, or requires genetic algorithms to find the best blend of features [4-5].

Automated feature extractions can fall into 2 main categories: techniques requiring neural network trainings and those using linguistic heuristics for extracting feature vectors. Examples using neural networks trainings include [5] which uses auto-encoders to directly encode log sequence bigrams into 300 dimensional vectors, and [14] where CNNs are trained to extract high level features from HTTP communication data. In the second category, hybrid systems like LogAnomaly [16] generate feature vectors by combining word synonyms/antonyms pre-determined by experts, with network trainings using lexical-contrast embeddings. The resultant vectors are then used for anomaly detection. While these automated systems take out the manual component in obtaining log sequence representations, they have their own shortcomings. The extra network trainings generate unnecessary overhead. These networks need to be separately tuned for obtaining meaningful representations, apart from the actual training of the anomaly detection system itself. Moreover the training data may fall short in fully training the feature extraction networks, which then needs to be separately dealt with [17]. Also while adding manual loops can improve detection outcomes, they can add delays and become a bottleneck in production settings.

A closely related category of automated techniques uses word embedding models (e.g. FastText [7]) to extract useful vectors from log data. These techniques don't require network trainings, but instead need tuning of word embedding models on available data. A fundamental drawback is that they do not appropriately account for neighbor-word contexts leading to incorrect vector representations [16]. For example, if 2 words NEXT and UP in separate log sequences have the same surrounding words, their word vectors would be similar despite them representing potentially different system statuses. Such a loss of context at the sequence level can be a drawback in systems relying solely on word level models. While this can be palliated with the addition of linguistic features, it again requires expert knowledge. Moreover such linguistic components need to be constantly updated in case of out-of-vocabulary (oov) words, that are often seen during run-times [18].

The work that comes closest to our approach is [19]: there the authors first apply BPE and TF-IDF weightings to raw HTTP logs, along with a simple separation of tokens based on special characters. The sequences are then tokenized with BPE vocabulary and passed on to an auto-encoder network, which converts BPE-tokenized sequences into 250 dimensional vectors. These vectors are then fed to anomaly detection algorithms and their performance is analyzed. While based on a similar strategy, our approach differs from this technique in the following crucial ways:

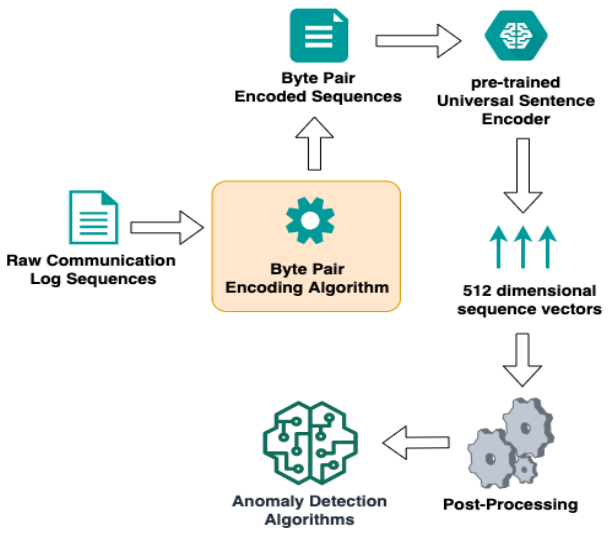

Figure 1: Log Sequence Vectorization with BPE and contextual embeddings

- After applying BPE segmentation to log sequences, we avoid using TF-IDF weights for encoding log sequence vectors. Instead, we apply a pre-trained sequence embedding model (USE) on BPE tokens for encoding log sequences into vectors.
- We don't perform any sequence partitioning based on non-alphabetical and non-numerical characters. Any grouping of characters is solely determined by the outcome of the BPE execution process.
- No auto-encoders are required for obtaining log sequence vectors from BPE segmented sequences. Instead the embeddings emitted from the USE model running over theses sequences, are directly fed into the anomaly detection algorithms.

In toto, we surmise that no supplemental word weightings, sequence separations or network trainings are necessary to obtain useful representations from communication data logs. In the next section, we elucidate our approach with an accompanying illustration.

3 APPROACH

Initially, a BPE tokenizer iterates over all sequences in our data and tokenizes them according to [11]. The number of BPE tokenizer merge operations is the only hyper-parameter needed in our approach. As note in [33], 30k-50k merge operations is the most common range of merge operations in NMT literature, and so we test our approach for 30k, 40k and 50k merge operations respectively. More details on the BPE implementation can be found in Appendix A.1.

After the tokenizer completes executing, the tokenized sequences are delivered to a publicly available pre-trained USE model. For each BPE tokenized sequence that the USE model ingests, it outputs a 512-dimensional embedding vector. Once the model has converted all sequences to their corresponding embedding vectors, they are converted and grouped together as *numpy* arrays, before being passed on to anomaly detection algorithms (this simple post-processing step, we observed, reduces the average total execution time by 29%). A quick primer on USE and its implementation in this work can be found in Appendix A.2. The illustration in Fig. 1 depicts our entire approach.

4 DATA USED

We use two public data sets for evaluating our approach: HTTP CSIC 2010 [20] and ISCX IDS 2012 [28]. Both these data sets contain training data consisting of benign traffic and testing data consisting of malevolent requests. We compare the performance of our technique with that of the others, by training them on normal traffic data and consequently testing their outcomes on the anomalous traffic data. This is done for both the data sets.

4.1 HTTP CSIC 2010

While numerous well-known attacks are present in the anomalous traffic set of HTTP CSIC 2010, there are also many examples which are trivial variants of normal traffic examples, obtained simply by adding/substituting existing groups of characters. These are not actually indicative of real-world anomalous behaviors. So for evaluations on HTTP CSIC 2010, we filter out such slightly mutated examples and only select those that actually represent attack-oriented traffic. These comprise attacks such as remote CLI script executions, cookie attacks, buffer overload attempts and web script executions amongst others. The result is about 1000 testing instances (including normal traffic requests). The number of training instances is 36,000.

4.2 ISCX IDS 2012

We similarly extract malicious requests for ISCX IDS 2012 - these are obtained from the traffic logs on 13th and 14th June (described in [28] as α profiles) from 7 days of available logs. The infiltrations in the 13th June subset mainly include internal scans and network mappings in addition to normal traffic. The traffic on 14th June consists of HTTP DOS attacks along with usual traffic. The malicious examples selected from both these days finally include network mapping/scanning traffic, vulnerability identification requests, DOS attacks and code injection attempts. The total number of testing instances comes to about 15,000 (including normal traffic requests). The training traffic is randomly sampled from the normal traffic on both the above days as well as from the traffic logs on 11th June, when there were no attacks. The total number of training instances is about 280,000.

5 EVALUATING OUR APPROACH

The anomaly detection algorithms used in our evaluation are One-class SVMs (OC-SVMs) [21] and Isolation Forests (iForests) [22]. These are the two most widely used algorithms for anomaly detection when only attack-free training data is available (this is usually the case in real-life settings). The goal is to compare our representations with features extracted manually as well as automatically. For this we compare the outcomes of anomaly detection algorithms when trained using our representations, versus manual features for HTTP CSIC 2010 and ISCX IDS 2012 from [20] as wells as features extracted automatically using Auto-Encoders in [6]. We

also compare our technique with the FastText-based features approach in [8] since this is another popular automated technique. Additionally we compare our technique's performance with similar BPE-based vectors from [19], to gauge the outcome differences of both the BPE-based approaches.

For each data set, the parameters for Isolation Forests are ranged at 100/200 iTrees and 256/8192 samples per iTree, similar to [4]. For OC-SVMs, both γ (kernel co-efficient) and ν (upper-bound on fraction of training errors) are ranged between 0.001 and 10000 each.

6 RESULTS AND ANALYSIS

Our results are divided into 4 sub-sections: (6.1) Quantitative comparison of detection results (using Table 1 and 2), (6.2) Qualitative analysis of the results obtained, (6.3) Comparison of average execution times of all techniques (Table 3) and (6.4) Comparing system memory trade-offs of all techniques.

For the only hyper-parameter in our work, *the number of BPE merge operations*, we found the best results at 30k merge operations for both data sets; and so we only report detailed results for our technique at 30k merge operations. For the sake of comparison, the degradation in results at higher merge operations (up to 50k), was up to an 8% decrease in accuracy, and an increase of up to 10% in FPRs, averaged across both the data sets and algorithms.

6.1 Quantitative Analysis

For both data sets and for both training algorithms, training on *BPE + USE* embeddings results either in higher accuracies compared to manual features, or equally good ones. For HTTP CSIC 2012 data set, BPE-based embeddings also provide lower FPRs than manual features for both algorithms (with at least a 4% improvement). A similar outcome is observed on the ISCX IDS 2012 data set. Our technique also outperforms all automated feature extraction techniques in terms of absolute accuracies. Its FPRs are also the lowest, when considered with comparable accuracies of other high accuracy automated techniques. Thus in toto, BPE-based sequence embeddings perform better or equally well compared with known feature-based techniques.

Also notedly, the algorithms trained on our representations outperform when compared to training on vectors from [19] (righmost 2 columns in both tables), which are similarly BPE-based but use further text-based processing for feature-vector extraction. The improvement is especially noticeable over the ISCX IDS 2012 data set. This demonstrates that vectorizing BPE tokens using pre-trained sequence embeddings, provides superior representations compared to features obtained using statistical analysis and network trainings. Moreover, sequence embeddings encode semantic contexts in their vectors (which is not the case when using TF-IDF weightings) and which have shown to deliver better anomaly detection outcomes [9]. Thus using our representations makes executing statistical text extractions and network trainings unnecessary for performing web log anomaly detection.

6.2 Qualitative Analysis

Based on the results obtained, we can infer the following observations about using BPE-based vectors for web log anomaly detection:

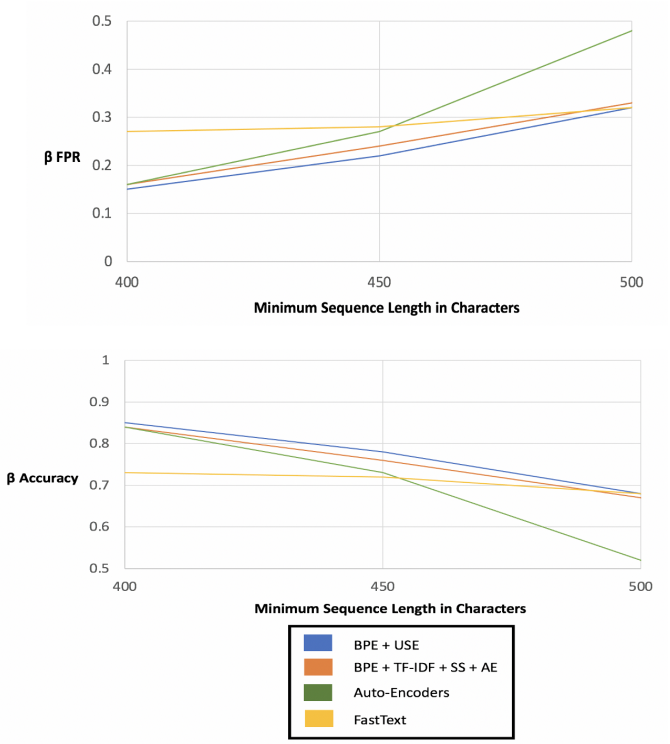

Figure 2: β FPRs and Accuracies vs URI Sequence Lengths

- **Observation**: BPE-based sequence embeddings provide better results compared to word-based embeddings. This is evident from the superior results obtained for our technique compared to the FastText approach in Tables 1 and 2. Sequence-based embeddings utilize the entire context of a sequence when encoding them into vectors, usually leading to better accuracies compared to word-based models, as similarly concluded in [24].

  Significance: Word-embedding techniques are more susceptible to word-level changes in log sequences, which can result in degraded detection outcomes. For example, GET requests for both URIs *http://localhost:8080/tienda1/publico/carrito.jsp* and *http://localhost:8080/tienda1/publico/miembros.jsp* are correctly marked benign by the our technique; however, the word-embedding technique marks the latter one as anomalous despite the URIs differing in only the final word. Thus BPE-sequence embeddings are able to consider the entire URI sequence when determining it as an expected request variant. But the word-based model is negatively affected by just a single word mutation, thereby degrading it's detection quality.

- **Observation**: There's negligible outcome variation when switching detection algorithms with BPE-based representations; but this is not the case with feature-based techniques. This can again be verified from Tables 1 and 2. Log

Table 1: Algorithm detection outcomes on HTTP CSIC 2010 data set

Training Algorithm	Manual Features		Bigrams + AE		FastText		BPE + USE		BPE + TF-IDF + Simple Separation + AE	
	Accuracy, FPR	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**
Isolation Forests	0.81 0.34	200 iTrees, 8192 samples per iTree	0.76 0.02	200 iTrees, 8192 samples per iTree	0.65 0.28	200 iTrees, 8192 samples per iTree	**0.93** **0.10**	**100 iTrees,** **256 samples** **per iTree**	0.91 0.12	100 iTrees, 256 samples per iTree
One Class SVM	0.93 0.12	kernel = rbf $\nu = 0.001$ $\gamma = 100$	0.81 0.08	kernel = rbf $\nu = 0.01$ $\gamma = 100$	0.76 0.42	kernel = rbf $\nu = 0.1$ $\gamma = 10000$	**0.94** **0.08**	**kernel = rbf** $\boldsymbol{\nu = 0.01}$ $\boldsymbol{\gamma = 10}$	0.88 0.14	kernel = rbf $\nu = 0.1$ $\gamma = 0.001$

Table 2: Algorithm detection outcomes on ISCX IDS 2012 data set

Training Algorithm	Manual Features		Bigrams + AE		FastText		BPE + USE		BPE + TF-IDF + Simple Separation + AE	
	Accuracy, FPR	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**	**Accuracy, FPR**	**Algorithm Parameters**
Isolation Forests	0.86 0.18	200 iTrees, 8192 samples per iTree	0.83 0.10	200 iTrees, 8192 samples per iTree	0.68 0.12	200 iTrees, 8192 samples per iTree	**0.93** **0.08**	**100 iTrees,** **256 samples** **per iTree**	0.82 0.10	100 iTrees, 256 samples per iTree
One Class SVM	0.91 0.10	kernel = rbf $\nu = 0.001$ $\gamma = 100$	0.88 0.07	kernel = rbf $\nu = 0.001$ $\gamma = 1$	0.62 0.08	kernel = rbf $\nu = 0.1$ $\gamma = 1000$	**0.92** **0.08**	**kernel = rbf** $\boldsymbol{\nu = 0.1}$ $\boldsymbol{\gamma = 0.01}$	0.81 0.11	kernel = rbf $\nu = 1$ $\gamma = 0.100$

sequences, in general, follow a fixed syntactic structure and BPE representations are known to learn sequence syntaxes better than other representations [27]. Thus BPE tokens provide for relatively un-varying representations, compared to feature-based techniques which don't account for sequential structures.

Significance: Representations using BPE tokens are robust and provide similar detection outputs across multiple downstream algorithms. This makes them highly useful in Auto-ML experiments when determining the best ML algorithm for any particular data set.

- **Observation**: Detection algorithms trained using BPE-based vectors perform better on shorter log sequences (i.e. with shorter HTTP URIs). This can be seen from Fig. 2, which depicts the detection outcomes for all automated techniques against increasing sequence lengths l in HTTP CSIC's development set β (NOTE: $400 < l < 520$ characters in β). β comprises of normal requests similar to the actual training data of HTTP CSIC. When all sequences are included in testing, FPRs for BPE-based models are the lowest, whereas their accuracies are the highest. As shorter length sequences are waxed out of the test set, FPRs for all techniques shoot up, while their accuracies tank down. The BPE-based models still provide the best detection outcomes at all sequence lengths. Their performance degradation at longer lengths is a known drawback, examined with multiple experiments in [25]. For the FastText-based technique, there's no discernible difference with increasing lengths; while for the one using ngrams and Auto-encoders, the performance degradation is steeper than the BPE-based technique.

  Significance: Recent research [29] has shown that the most visited internet URIs tend to have lengths < 100 characters. This is favorable for BPE-based systems, which perform comparatively better at lower URI lengths. Moreover, attack URIs tend to have much longer lengths compared to normal requests [30]. This is again advantageous for BPE-based systems, since even at higher lengths they perform better than all other feature-based techniques at detecting anomalous requests. In feature-based systems, these detections can only be achieved by actually considering URI lengths as individual features, which becomes a significant drawback of such systems.

6.3 Comparing Average Execution Times

This section compares the average execution times required by anomaly detection algorithms when trained using the different data representation techniques analyzed in this work. The execution times include the time needed for obtaining the features/embedding vectors, as well as the time required for training the algorithms on these representations.

From Table 3, we can see that the overall execution times on the ISCX IDS 2012 data set are far higher than those of the HTTP CSIC 2010 data set. This is firstly because there are about 240,000 more training instances in ISCX IDS 2012 compared to HTTP CSIC 2010. Secondly, the data in ISCX IDS 2012 also has comparatively

Table 3: Representation Extraction + Model Training Times

Technique	Time Taken for HTTP CSIC 2010 (in minutes)	Time Taken for ISCX IDS 2012 (in minutes)
Manual Features	1.12*	115*
Bigrams + AE	1.25*	327*
FastText-based features	5.8	575
BPE + USE embeddings	3.73	456
BPE + TF-IDF + SS + AE	3.83*	471*

more variation in components such as hostnames, ports and URLs compared to HTTP CSIC 2010. This causes the BPE tokenized vocabulary to be considerably larger as well.

Next, we can infer that extracting *BPE + USE* embeddings and training ML algorithms on them, takes at least twice the time compared to training with manual features (lowest) on both the data sets. The increased execution time is primarily needed for the training algorithms to run on BPE-based vectors, which have larger sizes (512 dimensions vs. 10 columns in manual features). But importantly, this is ALL the time required for obtaining detection results using BPE + USE embedding. This is not the case with other techniques (marked with *). For example, with manual features, considerable time is needed for obtaining the features in the first place, followed by feature selections which can altogether extend in multiple days [26]. Similarly for automated techniques, un-accounted time-frames include fine-tuning feature learning networks for obtaining suitable hyper-parameters. All these resources are not needed for either our representations, or the ones using FastText-based vectors. This is because both these techniques make use of efficient algorithms (BPE, FastText) for extracting text vectors. They require minimal-to-no external training. Thus extracting our representations neither contributes to significant execution delays, nor does it include unknown time-frames for data examination and preparation (which is required by contemporary techniques). Additionally, since no experts are needed and no feature learning networks are used, our technique becomes less cumbersome to re-use whenever new data becomes available for obtaining updated representations [1].

6.4 System Memory Trade-offs

While our proposed technique outperforms existing ones in terms of accuracies/FPRs, and completes in reasonable time compared to their end-to-end executions, it's also worthwhile discussing its memory consumption in relation to them. For evaluating the memory consumption of all the techniques, we make use of the *memory-profiler* module in python to monitor the entire process memory consumption.

Whereas BPE-based sequence embeddings don't require external network trainings or manual interference, they consume higher instantaneous system memory during execution. For example, the entire process of representation extraction and model training takes a maximum of 235 MB instantaneous memory when using manual features on HTTP CSIC 2010; whereas it takes a maximum of 5 GB instantaneous memory with our technique. This is expected since the representations of the entire training data need to be loaded into memory during training, and each data point extends to 512 dimensions with our technique. This amount is comparable to the maximum memory consumption of FastText vectors (2.9 GB) but is lower than the maximum demand of the BPE-based technique in [19] (7 GB). A similar observation is made on the ISCX IDS 2012 data set - BPE-based embeddings needed a maximum of 25GB to execute, while it was limited to 9 GB for manual features. FastText vectors in this case needed a maximum of 21 GB system memory

To reduce the maximum memory consumption, batch training is usually used with ML detection algorithms whenever applicable (the algorithms used in this work don't support batch learning). Instead when using them, alternative techniques can be utilized for improving in-memory space efficiency:

- PCA[44] and LDA [31] are popular techniques that can be used for reducing data dimensions while retaining enough discriminativeness for detecting useful anomalies.
- Application of specialized embedding dimensionality reduction (DR) techniques such as [32], can be explored for BPE-USE embeddings to meaningfully reduce their output dimensions, thereby reducing memory usage.

We thus observe that the memory consumption of our technique is on average higher than that of existing techniques; and thereby we plan to apply the above mentioned DR techniques for reducing memory consumption in future work.

7 CONCLUSION

In this work we proposed a new data representation technique for log anomaly detection using BPE tokens and pre-trained sequence embeddings. These representations provide better results compared to existing manual and automated feature-based techniques. Our technique's execution time is comparable with that of the others and importantly, does not require any feature preparation or feature learning and tuning processes. These usually take multiple hours or days to complete in contemporary techniques. Our representations are robust, contribute to better detection outcomes and are simple to obtain compared to existing mechanisms. Moreover, they can be applied to any category of logs (not just HTTP-based) since no assumptions are made characteristic just to HTTP logs. Thus we effectively abate the need of extracting features in web communication anomaly detection.

8 ACKNOWLEDGEMENT

This research was enriched with valuable feedback from Ali Torkamani and Baris Coskun. Thought-provoking discussions with Jack Ellis, Tyler Carlton and Adam Massachi were also frutiful in discussing applications of this research to our customers. Finally, we would like to thank Samuel Eddy for their talk on ML applications and also to IWSPA '22 reviewers for their valuable reviews.

REFERENCES

[1] S. He, J. Zhu, P. He and M. Lyu, "Experience Report: System Log Analysis for Anomaly Detection", 27th IEEE International Symposium on Software Reliability Engineering (ISSRE), 2016.
[2] C. Kruegel and G. Vigna, "Anomaly Detection of Webbased Attacks", ACM CCS, 2003.

[3] M. Kloft, U. Brefeld, P. Düssel, C. Gehl and P. Laskov, "Automatic Feature Selection for Anomaly Detection", AISec, 2008.
[4] D. Karev, C. McCubbin, and R. Vaulin, "Cyber Threat Hunting Through the Use of an Isolation Forest", 18th International Conference on Computer Systems and Technologies (CompSysTech), 2017.
[5] A. M. Vartouni, S. S. Kashi and M. Teshnehlab, "An Anomaly Detection Method to Detect Web Attacks Using Stacked Auto-Encoder", 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), 2018.
[6] Z. Chen; C. K. Yeo; B. Sung Lee and C. T. Lau, "Autoencoder-based network anomaly detection", Wireless Telecommunications Symposium (WTS), 2018.
[7] P. Bojanowski, E. Grave, A. Joulin and T. Mikolov, "Enriching Word Vectors with Subword Information", Transactions of the Association for Computational Linguistics, vol. 5, pp. 135–146, 2017.
[8] V. Tuulio, "Review of popular word embedding models for event log anomaly detection purposes", Helsingin yliopisto, 2020.
[9] Z. Cheng, B. Cui, T. Qi, W. Yang and J. Fu, "An Improved Feature Extraction Approach for Web Anomaly Detection Based on Semantic Structure", Hindawi Security and Communication Networks, Volume 2021, Article ID 6661124.
[10] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S.Yuan, C. Tar, Y. Sung, B. Strope and R. Kurzweil, "Universal Sentence Encoder", https://arxiv.org/abs/1803.11175.
[11] R. Sennrich, B. Haddow and A. Birch, "Neural Machine Translation of Rare Words with Subword Units", https://arxiv.org/abs/1508.07909.
[12] T. Gržinic, T. Kišasondi and J. Šaban, "Detecting anomalous Web server usage through mining access logs", Central European Conference on Information and Intelligent Systems , 2013.
[13] M.N. Sakib and C. Huang, "Using anomaly detection based techniques to detect HTTP-based botnet CC traffic", IEEE International Conference on Communications (ICC), 2016.
[14] M. Zhang, B. Xu, S. Bai, S. Lu and Z. Lin, "A Deep Learning Method to Detect Web Attacks Using a Specially Designed CNN", International Conference on Neural Information Processing (ICONIP), 2017.
[15] J. Li, H. Zhang and Z. Wei, "The Weighted Word2vec Paragraph Vectors for Anomaly Detection Over HTTP Traffic", IEEE Access, vol. 8, 2020.
[16] W. Meng, Y. Liu1, Y. Zhu, S. Zhang, D. Pei1, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun and R. Zhou, "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs", Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), 2019.
[17] R. Estepa, J. E. Díaz-Verdejo, A. Estepa and G. Madinabeitia, "How Much Training Data is Enough? A Case Study for HTTP Anomaly-Based Intrusion Detection", IEEE Access, vol. 8, 2020.
[18] W. Meng; Y. Liu, Y. Huang, S. Zhang, F. Zaiter, B. Chen and D. Pei, "A Semantic-aware Representation Framework for Online Log Analysis", 29th International Conference on Computer Communications and Networks (ICCCN), 2020.
[19] J. Zhan, X. Liao, Y. Bao, L. Gan, Z. Tan, M. Zhang, R. He and J. Lu, "19. An Effective Feature Representation of web log data by Leveraging Byte Pair Encoding and TF-IDF", Proceedings of the ACM Turing Celebration Conference, ACM TURC '19.
[20] C.T. Giménez, A.P. Villegas, and G.A. Marañón, "HTTP Dataset CSIC 2010", http://www.isi.csic.es/dataset/, 2012.
[21] https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassS VM.html
[22] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Isolat ionForest.html
[23] T. Nolle, A. Seeliger and M. Mühlhäuser, "Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders", International Conference on Discovery Science (DS), 2016.
[24] A. Joshi, S. Karimi, R. Sparks, C. Paris and C R. MacIntyre, "A Comparison of Word-based and Context-based Representations for Classification Problems in Health Informatics", Proceedings of the 18th BioNLP Workshop and Shared Task, 2019.
[25] M. Gallé, "Investigating the Effectiveness of BPE: The Power of Shorter Sequences", Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019.
[26] Q. Lin, H. Zhang, J. Lou, Y. Zhang and X. Chen, "Log Clustering based Problem Identification for Online Service Systems ", 38th IEEE/ACM IEEE International Conference on Software Engineering Companion, 2016.
[27] N. Durrani, F. Dalvi, H. Sajjad, Y. Belinkov and P. Nakov, "One Size Does Not Fit All: Comparing NMT Representations of Different Granularities", Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019.
[28] Brunswick, U. Intrusion detection evaluation dataset (ISCXIDS2012). Canadian Institute for Cybersecurity, 11 July 2010. Available online: https://www.unb.ca/cic/datasets/ids.html.
[29] https://backlinko.com/search-engine-ranking.
[30] https://docs.imperva.com/bundle/on-premises-knowledgebase-reference-guide/page/extremelylongurlparameter.html.
[31] D. M. Blei, A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation", Journal of Machine Learning Research 3, 2003.
[32] V. Raunak, V. Gupta and F. Metze, "Effective Dimensionality Reduction for Word Embeddings", Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019), August 2019.
[33] S. Ding, A. Renduchintala and K. Duh, "A Call for Prudent Choice of Subword Merge Operations in Neural Machine Translation", Proceedings of Machine Translation Summit XVII: Research Track, August 2019.
[34] Y. Wu and H. Zhao, "Finding Better Subword Segmentation for Neural Machine Translation", https://arxiv.org/abs/1807.09639.
[35] https://huggingface.co/docs/tokenizers/python/latest/.
[36] https://tfhub.dev/google/universal-sentence-encoder/1.
[37] https://tfhub.dev/google/universal-sentence-encoder-lite/2.
[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is All you Need", Advances in Neural Information Processing Systems 30 (NIPS 2017).
[39] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daume III, "Deep unordered compo- ´ sition rivals syntactic methods for text classification", Proceedings of ACL/IJCNLP, 2015.
[40] H. Chen, R. Xiao and S. Jin, "Unsupervised Anomaly Detection Based on System Logs", 33rd International Conference on Software Engineering and Knowledge Engineering, July 1– July 10, 2021.
[41] I. Provilkov, D. Emelianenko and E. Voita, "BPE-Dropout: Simple and Effective Subword Regularization", Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.
[42] B. Trevett, D. Reay and N. K. Taylor, "The Effectiveness of Pre-Trained Code Embeddings", 16th International Conference on Data Science (ICDATA'20), July 2020.
[43] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", https://arxiv.org/abs/1301.3781, 7th September, 2013.
[44] J. Shlens, "A Tutorial on Principal Component Analysis", https://arxiv.org/pdf/1404.1100.pdf, April 7th, 2014.

A APPENDIX

A.1 BPE Steps and Implementation

Algorithm 1 describes the steps used in generic BPE segmentations as laid down in [34]. The goodness score selected here is the frequency of substrings.

Algorithm 1 BPE Segmentation Algorithm

Input: The training data D, number of merge operations N and goodness score M

Output: tokenized data D" and vocabulary V

1: Initialize vocabulary V to the character vocabulary and represent each word in D as a sequence of characters, plus a special end-of-word symbol '·';
2: Obtain the goodness scores $M_i, M_j, ...M_n$ for all distinct consecutive substring pairs in D;
3: Search for the highest scored pair, then add it to V and merge all its occurrences in D as a single token;
4: Stop when the number of merge operations becomes $> N$.

The stopping criteria for the algorithm occurs when the required number of merge operations are reached. However, early stopping can automatically occur before the requisite number of merge operations are reached. This happens when the text is tokenized enough such that a further merge operation doesn't find any consecutive token pairs, which have frequencies >= 2

For implementing the BPE tokenizer we use the Hugging Face tokenizers [35] available in python. The number of merge operations which is the only hyperparameter in this algorithm, can be

customized, and in our experiments we set it at 30k, 40k and 50k operations separately.

A.2 USE Primer and Implementation

The Universal Sentence Encoder (USE) encodes textual strings into high dimensional vectors (embeddings) which can be used for language processing tasks such as text classification, semantic similarity, document clustering, neural machine translation and language generation amongst others. As opposed to popular embedding models such as as FastText and Word2Vec, which convert words into their best possible vectors; USE converts entire sequences as a whole into a single embedding vector of 512 dimensions.

Two flavors of the USE architecture were primarily developed for use in NLP applications - one based on the transformer architecture [38] and the other using Deep Averaging Networks (DANs) [39]. The transformer-based approach uses the encoding sub-graph of the transformer architecture which uses attention mechanism to obtain context-aware representations of words in a sentence that take into account both the ordering and identity of all the other words. These representations are converted to a fixed length vector by computing the element-wise sum of the representations at each word position. In the DAN-based model, input word embeddings and bi-grams are first averaged together and then passed through a feedforward deep neural network (DNN) to produce sentence embeddings.

The pre-trained USE models of both these architectures are publicly available on TensorFlow Hub ([36], [37]). In our work we use the *USE-lite* version of the available model [37], as opposed to the DAN model [37]. This is because as noted in [10] the transformer architecture delivers better overall results. Also the *lite* version of USE was trained to be more resource-efficient.