IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 16 May 2025, accepted 28 June 2025, date of publication 8 July 2025, date of current version 4 August 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3587029

RESEARCH ARTICLE

Convolutional Versus Large Language Models for Software Log Classification in Edge-Deployable Cellular Network Testing

ACHINTHA IHALAGE[1], SAYED TAHERI[1,2], (Member, IEEE), FARIS MUHAMMAD[1], AND HAMED AL-RAWESHIDY[2], (Senior Member, IEEE)

[1]VIAVI Solutions Inc., SG1 2AN Stevenage, U.K.
[2]Department of Electronics and Electrical Engineering, Brunel University of London, UB8 3PH London, U.K.

Corresponding author: Achintha Ihalage (achintha.ihalage@viavisolutions.com)

ABSTRACT Software logs generated by sophisticated network emulators in the telecommunications industry, such as VIAVI TM500, are extremely complex, often comprising tens of thousands of text lines with minimal resemblance to natural language. Only specialised expert engineers can decipher such logs and troubleshoot defects in test runs. While AI offers a promising solution for automating defect triage, potentially leading to massive revenue savings for companies, state-of-the-art large language models (LLMs) suffer from significant drawbacks in this specialised domain. These include a constrained context window, limited applicability to text beyond natural language, and high inference costs. To address these limitations, we propose a compact convolutional neural network (CNN) architecture that offers a context window spanning up to 200,000 characters and achieves over 96% accuracy (F1>0.9) in classifying multifaceted software logs into various layers in the telecommunications protocol stack. Specifically, the proposed model is capable of identifying defects in test runs and triaging them to the relevant department, formerly a manual engineering process that required expert knowledge. We evaluate several LLMs; LLaMA2-7B, Mixtral_8 × 7B, Flan-T5, BERT and BigBird, and experimentally demonstrate their shortcomings in our specialized application. Despite being lightweight, our CNN achieves strong performance compared to LLM-based approaches in telecommunications log classification while minimizing the cost of production. Our defect triaging AI model is deployable on edge devices without dedicated hardware and is applicable across software logs in various industries.

INDEX TERMS Cellular networks, LLM, log analysis, machine learning, NLP.

I. INTRODUCTION

As the telecommunications industry rapidly evolves into the 5G and 6G era, the scope and complexity of network testing have expanded at an unprecedented rate. Such testing solutions entail emulating user equipment (UE) and their intricate interactions with the network infrastructure. This enables network operators and equipment manufacturers to evaluate the quality and capacity of their wireless networks, ensure compliance with industry-standard protocols, and effectively troubleshoot issues while optimizing network performance. Meeting these needs involves employing advanced dedicated hardware (such as VIAVI TM500).

Invariably, these sophisticated systems generate voluminous and highly complex software logs that are indispensable for troubleshooting and defect triaging. The telecom raw logs analyzed in this context stand apart in their complexity and nature from conventional software logs, exhibiting a vast diversity of commands and parameters configured in real-time under dynamic industrial conditions. Moreover, their structure and semantics bear little correlation to natural language. Therefore, only expert engineers with extensive

The associate editor coordinating the review of this manuscript and approving it for publication was Rentao Gu.

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

experience in the field can navigate through these logs to identify issues in a network emulation. However, this manual analysis is inefficient, error-prone, less scalable, and vulnerable to knowledge silos. This inefficiency in manual analysis is further compounded by the high stakes involved; delays in defect resolution can have significant financial and reputational repercussions for both service providers and their clients.

Recently, the application of machine learning (ML) and natural language processing (NLP) techniques has demonstrated remarkable success in the classification and extraction of insights from text-based software logs [1], [2], [3], [4], [5], [7], [8], [18]. Additionally, a multitude of research studies have been dedicated to log analysis using ML techniques for anomaly detection, proposing ML solutions ranging from supervised classical models [6], [16] to convolutional neural networks (CNNs) [15], recurrent neural network (RNNs) and long short-term memory (LSTM) based networks or hybrid models [10], [19], large language models (LLMs) [12], [14], other transformer-based models [13] and unsupervised approaches [17]. Classical ML approaches commonly regard text input as static, relying on a global input text embedding, whereas CNNs, LSTMs, and LLMs handle text by processing it as a sequence of word or token embeddings.

Nevertheless, studies that employ ML for industrial telecommunications logs classification are few and far between. Closest related work [1] explores term frequency-inverse document frequency (TF-IDF) and bag of words (BOW) methods for representing software log lines in voice over internet protocol (VoIP) soft-switch systems and classifying them using various classical ML algorithms such as random forest (RF), support vector machine (SVM) and boosting methods. The objective of this study is to distinguish specific log entries into the error, debug, or info classes rather than diagnosing underlying root causes indicated by the logs. Furthermore, Ramachandran et al. [9] propose a CNN-LSTM architecture to categorize individual lines in industrial telecom-related log files as either errors or non-errors. However, it does not consider the entirety of the software logs for classification. In scenarios like ours, where it is essential to aggregate all errors, warnings, and indications within the logs to identify issues, such methods may not capture the full context necessary for accurate problem determination. In fact, none of the prior studies in the realm of log classification has demonstrated the capability to classify massive software logs containing tens of thousands of text lines while taking into account their complete content.

It is worthwhile noting the drawbacks of state-of-the-art (SOTA) LLMs in our specific application, despite their remarkable success in text classification [21], text generation [22] and many other NLP tasks [23], [24], [37]. LLMs are primarily pretrained and fine-tuned for natural language understanding, including code. Their adaptability to novel text structures, like our proprietary logs, remains constrained due to limited prior exposure to such formats. Therefore, it is essential to pretrain off-the-shelf models on our own extensive software log corpus before fine-tuning them for a log classification downstream task to achieve optimal performance. However, this requires enormous amount of data (e.g., 20 times the number of parameters in the model to achieve compute-optimal state according to recent research [38]) and computing power, hampering the adaptability of LLMs to our use case. Moreover, LLMs typically operate with a relatively narrow context window, ranging from 512 to 2048 tokens. This window is considerably smaller than the volume of text contained in our software logs, resulting in truncation and potential loss of critical information. Recent findings indicate that even the long-context LLMs may exhibit suboptimal performance, particularly when relevant information is situated in the middle of the context window rather than at the beginning or end [39]. Lastly, LLMs demand significant memory and computational resources even for inference, which may be undesirable in industrial settings due to higher recurring costs in model serving and deployment.

In this work, we propose a compact residual CNN architecture designed for the classification of software logs produced by VIAVI TM500. Its primary function is to identify the specific protocol stack layer in which a defect has occurred, thereby facilitating a more streamlined defect triage process. Our model can process text sequences up to 200,000 characters at a time and achieves over 96% accuracy (F1_score $>$ 0.9) in classifying complex software logs. To construct an initial embedding matrix, we trained a separate LSTM-based model with a sequence-to-sequence (seq2seq) objective on our sufficiently-sized text corpus. This process is analogous to *software log language* understanding and inspired by recent work that underscores the importance of industry-specific word embeddings in downstream log classification tasks [20]. Finally, we benchmark the performance of our CNN against latest open-source LLMs, namely Mixtral_8 $\times$ 7B [40], LLaMA2_7B [41], Flan-T5 [42], BigBird [43] and BERT [21].

The main contributions and novelty of this article are as follows.

1) We introduce a lightweight residual CNN architecture that can classify massive software logs accurately for defect detection in 5G/6G cellular network testing. The proposed model can support input text sequences up to 200,000 characters in telecommunications domain, with broad implications for software log classification across disciplines. Our model is only 3 Mega-bytes in size and has less than 0.8 Million parameters, making it edge-deployable and production-ready.
2) We experimentally demonstrate the drawbacks of off-the-shelf LLMs in specialized applications. To tailor these models for our specific domain, we perform domain-specific pretraining and subsequent finetuning on our data using techniques such as Low-Rank Adaptation (LoRA) [44] and quantization where applicable.

3) We propose an adaptable overlapping sliding window approach to extract pretrained-LLM embeddings of lengthy software logs that often exceed the context window of LLMs. Embeddings are extracted from several LLMs, including the latest models such as Mixtral_8 × 7B LLaMA2_7B that contain Billions of parameters. Separate classifiers are applied on top of these embeddings, showcasing acceptable classification performance.
4) We develop a sequence-to-sequence model based on LSTM units for understanding raw logs in telecommunications. Meaningful token embeddings extracted from this model are utilized to initialize the Embedding layer of the proposed CNN, helping to achieve optimal performance.

The remainder of this article is organized as follows. Section II presents an overview of the literature in the field of log classification and log anomaly detection. Section III elucidates the proposed ML architectures with theoretical foundations and provides a comprehensive overview of our data processing pipeline. In Section IV, we present the results and analysis of our model and conduct a benchmarking study with LLMs. Experimental details of model training and evaluation are described in Section V. Finally, Section VI concludes this article.

II. RELATED WORK

Analysing software logs has been studied under both supervised and unsupervised learning categories in the literature. Below, we discuss some notable work on log classification and log anomaly detection highlighting key methodologies and their outcomes, which partly inspired our choice of CNN-based architecture for our industrially-motivated telecommunications logs classification use case.

A. SUPERVISED METHODS

1) CLASSICAL ML

Chen et al. [25] presented a decision tree (DT) learning approach to diagnosing failures in large internet sites. They recorded the runtime properties of each request and trained a DT algorithm on the request trace to identify the causes of failures. Their method successfully identified 13 out of 14 true causes of failure in the logs produced by a large internet services system in a production environment, proving its effectiveness in real-world applications. Liang et al. [26] developed a methodology for predicting failures in IBM BlueGene/L supercomputers using event logs. They converted event logs into datasets suitable for classification techniques and applied several classifiers, including a rule-based classifier, support vector machines (SVMs), and a customized nearest neighbor method. Their customized nearest-neighbor approach outperformed others in coverage and precision, suggesting its viability in alleviating the impact of failures. One key benefit of some of the classical ML algorithms is their interpretability. For example, a DT may be indicative of which events or log messages are more likely to be related to a failure. Nevertheless, more advanced algorithms have since been proposed, particularly with the progress of deep learning.

a: RNN AND CNN

Recurrent neural networks, particularly those based on LSTM units, have been extensively applied to log classification due to their capability to handle sequential data and capture long-term dependencies within log sequences. In one of the pioneering studies, Du et al. [27] proposed DeepLog, a deep neural network utilizing LSTM units that can detect anomalies when the log patterns deviate from the normal execution. DeepLog's architecture includes three main components: a log key anomaly detection model, a parameter value anomaly detection model, and a workflow model for diagnosing anomalies. The models are trained with log entries from the normal system execution path. Each log entry is parsed to a log key and parameter value vector, and the models are tasked to predict the probability distribution of the next log key or parameter value vector in their respective sequences. Anomalies are detected at the inference stage by comparing the actual next log key or parameter value with the top predicted ones which model the normal behaviour. If the predictions significantly deviate from the ground truth, as found systematically using thresholds, then the entry is flagged as an anomaly. Zhang et al. [28] proposed LogRobust, an attention-based bidirectional-LSTM-based neural network for detecting log anomalies in both synthetic public datasets and a real industrial dataset. They model each log event as a semantic vector and use the attention mechanism to weigh different events, as well as a Bi-LSTM architecture to produce anomaly likelihood. The proposed architecture is benchmarked against classical ML models, including SVM and logistic regression, and is demonstrated to achieve consistently high performance. Several other deep learning architectures with LSTM as the key component have been proposed for log anomaly detection [10], [19], [29], [30].

On the other hand, CNN architectures can also be tailored for log classification with a much lower computational load compared to RNN architectures, especially for large context windows. CNNs may work well, especially when the existence of specific log events is important in detecting critical incidents rather than the long-range dependency between the events. A one-dimensional convolutional architecture has been proposed for detecting anomalies in big data system logs such as Hadoop Distributed File System (HDFS) logs [15]. Here, the proposed CNN achieved slightly higher performance in relation to a general multi-layer perceptron (MLP) model and an LSTM-based architecture. Ren et al. [8] propose a more feature engineering-heavy strategy followed by a two-dimensional CNN to classify CMRI-Hadoop and bluegene/L logs with critical events into

13 different categories. The proposed CNN showcases the best performance among other models, including against classical ML and LSTM-type models.

B. UNSUPERVISED METHODS

Unsupervised log anomaly detection methods can be quite useful, especially when extracting the labels for logs is too expensive or impractical. Unsupervised log analysis has been studied using a wide variety of algorithms, from classical models to state-of-the-art transformer-based approaches. Lin et al. [31] presented LogCluster, an agglomerative hierarchical clustering approach to detect anomalies in Hadoop-based application logs and large-scale online service platforms. Principal component analysis (PCA) has also been applied for log anomaly detection in the absence of a labelled dataset. PCA projects high-dimensional data ($\in \mathbb{R}^N$) into a lower dimensional coordinate system composed of k principal components ($k < N$) that capture the most variance in the original high-dimensional data. Xu et al. [32] represented log sequences as event count vectors and used PCA to obtain a lower dimensional representation of logs. They identified anomalies by thresholding the norm of the low-dimensional vectors.

With the remarkable success of the self-attention mechanism and the transformer architecture in language understanding, an increasing number of LLMs have been adapted for log analysis. Lee et al. [12] proposed an unsupervised log anomaly detection method based on the BERT architecture [21]. They first pretrain BERT on normal log data with a masked language modelling (MLM) objective. They then adopt the assumption that the context of a normal system log is notably different from that of an abnormal system log. Under this assumption, a normal log should exhibit a low error and high probability of prediction for masked tokens, whereas an abnormal log may produce a high error and a flatter probability distribution, which allows the detection of anomalies. Almodovar et al. [33] presented LogFiT, a finetuned version of the pretrained BERT model for log anomaly detection. They used the masked sentence prediction objective on the normal log data so the model learns the distribution of normal logs. Subsequently, when the model is presented with new log data, its top-k token prediction accuracy can be used with a threshold to identify the anomalous logs. Recently, Hou et al. [34] proposed HLogformer, a hierarchical transformer architecture designed for structured log data with nested, dictionary-like formats. Instead of treating logs as flat sequences, HLogformer builds hierarchical representations by segmenting logs into structured components and progressively summarizing them through a top-down and bottom-up summary-passing mechanism. This approach reduces memory overhead and improves the quality of log representation, particularly when logs are hierarchically structured. Furthermore, several other transformer-based models have been reported for log anomaly detection, including LogBERT [35] and CCT [36].

Although the literature is rich on general system logs classification and anomaly detection, the space of industrial telecommunications logs classification is largely unexplored. In the following sections, we discuss our approach in classifying logs generated by our network emulation device hardware and associated software stack into various layers in the telecommunications protocol stack.

III. METHODS

In this section, we discuss and formulate the proposed models, in particular our seq2seq model for language understanding and the residual CNN for classification.

A. SEQUENCE-TO-SEQUENCE EMBEDDING METHOD

The objective here is to train a ML model with a seq2seq objective for acquiring learned token embeddings from an industrial corpus. The process of transitioning from a textual representation to a numerical one typically encompasses multiple stages. Fig. 1 shows the steps we follow for preparing the input raw logs into primarily encoded text so that it can be used by our seq2seq model for training. Raw logs are voluminous, often riddled with noise, inconsistencies, and occasional ambiguity. Once a historical raw logs database is collected, we send it through a pre-processing unit (PPU). PPU captures logs related to several network testing categories, namely, single-UE, single-cell, multi-UE, multi-cell, New Radio (NR) 5G tests, Long-Term Evolution (LTE) 4G tests, and L3 tests. Additionally, the PPU removes redundant information unrelated to detecting defects, such as very long words and lines, and numbers. Next, we identify outlier logs based on their size using the Tukey's method. A box plot (see Appendix Fig. 9) is created, showing the distribution of the number of characters in logs. Outliers are then defined as observations that fall below $Q1 - 1.5 * IQR$ or above $Q3 + 1.5 * IQR$, where Q1 and Q3 are first and third quartiles, respectively, and IQR is the interquartile range ($Q3 - Q1$). These outliers, along with files $>$ 300 kilobytes, have been removed to establish our final dataset. Such filtering is necessary as our raw logs can occasionally go up to hundreds of Megabytes in size. Fig. 2 depicts the histograms of our dataset before and after preprocessing.

The resulting software logs are then concatenated to create a training corpus (TC). Given the little correlation between our logs and natural language, we opted to use unique characters present in our corpus as tokens. This approach yielded in a vocabulary consisting of 97 unique characters, allowing us to encode any piece of text within the TC and obtain a corresponding numerical representation. The decision to use character-level tokenization over word or subword tokenization is inspired by the unique nature of telecom software logs, which differ significantly from natural language text. These logs often contain a mix of proprietary code, hexadecimal values, technical symbols, vendor-specific abbreviations and non-standard patterns. Consequently, word or subword tokenization would result in unnecessarily large vocabularies, possible out-of-vocabulary

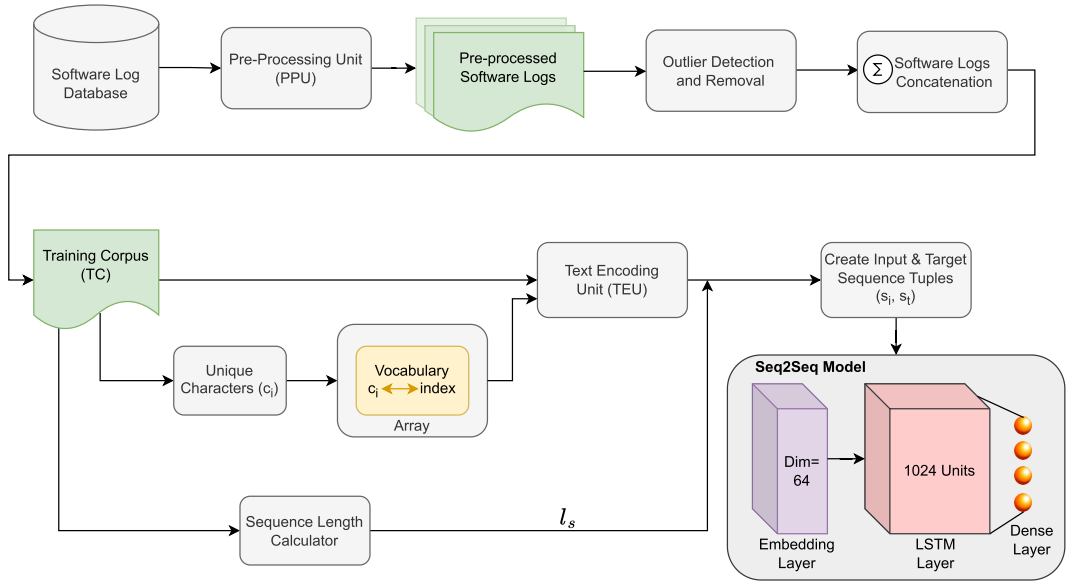


FIGURE 1. Workflow demonstrating data preparation and architecture of CCOS model.

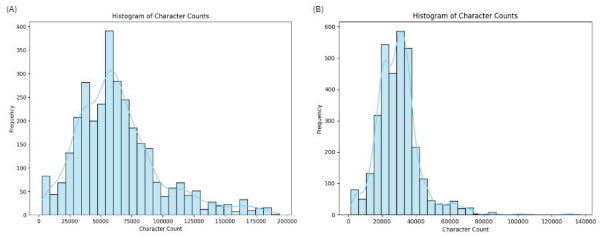


FIGURE 2. Character length histograms of log files before (A) and after (B) cleaning.

tokens, and would extensively expand the embedding layer. To mitigate these issues and ensure broader coverage, we adopt a character-level tokenization approach. Character-level tokenization shows better resilience to noisy, non-standard text, but it may expand the sequence length slightly.

In order to create training sequences, we choose a sequence length (l_s) equal to the median length of *message blocks* (in characters) in our software command logs. A *block* refers to the set of information contained within so-called Indications (`I:`) that record a set of events that have come back from the network, and Confirmations (`C:`) that record the state of execution of system commands. These are the information that is critical for the AI system to be able to learn and capture defects, justifying the rationale behind the selected l_s. Next, we create tuples of input and target sequences (s_i, s_t) with matching lengths (l_s), where s_t is formed by shifting a window of l_w characters across s_i within the continuous TC. To maintain simplicity, we adopt the assumption that $l_w = 1$.

Having equal and fixed-length input and target sequences allows us to design a simpler recurrent neural network architecture (RNN) without an explicit decoder for seq2seq learning. The proposed architecture as illustrated in Fig. 1 has an embedding layer to represent every character in the vocabulary with an embedding of 64 dimensions, chosen heuristically. This is followed by an LSTM layer with 1024 units that returns processed sequences, and an output fully-connected dense layer with the number of units equal to the vocabulary size. The dense layer is applied across all returned sequences by the LSTM layer. As such, our optimization objective is to minimize the negative log-likelihood of the true next sequence.

$$\text{argmin}_\theta \frac{1}{N} \sum_{j=1}^{N} -\log P(s_j | s_{j-1}; \theta) \tag{1}$$

where N is the total number of input and target sequence tuples in the dataset. $P(s_j|s_{j-1};\theta)$ is the conditional probability of generating the true next sequence s_j given the previous sequence s_{j-1} and the model parameters θ. With this notation, the target sequence s_t equals to s_j, and the input sequence s_i equals to s_{j-1}. Equation 1 can be further decomposed as follows, considering individual characters.

$$\text{argmin}_\theta \frac{1}{N} \sum_{j=1}^{N} \sum_{t=1}^{l_s} -\log P(c_t^{s_j} | c_1^{s_{j-1}}, \ldots, c_{l_s}^{s_{j-1}}; \theta) \tag{2}$$

where $c_t^{s_j}$ is the character at the t^{th} index of sequence s_j. Probability logits over the vocabulary for every character in the target sequence are calculated through a softmax function applied at the output layer. Consequently, the model is trained to minimize the multi-class categorical cross-entropy loss. Once the model is trained, character embeddings are extracted from the weights stored within the Embedding layer of the model. It is worth noting that the trained seq2seq model, primarily utilized for extracting character embeddings, has

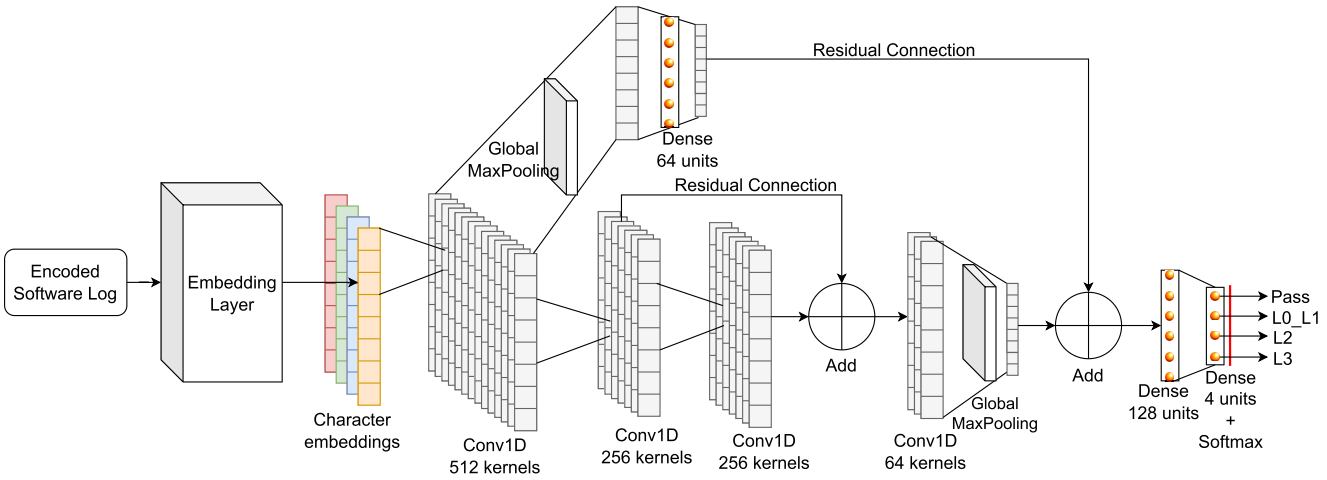


FIGURE 3. Proposed residual CNN architecture for software log classification.

the potential for software log text generation. However, this aspect falls outside the scope of this paper.

B. RESIDUAL CNN FOR SOFTWARE LOG CLASSIFICATION

The proposed architecture for lengthy software log classification is primarily based on one-dimensional convolutional layers (Conv1D). CNNs are generally lightweight and consume less computational resources. This architectural choice stems partly from practical considerations in industrial production, as many target edge devices lack dedicated hardware like GPUs. Nevertheless, as we demonstrate later in this article, the proposed CNN, in fact, consistently outperforms other benchmarked models. Formally, Conv1D operation applied on a discrete 1D input sequence s at a time index t with single filter w is given by;

$$y(t) = (w * s)(t) = \sum_{k=-\frac{(K-1)}{2}}^{\frac{(K-1)}{2}} w(k) \cdot s(t-k) \qquad (3)$$

where $*$ indicates the convolution operation, $y(t)$ is the feature map resulting from the filter applied at position t, and K is the kernel size.

Concretely, our CNN consists of an embedding layer initialized with character embeddings extracted from the seq2seq model, followed by a block of Conv1D layers before applying global max pooling operation on the processed sequences, resulting in a fixed-size representation of the input sequence. This is then processed through a set of fully-connected layers followed by softmax activation for multi-class classification. We further apply residual connections to improve model performance. The end-to-end (E2E) architecture is depicted in Fig. 3.

The model is trained to classify various software logs into four distinct classes: *Pass*, *L0_L1*, *L2*, and *L3*. *Pass* class represents software logs that do not indicate any issues. *L0_L1* denotes defects at the physical layer, *L2* pertains to issues within the data link layer, and *L3* encompasses problems related to the network or higher layers, in accordance with the Open Systems Interconnection (OSI) model. These labels for the software logs in our dataset are extracted from historical data. Undoubtedly, most test runs are completed without issues, resulting in a highly imbalanced class distribution within our dataset, as can be seen in Fig. 4(A). To reduce the impact of class imbalance, ML models studied in this work are trained with appropriate class weights where applicable. There are 3262 samples in total in our dataset which is randomly divided into 70% training and 30% test sets. The class distribution for the test set is shown in Fig. 4(B).

IV. RESULTS AND DISCUSSION

In this section, we present the classification results of the proposed CNN on the test set and benchmark it against several other approaches, including LLMs. Note that the test set is kept intact across all experiments.

A. CNN RESULTS

Fig. 5 illustrates the training progress of our CNN, namely the categorical cross-entropy loss, accuracy, and F1-score curves. Our defect triaging model achieves a notable 96% accuracy, markedly improving both precision and efficiency in manual engineering workflows. A more comprehensive summary of results is shown in Table 1. Although our model can support text sequences up to 200,000 characters, we used a sequence length of 50,000 characters to obtain results shown in Table 1. This is because nearly 95% of our cleaned software logs are less than 50,000 characters in length. The trained model, with its compact size of just 3 megabytes, is ideally suited for deployment on edge devices with basic hardware resources.

On the other hand, LSTM networks are recognised for their effectiveness in sequence classification tasks, owing to

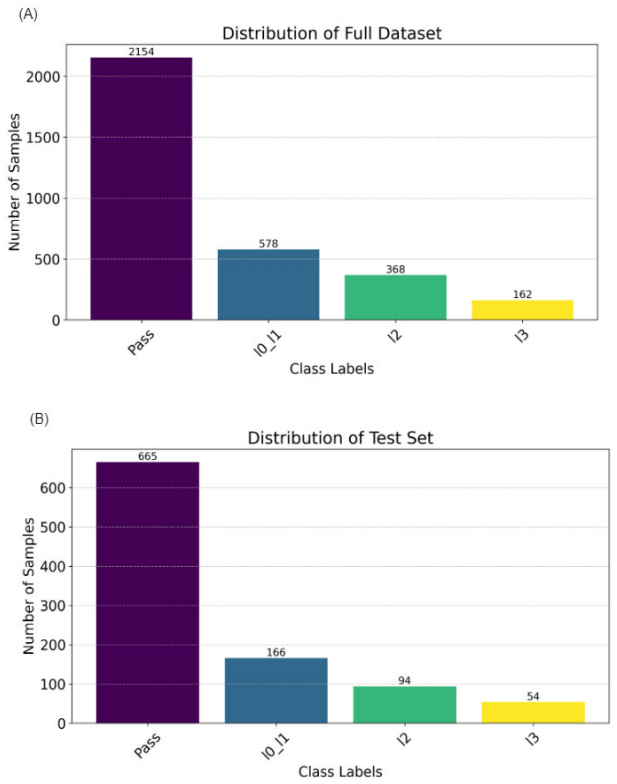


FIGURE 4. Distribution of classes in our full dataset (A) and test set (B).

their ability to learn long-term dependencies with memory cells and gating mechanism. It is interesting to include LSTM components in our model and assess the impact on performance. In fact, such architectures have been proposed for sequence learning tasks in telecommunications including channel estimation [47]. We added a bidirectional LSTM layer right after the embedding layer of the proposed CNN (denoted as BiLSTM+CNN model) and evaluated the performance on the test set under the same conditions. This resulted in a slightly downgraded accuracy of 94.2%, despite a nearly three-fold increase in model size (see Table 1). This potentially suggests that in the context of our software logs, it is not so much the long-range dependencies or the semantic structure that are crucial for identifying defects but rather the presence of specific combinations of log messages. The confusion matrices of the CNN and CNN+BiLSTM models are displayed in Fig. 7. Both models can accurately detect even the least represented class ($L3$) that has less than one-tenth of samples compared to the dominant class.

We subsequently explored how the input sequence length impacts the performance of our CNN. Illustrated in Fig. 6, we observed a steady improvement in performance as the sequence length increases, up until a point where it begins to marginally decline for extremely lengthy sequences (>80k characters). This trend may occur because such expansive context windows necessitate excessive padding in most input text sequences with redundant tokens, thereby diluting the pertinent information. Furthermore, we noticed that the proposed CNN model is generally robust in terms of the number of Conv1D layers present in the architecture. Employing just a single Conv1D layer while maintaining the rest of the architecture (embedding & dense layers) intact yielded an accuracy of 93.6%. Further increasing the number of Conv1D layers, up to four, repeatedly led to a classification accuracy of above 95% with consistent distribution of other performance metrics.

B. LLMS FOR SOFTWARE LOG CLASSIFICATION

While LLMs have been generally successful in learning to categorize software logs [12], [13], raw logs generated by software and hardware stacks prevalent in the telecommunications industry pose unique challenges to LLMs, particularly due to their vast size and little relevance to the natural language. It may be possible to intuitively reason that the small context windows of off-the-shelf LLMs cannot possibly capture all necessary information from large logs, which may lead to poor downstream performance. Similarly, conducting further pretraining (also known as domain adaptation) on a domain-specific corpus could be beneficial in enhancing downstream task accuracy. To ensure a broad and representative evaluation for our specific application, we selected five fundamentally different pretrained LLMs, namely, LLaMA2-7B, Mixtral_8 × 7B, Flan-T5, BERT, and BigBird. This covers encoder-only (BERT), decoder-only (LLaMA2-7B), encoder-decoder (Flan-T5), mixture of experts (Mixtral_8 × 7B), and long-sequence transformers (BigBird).

LLaMA2-7B [41] is a transformer-based decoder model, exhibiting strong text-generation performance. Mixtral_8 × 7B [40] is a transformer-based sparse mixture of experts (MoE) language model. Each transformer block is composed of 8 feedforward layers (i.e., experts), of which only 2 are activated per forward pass, as selected by a router network. This design allows the model to scale to a larger parameter count while keeping computation per inference step relatively low. Flan-T5 [42] is an encoder-decoder model fine-tuned for instruction-following tasks across multiple benchmarks. BERT [21] is a bidirectional encoder trained with masked language modeling, well-suited for understanding contextual semantics. BigBird [43] extends transformer capabilities to longer sequences using sparse, sliding window attention patterns, making it efficient for processing large input logs. All LLM architectures considered in this study are opensource and their weights are available publicly.

While every model undergoes evaluation in its original pretrained state, both Flan-T5 and BERT are additionally chosen for domain adaptation, owing to their distinctive natures and more manageable sizes. Accordingly, BERT is domain-adapted with a masked language modelling objective (i.e. a percentage of tokens are masked randomly, and the model is tasked to predict these tokens accurately). Flan-T5 is domain-adapted by training the model to predict a shifted input sequence of tokens, emulating a sequence-to-sequence learning objective. Next, we discuss the exact steps that we

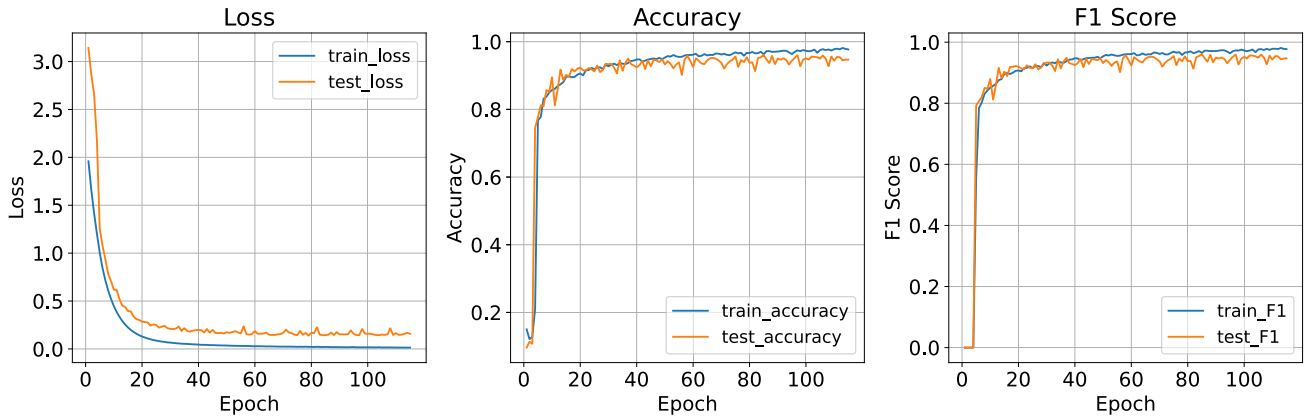


FIGURE 5. Training progress curves for the proposed residual CNN model. Note that the F1-score shown here is the micro average computed by obtaining the overall true positives, false positives, and false negatives across all classes.

TABLE 1. Multi-class performance metrics of the residual CNN and BiLSTM+CNN models on the test set.

Model	# Parameters	Overall Accuracy (%)	Overall F1-score (macro)	Per-Class Metrics			
				Class	F1-score	Precision	Recall
CNN	0.79M	**96.01**	**0.902**	Pass	0.999	1	0.998
				L0_L1	0.912	0.915	0.91
				L2	0.818	0.779	0.862
				L3	0.88	0.957	0.815
BiLSTM+CNN	2.1M	94.2	0.856	Pass	0.999	1	0.998
				L0_L1	0.861	0.848	0.873
				L2	0.756	0.737	0.777
				L3	0.808	0.889	0.741

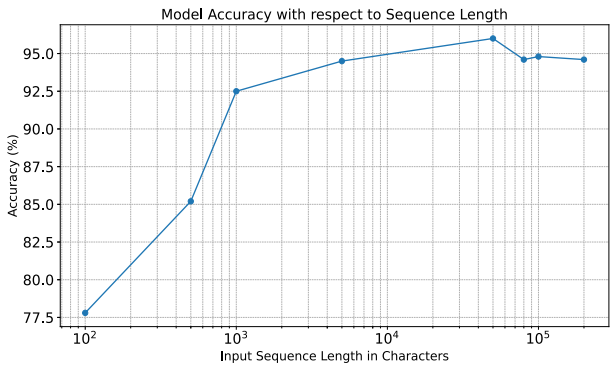


FIGURE 6. Performance of residual CNN with varying context size. Note that the x-axis is in the log scale.

follow for pretraining LLMs on our own industrial software log corpus.

Domain adaptation or further **pretraining** of LLMs may significantly elevate the downstream task performance of LLMs, especially in domains where the morphology of text is considerably different from the natural language on which most models are trained on. First, we identify some specific pieces of text, such as hexadecimal strings, standalone numbers, IP addresses, and new-line characters, that are generally unrelated to identifying defects. These are then masked with special tokens such as `<hex>`, `<num>`, `<ipaddr>` and `<newline>`. Furthermore, the end of a software log sample in our concatenated training corpus is indicated with an `<endsample>` token. Tokenization is a critical step in preparing data for LLM training that involves breaking down raw text into numerical representations. The models investigated in this study use subword tokenization, which balances the need to represent common words as they appear and decompose less common ones into smaller, understandable sub-units, resulting in efficient handling of out-of-vocabulary words. Here, we use the pre-trained WordPiece tokenizer for BERT and the SentencePiece tokenizer for Flan-T5. The special tokens identified above are added to both tokenizers before tokenizing the TC.

We then train BERT for domain adaptation with a warm start from its pre-trained base model checkpoint. The relatively small size of BERT allows for updating all model weights during domain adaptation, conducted with a masked language modeling (MLM) objective on the same software log TC constructed in the workflow shown in Fig. 1, further processed to add special tokens discussed above. Note that one of the pre-training strategies of BERT, next sentence prediction (NSP), is not used here as performing domain adaptation using MLM is deemed sufficient [43]. In MLM, a portion of tokens are randomly masked (∼30% in our case), and the model is tasked to predict these masked tokens based on their context, enabling it to gain a deep understanding

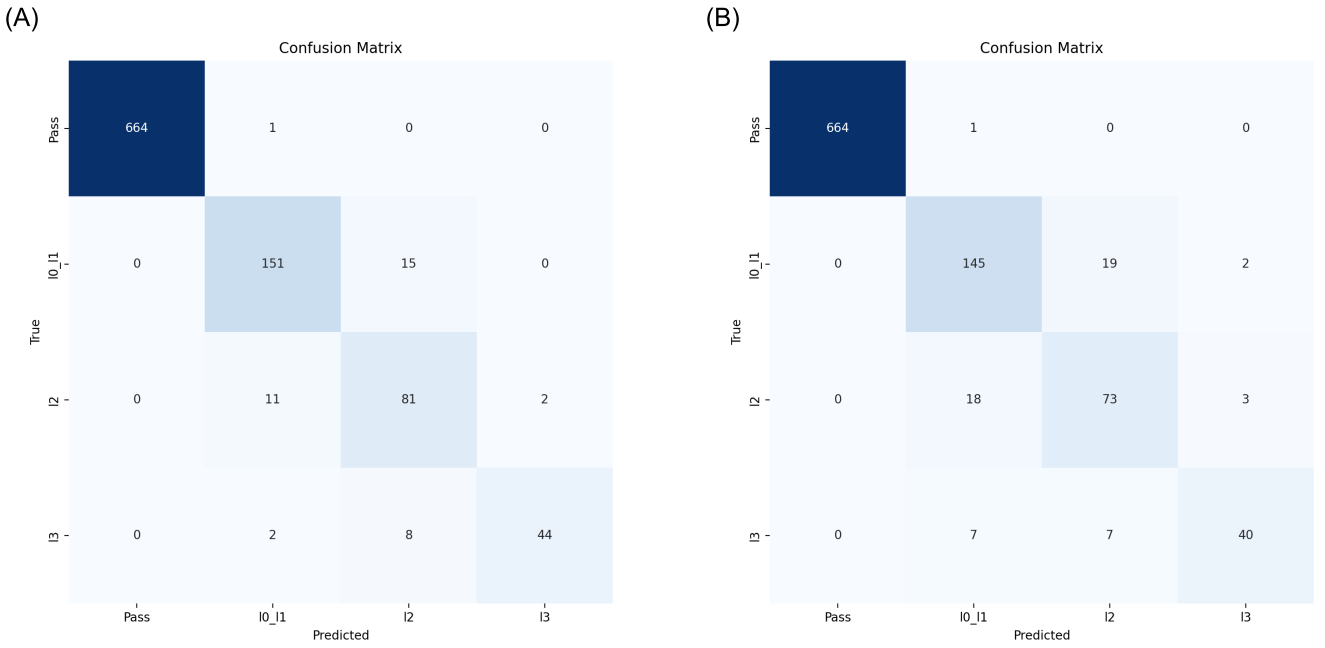


FIGURE 7. Confusion matrices of CNN (A) and CNN+BiLSTM (B) models for the test set.

of the syntax, semantics, and morphology of a given text corpus.

Flan-T5 is trained starting from `flan-t5-large` checkpoint using a sequence-to-sequence objective where the target sequences are created by shifting the input sequences by k ($\geq$ 1) number of tokens. TC is then tokenized with Flan-T5's own pre-trained tokenizer, extended to include our special tokens. Due to the large size of Flan-T5 architecture (∼780M parameters), we opted to use LoRA method for domain adaptation. LoRA involves introducing trainable low-rank matrices to modify existing weights in selected layers of the model, such as attention and feed-forward layers, without altering the original pretrained weights. This allows for effective adaptation of the model for specific tasks by reducing the computational burden while maintaining model performance. Both BERT and Flan-T5 are trained for 5 epochs, and the trained checkpoints along with their associated tokenizers are recorded for extracting software log embeddings.

LLM embedding of a given text input can be treated as a learned numerical representation that encodes the contextual meaning of that text. As discussed, the small context window of LLMs necessitates an alternative strategy for them to sift through the entirety of long documents. We propose an overlapping sliding window approach, as depicted in Fig. 8, to extract a single embedding from lengthy text segments. Formally, consider a long text document of length L tokens and a context window (i.e., sequence length) of l_c ($< L$) tokens. For an adaptable overlapping window of size w ($< l_c$) tokens, the number of overlapping text chunks that need to pass through the model to cover the entire document, M, can be expressed as follows.

$$M = \frac{L - w}{l_c - w} \tag{4}$$

Under the proposed approach, the document global embedding E_g can be computed as follows.

$$E_g = \frac{1}{M}\frac{1}{l_c}\sum_{k=1}^{M}\sum_{i=1}^{l_c} TE_{k,i} \tag{5}$$

Here, $TE_{k,i}$ is the token embedding of the i^{th} token of the k^{th} chunk. Token embeddings are extracted at the last hidden layer of the model. $TE_{k,i}$ is of dimension d_{TE} equal to the number of units in the last hidden state of the model.

Concretely, we divide each long log into smaller, overlapping chunks. Each chunk is passed through the LLM to generate token-level embeddings. For each chunk, we average its token embeddings (ignoring padding) to get a single chunk-level representation. Then, we average the embeddings from all overlapping chunks to produce one final embedding for the entire log. The overlap between chunks ensures that contextual information at the boundaries is preserved, making the overall representation more coherent and robust.

Essentially, the mean-pooling operation (element-wise mean of node embeddings) is applied across token embeddings of a chunk to obtain the chunk embedding, and the same operation is repeated across all chunks to obtain the document embedding. The attention mask is considered

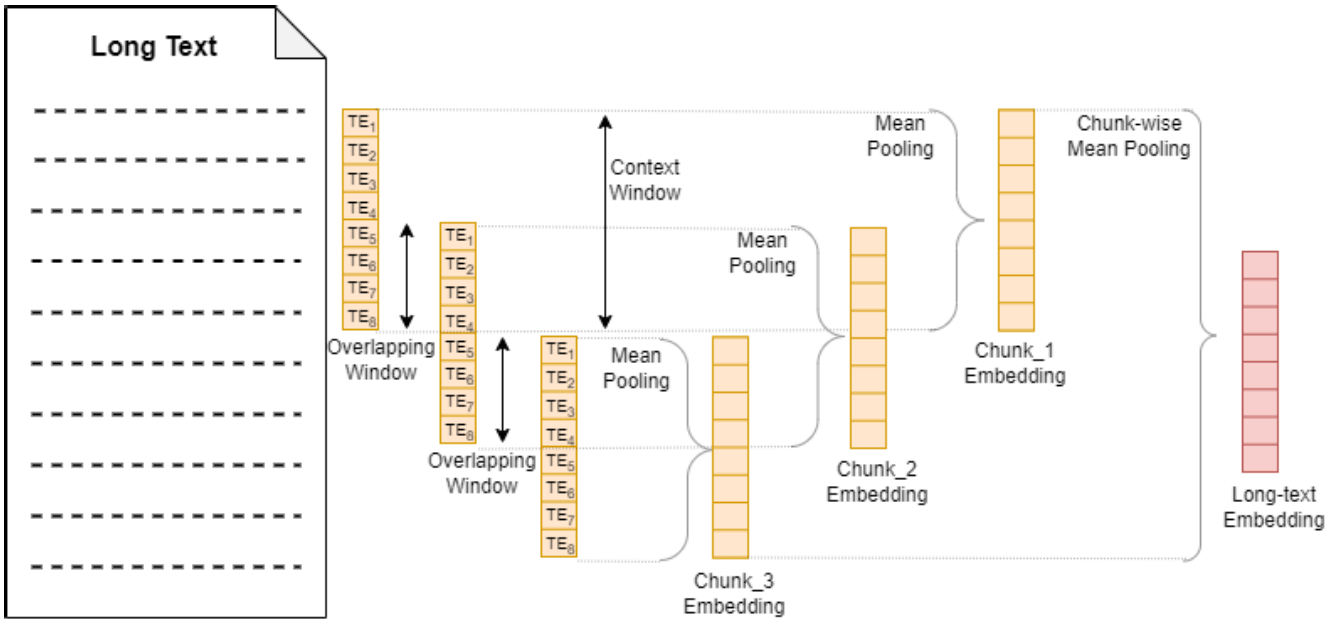


FIGURE 8. Overlapping sliding window approach to extract embeddings of long documents using LLMs. TE_i refers to the token embedding of i^{th} token at the final hidden layer of the LLM. Token embedding dimension typically ranges from 512 to 4096 for larger models.

when applying the pooling operation (i.e., pad tokens are disregarded). Text chunks are selected with an overlap w so that some information and context from the previous windows are retained as the model slides through the text. Likewise, embeddings of all logs in our dataset are extracted independently using every LLM examined. Note that due to memory constraints, the context window used for embedding extraction in this work may be smaller than the maximum context window supported by some LLMs. In order to assess the quality of these embeddings, we applied separate classifiers, namely, random forest, XGBoost, and decision tree, with the same defect detection objective. This facilitates benchmarking the proposed LLM-embedding method against our residual CNN. In addition to classical ML models, we utilize a similar residual 1D CNN to classify LLM-embeddings and report the performance on the test set. As a classical baseline, we implemented a TFIDF (Term Frequency-Inverse Document Frequency) embedding pipeline followed by a logistic regression classifier. TFIDF transforms raw log text into a sparse numerical representation based on the frequency of terms, adjusted by how rare those terms are across the dataset. TFIDF embeddings capture the relative importance of words within individual logs. Logistic regression is extended to handle multiclass classification using the multinomial formulation.

Table 2 summarizes the experimental results. LLaMA2-7B embeddings used with the XGBoost classifier provide the best accuracy of 82.2% across all experiments. Although this is reasonable, it represents a notable decline in performance when contrasted with the results achieved by our standalone residual CNN, indicating that even some of the most powerful general-purpose LLMs may perform sub-optimally in challenging domain-specific tasks such as large and complex software log classification. Nevertheless, this series of experiments yields valuable insights. As expected, large and more potent LLMs, specifically, LLaMA2-7B and Mixtral_8×7B, produce higher quality embeddings, resulting in higher downstream performance compared to other models. The baseline approach (TFIDF+Logistic Regression) achieves a moderate accuracy of 71.6%, notably below the performance achieved by both the XGBoost classifier and the proposed lightweight CNN.

Moreover, domain adaptation enhances the performance to some extent, as the models may have gained some understanding of the specific text format by being exposed to a targeted corpus. This enhancement, however, is more prominent for BERT compared to Flan-T5 likely due to distinct domain adaptation strategies. Flan-T5 is pretrained on our corpus by applying LoRA exclusively to the query and value matrices in the transformer block, and therefore the initial token embeddings or model weights that have been optimized for natural language understanding do not get updated. These tokens derived from a corpus of natural language, and their embeddings, may not be very relevant in our specific context. On the contrary, BERT is end-to-end trained during domain adaptation, allowing us to learn token embeddings and model parameters tailored to our corpus, leading to a more substantial boost in performance. Moreover, the downstream task performance is majorly reliant on the quality of embeddings, as more advanced architectures such as 1D-CNNs do not lead to higher performance relative to some classical ML models.

TABLE 2. Classification performance of ML models on the test set using software log embeddings extracted by LLMs. RF and DT refer to random forest and decision tree classifiers, respectively. NA- not applicable.

LLM Embedding Model	Approximate # Parameters	Used Context Window	Embedding Dimension	Overall Accuracy (%)				Overall F1-score (macro)			
				RF	DT	XGBoost	CNN	RF	DT	XGBoost	CNN
LLaMA2-7B	7B	2048	4096	80.8	71	**82.2**	77.1	0.683	0.59	**0.705**	0.632
Mixtral_8x7B	47B	512	4096	79.2	67.2	80.2	77.4	0.626	0.517	0.658	0.617
Flan-T5	770M	512	1024	75.7	63.9	79.5	NA	0.531	0.466	0.649	NA
Flan-T5 (Domain Adapted)	770M	512	1024	77	63.3	79.2	NA	0.555	0.4291	0.634	NA
BERT	110M	512	768	73.9	62.3	78.2	74.6	0.45	0.438	0.596	0.527
BERT (Domain Adapted)	110M	512	768	76.9	69.1	79.4	74.6	0.568	0.507	0.641	0.568
BigBird (E2E Finetuned)	110M	4096	NA	81				0.581			
TFIDF+Logistic Regression	NA	NA	10000	71.6				0.358			

Finally, we finetune the original pretrained BigBird base model on our labelled dataset with the hope that its relatively large context window of 4096 tokens can capture as much information as possible from software command logs. Instead of extracting embeddings, here, the model is end-to-end finetuned with a classification head, and all model parameters are updated during training. This resulted in accuracy and F1-score of 81% and 0.581 on the test set, respectively, consistent with LLM-embedding classification results presented in Table 2. An analysis of similar-size models reveals that, when the model context window is sufficiently large, finetuning LLMs end-to-end on a downstream classification task leads to higher performance compared to employing separate classifiers on pretrained LLM embeddings. Nevertheless, none of the LLM-based approaches achieves comparable performance to the proposed CNN, indicating that domain-tailored ML architectures may be better suited for industrial applications that require processing miscellaneous formats of text. In addition to delivering superior performance, such lightweight ML models minimize the cost of production and are practically feasible for in-house or field deployment.

It should be noted that this study focuses on telecommunications software logs, which exhibit domain-specific structures and non-natural language patterns. While the proposed CNN is tailored to this setting, the underlying approach including character-level modelling, long-context handling and the lightweight CNN can be adapted to other technical log formats with similar characteristics. It should be noted that while residual CNNs outperform LLMs in this specific telecommunications log classification task, their primary limitation lies in their inability to capture long-range dependencies or semantic context, which may be crucial in more linguistically structured or semantically rich domains. Likewise, LLMs may be more effective in domains where log data more closely resembles natural language or where pretrained linguistic knowledge is advantageous, particularly when compute and inference resources are not a constraint.

V. EXPERIMENTAL DETAILS

All training and evaluation experiments in this study were performed on KubeFlow, which enables the orchestration of machine learning workflows on the Kubernetes cluster. Ubuntu 20.04 was used as the operating system. The hardware infrastructure consisted of an Nvidia DGX server carrying 8xA100 graphics processing units (GPUs), each with 80GB memory. The system is equipped with 1 terabyte of physical memory. MLFlow [45] integrated within the KubeFlow environment was used for experiment tracking and logging artifacts.

CNN architecture and hyperparameters were optimized empirically. The optimized architecture contained four Convolutional-1D layers with number of filters ranging from 64 to 512 (see FIGURE 3). All convolutional layers use a kernel size of 3 and are followed by Rectified Linear Unit (ReLU) activation [46]. The model was trained for up to 200 epochs with an early stopping patience of 30 epochs. Adam optimizer with a learning rate of 10^{-4} was used to optimize model parameters. L_2 regularization was applied to selected layers to reduce overfitting and improve generalization of the model. The batch size was adjusted accordingly within a range of 16 to 512 to accommodate large context sizes. For the largest context size tested (200k), the model completed training in under one hour. A similar hyperparameter setting was employed for the seq2seq LSTM model that had 4.6 Million parameters. We used *Keras* package with *TensorFlow* backend to implement and train the models.

LLMs analyzed in this study were adapted from Hugging Face [48] using the *transformers* package. BERT and Flan-T5 models were each trained for 5 epochs for domain adaptation. LoRA adapters, with a rank of 16 and a scaling factor of 32 were used to train Flan-T5. BERT was trained in less than one hour on our TC whereas the training of Flan-T5 took about seven hours. The batch size is set to 64 for BERT and 12 for Flan-T5. Larger models (LLaMA2-7B and Mixtral_8 $\times$ 7B) were 4-bit quantized using the *bitsandbytes* package before extracting embeddings for computational efficiency. LLaMA2-7B, Mixtral_8 $\times$ 7B, Flan-T5, and BigBird use SentencePiece-based subword tokenizers (with BPE or Unigram models), while BERT uses the WordPiece tokenizer. Flash attention 2 [49] implementation in the *transformers* package was used to further reduce memory requirements. The overlapping window size was set to half the sequence length for embedding extraction. *torchrun* was used for distributed training of LLMs where applicable.

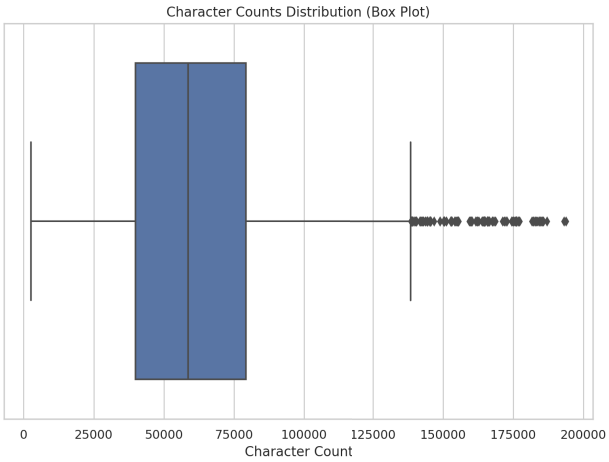


FIGURE 9. Boxplot indicating outlier software logs.

BigBird base model was finetuned for up to 200 epochs with a batch size of 8 and an early stopping patience of 30 epochs. In general, the finetuning of BigBird took about 4 hours. The hyperparameters of the ML models used with LLM-embeddings were optimized using a 5-fold cross-validation strategy with grid search. Classical ML implementations were adopted from Scikit-Learn [50] with default parameters. Classification performance metrics are calculated as follows where TP, FP and FN indicate true positives, false positives and false negatives, respectively.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

$$\text{F1\_score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

VI. CONCLUSION

In this research, we presented a robust CNN architecture tailored for the classification of intricate large-scale software logs in the telecom sector to meet the edge-deployability requirement of defect detection systems. The proposed model, adept at processing extensive text sequences up to 200,000 characters, markedly outperforms some LLM-based approaches examined. This advancement is critical in mitigating the limitations of manual log analysis, such as inefficiency and error-proneness, offering a streamlined, automated approach for defect triage in 5G/6G network testing. Our research demonstrates that custom, compact models like our CNN architecture can not only offer a practical and efficient alternative to resource-intensive LLMs in specific industrial applications, but can even surpass their performance. While our results do not establish that custom CNN architectures can universally surpass LLMs in all log classification tasks, this study underscores the importance of considering architectural choices in light of real-world constraints and production overheads. In conclusion, this study not only presents a carefully-executed empirical approach to developing an edge-deployable ML model for accurate defect detection from raw software logs in the telecommunications industry but also provides valuable insights into the application of artificial intelligence in industrial settings, paving the way for future innovations in software log classification.

APPENDIX

See Figure 9.

ACKNOWLEDGMENT

The authors acknowledge the support of VIAVI Solutions Inc., for their provision of data, funding, and MLOps infrastructure, including GPUs, which contributed significantly to the completion of this project.

(Achintha Ihalage and Sayed Taheri contributed equally to this work.)

REFERENCES

[1] O. Ülkü, N. Gözüaçik, S. Tanberk, M. A. Aydin, and A. H. Zaim, "Software log classification in telecommunication industry," in *Proc. 6th Int. Conf. Comput. Sci. Eng. (UBMK)*, Sep. 2021, pp. 348–353.

[2] A. Catovic, C. Cartwright, Y. T. Gebreyesus, and S. Ferlin, "Linnaeus: A highly reusable and adaptable ML based log classification pipeline," in *Proc. IEEE/ACM 1st Workshop AI Eng.-Softw. Eng. AI (WAIN)*, May 2021, pp. 11–18.

[3] T. Pitakrat, J. Grunert, O. Kabierschke, F. Keller, and A. V. Hoorn, "A framework for system event classification and prediction by means of machine learning," *EAI Endorsed Trans. Self-Adapt. Syst.*, vol. 1, no. 3, pp. 1–8, Jan. 2015.

[4] M. Aljabri, A. A. Alahmadi, R. M. A. Mohammad, M. Aboulnour, D. M. Alomari, and S. H. Almotiri, "Classification of firewall log data using multiclass machine learning models," *Electronics*, vol. 11, no. 12, p. 1851, Jun. 2022.

[5] J. Voloskin, "Learning-based classification of software logs generated by a test automation framework," Tech. Rep., 2022.

[6] C. Bertero, M. Roy, C. Sauvanaud, and G. Tredan, "Experience report: Log mining using natural language processing and application to anomaly detection," in *Proc. IEEE 28th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2017, pp. 351–360.

[7] R. R. Abdalla and A. K. Jumaa, "Log file analysis based on machine learning: A survey," *UHD J. Sci. Technol.*, vol. 6, no. 2, pp. 77–84, Oct. 2022.

[8] R. Ren, J. Cheng, Y. Yin, J. Zhan, L. Wang, J. Li, and C. Luo, "Deep convolutional neural networks for log event classification on distributed cluster systems," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 1639–1646.

[9] S. Ramachandran, R. Agrahari, P. Mudgal, H. Bhilwaria, G. Long, and A. Kumar, "Automated log classification using deep learning," *Proc. Comput. Sci.*, vol. 218, pp. 1722–1732, Jan. 2023.

[10] R. B. Yadav, P. S. Kumar, and S. V. Dhavale, "A survey on log anomaly detection using deep learning," in *Proc. 8th Int. Conf. Rel., Infocom Technol. Optim. (Trends Future Directions) (ICRITO)*, Jun. 2020, pp. 1215–1220.

[11] X. She and D. Zhang, "Text classification based on hybrid CNN-LSTM hybrid model," in *Proc. 11th Int. Symp. Comput. Intell. Design (ISCID)*, vol. 2, Dec. 2018, pp. 185–189.

[12] Y. Lee, J. Kim, and P. Kang, "LAnoBERT: System log anomaly detection based on BERT masked language model," *Appl. Soft Comput.*, vol. 146, Oct. 2023, Art. no. 110689.

[13] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "HitAnomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 4, pp. 2064–2076, Dec. 2020.

[14] H. Ott, J. Bogatinovski, A. Acker, S. Nedelkoski, and O. Kao, "Robust and transferable anomaly detection in log data using pre-trained language models," in *Proc. IEEE/ACM Int. Workshop Cloud Intell. (CloudIntelligence)*, May 2021, pp. 19–24.

[15] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *Proc. IEEE 16th Intl Conf Dependable, Autonomic Secure Comput., 16th Intl Conf Pervasive Intell. Comput., 4th Intl Conf Big Data Intell. Comput. Cyber Sci. Technol. Congress(DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 151–158.

[16] W. Meng, Y. Liu, S. Zhang, F. Zaiter, Y. Zhang, Y. Huang, Z. Yu, Y. Zhang, L. Song, M. Zhang, and D. Pei, "LogClass: Anomalous log identification and classification with partial labels," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 1870–1884, Jun. 2021.

[17] T. Ergen and S. S. Kozat, "Unsupervised anomaly detection with LSTM neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 8, pp. 3127–3141, Aug. 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," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–7.

[19] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2017, pp. 236–242.

[20] E. Khabiri, W. M. Gifford, B. Vinzamuri, D. Patel, and P. Mazzoleni, "Industry specific word embedding and its application in log classification," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, Nov. 2019, pp. 2713–2721.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[22] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.

[23] W. X. Zhao et al., "A survey of large language models," 2023, *arXiv:2303.18223*.

[24] L. Bariah, H. Zou, Q. Zhao, B. Mouhouche, F. Bader, and M. Debbah, "Understanding telecom language through large language models," 2023, *arXiv:2306.07933*.

[25] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *Proc. Int. Conf. Autonomic Comput.*, 2004, pp. 36–43.

[26] Y. Zhang and A. Sivasubramaniam, "Failure prediction in IBM BlueGene/L event logs," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2008, pp. 583–588.

[27] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[28] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, and F. Shen, "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 807–817.

[29] Y. Fu, K. Liang, and J. Xu, "MLog: Mogrifier LSTM-based log anomaly detection approach using semantic representation," *IEEE Trans. Services Comput.*, vol. 16, no. 5, pp. 3537–3549, May 2023.

[30] Z. Xueqing, Z. Zhansong, and Z. Chaomo, "Bi-LSTM deep neural network reservoir classification model based on the innovative input of logging curve response sequences," *IEEE Access*, vol. 9, pp. 19902–19915, 2021.

[31] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2016, pp. 102–111.

[32] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.*, Oct. 2009, pp. 117–132.

[33] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "LogFiT: Log anomaly detection using fine-tuned language models," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 2, pp. 1715–1723, Apr. 2024.

[34] Z. Hou, M. Ghashami, M. Kuznetsov, and M. Torkamani, "HLogformer: A hierarchical transformer for representing log data," 2024, *arXiv:2408.16803*.

[35] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.

[36] R. Larisch, J. Vitay, and F. H. Hamker, "Detecting anomalies in system logs with a compact convolutional transformer," *IEEE Access*, vol. 11, pp. 113464–113479, 2023.

[37] K. Singhal et al., "Large language models encode clinical knowledge," *Nature*, vol. 620, no. 7972, pp. 172–180, Jul. 2023.

[38] J. Hoffmann et al., "Training compute-optimal large language models," 2022, *arXiv:2203.15556*.

[39] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," 2023, *arXiv:2307.03172*.

[40] A. Q. Jiang et al., "Mixtral of experts," 2024, *arXiv:2401.04088*.

[41] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*.

[42] H. Won Chung et al., "Scaling instruction-finetuned language models," 2022, *arXiv:2210.11416*.

[43] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontañón, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2020, pp. 17283–17297.

[44] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," 2021, *arXiv:2106.09685*.

[45] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar, "Accelerating the machine learning lifecycle with MLflow," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, Apr. 2018.

[46] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.

[47] C. Nguyen, T. M. Hoang, and A. A. Cheema, "Channel estimation using CNN-LSTM in RIS-NOMA assisted 6G network," *IEEE Trans. Mach. Learn. Commun. Netw.*, vol. 1, pp. 43–60, 2023.

[48] T. Wolf et al., "HuggingFace's transformers: State-of-the-art natural language processing," 2019, *arXiv:1910.03771*.

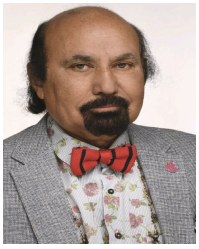
[49] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," 2023, *arXiv:2307.08691*.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

ACHINTHA IHALAGE received the Ph.D. degree in applied machine learning from Queen Mary University of London (QMUL), in 2022. From 2022 to 2023, he was a Postdoctoral Research Assistant with QMUL, contributing to multidisciplinary research efforts on AI-driven materials discovery for reconfigurable antenna design. He delivered undergraduate modules related to AI/ML as a Teaching Fellow with QMUL. He is currently in consulting industry, as an Associate Director, Data and AI, having also worked in the telecommunications sector as an AI/ML Specialist. His expertise lies in leveraging AI/ML within the antenna and telecommunication industry, with a specific focus on transformer-based language models and graph neural networks. His focus is on building generative and agentic AI systems with large language models (LLMs) for enterprise modernization and transformation. He also leads the integration of AI into the radio access network (RAN) infrastructure and RAN intelligent controller (RIC). He is the winner of the 2022 Mansel Davies Award for Dielectrics, awarded to the most outstanding early career researcher in the field by the Institute of Physics, U.K.

SAYED TAHERI (Member, IEEE) received the master's degree (Hons.) in electrical engineering—signal processing and communications with AI from The University of Edinburgh, U.K., and the Ph.D. degree from the University of London, Brunel. Since 2020, he has been a Research Fellow with VIAVI Solutions Inc., while holding a position as a Doctoral Researcher with Brunel University of London, working in cutting-edge AI/ML toward his Ph.D. He has been holding the position of AI/ML Solutions Architect and Team Lead with VIAVI Solutions Inc., since July 2022, where he leads cutting-edge research AI/ML initiatives and the productization of several PoCs in the telecom domain. He has over 15 years of experience in numerous large enterprises in the AI and data science fields. He is the main inventor of nine U.S. patents and the author and co-author of two books, one book chapter, and several peer-reviewed research articles. In 2024, he won the prestigious "Vice Chancellor's Prize for Doctoral Research" from the University of London, which is awarded yearly to the highest-achieving Ph.D. graduate across all fields and colleges. In the same year, he also won the "Dean's Prize for Doctoral Research Innovation and Impact" from the College of Engineering, Brunel University of London. He has won several scholarships and was the only winner of the USA's SPIE Education Scholarship Award from Asia in 2013. He has been a designated reviewer for several journals in IEEE and Springer. In January 2022, he was selected as the joint Technical Lead and SME in VIAVI's Centre of Excellence (COE) for AI. He was a distinguished student in the Olympiad of Mathematics in 2004.

FARIS MUHAMMAD received the M.Sc. and Ph.D. degrees from Strathclyde University, the P.G.C.E. degree from MMU, and the M.B.A. degree from Imperial College Business School, Imperial College London, with a focus on customer-facing and commercial skills. He is a Chartered Engineer (CEng). He has over 27 years of professional academic and industrial experience. Following five years of academic experience as a Senior Lecturer, he progressed his career in the telecoms industry, where he has developed expertise in software development, telecommunications, project management, and leadership. He has a distinguished track record in delivering complex projects, leading diverse teams, and driving successful change initiatives. He is also a Certified SAFe Agilist. He excels in Project and Program Management, employing both agile and waterfall methodologies. He is skilled in commercial and customer management, wireless technologies, systems engineering, and leadership and team management. In addition to his engineering and project lifecycle management skills, he has directed large and complex development programs encompassing software, systems engineering, DevOps, testing, AI/ML, and to a lesser extent, hardware. He has authored one book, four theses, six patents, and over 50 conferences and journal papers. He is a fellow of IET.



HAMED AL-RAWESHIDY (Senior Member, IEEE) received the Ph.D. degree from Strathclyde University, Glasgow, U.K., in 1991. He was with the Space and Astronomy Research Centre, Iraq, PerkinElmer, USA, Carl Zeiss, Germany, British Telecom, and U.K.; Oxford University, Manchester Metropolitan University, and Kent University. He is currently a Professor of communications engineering. He is also the Group Leader of the Wireless Networks and Communications Group (WNCG) and the Director of PG studies (EEE) with Brunel University London, U.K. He is the Co-Director of the Intelligent Digital Economy and Society (IDEAS) and the New Research Centre, which is a part of the Institute of Digital Futures (IDF). He is the Course Director for the M.Sc. Wireless Communication and Computer Networks. He is an Editor of the first book in *Radio Over Fibre Technologies for Mobile Communications Networks*. He acts as a consultant and involved in projects with several companies and operators, such as Vodafone, U.K.; Ericsson, Sweden; Andrew, USA; NEC, Japan; Nokia, Finland; Siemens, Germany; Franc Telecom, France; Thales, U.K. and France; and Tekmar, Italy, Three, Samsung and VIAVI Solutions, USA, and U.K.—actualizing several projects and publications with them. He is a Principal Investigator for several EPSRC projects and European projects, such as MAGNET EU Project (IP) 2004–2008. He has published more than 500 journals and conference papers, and his current research interests include 6G with AI and Quantum and the IoT with AI and Quantum. He is also an External Examiner for Beijing University of Posts and Telecommunications (BUPT)—Queen Mary University of London. Further, he was an External Examiner for a number of the M.Sc. communications courses with King's College London, from 2011 to 2016. He has also contributed to several white papers. Specifically, he was an Editor of Communication and Networking (White Paper), which has been utilized by EU Commission for research. He has been invited to give presentations at an EU workshop and delivered two presentations at Networld2020, and has been the Brunel Representative for NetWorld2020 and WWRF (for the last 15 years).