Leveraging Explainable AI for Reducing Queries of Performance Indicators in Open RAN

Chinenye Tassie, Brian Kim, Joshua Groen, Mauro Belgiovine, Kaushik R. Chowdhury
Institute for the Wireless Internet of Things, Northeastern University, Boston, USA
{tassie.c, br.kim, groen.j, belgiovine.m }@northeastern.edu, krc@ece.neu.edu

***Abstract*—Open Radio Access Network (O-RAN) is positioned to play a pivotal role in shaping the future of telecommunications networks through open interfaces and virtualization, allowing interoperability between different vendors. As a key departure from single-operator managed RAN, a remote RAN intelligence controller (RIC) queries the gNB for the Key Performance Indicators (KPIs) that are required for making RAN control decisions, often leveraging advanced machine learning (ML) models. However, this repeated querying increases control traffic overhead on the so called E2 interface connecting the gNB to the RIC. To address this challenge, we utilize a method from Explainable Artificial Intelligence (XAI), specifically SHapley Additive exPlanations (SHAP), which quantifies the contribution of each requested KPI to a model's prediction. Furthermore, we explore two different methods of choosing the most discriminative KPIs influencing model's performance, so that a smaller subset of KPIs may be queried, thus lowering the overhead on the E2 interface. Our analysis reveals that a model trained for the task of traffic classification using as input only the fraction of the top contributing KPIs identified by SHAP reduces control traffic overhead by up to 33% with only 7% reduction in ML classification accuracy.**

I. Introduction

Open Radio Access Network (O-RAN) is set to transform 5G and future networks with its four key specifications: disaggregation, intelligent controllers, virtualization, and open interfaces. It champions virtualized RANs, linking disaggregated elements through open interfaces and enhancing them with RAN intelligent controllers (RICs) [1] leading to networks that are interoperable across various vendor components and adaptable to dynamic resource requirements [2].

Problem. The O-RAN framework introduces programmable control namely near-real time (RT) RIC and non-RT RIC which process data and leverage machine learning (ML) to determine policies to be applied by the RAN [1]. The non-RT RIC is integrated with the network orchestrator and supports the near-RT RIC on longer term network management tasks ($t \geq$1s). Meanwhile, The near-RT RIC is deployed at the edge of the network and operates on a faster time scale (10 ms $\leq t \leq$ 1 s). It hosts xApps, which receive real-time data such as key performance indicators (KPIs) over the E2 interface that connects the gNB and the near-RT RIC. With these KPIs, the xApp can run a pre-trained ML model and send back control actions to quickly handle tasks such as dynamic resource allocation, load balancing, and interference management [3]. However, utilizing xApps for network management requires frequent probing for KPIs which introduces a large communication overhead. For example, consider a scenario

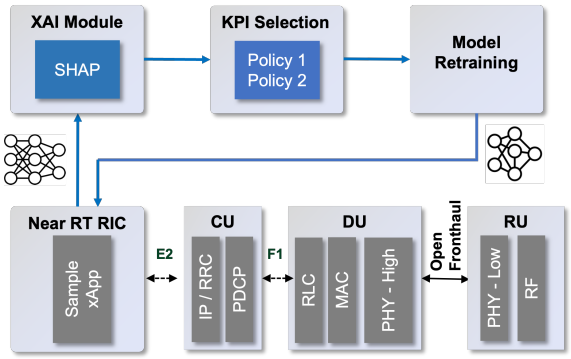


Fig. 1: O-RAN architecture with RT-RIC running an ML xApp that uses KPIs sent over the E2 interface from the base station. The SHAP method for Explainable AI identifies the KPIs that are most discriminative, selects a subset depending on the chosen policy and the re-trains a new model with fewer KPI inputs. This simpler model is returned back to the RT-RIC.

where all 17 KPIs described in Table I are reported for one UE. Requesting all 17 KPI results in median packet sizes of about 280 bytes, compared to 240 bytes and 200 bytes for 8 and 4 KPIs respectively. Scaling that up to 20UEs, the packet size can go up to 5088 bytes. Furthermore, these KPIs are transmitted in plain text, which raises concerns of data poisoning and eavesdropping by unauthorized parties. Encryption of the KPIs at the gNB followed by decryption at the RIC can address this concern, but again the processing overheads of cryptography and computing hashing functions are considerable. We showed earlier in [4], that computation is generally the bottleneck when large number of KPIs undergo the encryption/decryption process to secure the E2 interface, resulting in a delay of $\leq 50\mu$s.

Solution. Given that most examples of RT-RIC-based RAN control involve operations performed by ML models [3], our proposed solution is to explore if the same model outcomes can be achieved with fewer KPIs. This will result in less computational and communication overheads compared to the baseline case, without impacting the network performance. In recent years, there has been significant interest in the field of explainable artificial intelligence (XAI), which provides insight into a model's inner workings. Such methods have been used primarily for two applications: (i) interpretability/explainability (ii) optimization. *Interpretability* refers to presenting the intuition of the model in human-understandable terms [5], and *explainability* refers to understanding the fundamental operation of the model [6]. While explainabil-

ity/interpretability are relevant to xApp development, in this paper, we will focus on optimization which leverages the explanations to reduce the overhead at the E2 interface within the O-RAN. A summary of our approach is shown in Fig. 1.

Only a few works have explored XAI in the context of interpretability/explainability for the wireless domain. In [7], SHapley Additive exPlanation (SHAP) is used to understand which input features, such as traffic demand, active users, and time of delay, are important to the real-time decision of the model for short-term resource reservation (STRR) in 5G. Similarly, [8] proposes a method to describe the decision rationale of a DNN that predicts random environmental conditions for autonomous edge devices. These works on understanding how models deduce their results help in building better models, streamline data collection for model training, increase trust in the model, and identify biases or flaws in logic. For these reasons, we propose to use SHAP to understand which KPIs are most useful to the ML model in the xApp. However, this is not a trivial selection of KPIs simply ranked by SHAP: in a multi-class inference model, SHAP reveals the extent to which a given KPI contributes to each class. Thus, since such contribution levels vary widely on the SHAP scale, how to leverage the outcomes of SHAP to decide the overall set of KPIs is an open challenge.

Contributions. To demonstrate how XAI can help in the design of xApps, we present a case study of an ML model trained for traffic classification using O-RAN KPIs. We leverage SHAP to explain which KPI contributes most to the model's prediction. Then, we train the model only with selected KPIs to reduce the overhead of transmitting all KPIs and observe the model's accuracy. While there exist works that aim to show feature contribution to a model's prediction, to the best of our knowledge, this is the first paper that leverages XAI to train the model using selected features based on contribution to the model's prediction. Our contributions are as follows:

- We design a SHAP-based XAI software framework to identify the KPIs that contribute the most to the performance of an ML model trained for the example xApp of traffic classification.
- We present two methods to reduce the input size (number of KPIs) of the model, namely, *top K overall* and *top K per class*. We show that with reduced KPIs, we can still achieve high accuracy with only a 7% drop.
- We analyze the impact of communication overhead using fewer KPIs and observed a reduction of up to 33% of the data rate for control traffic compared to using all available KPIs while maintaining accuracy.

II. XAI Methodology

XAI methods provide an opportunity to understand the inner workings of a model and increase trust in the model. However, there are various considerations when choosing an XAI method as outlined in [6]. These considerations include model dependency, the scale of interpretability, and the data type. Model dependency distinguishes between methods tailored to specific models (model-specific) and those applicable to any model or algorithm (model-agnostic). Another factor is the scale of interpretability, with local explanations addressing individual predictions and global explanations shedding light on the overall workings of the entire model. Finally, data type refers to the data formats compatible with an XAI method such as text, tabular, and image data being the most popular.

With these considerations in mind, we chose SHAP [9] as our XAI method. SHAP provides a powerful framework that quantifies the contribution of each input feature to a model's prediction. It uses a game theory approach that considers all possible combinations of feature values and computes the marginal average contribution of each feature across these combinations. Specifically, we first define a subset of features $S \subseteq F$ where F is a list of all possible permutations of features. Next, we calculate the marginal contribution for each permutation of the features. Consider a model prediction $f(x_{S\cup\{i\}})$ that is trained with feature i present compared to a model $f(x_S)$ that is trained without feature i, the change in the model's prediction is the marginal contribution of that feature i. Given that the effect of a feature on the model's prediction also depends on the other features in the model, the marginal contributions have to be weighted. Finally, the weighted marginal contributions are averaged over all possible permutations. In summary, shapley value $\phi(i)$ is the weighted average marginal contributions that feature i makes to all possible feature subsets in F. Our formulation below is obtained from [9]:

$$\phi(i) = \frac{1}{|F|!} \sum_{S\subseteq F\setminus\{i\}} |S|!(|F| - |S| - 1)![f(x_{S\cup\{i\}}) - f(x_S)]$$

In our work, we use DeepSHAP, an extension of SHAP that integrates DeepLift [10] and Shapley values. DeepSHAP is specifically tailored for deep neural networks (DNNs) which generates more accurate and fine-grained explanations. It uses the same game theory approach as above but assesses the contribution of each feature to a prediction by considering how they interact within the neural network's layers.

III. Traffic Classification Model

In this section, we discuss three stages of the data collection process for capturing KPIs, the details of the traffic classification model, and the pre-processing done to the model's input.

A. Data Collection and Pre-Processing

To train our traffic classification model, we generate a dataset using a three stage process. First, we collect real-world 5G user traffic using Google Pixel 6 Pro smartphones. Per 5G specifications, there are 3 defined types of network slices for different types of traffic where Ultra Reliable Low Latency Communications (URLLC) focuses on low latency and high reliability, enhanced Mobile Broadband (eMBB) aims to provide high-speed data rates, and massive Machine Type Communications (mMTC) maximizes the number of concurrent connections. In our scenario, we generate eMBB traffic through Netflix streaming, URLLC traffic through video chat on Facebook Messenger (including one audio-only period),

KPI Name	Description
dl_mcs	downlink modulation and coding
dl_n_samples	number of download samples in previous 250 ms
dl_buffer_bytes	downlink queue length in bytes
tx_brate_downlink_Mbps	downlink bitrate in Mbps
tx_pkts_downlink	downlink number of packets transmitted in previous 250 ms
dl_cqi	downlink channel quality indicator
ul_mcs	uplink modulation and coding
ul_n_samples	uplink number of samples in previous 250 ms
ul_buffer_bytes	uplink queue length in bytes
rx_brate_uplink_Mbps	uplink bitrate in Mbps
rx_pkts_uplink	uplink number of packets recieved in previous 250 ms
rx_errors_up_perc	uplink percent of packets with errors in previous 250 ms
ul_sinr	uplink signal to interference and noise ratio
phr	UE power head room
sum_reqsted_prbs	sum of the resource blocks requested in previous 250 ms
sum_granted_prbs	sum of the resource blocks granted in previous 250 ms
ul_turbo_iters	uplink turbo encoding

TABLE I: List of the 17 KPIs used for traffic classification.

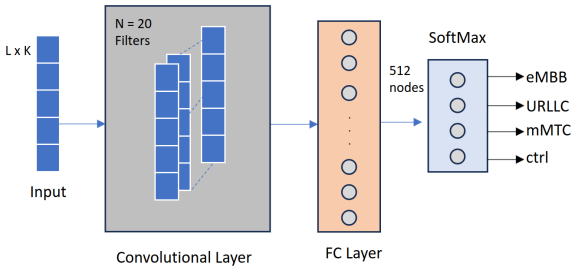


Fig. 2: Overview of CNN architecture where L represents slice length and K represents the number of KPIs.

and mMTC traffic through text messaging while capturing all background application data.

In the second stage, we use the collected packets to generate traffic between the UE and gNB on Colosseum [11], the world's largest publicly available RF emulator with actual SDRs in the loop. The experimental setup consists of one UE, a gNB, and the near-RT RIC which are implemented in LXC containers each connected to separate software Defined Radios (SDRs) [4]. Each UE is allocated a distinct traffic slice in accordance with the 3 defined in the 5G specifications: URLLC, eMBB, and mMTC, while the gNB is connected to the near-RT RIC over a wired 10 Gbps called the E2 interface. This E2 interface functionality is integral for the later collection of KPIs. Our replay script allows us to replicate the timing and length as if the original communication was taking place on Colosseum.

In the final stage, we capture the KPIs as they traverse the E2 interface. We modify the sample KPI monitor xApp that is integrated with ColO-RAN [12] and probe the gNB for the requested KPIs every 250 ms. We use tcpdump to capture every packet traversing the E2 interface to analyze the actual throughput and packet level details in an O-RAN system.

B. Training CNN model

Before training the model, we pre-process the collected KPIs. To protect user privacy, we remove any KPIs that contain uniquely identifiable information for device or user identification. The 17 KPIs used to train our model are listed in Table I. Next, we trim surplus time from both the beginning and the end of the KPI capture. Although the replay script reproduces the initial capture, there are intervals before and after its execution when KPIs continue to be captured. Finally, certain slices may experience extended periods of no traffic so we manually examine and eliminate those silent intervals to ensure suitability for training.

To create the input for our model, we divide the KPIs into batches of L consecutive time samples for each KPI where $L = \{8, 16, 32\}$. For the traffic classification task, we train a Convolutional Neural Network (CNN) using PyTorch. Our model consists of 3 layers: a single 2D CNN layer with 20 kernels using ReLU activation, 1 fully connected (FC) hidden layer of 512 neurons with ReLU activation function and the Softmax output layer with CrossEntropyLoss as loss function. For the CNN layer, the spatial size of the filters is set to be 4×1 to capture the local patterns in time dimension for each individual feature value, given that they are not spatially correlated on the KPI features dimension. We train our model for up to 350 epochs with early stopping strategy. The CNN model is depicted in Fig. 2. We have 4 classes, i.e. $\{$*eMBB, URLLC, mMTC, ctrl*$\}$, where *ctrl* represents packets with periods of inactivity.

We divide the dataset into an 80/20 split for training and testing, conducting multiple training iterations of our model. We then choose the model with the highest accuracy for the subsequent evaluation of feature contributions in Section IV.

IV. Proposed Methods For Model Optimization

Now, we introduce our proposed methods for identifying the most influential KPIs for the model's performance. Our approach to identifying the most influential KPIs can be split into two steps. In the first step, we use DeepSHAP to generate feature importance explanations for the traffic classification model. In the second step, we select a subset of KPIs that are positively contributing to the model prediction where we propose two methods to select KPIs, top K KPIs overall and top K KPIs per class.

A. Feature Importance Explanations - DeepSHAP

To obtain the feature importance of the KPIs, we used DeepSHAP, an extension of SHAP specifically for neural networks. First, we initialize an object called *explainer* which takes the trained CNN model and training data as input. Then, using the *explainer* object, we calculate the SHAP values for the test data. This process is repeated for all instances and feature permutations. Fig. 3 shows, in a human understandable way, how much each KPI (feature) contributes to the prediction of the traffic classification model.

B. Method 1 - Top K KPIs Overall

After obtaining the feature contribution for all KPIs, method 1 (MTH1) proposes to pick the top K KPIs across all classes. The returned SHAP values have dimensions [C, N, L, 17] which represents the number of class, number of samples, the slice length, and the number of all available features respectively. The SHAP values are computed per class and

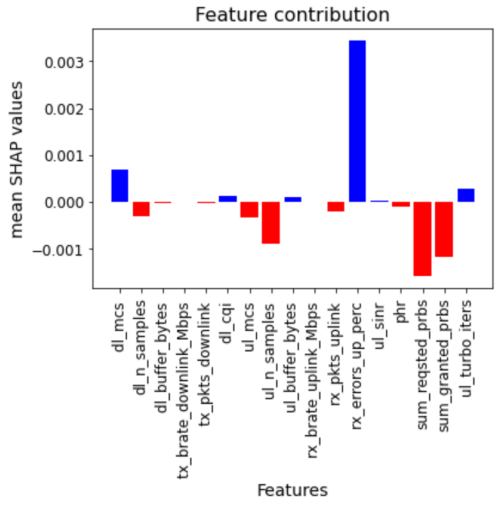


Fig. 3: DeepSHAP results showing the feature contribution of the 17 KPIs for the traffic classification model for slice length $L = 32$. Blue represent positive contribution while red represents negative contribution.

for each instance, but to determine the overall contribution of KPIs, we apply averaging. First, we reshape the SHAP value array from 4D to a 2D array with dimensions [$C \times N \times L$, 17]. Next, we average each column to obtain an array of dimension 1×17, containing the averaged SHAP values across all classes (i.e., overall) for all KPI which can be visualized as Fig. 3. By sorting these averaged SHAP values, we identify the top K contributing features. Here, we opt for average SHAP value because it offers a holistic view of the net feature contribution. This choice accommodates situations where a KPI might have a strongly negative impact on one class but significantly positive impact on another - helping us identify features that have a stable influence across various classes. To further analyze the impact of K, we retrain the model with different K values in Section V-A.

C. Method 2 - Top K KPIs per class

For method 2 (MTH2), we use the mean absolute SHAP values or the average of the absolute SHAP values. We start with the 4D array of SHAP values with dimensions [C, N, L, 17] returned by the SHAP *explainer* object. For each class C, we extract the SHAP values array with dimensions [N, L, 17] and average along the L dimension to obtain an array with dimension [N, 17]. Next, we take the mean absolute which will result in an array of dimension 17 for each KPI and for that class. We repeat the process for all classes and the resulting array has dimensions [C, 17] which represents the mean absolute SHAP values for all KPI and for each class. When choosing the top K per class, we start selecting K KPIs for one class and move on to the other class when selecting is finished. However, there is a possibility of selecting the same KPI for different classes. Then, we choose the next top KPI when the KPI overlaps with that from another class. Again, to further analyze the impact of K, we retrain the model with different values of K in Section V-A.

Method (L= 32, K=8)	**Model Accuracy**
MTH1 (mean SHAP)	83.6%
MTH1 (mean absolute SHAP)	77.5%
MTH2 (mean SHAP)	77.8%
MTH2 (mean absolute SHAP)	80.8%

TABLE II: Average model accuracy comparing MTH1 (mean and mean absolute) to MTH2 (mean and mean absolute). For MTH1, mean gives better results while mean absolute gives better results for MTH2.

Here, we order SHAP values by their absolute mean because it outperforms mean SHAP values (as illustrated in Table II). Since this is on a per class basis, we are more concerned with the most discriminative features (regardless of if positive or negative). Using the mean on a per class basis will overcompensate and risk excluding features that are discriminative but have very negative contributions.

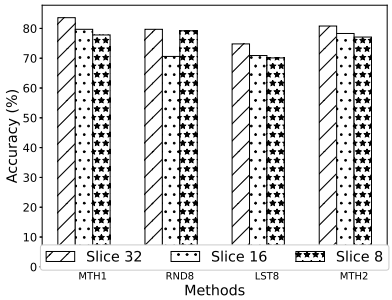
V. Performance Evaluation

In this section, we present the results of method 1 and method 2 which use a reduced number of KPIs as input and compared to the result of using all 17 KPIs. We also analyze the impact of using fewer KPIs on the communication overhead over the E2 interface. For our KPI reduction analysis, we partition our dataset into training and testing data. We opt for a balanced training and test dataset of 200 samples for 4 classes *(eMBB, URLLC, mMTC, ctrl)*, resulting in a total of 800 samples for SHAP analysis. Next, we use DeepSHAP to calculate the SHAP values for the features resulting in an array of dimension $[4 \times 800 \times 32 \times 17]$, where 4 represents the number of class C, 800 represents the number of samples N, 32 represents the slice length L and 17 represents the number of available features.

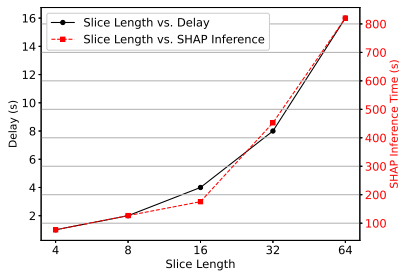
A. Reduced KPIs for Traffic Classification

To evaluate how the model's accuracy is affected by using a reduced set of KPIs selected by MTH 1 and MTH2 2, we compare its performance with using all 17 KPIs. We also compare our proposed methods to two benchmark alternatives, namely, random K KPIs (RND K) and least K KPIs overall (LST K) where we choose K KPIs randomly or pick K KPIs with the most negative contributions over all classes. For the random K case, K KPIs are randomly picked once and fixed for the simulation.

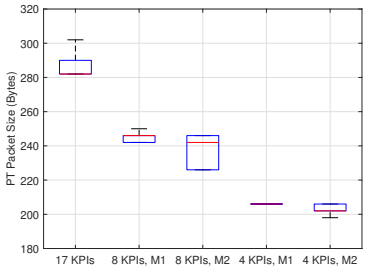
In Fig. 4(a), we observe that method 1 consistently outperforms method 2 for all slice lengths. However, for slice length 8, RND8 outperforms method 1. From our analysis, we have observed that model accuracy tends to decrease with shorter slice lengths where slice length 8 yields the lowest accuracy even with utilizing all 17 KPIs. Since SHAP feature contributions rely on the trained model, it is reasonable to conclude that KPIs obtained from slice length 8 may not accurately represent the contributions of the selected KPIs compared to those derived from slice length 32. As anticipated, our method 1 outperforms LST8 for all slice lengths. For method 2, we consider $K = 2$ for each class, resulting in 8 KPIs. Compared to $K = 8$ for method 1, we observe that



(a) Model accuracy across different methods when $K = 8$ for MTH 1 and $K = 2$ for MTH2.



(b) CNN classification time and SHAP inference time for various slice lengths where $L = \{4, 8, 16, 32, 64\}$.



(c) Packet sizes for different number of KPIs for both MTH1 and MTH2. MTH 2 offers up to 28% reduction for $K = 4$.

Fig. 4: Results showing the model accuracy for the proposed methods and the CNN delay cost for various slice lengths.

K	Avg. Model Accuracy
4	76.0%
5	78.0%
8	80.3%
10	76.7%
12	71.5%
17	87.1%

TABLE III: Average model accuracy for 17 KPIs and method 1 (Top N) where $N = \{4, 5, 8, 10, 12\}$. The accuracy is averaged across slice lengths and $N = 8$ offers the best performance among method 1.

method 2 performs worse than method 1, as illustrated in Fig. 4(a). We attribute this drop in performance to the varying contributions of the 4 classes to the model where method 2 weights contribution equally. For example, the top KPI for a class might have a low SHAP value when compared to all KPIs. Overall, method 1 outperforms method 2 as seen in Fig. 4(a) and we observe only a slight drop of approximately 7% in model accuracy (considering the best-case scenario of a slice length of 32 and method 1 with $K = 8$) when compared to a model trained with all 17 KPIs.

Further analysis of method 1 is presented in Table III where we consider $K = \{4, 5, 8, 10, 12\}$. Here we observe two key things: 1) the accuracy of the model increases with slice length and 2) increasing K beyond a limit can reduce accuracy due to the inclusion of features that might have negative contributions to the model. For example, Fig. 3 shows that 8 features contribute non-negatively to the model. In this case, selecting more than 8 features will introduce features that have a negative impact on the model's performance, potentially lowering its accuracy. As seen in Table. III, the accuracy of the model is highest for $K = 8$ and there is a downward trend in accuracy when $K = \{10, 12\}$. Note that we average over different slice lengths ($L = 8, 16, 32$) for K KPIs chosen by method 1.

It is important to note that the consideration for choosing the best method should not be solely based on accuracy; there is an associated cost to consider. For each slice length, there is a cost that relates to the time it takes for the CNN to make a decision. The various slice lengths $S = \{4, 8, 16, 32, 64\}$ translate to a CNN classification time $T = \{1, 2, 4, 8, 16\}s$, as illustrated in Fig. 4(b). Thus, choosing the best method and set of KPIs should give consideration to the model slice length. We also include the time it takes SHAP to generate the feature contributions for the various slice lengths in Fig. 4(b).

B. Impact on encryption overhead

In O-RAN, there is always a cost to moving data between different nodes. In systems with open interfaces, there is an additional cost of securing the data while in transit. While the cost of securing the E2 interface is generally low [4], the cost grows as the system scales in size. Thus, reducing the size of the metrics or KPIs that must be sent across this E2 interface, without reducing the performance of the ML agents using them, directly reduces the cost of O-RAN.

We design an experiment using our emulation environment described in Sec. III-A to empirically determine the reduction in the data sent across the E2 interface based on our proposed methods. We use a single UE connected to the gNB and send a mixed data trace containing all three traffic slices. The gNB generates KPIs and sends them over the E2 interface to our near-RT RIC. It is important to note that KPIs are encoded using ASCII and are not fixed size. In other words, if a specific KPI has values in the range $[0, 12500]$, the data size of that KPI can change from 1 to 5 Bytes. We capture all traffic traversing the near-RT RIC to observe the actual packet size based on the number of KPIs. We created a baseline by sending all 17 KPIs, and then repeated the experiment using 4 and 8 KPIs with method 1 and method 2.

The experiment generates 5,433 to 14,868 individual packets captured on the E2 interface per trial. Fig. 4(c) is a box-plot of the actual plain text (PT) packet sizes for each experiment. It is clear that reducing the number of KPIs decreases the amount of data sent across the E2 interface, by up to 28% when sending 4 KPIs using Method 2.

While these results clearly show the reduction in data sent across the E2 interface, more analysis is needed to fully understand the cost savings. These KPIs were sent without any encryption in order to capture and correctly identify the packets. However, the O-RAN Alliance specifications call for using IPsec to secure all traffic crossing the E2 interface. Adding IPsec adds additional headers increasing the overhead by at least 57 Bytes. Additionally, these KPIs were generated

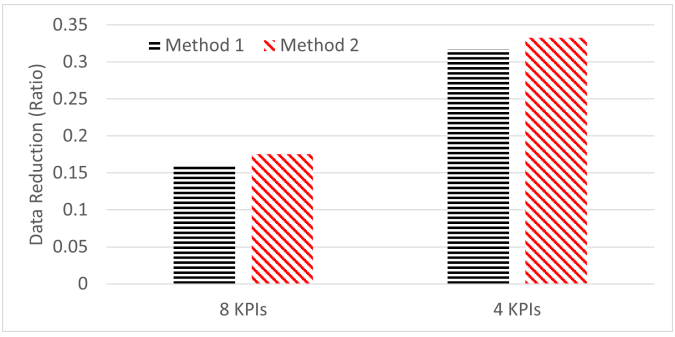


Fig. 5: Data reduction ratio for $U = 20$ UEs. Method 2 with 4 KPIs reduces the bit on the wire by $1/3$.

from a single UE connected to a gNB. As the number of UEs served by the gNB increases, the amount of KPIs sent must also increase. However, multiple UEs' KPIs can be reported in a single packet, reducing some of the overhead associated with the various headers at each network layer. Therefore, we build a model to analyze these variables and understand the data reduction when IPsec is used and the number of UEs scales.

We use the median packet size for each experiment shown in Fig. 4(c) and subtract all headers to find the median payload size. We use this median payload size for a single UE and add on a fixed overhead of 60 Bytes for IPsec as well as the transport layer (SCTP), network layer (IP), and data link layer (Ethernet) header overheads. We then increase the number of UEs until the packet reaches the MTU size of 1500 Bytes, at which point we form a new packet. We can then calculate the total data sent over the E2 interface for a specific number of UEs. In Fig. 5, we plot the data reduction ratio for each method and number of KPIs for $U = 20$ UEs. The data reduction ratio is given by

$$1 - \frac{\sum_{u=1}^{U} b_{m,k}}{\sum_{u=1}^{U} b_{0,17}}$$

where $b_{m,k}$ is the total bits on the wire sent for method m and number of KPIs k summed for U number of UEs. This ratio gives the relative reduction in bits sent across the E2 interface for a particular method and number of KPIs compared to sending all 17 KPIs for a certain number of UEs.

As Fig. 5 shows, reducing the number of KPIs can reduce the total bits sent over the E2 interface by up to a third. This means the actual throughput decreases by up to $1/3$, freeing network resources to carry other traffic. Perhaps more importantly, this also decreases the burden on the local processor. In fact, CPU utilization increases linearly with throughput when using IPsec [4]. Therefore, reducing the throughput will reduce the CPU utilization by the same factor. For a distributed, edge device with limited computing resources, reducing the CPU utilization due to IPsec by $1/3$ represents a huge savings in the cost of securely transferring the KPIs.

VI. Conclusion

ORAN is poised to transform the telecommunications landscape by embracing open interfaces and virtualization for enhanced interoperability among diverse vendors. The integration of RIC and xApps for network management provide an agile and dynamic system. However, the repeated probing for KPIs introduces a large communication overhead on the E2 interface connecting the gNB to the RIC. In this work, we propose XAI techniques, specifically SHAP, to precisely quantify the impact of individual KPIs on a traffic classification model's predictions. Furthermore, we explore 2 strategies for selecting the most influential KPIs, Top K overall and Top K per class, effectively reducing the query load on the E2 interface. Our findings demonstrate that by training models using only the top-contributing KPIs identified by SHAP for traffic classification, we can curtail control traffic overhead by up to one-third, with a modest 7% reduction in ML classification accuracy. Our study highlights the possibility of achieving a balance between network efficiency and predictive accuracy, ushering in a promising era of innovation and deployment in the realm of O-RAN.

References

[1] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges," 2022.

[2] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, programmable, and virtualized 5G networks: State-of-the-art and the road ahead," *Computer Networks*, vol. 182, p. 107516, dec 2020. [Online]. Available: https://doi.org/10.1016%2Fj.comnet.2020.107516

[3] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and learning in O-RAN for data-driven NextG cellular networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 21–27, oct 2021. [Online]. Available: https://doi.org/10.1109%2Fmcom.101.2001120

[4] J. Groen, B. Kim, and K. Chowdhury, "The Cost of Securing O-RAN," in *IEEE International Conference on Communications (ICC)*, 2023.

[5] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017.

[6] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable AI: A review of machine learning interpretability methods," *Entropy*, vol. 23, no. 1, 2021. [Online]. Available: https://www.mdpi.com/1099-4300/23/1/18

[7] P. Barnard, I. Macaluso, N. Marchetti, and L. A. DaSilva, "Resource reservation in sliced networks: An explainable artificial intelligence (XAI) approach," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1530–1535.

[8] P. M. Dassanayake, A. Anjum, A. K. Bashir, J. Bacon, R. Saleem, and W. Manning, "A deep learning based explainable control system for reconfigurable networks of edge devices," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 7–19, 2022.

[9] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[10] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," *CoRR*, vol. abs/1704.02685, 2017. [Online]. Available: http://arxiv.org/abs/1704.02685

[11] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, A. Bagga, P. Patel, V. Petkov, M. Seltser, F. Restuccia, A. Gosain, K. R. Chowdhury, S. Basagni, and T. Melodia, "Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2021, pp. 105–113.

[12] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing machine learning-based xApps for Open RAN closed-loop control on programmable experimental platforms," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5787–5800, 2023.