# Encrypted Network Traffic Classification with Higher Order Graph Neural Network

Zulu Okonkwo*, Ernest Foo, Zhe Hou, Qinyi Li, and Zahra Jadidi

Griffith University, Australia
{z.okonkwo,e.foo,z.hou,qinyi.li,z.jadidi}@griffith.edu.au

**Abstract.** Encryption protects internet users' data security and privacy but makes network traffic classification a much harder problem. Network traffic classification is essential for identifying and predicting user behaviour which is important for the overall task of network management. Deep learning methods used to tackle this problem have produced promising results. However, the conditions on which these experiments are carried out raise questions about their effectiveness when applied in the real world. We tackle this problem by modelling network traffic as graphs and applying deep learning for classification. We design a graph classifier based on higher order graph neural network with the aim of optimum generalisation. To demonstrate the robustness of our model, we cross validate it on the ISCXVPN and USTC-TFC datasets with varying input specifications. We use our model to demonstrate the impact of network data truncation on traffic classification and define benchmarks for input specifications. Our best results outperform the state-of-the-art in terms of generalisation strength. Our tool is available online[1].

**Keywords:** Graph Neural Network · Encrypted Network Traffic · Classification · Network Security · Deep Learning.

## 1 Introduction

In the design of network security systems, the core aim is an accurate classification of network traffic by the security apparatus [2]. Traditional network security involves combining various layers of security in a defence-in-depth approach. Each layer plays a role with the overall aim of controlling what data enters or exits the network. Encryption aids security by strengthening privacy amongst communicating channels, leading to its wide adoption across the internet. Over 94% of Google's traffic now runs over TLS or SSL [3]. While improving privacy over the internet, encryption has created new attack surfaces for bad actors to traverse unnoticed in and out of networks. Due to the sophisticated nature of encryption algorithms, security systems struggle to accurately classify network traffic. Recently, a new wave of encrypted attacks has spread across the internet,

* Corresponding Author

[1] https://github.com/zuluokonkwo/Encrypted-Network-Traffic-Classification-with-Higher-Order-Graph-Neural-Network

with the rise of cyber physical systems(CPS) and internet of things (IoT) cyber threats over encrypted channels saw a 132% increase in 2022, and malware over HTTPS continues to rise [1].

Machine learning (ML) and Deep Learning (DL) based methods have experienced huge success in the overall task of network traffic classification. However, their results are questionable because of underlining issues. State-of-the-art ML based classifiers [2,4,5,6] have more emphasis on the model design stage while testing on a small set of data and not paying much attention to the data processing/feature extraction stage. This process yields good but questionable results as the generalisation capability of the model isn't guaranteed. State-of-the-art Graph neural networks (GNN) based designs [22,15] are built with low-order GNNs. This method utilises node locality for classification tasks, not taking advantage of higher-order information that has proven to preserve structural embeddings necessary for graph classification.

The imbalanced nature of network traffic datasets is a concern for ML/DL based classifiers; training with imbalanced data directly impacts a model's performance. Results produced with such datasets are biased towards the majority classes. State-of-the-art classifiers [5,4,6] utilised imbalanced datasets for their training process not clearly stating how this issue is contained. DL/ML classifiers deal with a lot of parameters that increase during model training to enable the network generalise optimally. An increase in parameters also accounts for an increased tendency of an over-fitting model. State-of-the-art classifiers [22,5,4,6] use dropout to deal with this tendency. Research by Garbin et al. [8] advised that dropouts be used with caution and when in doubt batch normalisation be used instead. Garbin et al. [8] also advised that dropouts be used with different rates to find the optimum spot which yields the best accuracy. Cross-validating and training a model across a range of datasets is a way to show a model is not overfitting. Network traffic is dynamic and varies in size. The number and sizes of packets in a traffic session vary for different applications. For ML/DL classification approaches, defining input specifications is a requirement for models. Therefore, defining input sizes that yield the best results when variable-size inputs cannot be used is important. State-of-the-art classifiers [4,6] truncate network input during training. A trade-off of such should be backed with a clear explanation or leverage DL methods that can perform computation on variable-sized inputs.

In this paper, we design an encrypted traffic classifier based on a higher-order graph neural network. Our design is based on a GNN blueprint by Morris et al. [21] that preserves structural information necessary for graph classification. We test our model on two datasets with different input specifications to investigate which is optimum for the task of encrypted traffic classification. Our model can be applied to the network layer of CPS, IoT and smart critical infrastructures to identify traffic that pose risks to the systems. In our training phase, we implement a weighted random sampler that ensures every class is well represented, making our model unbiased towards any class. We perform stratified cross validation across our dataset with a fold of 5 to ensure our results are as realistic as possible. Our model demonstrated better generalisation strength when compared to state-of-the-art that used the same dataset. Our best results for the VPN-dataset are above 97% across all evaluation metrics which is better than the compared literature. For the non-VPN dataset our best results are above 87% for all evaluation metrics. For the Benign and Malware classes of the USTC-TFC dataset, we achieve well over 95% for all evaluation metrics. Our contributions can be itemized as follows:

- Development of a higher order GNN-based model that can identify and classify encrypted network traffic with optimum generalisation strength.
- Improved the feature extraction process for encrypted traffic to better represent and show relationship of traffic flows.
- Evaluate the impacts of data truncation and padding for encrypted network traffic classification at the data-link layer (L2) and sessions layer (L5).

The rest of this paper is structured as follows: In Section 2, we introduce the preliminaries that lay foundations to our data processing and model design. In Section 3, we introduce our GNN-based classifier and define the data processing methodology. In Section 4, we evaluate our model on the ISCXVPN and USTC-TFC datasets. We also discuss the results gotten from our experiments and conduct a comparative analysis with state-of-the-art models. In Section 5, we highlight some related papers and discuss gaps which our paper addressed. In Section 6, we conclude the work and highlight future directions.

## 2 Preliminaries

The advancements in neural networks for tasks like pattern recognition, image classification, and predictive analytics have made them valuable tools for researchers in tackling complex issues. They excel in uncovering hidden patterns in data that can be plotted on a plane but struggle with non-euclidean data [9]. This has led to the development of Geometric Deep Learning to better handle such data. Graphs, which are non-euclidean representations of entities and their relationships, can effectively show connections between two or more elements. They are versatile, as they can be used to model anything, and are particularly useful for modelling non-euclidean network data, which contains a wealth of information. To fully harness its potential, it must be represented formally.

### 2.1 Graph Neural Network

A simple graph $G$ consists of a non-empty finite set $V(G)$ of elements called nodes (or vertices), and a finite set $E(G)$ of distinct unordered pairs of distinct elements of $V(G)$ called edges. Mathematically, a graph $G$ is defined by Equation 1 as:

$$G = VE, \tag{1}$$

where $V$ is the vertex set and $E$ is the edge set. For example, a visual representation of a graph $G$, with vertex set $V(G) = \{a, b, c, d\}$ and edge set $E(G) = \{ab, ac, bc, cd\}$ is shown in Figure 1.

In Figure 1, we assume every vertex possesses a feature that defines itself called vertex or node attributes. Graph neural network (GNN) falls under the branch of geometric deep learning, a type of deep learning for emerging methods aiming to generalise deep learning models to non-euclidean domains mainly graphs and manifolds [9]. Early GNN methods were generated with the main goal of constructing a generalisation of CNN architecture on non-euclidean domains. In neural networks, activations are received from preceding layers in order to propagate features to succeeding layers. This is defined in Equation 2 as:

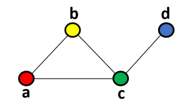


**Fig. 1.** An example graph.

$$H^{[l+1]} = \sigma(W^{[l]}H^{[l]} + b^{[l]}), \tag{2}$$

where $H^{[l+1]}$ is the feature representation at layer $[l+1]$, $\sigma$ is a non-linear activation function, $W^{[l]}$ is the weight at the $l^{th}$ layer, $H^{[l]}$ is the feature representation at the $l^{th}$ layer and $b^{[l]}$ is the bias at the $l^{th}$ layer. Kipf et al. [10] defined a Graph Convolution Network (GCN) propagation rule by taking into consideration the adjacency matrix. The adjacency matrix in graph theory shows how nodes are connected to each other. This helps nodes learn information about their neighbours during forward propagation. The adjacency matrix forward propagation can be defined in Equation 3 as (we eliminate the bias $b$ for simplicity):

$$H^{[l+1]} = \sigma(W^{[l]}H^{[l]}A^{*}). \tag{3}$$

$$H^{[norm]} = D^{-1}AH. \tag{4}$$

$$H^{[norm]} = D^{-1}\tilde{A}H. \tag{4a}$$

$$H^{[norm]} = D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H. \tag{4b}$$

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}W^{l}). \tag{5}$$

The $A^{*}$ in Equation 3 represents the normalised adjacency matrix, normalisation prevents numerical instabilities like vanishing and exploding gradients. Matrix normalisation is the dot product of the inverse of the degree matrix, $D^{-1}$ with the adjacency matrix, $A$ and the node features $H$ as shown in Equation 4. By implementing a self-loop, a node takes into account its own feature during forward propagation. Setting the diagonal elements of an adjacency matrix to 1 implements self-loops, $\tilde{A}$ represents the refined adjacency matrix in Equation 4a. Due to varying node degrees, higher degrees nodes tend to dominate. Kipf et al. [10] suggest symmetric normalisation to reduce this dominance in Equation 4b. After normalizing $A$ with its degree matrix and enforcing self-loop by adding the identity matrix. We get the final equation for forward propagation in GCN as Equation 5, $\sigma$ represents a nonlinear activation function like ReLU or Tanh used during neural network computation.

GNN's evaluation and analysis have been more empirical than theoretical, making the entire process seem like a black box. Morris et al. [21] tried to give clarity to this issue by relating GNN operations to the Weisfeiler-Leman graph isomorphism heuristic (1-WL). They show that GNNs possess the same expressiveness as the 1-WL in terms of distinguishing non-isomorphic graphs. Meaning that both algorithms have the same imperfections. They propose a higher-order generalisation of GNNs called $k$-GNNs which can capture structural information not visible at the node level. In $k$-GNNs, messaging passing is between sub-graph structures rather than just nodes. Let $V(G)^a$ be a sub-graph of $V(G)$ and $b$ be a set of nodes in $V(G)^a$. The neighbourhood of $b$ is defined below.

$$N(b) = \{t \subset [V(G)]^a \text{ s.t. } |b \cap t| = a - 1\}, \tag{6}$$

Basically, the local neighbourhood $N_L(b)$ consists of all elements in the set $t \subset N(b)$ such that $(v, w) \in E(G)$ for unique $v \in b \setminus t$ and unique $w \in t \setminus b$. The local propagation formula of feature vectors for layer $\ell > 0$ becomes:

$$f^{(\ell)}_{a,\ \mathrm{L}}(s) = \sigma\left( f^{(\ell-1)}_{a,\ \mathrm{L}}(s) \cdot W^{(t)}_1 + \sum_{u \in N_L(s)} f^{(\ell-1)}_{a,\ \mathrm{L}}(u) \cdot W^{(\ell)}_2 \right), \tag{7}$$

where $f^{(\ell-1)}_{a,\ \mathrm{L}}(s)$ represents the feature vector of the set of nodes $s$ at layer $t-1$. The aggregation of the local neighbourhood set of $s$ represented by $u$ is $[\sum_{u \in N_L(s)} f^{(\ell-1)}_{a,\ \mathrm{L}}(u)]$. For simplicity, Equation 7 is denoted as:

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j, \tag{8}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are trainable weight parameters of the GNN, $\mathbf{X}_i$ signifies the feature vectors of set of nodes $i$. $\sum_{j \in \mathcal{N}(i)}$ denotes the aggregation of the local neighbourhoods of $i$ denoted by $j$. The edge weight of the local neighbour is denoted by $e_{j,i}$ and $\mathbf{x}_j$ signifies the feature vector of the neighbours. A non-linear function such as ReLU is utilised during computation.

### 2.2 Pooling

Pooling is popular for dimensionality reduction in convolutional-based systems. In this process, the dimension of the feature map is reduced while retaining useful information and eliminating irrelevant information from the input data. Pooling reduces the complexity of upper layers and simplifies computation by reducing the weight parameters. It also controls over-fitting to a reasonable extent.

*Global mean pooling* This method of pooling was introduced by Yann LeCun [11]. It returns batch-wise graph-level-outputs by averaging node features across the node dimension. The output for a single graph $G_i$ is computed by:

$$\mathbf{r}_i = \frac{1}{N_i} \sum_{n=1}^{N_i} \mathbf{x}_n \tag{9}$$

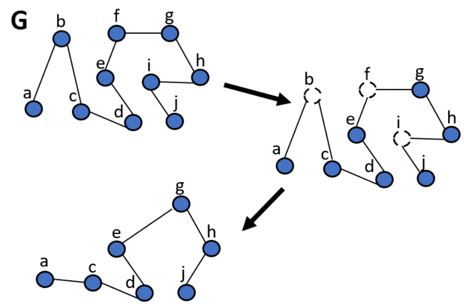


**Fig. 2.** Condensed graph after top-K pooling. Nodes are dropped after every pooling layer making neighbourhood computation for generating graph embedding easier.

*Global max pooling* The max pooling [12] method simply passes to the next layer the maximum value within a group of R activations. It returns batch-wise graph-level-outputs by taking the channel-wise maximum across the node dimension. The output for a single graph $G_i$ is computed by:

$$\mathbf{r}_i = \max_{n=1}^{N_i} \mathbf{x}_n \tag{10}$$

For Equations 9 and 10, $\mathbf{x}$ represents the node feature matrix.

*Top-k pooling* This is a sort-based method for dimensionality reduction. A projection vector [13] is used to score nodes and only nodes with TopK scores along with related edges are retained. This is an important part of our model design as we perform TopK pooling at every layer to get the optimum embedding for network graph classification. Assuming we have a graph $G$ with vertex set $V(G) = \{a, b, c, d, e, f, g, h, i, j\}$ and edge set $E(G) = \{ab, bc, cd, de, ef, fg, gh, hi, ij\}$, we also assume $G$ is undirected. A visual representation of $G$ is shown in Figure 2. Suppose we want to train a NN model to get optimum embedding that best defines graphs similar to $G$. Applying top-K pooling will cause our graph to drop some nodes, as demonstrated in Figure 2.

It is crucial to note that the nodes carried forward are mathematically and not arbitrarily determined. A measure of how much information is retained after node feature vectors $\mathbf{x}_i$ are projected in the direction of the vector $p$ determines which nodes are dropped and preserved.

## 3 Proposed Model

In this section, we first describe the process used in generating our network traffic graphs then define the model used for classification.

### 3.1 Data Processing

Traffic sessions show bidirectional flows of communication between two or more parties; this makes it a better way of describing packet relationships for classification. Wang et al. demonstrated this [4]. Network communications are processed as packet capture files (pcap or pcapng) for analysis. After collecting packet capture files, we split them into sessions with an open-source tool called splitCap. Packet capture files with *pcapng* extension are converted to *".pcap"* files before splitting occurs. An open-source tool "LibCap" is used to achieve this. The next phase deals with the removal of unwanted information. Network traffic data is collected at the data-link layer that carries information about their physical interfaces. Information contained in the Ethernet header is stripped off as it is not useful for classification and can be spoofed by attackers. Network or flow data like IP addresses and port numbers are not used as features. This restriction forces the model to utilise only the encrypted traffic for classification. At the network layer, the source and destination IP addresses of every packet are masked as they can influence the learning process of the model and can also be easily manipulated by bad actors. At the transport layer, TCP and UDP are used. Since these protocols have different connection orientations with different header sizes (TCP=20 and UDP=8), we pad UDP headers with zeros to match TCP. Since our main aim is to classify encrypted traffic we limit our features to information that can't be easily manipulated. Pcap files are then converted to their raw byte format. The maximum transmission unit (MTU) of a packet is 1500 bytes, to maximize information, we extract all packet info and convert it to raw bytes. Where a packet is not up to the MTU, padding is applied. By doing this, every packet has a length of 1500. Next, we normalise every byte to fall within the range of 0-1 by converting every byte to decimal and dividing by 255. Although we conduct tests with different MTU and session sizes, this initial data processing method is the basis for our experiment.

### 3.2 Model Description

**Graph Data Extraction and Creation** The next phase deals with defining and extracting graph information from the pre-processed files. Since network traffic is collected over time, it can be represented as time series information. Traffic sessions are also represented as a back-and-forth exchange over time. Pang et al. [14] generated chained graph structures to represent traffic sessions. Each packet in a session is modelled as a node or vertex, and the edge shows the chronological (time) relationship between packets. Peng et al. [20] conducted experiments to find the number of packets most effective for traffic identification. Their experiment [20] showed the first five-to-seven packets are optimal for classification. We extend this number to ten and extract the first ten packets per session, we choose ten to ensure sufficient exchange of encrypted application data after the completion of the TLS handshake process which is usually three round trips making six packets in Wireshark. A session with ten packets will be modelled as a graph with ten nodes, the edges are bidirectionally connected

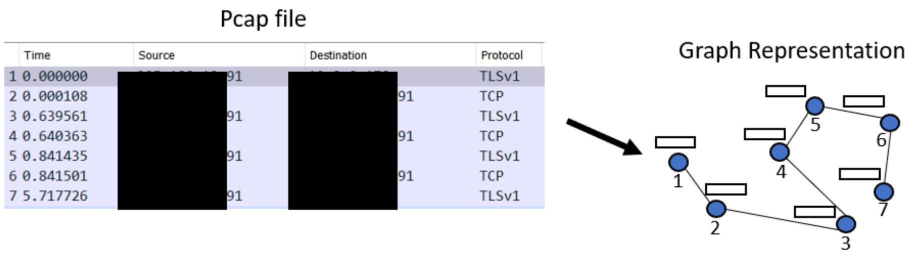


**Fig. 3.** Graph representation of traffic session

to their adjacent preceding and succeeding nodes. The entire session is labelled with the application traffic it conveys. Figure 3 demonstrates the structure of our traffic session graph; our network traffic can be represented mathematically as a path graph. To create network graphs, we extract the following information from the pre-processed pcap files. The extracted information is classified into four files, each file consisting of mappings as follows:

**Node attributes file:** Node information is extracted from the packet, and every node is mapped to its attribute which has an initial feature length of 1500. Nodes represent packets and attributes are packet contents (raw byte).

**Edge file:** The edge information shows the relationship between nodes in a graph. In the edge file, source nodes are mapped to destination nodes. This preserves the chronological relationship of traffic sessions.

**Graph to Label file:** After assigning labels to sessions, we extract this mapping of sessions (graphs) and their corresponding class label.

**Node to Graph file:** As packets belong to particular sessions, we extract this mapping of packets (nodes) and sessions (graphs).

The files are used to create labelled graphs, every graph has nodes with attributes (raw byte) and time series relationships represented by edges.

**Architecture** Our architecture as described in Figure 4, consists of two major parts, the learner and the categorizer. The learner consists of five sub-layers, each sub-layer comprising of a GraphConv or $k-$GNN layer as described by Morris et al, in [21], not be confused with the Graph Convolution layer defined by Kipf et al, in [10]. For simplicity, we refer to the $k-$GNN as GraphConv for the rest of this literature. The next layer is a batch normalisation layer followed by a top-K pooling layer. The aim of the learner is to produce optimum embedding used for classification.

GraphConv [21] is based on a localised higher-order approximation of the Weisfeiler-Leman graph isomorphism heuristic with the propagation rule defined in Equation 8. This is in contrast to the basic convolutional filter which allows weight sharing (by means of a filter with a fixed kernel size sliding over an input). Convolution in spatial domains recognises similar features irrespective of their spatial location. Graphs on the other hand do not have a defined

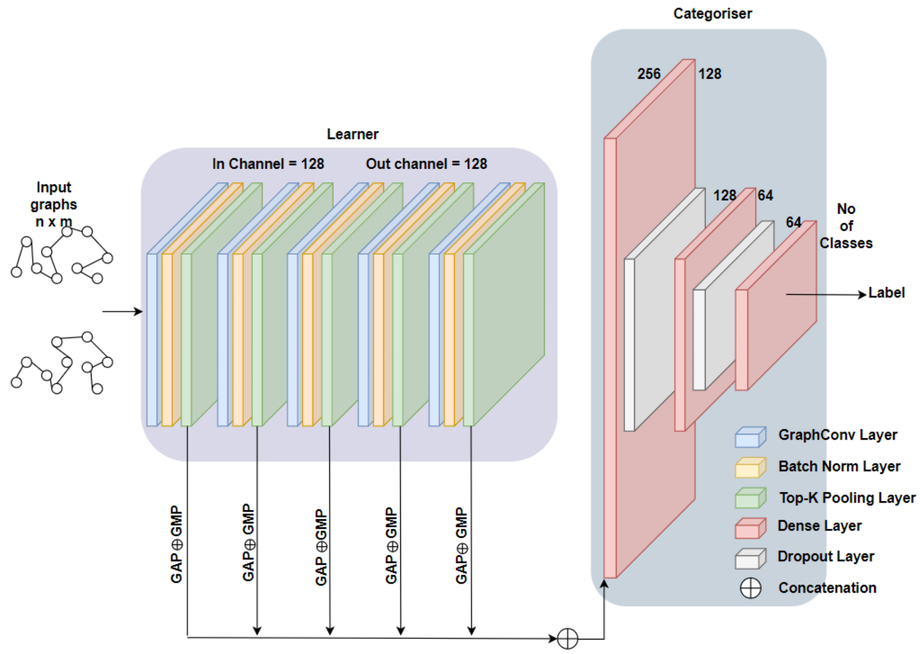


**Fig. 4.** Proposed encrypted network traffic classifier. Our model consists of two major parts that work independently. The learner, which focuses on generating graph embeddings, and the categorizer which classifies the embedding to its label. The learner consists of five sub-layers, each sub-layer with three internal layers (a GraphConv, batch normalisation and top-k polling layer). The graphconv layer leverages sub-graphs rather than nodes information to perform computation, thereby utilising higher order details of graphs capable of preserving structural info. The number 128 above the internal layers signifies its input and output channel size. The input and output channel sizes remain the same for all sub-layers to deal with the issue of information loss. It is important to note that the input graph reduces in size after every sub-layer, the top-K pooling layer makes this possible. Every sub-layer produces an output which is the concatenated form of its global mean pool and global max pool values, $GMP \oplus GAP$. The five sub-layers produce five embeddings which are concatenated and fed to the categorizer. The symbol $\oplus$ between the learner and categorizer signifies the concatenation function. The number behind and in front of the categorizer's dense layer signifies its input and output channel size. The categorizer takes the concatenated embedding from the learner, passes it through a series of dense layers and finally classifies it. The final layer is a softmax classifier which assigns a label to the input graph.

spatial concept. Hence, Kipf et al. [10] utilises spectral graph convolution to lay the mathematical foundation for propagation. Although graph convolution works optimally for node classification, we utilise its potency for the entire graph classification. The model accepts graphs of dimensions $n \times m$, the first GraphConv layer maps the input feature to an output channel of 128 neurons. The selection of this neuron size is based on ease of computation and experiments carried out during this work, literature like [15,14] also use similar sizes. Every GraphConv layer contains 128 neurons. We keep this number constant for all sub-layers in our learner because we extract optimal graph embedding after every sub-layer, since we implement another layer for dimensionality reduction, generating proper embedding becomes key. During our experiments, we realise that a reduction in feature length negatively impacts classification accuracy, a phenomena that occurs in all conducted experiments. This shows that constant reduction of the input and output channel during the learning stage will also impact accuracy. To deal with this issue of loss of information we keep the channel sizes constant and implement dimensionality reduction with a top-K pooling layer. This way, node features defining the graphs are well represented before pooling. The non-linear function used is the Rectified Linear Unit (ReLU).

Following the GraphConv is the batch normalisation (BN) layer. Normalisation is an important aspect of our whole process and it is implemented at the data processing and model training level. We introduce this layer to deal with the internal covariate shift. BN layers standardise the mean and variance of each unit in order to stabilise learning, making the gradients more predictive and increasing convergence time. We utilise BN to achieve a stable distribution of activation values during the training process, hence, it is introduced after the non-linearity. BN layers have also been proven to reduce over-fitting in convolutional NN. Garbin et al. [8] in an experiment found BN layers a better choice to improve accuracy and advised BN layers to be considered as an initial means to deal with over-fitting before other methods.

The next layer is the top-K pooling layer, as described in Section 2.2 the aim of this layer is overall dimensionality reduction of graphs. We employ this method because of the irregular sizes of network packets per session, during data processing we noticed some sessions with over 50 packets. Similar to pooling in CNN, top-K pooling reduces the number of nodes in the input graph every time it is used. The nature of our graphs demands a way to optimise feature propagation for ease of computation. Hence, we apply a top-K pooling layer to make our model robust. We still perform more experiments with the truncated network sessions for impact analysis. The Top-K pooling layer uses a trainable projection vector [13] to select top-K nodes for the next layer. Dimensionality reduction is also implemented to deal with over-smoothness in our model. Oversmoothing is a phenomenon where GNN performance gradually reduces with increasing layers, this occurs when node embedding becomes similar the GNN does not seem to learn anything new. Sadowski et al. [16] demonstrated how dimensionality reduction alleviates this issue.

To get embeddings for graph classification, we compute the global mean pooling and global max pooling after every sub-layer. This embedding is concatenated for every sub-layer of the learner, we end up with five embeddings, one per sub-layer of our learner. The five embeddings are concatenated and fed to the categorizer.

The categorizer consists of two sub-layers and a softmax classifier. The categorizer can be seen as a simple feed-forward NN with three linear or dense layers. The concatenated embeddings from the learner are fed to the categorizer which then predicts the label of the graph based on learned parameters. Since the output of the learner is of dimension 2 × 128, the first sub-layer of the categorizer has an input channel of 256 and an output channel of 128. We introduce a dropout layer with a probability of 0.5 to tackle over-fitting. the second sub-layer has an input channel of 128 and an output channel of 64, we also implement a dropout layer with the same probability here. The final layer has an input channel of 64 and an output channel equal to the total number of labels. Each with a probability that defines how well a graph fits the label, this is achieved with a softmax classifier.

# 4 Evaluation

To validate the robustness of our model, we subject it to a series of tests with a range of datasets. The evaluation process is divided into four sections as follows. The dataset, here we give an in-depth description of the dataset and why we choose it for evaluation. In the experimental methodology section, we explicitly define the processes and types of experiments conducted, the parameters with respect to the model training and testing phase are also defined. In section 4.3 we discuss the results of our experiments and finally compare our results with state-of-the-art.

## 4.1 Datasets

The VPN-nonVPN dataset (ISCXVPN2016) [17] captures real traffic of users Alice and Bob created to use services (applications) described in Table 1. Seven traffic classes are captured for VPN and non-VPN (at the time of dataset collection the P2P class of the non-VPN traffic was removed from the repository). The VPN-nonVPN dataset [17] is one of the most popular datasets for encrypted traffic classification tasks, making it suitable for comparative analysis. Reviewed literature like [4,15] utilised this dataset for their analysis. The dataset is in *.pcap* and *.pcapng* format and 28GB in size.

**Table 1.** VPN-nonVPN Dataset Summary

| **Traffic** | **Content** |
|---|---|
| Web Browsing | Firefox and Chrome |
| Email | SMPTS, POP3S and IMAPS |
| Chat | ICQ, AIM, Skype, Gmailchat, Facebook and Hangouts |
| Streaming | Vimeo, Youtube and Spotify |
| File Transfer | Skype, FTPS and SFTP |
| VoIP | Facebook, Skype and Hangouts voice calls |
| P2P | Vimeo, Youtube and Spotify |

**Table 2.** USTC-TFC2016 Dataset Summary

| Traffic | Content |
|---|---|
| Benign | BitTorrent, Facetime, FTP, Gmail, MySQL, Outlook, Skype, SMB, Weibo, WorldOfWarcraft |
| Malware | Cridex, Geodo, Htbot, Miuref, Neris, Nsis-ay, Shifu, Tinba, Virut, Zeus |

The USTC-TFC2016 dataset [18] consists of two parts, the malware class and benign class. The malware class consists of ten types of malware traffic collected from public websites in a real network environment. The benign class consists of ten types of normal traffic collected using a professional traffic simulation equipment. This dataset is popular for traffic classification tasks and suitable for comparative analysis. The dataset is in pcap format and 3.71GB in size. Table 2 gives a summary of the USTC-TFC2016 dataset.

### 4.2 Experimental Methodology

To explicitly demonstrate our contribution and prove the robustness of our model, we conduct experiments using the datasets defined in Section 4.1. The VPN-nonVPN dataset is split so separate experiments are carried out on the *VPN traffic* and *NonVPN traffic*. For the USTC-TFC dataset, separate experiments are conducted on the *benign* and *malware* traffic respectively. Network graphs are created from traffic sessions. The traffic session distribution for the VPN non-VPN dataset is described in Table 3. Real-world datasets are naturally imbalanced, and classifiers have to incorporate augmentation techniques during training. The USTC-TFC dataset has an abundance of samples, so we extracted 2000 sessions per application.

The different input parameter sizes for our experiments are defined in Table 4. For every input parameter, we process the datasets as required and conduct experiments. For the first experiment, we truncate at the sessions layer, making every session and resulting graph have a fixed number of packets (nodes). For the second experiment, truncation isn't applied, the number of nodes per graph varies as packets per session vary. For experiment three, we truncate the data-link and session layer. We slash the packet size by almost half the MTU, ending up with 784 bytes per packet and also keeping the sessions fixed at 10 packets per session. The choice of 784 is motivated by literature that utilise images from network traffic [6,19] to address the classification problem. Zou et al [6] proved that network images of size 784 bytes are effective for traffic classification, and are more lightweight than 1500 bytes. Wang et al [19] generated network

**Table 3.** VPN-nonVPN dataset sessions sample distribution

| Class | Chat | Email | File | P2P | Stream | VoIP |
|---|---|---|---|---|---|---|
| VPN | 4029 | 298 | 1020 | 477 | 659 | 11985 |
| non-VPN | 6523 | 7312 | 276 | - | 445 | 1781 |

**Table 4.** Input Graph Specifications

| Experiment | Nodes per graph | Node attribute size |
|---|---|---|
| 1 | 10 nodes (fixed) | 1500 |
| 2 | Variable (not fixed) | 1500 |
| 3 | 10 nodes (fixed) | 784 |
| 4 | Variable (not fixed) | 784 |

traffic images and used only the first 784 bytes to get fixed sized input for their CNN model. For experiment four, truncation is applied only at the data-link layer with packet sizes slashed down to 784 bytes.The first experiment lays the foundation for comparative analysis while demonstrating the generalisation strength of our model. Other experiments are conducted for impact analysis of variable length input (data truncation) on encrypted network traffic classification and to demonstrate the model generalisation strength.

**Training Specification** The following specifications are used for developing, training and testing. The PyTorch geometric library with a python 3.9.13 back-end is used to generate the graphs, build, train and test our model. The hardware specification is a Linux dell 5.15.0-56-generic server, the processor is a 12th Gen Intel(R) Core(TM) i9-12900, 125GB of physical RAM and a NVIDIA RTX A4000 GPU.

During the training process, weighted random sampling is implemented to deal with the imbalanced nature of the VPN Non-VPN dataset. Weighted random sampling ensures minority classes are properly represented during the training process. For all experiments, we split our datasets into two, 80% for training and 20% for testing. The hyper-parameters of the model are as follows: batch size is 256, the number of epochs is 500 for regular training and 200 during cross validation. The Adam optimizer is used to improve the categorical cross-entropy loss function with a learning rate of 0.0003, and the decay rate is 0.00001. We used the same training specifications for all experiments. We evaluate our model using the four standard classification metrics namely, Precision, Recall, $f1$Score and Accuracy.

*Test for Over-fitting* The increase in parameters during training can cause neural networks to adapt so much to a particular data that it performs poorly when unseen data is introduced. Ensuring a model does not overfit is a crucial aspect of neural network training. To demonstrate the generalisation of our model, we subject it to a cross-validation test. For this test, we perform stratified cross-validation on our model with the dataset. Our choice of stratified cross-validation is motivated by the imbalanced nature of the dataset, and we need to ensure that classes are well represented during training. We use the same training specification as Section 4.2 with few modifications. We split the data set into 5 folds and reduce the epoch to 200 to limit the tendency of the model adapting to the training data.

### 4.3 Results

A total of 12 experiments are conducted with our model on two datasets. Four experiments on the VPN and four on the non-VPN traffic of the ISCXVPN dataset. Two experiments on the benign and two on the malware traffic of the USTC-TFC dataset. To validate our results, we perform stratified cross-validation with a fold of 5 on the dataset.

Table 5 shows the classification of VPN traffic. Our model yields its best result in the first experiment when truncation is applied at the session layer and padding at the data-link layer. The model generalises optimally for the first three experiments but struggles in the last experiment when truncation is applied only at the data-link layer. For experiments 2 and 4 we use graphs with variable vertex cardinality for classification. The results demonstrate that geometric deep learning, like other DL methods, thrives when the input size is fixed. In situations where the input parameter sizes vary, such as in experiment 4, a compensation to improve accuracy should be well-defined node attributes of an adequate size.

Table 6 shows very similar results to Table 5 for the non-VPN traffic classification task. The best result is achieved in the first experiment. Applying truncation at the data-link layer yields the worst result. Notice how the model struggles to classify the chat and email traffic across all experiments. This is attributed to poor distinction of applications within network sessions. In our work, every session has a label that defines its traffic class. It is possible to have multiple application leveraging the same session or have a case of tunnelling where packets are wrapped inside packets. Our model does not pay attention to this dynamics rather it focuses on distinguishing a particular traffic session by labelling it. The imbalanced nature of the traffic also contributes a great deal to this confusion. Figures 5 and 6 show the confusion matrix for the VPN and Non-VPN traffic of the ISCXVPN dataset.

**Table 5.** VPN Traffic classification result

| **Input Spec** | **Metric** | **Chat** | **Email** | **File** | **Stream** | **P2P** | **VoIP** | Accuracy |
|---|---|---|---|---|---|---|---|---|
| 10nodes/1500 | Precision | 0.9738 | 0.9672 | 0.9069 | 0.8993 | 0.9877 | 0.9850 | 0.9784 |
| | Recall | 0.9595 | 0.9363 | 0.8685 | 0.9921 | 0.9877 | 0.9891 | |
| | $F1$score | 0.9666 | 0.9516 | 0.8873 | 0.9434 | 0.9877 | 0.9871 | |
| v-nodes/1500 | Precision | 0.9831 | 0.8983 | 0.6916 | 0.7484 | 0.9444 | 0.9867 | 0.9437 |
| | Recall | 0.9509 | 0.8413 | 0.9181 | 0.8406 | 0.9533 | 0.9512 | |
| | $F1$score | 0.9409 | 0.8689 | 0.7889 | 0.7918 | 0.9488 | 0.9687 | |
| 10nodes/784 | Precision | 0.9674 | 0.8955 | 0.7103 | 0.8529 | 0.9700 | 0.9838 | 0.9548 |
| | Recall | 0.9601 | 0.9231 | 0.8861 | 0.8722 | 0.9417 | 0.9648 | |
| | $F1$score | 0.9637 | 0.9091 | 0.7885 | 0.8625 | 0.9557 | 0.9742 | |
| v-nodes/784 | Precision | 0.2273 | 0.1908 | 0.2559 | 0.5215 | 0.3333 | 0.7080 | 0.4312 |
| | Recall | 0.4065 | 0.3906 | 0.4645 | 0.6693 | 0.6790 | 0.4168 | |
| | $F1$score | 0.2916 | 0.2564 | 0.3300 | 0.5862 | 0.4472 | 0.5247 | |

**Table 6.** non-VPN Traffic classification result

| **Input Spec** | **Metric** | **Chat** | **Email** | **File** | **Stream** | **VoIP** | Accuracy |
|---|---|---|---|---|---|---|---|
| 10nodes/1500 | Precision | 0.8683 | 0.8389 | 0.9412 | 1.000 | 1.000 | 0.8707 |
| | Recall | 0.8139 | 0.8881 | 0.9412 | 0.9867 | 0.9745 | |
| | $F1$score | 0.8402 | 0.8586 | 0.9412 | 0.9933 | 0.9871 | |
| v-nodes/1500 | Precision | 0.8279 | 0.6483 | 0.6667 | 0.8493 | 0.8697 | 0.7160 |
| | Recall | 0.4345 | 0.9241 | 0.7568 | 0.8732 | 0.8566 | |
| | $F1$score | 0.5699 | 0.7620 | 0.7089 | 0.8611 | 0.8631 | |
| 10nodes/784 | Precision | 0.6823 | 0.5266 | 0.8182 | 0.7719 | 0.8392 | 0.5859 |
| | Recall | 0.2465 | 0.9023 | 0.7660 | 0.7333 | 0.5604 | |
| | $F1$score | 0.3622 | 0.6650 | 0.7912 | 0.7521 | 0.6720 | |
| v-nodes/784 | Precision | 0.5852 | 0.5516 | 0.2400 | 0.4017 | 0.5736 | 0.5435 |
| | Recall | 0.2469 | 0.7662 | 0.6000 | 0.6026 | 0.6632 | |
| | $F1$score | 0.3473 | 0.6414 | 0.3429 | 0.4821 | 0.6151 | |

Figure 7 shows the confusion matrix for the benign class of the USTC-TFC dataset. With a balanced dataset, our model perfectly classifies six applications for the first experiment and classifies five applications perfectly when the feature vector is slashed by almost half. As demonstrated in Figure 8 our model shows powerful generalisation strength for the malware class of the USTC-TFC dataset. We get very similar results for both experiments regardless of data truncation. This time we test to see the best feature length for optimum generalisation. Hence, we keep the nodes length fixed at 10 while varying the attribute size. Our model maintains an accuracy of over 90% after we slash the feature vector by almost half. For other evaluation metrics, our model again maintains accuracies of over 90%. The experiments proved that an increase in features size while keeping vertex cardinality fixed, improves generalisation strength. Table 7 shows the overall performance of our model after all 12 experiments are carried out.

### 4.4 Performance comparison

We compare our best results with state-of-the-art results that use the same dataset and a similar data processing approach. For VPN classification, our

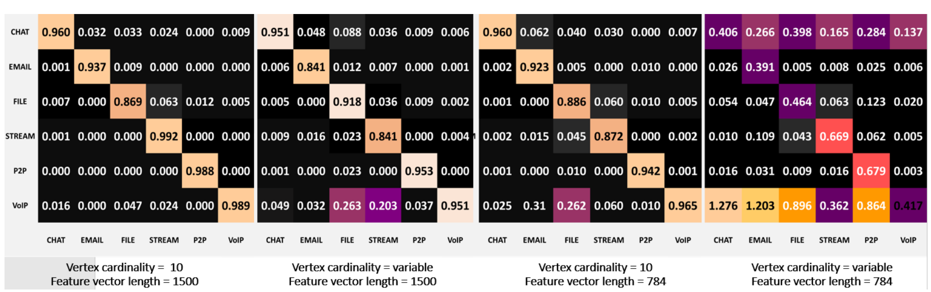


**Fig. 5.** Confusion matrix for VPN dataset

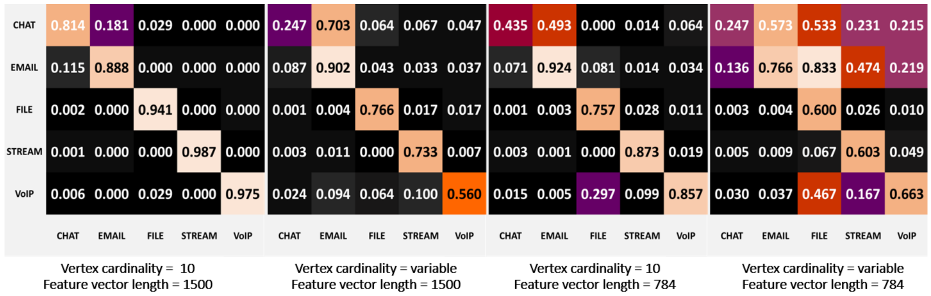


**Fig. 6.** Confusion matrix for non-VPN dataset

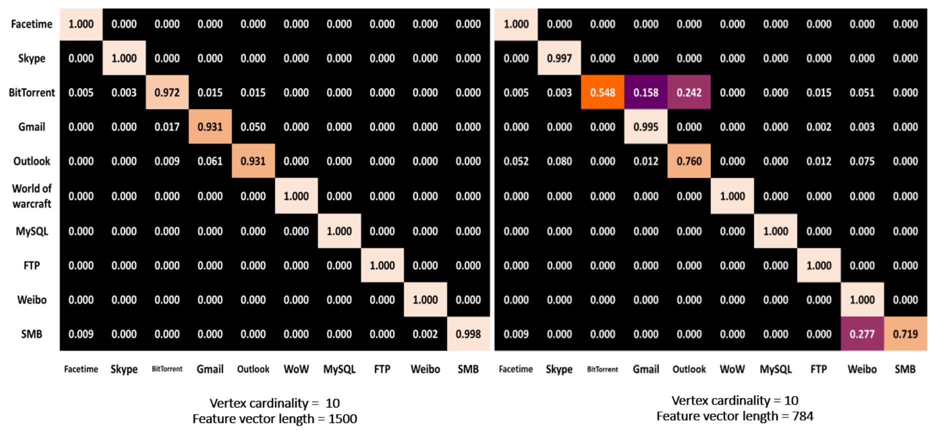


**Fig. 7.** Confusion matrix for USTC-TFC Benign Traffic

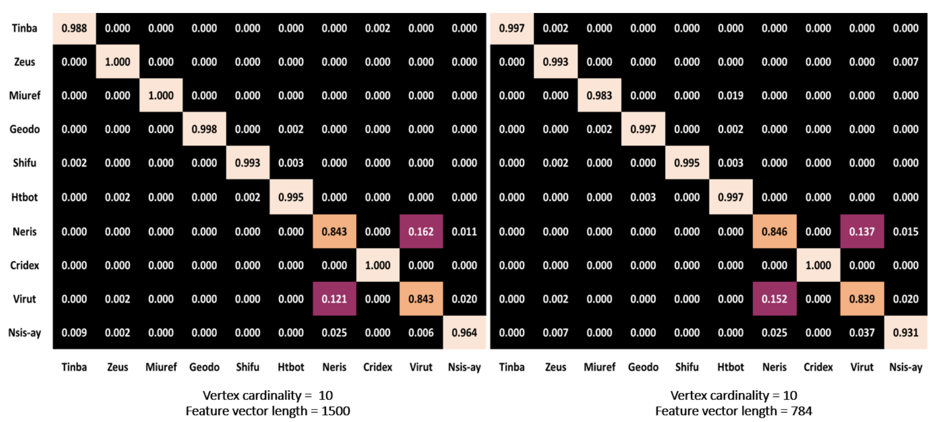


**Fig. 8.** Confusion matrix for USTC-TFC Malware Traffic

**Table 7.** Overall Performance Summary

| Dataset | Input Spec | Precision | Recall | $F1$Score | Accuracy |
|---|---|---|---|---|---|
| VPN | 10nodes/1500 | 0.9749 | 0.9748 | 0.9747 | 0.9748 |
| | 10nodes/784 | 0.9586 | 0.9548 | 0.9561 | 0.9548 |
| | v-nodes/1500 | 0.9494 | 0.9437 | 0.9455 | 0.9437 |
| | v-nodes/784 | 0.5543 | 0.4312 | 0.4587 | 0.4312 |
| non-VPN | 10nodes/1500 | 0.8722 | 0.8707 | 0.8706 | 0.8707 |
| | 10nodes/784 | 0.6398 | 0.5859 | 0.5466 | 0.5859 |
| | v-nodes/1500 | 0.7505 | 0.7160 | 0.6978 | 0.7160 |
| | v-nodes/784 | 0.5588 | 0.5434 | 0.5144 | 0.5435 |
| Benign | 10nodes/1500 | 0.9830 | 0.9831 | 0.9831 | 0.9830 |
| | 10nodes/784 | 0.9177 | 0.9030 | 0.8983 | 0.9030 |
| Malware | 10nodes/1500 | 0.9655 | 0.9655 | 0.9655 | 0.9655 |
| | 10nodes/784 | 0.6578 | 0.9573 | 0.9575 | 0.9573 |

model generalises optimally with approximately the same value across all evaluation metrics. Our model maintains an accuracy that doesn't fall below 97%, an occurrence not demonstrated by state-of-the-art models. The results of compared literature remained unstable for different evaluation metrics.

**Table 8.** Performance Comparison

| Dataset | Metric | DeepPacket (GAE)[5] | Gil et al. [17] | Song et al. [23] | 1DCNN [4] | Huoh et al. [15] | k-NN. [24] | Our work |
|---|---|---|---|---|---|---|---|---|
| VPN | Precision | 0.97 | 0.89 | 95.2 | 0.949 | 0.913 | - | **0.9748** |
| | Recall | 0.80 | - | 97.2 | 0.973 | 0.940 | - | **0.9784** |
| | $F1$Score | 0.97 | - | 96.1 | - | 0.926 | - | **0.9747** |
| | Accuracy | - | - | - | - | - | 0.94 | **0.9748** |
| non-VPN | Precision | 0.87 | **0.906** | 87.6 | 0.855 | 0.882 | - | 0.8722 |
| | Recall | **0.88** | - | 87.3 | 0.858 | 0.866 | - | 0.8707 |
| | $F1$Score | 0.87 | - | 87.5 | - | 0.871 | - | **0.8706** |
| | Accuracy | - | - | | - | - | | 0.8707 |

For the task of non-VPN classification, we obtain similar results to the compared literature. Our model still demonstrates optimum generalisation strength producing accuracies that do not fall below 87%. Gil et al. [17] using flow-based features got a precision score of 90.6% on the non-VPN dataset. While this result is impressive, applications share similar flow-based features which can impact classification. Huoh et al. [15] GNN-based classifier performed well for both tasks; their model does not generalise optimally for VPN classification. Comparatively, Song et al. [23] achieved the closest accuracy to our work; they used text-based CNN for classification. This closeness in results can be attributed to "sense of locality". GNNs and CNNs utilise locality relationships for the classification tasks. While GNNs can be built with higher order capability to perform better, results from both methods should not be far apart. Most state-of-the-art models avoid the use of accuracy as a metric for evaluation. This is a normal occurrence when the utilised dataset is imbalanced. The accuracy of classifier on imbalanced dataset does not depict the true nature of the model's performance as minority classes can negatively impact accuracy. For our work, we consider all four evaluation metric (including accuracy) as in the real world, dataset is mostly imbalanced.

## 5 Related Work

Deep learning methods have been immensely applied to the task of traffic classification, recurrent(RNN) and convolutional neural networks (CNN) [7] account for the most usage. In analysing network traffic, the data processing stage is crucial to the overall process. A common practice by state-of-the-art literature [5,4,6] is utilising raw packet bytes for classification. Extracted byte information is processed to have a fixed length as this is a requirement for most deep learning approaches. Data truncation is applied, and inputs are fed to the network for classification. The model is left with the task of making sense of the truncated input. While reviewed state-of-the-art models provide good accuracy, it is important to note that generalisation isn't guaranteed as information lost during data processing is necessary for classification.

Geometric deep learning is an emerging concept that generalise neural networks to non-euclidean domains. Graphs are a visual way to demonstrate relationships between two or more entities. Since network data exists in non-euclidean space, they can be represented as graphs. Unlike RNN and CNN, GNN can perform computation on variable-sized inputs. Huoh et al. [15] applied GNN to traffic classification, they conduct a range of experiments which produced good results. when they use raw bytes for classification, they experience significant drop in accuracy. Their design does not utilise structural information necessary for graph classification. GNNs rely on message passing to propagate information across nodes, this propagation method performs well for the task of node classification. To classify the entire graph, a stronger propagation function is necessary. Shen et al [22] used GNN to distinguish decentralised application. They design simple traffic dispersion graphs representing applications then use GNN for classification. They achieve promising results, attributed to the simplistic nature of their graphs. When network traffic sessions are modelled as variable sized inputs, the tendency of having graphs with huge vertex cardinality increases. In such scenarios, the task of graph classification demands higher order GNNs that generate optimum embedding for graph classification.

A process not demonstrated by the reviewed literature is defining the impact of lost information on a model's performance. The practice of truncating network packet data leads to loss of classification information, discarding information impacts the generalisation strength of classifiers and should be analysed before implemented. Literature that utilise GNN for the task of classification failed to utilise the potential of graphs by taking advantage of structural details for graph classification. The imbalance nature of dataset should always be taken into consideration during training to produce model that generalise optimally.

## 6 Conclusion

This work presents a higher order GNN for the task of encrypted traffic classification. We build on theoretical foundations that relates the short fall of basic GNN to that of the Weisfeiler-Leman graph isomorphism. We observe that traditional machine learning and deep learning based methods demands fixed sized input for classification. This requirement causes information discarding during data processing leading to ordinary representations of traffic data. As graphs are a powerful way to demonstrate relationships between entities, we model traffic session as graphs. Nodes represents packets and edges define chronological relationship between nodes. We harness this expressive nature of graphs to model network traffic with different input specifications, defining a baseline data processing method that conforms to state-of-the-art. The selected datasets are based on popularity within the research domain, and their suitability for comparative analysis. Our Proposed model which consist of two major parts is capable of preserving vertex and structural information of graphs suitable for classification. In training our model we consider the imbalance nature of network data and define ways to curb its ripple effect. When compared to state-of-the art, our model demonstrates optimum generalisation strength on all dataset across all experiment conducted. We conduct further test to determine the impact of truncation on traffic classification. Based on the evaluation metric from series of test, our model's overall aim was achieved. Higher order graphs proved suitable for the task of encrypted network traffic classification. In improving the current work, the focus needs to shift from the classification stage to feature extraction stage. To harness the full potential of GNN, fined grained graph structures need to be developed, as graphs can be spectral or spatial in nature. A hybrid of both methods for feature representation can aid classification. An extension of the current method to restricted network contexts, such as IoT, IIoT or CPS is important. Finally, unknown traffic classification should be explored.

## References

1. SonicWall 2022 SonicWall Cyber Threat Report: Cyberattacks Climb Due To Seismic Shift In Geopolitical Landscape. (SonicWall,2022)
2. Zhang, J., Chen, X., Xiang, Y., Zhou, W. & Wu, J. Robust network traffic classification. *IEEE/ACM Transactions On Networking*. **23**, 1257-1270 (2014)
3. Google Google Transparency Report, HTTPS encryption on the web. *Https://transparencyreport.google.com/https/overview?hl=en.* (2023)
4. Wang, W., Zhu, M., Wang, J., Zeng, X. & Yang, Z. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. *IEEE International Conference On Intelligence And Security Informatics (ISI)*. pp. 43-48 (2017)
5. Lotfollahi, M., Jafari Siavoshani, M., Shirali Hossein Zade, R. & Saberian, M. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*. **24**, 1999-2012 (2020)
6. Zou, Z., Ge, J., Zheng, H., Wu, Y., Han, C. & Yao, Z. Encrypted traffic classification with a convolutional long short-term memory neural network. *IEEE 20th International Conference On High Performance Computing And Communications; IEEE*

*16th International Conference On Smart City; IEEE 4th International Conference On Data Science And Systems (HPCC/SmartCity/DSS)*. pp. 329-334 (2018)
7. Okonkwo, Z., Foo, E., Li, Q. & Hou, Z. A cnn based encrypted network traffic classifier. *Australasian Computer Science Week 2022*. pp. 74-83 (2022)
8. Garbin, C., Zhu, X. & Marques, O. Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia Tools And Applications*. **79**, 12777-12815 (2020)
9. Bronstein, M., Bruna, J., LeCun, Y., Szlam, A. & Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*. **34**, 18-42 (2017)
10. Kipf, T. & Welling, M. Semi-supervised classification with graph convolutional networks. *ArXiv Preprint ArXiv:1609.02907*. (2016)
11. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings Of The IEEE*. **86**, 2278-2324 (1998)
12. LeCun, Y., Kavukcuoglu, K. & Farabet, C. Convolutional networks and applications in vision. *Proceedings Of 2010 IEEE International Symposium On Circuits And Systems*. pp. 253-256 (2010)
13. Gao, H. & Ji, S. Graph u-nets. *International Conference On Machine Learning*. pp. 2083-2092 (2019)
14. Pang, B., Fu, Y., Ren, S., Wang, Y., Liao, Q. & Jia, Y. CGNN: Traffic Classification with Graph Neural Network. *ArXiv Preprint ArXiv:2110.09726*. (2021)
15. Huoh, T., Luo, Y., Li, P. & Zhang, T. Flow-based Encrypted Network Traffic Classification with Graph Neural Networks. *IEEE Transactions On Network And Service Management*. (2022)
16. Sadowski, K., Szarmach, M. & Mattia, E. Dimensionality reduction meets message passing for graph node embeddings. *ArXiv Preprint ArXiv:2202.00408*. (2022)
17. Draper-Gil, G., Lashkari, A., Mamun, M. & Ghorbani, A. Characterization of encrypted and vpn traffic using time-related. *The 2nd International Conference On Information Systems Security And Privacy (ICISSP)*. pp. 407-414 (2016)
18. CTU-University The Stratosphere IPS Project Dataset. *Https://stratosphereips.org/category/dataset.html,*. (2016)
19. Wang, W., Zhu, M., Zeng, X., Ye, X. & Sheng, Y. Malware traffic classification using convolutional neural network for representation learning. *2017 International Conference On Information Networking (ICOIN)*. pp. 712-717 (2017)
20. Peng, L., Yang, B., Chen, Y. & Wu, T. How many packets are most effective for early stage traffic identification: An experimental study. *China Communications*. **11**, 183-193 (2014)
21. Morris, C., Ritzert, M., Fey, M., Hamilton, W., Lenssen, J., Rattan, G. & Grohe, M. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. *CoRR*. **abs/1810.02244** (2018), http://arxiv.org/abs/1810.02244
22. Shen, M., Zhang, J., Zhu, L., Xu, K. & Du, X. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Transactions On Information Forensics And Security*. **16** pp. 2367-2380 (2021)
23. Song, M., Ran, J. & Li, S. Encrypted traffic classification based on text convolution neural networks. *2019 IEEE 7th International Conference On Computer Science And Network Technology (ICCSNT)*. pp. 432-436 (2019)
24. Yamansavascilar, B., Guvensan, M., Yavuz, A. & Karsligil, M. Application identification via network traffic classification. *2017 International Conference On Computing, Networking And Communications (ICNC)*. pp. 843-848 (2017)