Expert Systems With Applications 241 (2024) 122564

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

# FlowTransformer: A transformer framework for flow-based network intrusion detection systems

Liam Daly Manocchio, Siamak Layeghy *, Wai Weng Lo, Gayan K. Kulatilleke, Mohanad Sarhan, Marius Portmann

*School of ITEE, The University of Queensland, Brisbane, Australia*

ARTICLE INFO

*Keywords:*
Transformers
Network intrusion detection system (NIDS)
Machine learning (ML)
Generative pre-trained transformer (GPT)
Network flow

ABSTRACT

This paper presents the FlowTransformer framework, a novel approach for implementing transformer-based Network Intrusion Detection Systems (NIDSs). FlowTransformer leverages the strengths of transformer models in identifying the long-term behaviour and characteristics of networks, which are often overlooked by most existing NIDSs. By capturing these complex patterns in network traffic, FlowTransformer offers a flexible and efficient tool for researchers and practitioners in the cybersecurity community who are seeking to implement NIDSs using transformer-based models. FlowTransformer allows the direct substitution of various transformer components, including the input encoding, transformer, classification head, and the evaluation of these across any flow-based network dataset. To demonstrate the effectiveness and efficiency of the FlowTransformer framework, we utilise it to provide an extensive evaluation of various common transformer architectures, such as GPT 2.0 and BERT, on three commonly used public NIDS benchmark datasets. We provide results for accuracy, model size and speed. A key finding of our evaluation is that the choice of classification head has the most significant impact on the model performance. Surprisingly, Global Average Pooling, which is commonly used in text classification, performs very poorly in the context of NIDS. In addition, we show that model size can be reduced by over 50%, and inference and training times improved, with no loss of accuracy, by making specific choices of input encoding and classification head instead of other commonly used alternatives.

## 1. Introduction

ChatGPT (OpenAI, 2022) has seen an explosion in use, in academic, private, and professional domains, for its ability to answer complex prompts and generate high-quality human-like text. ChatGPT, like many other large language models, is based on a transformer architecture. Transformer architectures (Vaswani et al., 2017) are extremely powerful for natural language processing (NLP) tasks due to their ability to capture long-range dependencies and relationships between different elements of a sequence, without requiring prior domain-specific knowledge or feature engineering. Although initially designed for NLP, transformer architectures have proven to be versatile and powerful tools for capturing complex patterns and relationships in various types of sequential data, including but not limited to image, graph, and speech (Parmar et al., 2018; Vaswani et al., 2017; Yun et al., 2019). This adaptability has made transformer-based models particularly attractive for use in machine learning (ML)-based Network Intrusion Detection Systems (NIDSs), where data is captured as sequences of packets or flows, and where the ability to identify subtle and complex patterns in this traffic is critical for NIDS performance.

Despite the sequential nature of network communications, current ML-based NIDS research often overlooks sequential data (Kumar et al., 2021; Singh & Khare, 2022; Walling & Lodh, 2022), focusing instead on classifying individual network flow records in isolation. This is partly due to the challenges posed when trying to apply traditional RNNs due to their inability to be parallelised (Vaswani et al., 2017). Transformer architectures, on the other hand, provide an effective means of applying machine learning to a sequence of data, that can be parallelised (Vaswani et al., 2017). Transformers have been shown to have faster training times and increased model performance than traditional recurrent approaches (Vaswani et al., 2017). In addition, transformers are able to support both long and short term dependencies (Vaswani et al., 2017), and there are prominent attacks in networks

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
*E-mail addresses:* liam@riftcs.com (L.D. Manocchio), siamak.layeghy@uq.net.au (S. Layeghy), w.w.lo@uq.net.au (W.W. Lo), g.kulatilleke@uqconnect.edu.au (G.K. Kulatilleke), m.sarhan@uq.net.au (M. Sarhan), marius@ieee.org (M. Portmann).

https://doi.org/10.1016/j.eswa.2023.122564
Received 3 July 2023; Received in revised form 8 November 2023; Accepted 10 November 2023
Available online 23 November 2023
0957-4174/

that require considering the long-term behaviour and characteristics of the network to detect, such as SlowLoris attacks (Sikora et al., 2019). The transformers' ability to access long-range information (Vaswani et al., 2017) can help detect complex patterns of network traffic that may indicate attacks, even if they are distributed over an extended period of time. Therefore, we believe there is a great potential in investigating transformer architectures such as GPT (OpenAI, 2019) and BERT (Devlin et al., 2018), or smaller transformers, in the context of NIDS.

Applying transformers in the networking domain is not as straightforward as in the natural language domain. Although established architectures exist for handling text data with transformers, this is not the case for network traffic. There are several critical decisions that must be made, independently of the transformer model itself, both to ingest network data, as well as produce a classification from the transformer output. To the best of our knowledge, there are no works that provide an extensive and systematic evaluation of transformer models and parameters in the context of NIDS. To address this gap, this paper proposes 'FlowTransformer', a framework that allows comprehensive evaluation of transformer-based NIDSs, by enabling the efficient interchange of key components, such as input encoding, transformer model and classification head. We choose to use flow-based network data for this paper for several reasons. Firstly, flow records aggregate network data and provide a compact representation of network communication between two endpoints. This reduces the volume of data that needs to be analysed, making it more scalable for NIDS to process large amounts of network traffic. The majority of medium to large networks (10 Gbps+) are of a scale where packet captures are impractical or impossible (Schultz & Crowley, 2012). Furthermore, widely utilised flow-based traffic collectors are capable of handling such volumes even on commodity hardware (Schultz & Crowley, 2012), making it a practical data format for NIDSs.

The key contributions of this paper are that we propose the FlowTransformer framework, to address the challenges, including dataset ingestion and processing, choice of input encoding, model hyperparameters, classification heads, and evaluation over sequential datasets, associated with implementing transformer-based NIDSs in a systematic manner, and provide the implementation for public use.[1] It provides researchers the flexibility to select the most appropriate hyperparameters and architecture configurations that align with the requirements of NIDS applications. FlowTransformer provides a systematic evaluation methodology to assess the performance of different transformer models in the context of NIDS. It takes into account model components, hyperparameter configurations, and their impact on model size, speed, and accuracy, allowing for a comprehensive analysis of various transformer architectures in NIDS use cases. Our analysis highlights the trade-offs between model size, performance, and inference time in network intrusion detection tasks, and we demonstrate an over 50% reduction in model size and a reduction in inference time for the same accuracy when using a sensible choice of transformer components versus other common approaches. Finally, as a result of this comparative analysis, we recommend an overall transformer architecture that we believe represents the best choice of components as a starting point for future transformer-based NIDS research.

## 2. Background - NetFlow

Network traffic monitoring is a critical aspect of network security and management. The two primary approaches for this purpose include packet-based and flow-based monitoring. Packet-based monitoring involves capturing both packet headers and payloads as they traverse the network, while flow-based monitoring collects summary information based on a sequence of packets between two endpoints. However, due to its resource-intensive nature, continuous packet-based monitoring is difficult to implement in large-scale networks. Furthermore, packet capture raises privacy concerns, as it may collect sensitive information. In contrast, flow-based monitoring provides a highly compressed summary of network traffic, making it a more scalable alternative. It is widely used in large-scale networks, and numerous tools are available for flow-based traffic exporting and collection.

NetFlow is a widely adopted flow-based network traffic information collection and monitoring protocol developed by Cisco (Delgadillo & Marketing, 1996). It operates by consolidating a series of packets within a communication sequence, either unidirectional or bidirectional, that share similar characteristics, such as the same source and destination IP, source, and destination port, and transport protocol. Bidirectional NetFlow is equipped to capture the number of packets and bytes in both directions, along with other features.

Although in this work we use NetFlow version of benchmark NIDS datasets, there are other flow standards, such as IPFix (Aitken et al., 2013) or SFlow (Phaal et al., 2001), and these flow formats are also natively supported by the FlowTransformer implementation.

## 3. Related works

This work considers related works that also focus on transformer or RNN based approaches to NIDS. This is because unlike traditional ML-based NIDS that act on single flow, these approaches handle sequences of flows.

Wu et al. (2022) present a Transformer-based Intrusion Detection System called RTIDS, which incorporates the positional embedding technique to associate sequential information between features. To train and evaluate the model, a variant stacked Transformer encoder–decoder neural network has been utilised. The approach has been evaluated using the CICDDoS2019 dataset, and its performance has been compared to baseline machine learning models, including support vector machines (SVM), as well as deep learning algorithms such as recurrent neural network (RNN), fuzzy neural network (FNN), and long short-term memory (LSTM).

Yangmin et al. (Li et al., 2022) present a novel approach, the Extreme Semi-Supervised Framework based on Transformer (ESeT), for network intrusion detection. This framework utilises a small amount of labelled data to achieve superior detection performance. ESeT includes a multi-level feature extraction module and a semi-supervised learning module, which incorporates a dual-encoding transformer, credibility selector, and feature augmentor. The efficacy of the proposed framework is evaluated on two large, real-world NIDS datasets, and the results demonstrate improved performance compared to existing state-of-the-art methods with only a limited quantity of labelled data.

Wei et al. (Wang et al., 2023) propose a self-supervised-based network intrusion detection system based on transformer. They use a transformer-based architecture and a masked context reconstruction module to detect intrusions. The authors evaluate the algorithm on three different datasets: KDD, UNSW-NB15, CICIDS-17 and also investigate the impact of different parameters of the proposed model, such as the mask ratio and context size, on the algorithm's performance. They show that the algorithm's performance is sensitive to the mask ratio but relatively insensitive to the context size.

Nam et al. (2021) present a novel method for detecting spoofing attacks in Controller Area Networks (CAN) by training a transformer-based language model (GPT) on normal CAN ID sequences. The proposed method outperforms other state-of-the-art methods in terms of accuracy and efficiency. However, the proposed method assumes that the attacker does not have access to the normal CAN ID sequences, which may not be a realistic assumption in practice.

Loc et al. (Nguyen & Watabe, 2022) propose a method to improve the domain adaptation capability of Network Intrusion Detection Systems (NIDS) by employing the Bidirectional Encoder Representations

[1] FlowTransformer source code https://github.com/liamdm/FlowTransformer.

from Transformers (BERT). The proposed method uses sequences of flows to overcome the limitation of modelling the distribution of features within a flow. The BERT framework is used to utilise the context information from a sequence of flows, allowing the classifier to further model the distribution of a flow in relation to other flows. The BERT model is pre-trained with only the Masked Language Modeling (MLM) task and is fine-tuned with a linear layer with softmax output for detecting intrusion.

Han et al. (2023) present a new intrusion detection method for encrypted traffic called GTID. It combines n-gram frequency and time-aware transformer methods to address the limitations of deep learning-based and n-gram-based methods. The model processes packet header and payload features separately, uses n-gram frequency to handle variable-length sessions and packets, and incorporates a time-aware transformer to consider the time intervals between packets. However, the performance of GTID varies with the proportion of encrypted traffic on the dataset, and the method is less effective in detecting encrypted traffic with smaller payload sizes. Also, the use of n-grams results in higher computational complexity.

Mohammad et al. (Hassan et al., 2019) propose a NIDS (Network Intrusion Detection System) that combines Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) architectures using deep learning techniques. The CNN layers are utilised to efficiently extract significant features from network data because of their weight-sharing property, resulting in faster processing speed. The LSTM, on the other hand, maintains the long-term temporal relationship between the extracted data features. The hyper-parameters of the system were optimised through a trial and error process. The hybrid model achieved an overall classification accuracy of 97.1% for binary classification and 98.43% for the multi-class case when tested on the UNSW-NB15 dataset.

In terms of related works leveraging transformers, the concept of a transformer-based Intrusion Detection System (RTIDS) is introduced in (Wu et al., 2022), employing position embeddings to link sequence information among features and self-attention for learning low-dimensional representations. The effectiveness of their approach, of low-dimensional encoder representations being fed into the decoder, then transformed back into high dimensions, was examined in (Liu & Wu, 2023) and shown to be ineffective. Visual Transformer (ViT) utilises a sliding window mechanism for local feature modelling and a hierarchical focal loss function to tackle data imbalance, as outlined in (Yang et al., 2022). While ViT applies focal loss with weight modifications to address data imbalance, it is acknowledged in (Liu & Wu, 2023) that this does not fully counteract quantitative data imbalance. A fusion of CNN and transformer, called CNN-Transformer, is introduced, capturing both local and global information in (Zhang & Wang, 2022). In (Ullah et al., 2023), transformer-based transfer learning is employed alongside a hybrid CNN–Long Short-Term Memory (CNN–LSTM) architecture to learn network feature representations. However, CNN–LSTM's high computational cost is noted. (Liu & Wu, 2023) employs stacked auto-encoders to reduce data dimensionality, utilises under-sampling of normal samples via KNN, and employs hybrid sampling with Borderline-SMOTE for over-sampling of abnormal samples in a transformer encoder with position encoding for binary classification. We include the reported classification accuracy of these works as comparisons to our FlowTransformer-based model in our Results section.

In contrast to existing works, which propose individual models, our paper proposes the FlowTransformer framework, and its use in the systematic and comprehensive evaluation of various transformers models in the NIDS domain, systematically exploring the impact of various model components and configurations on performance, model size, and inference time.

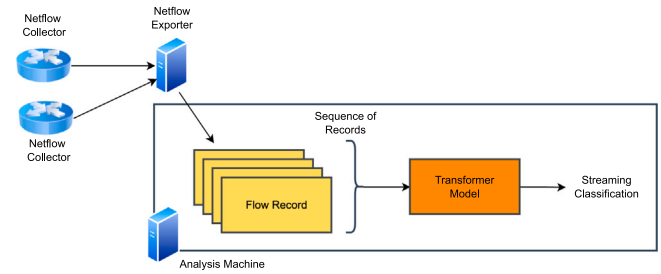


**Fig. 1.** A simple framework for online NetFlow processing using a pre-trained transformer model.

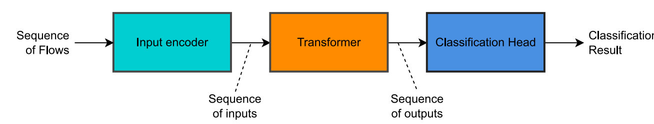


**Fig. 2.** Architecture of a transformer-based NIDS consisting of the input encoding, transformer and classification head.

## 4. Transformers for NIDS

As discussed, there has been limited investigation into NIDS that can effectively handle sequences of traffic flows. Transformers provide an attractive option for researchers, given their inherent ability to process sequences and capture complex relationships between items within a sequence. In this context, we will begin by discussing the main components of a transformer-based system when applied to the NIDS task.

Fig. 1 presents a potential framework that employs a transformer-based approach for NetFlow processing. This approach involves using a transformer to analyse sequences of network data. Specifically, the flow records aggregated by a NetFlow exporter are fed directly into a transformer model, which provides classifications for each of these records.

Using a transformer-based approach offers several benefits for this framework. Firstly, it enables the analysis of network data in a highly efficient and scalable manner, given that transformers support parallel execution, unlike many other methods for processing sequential data. Additionally, the transformer model can effectively capture the complex relationships between different network flows, making it ideal for use in NetFlow processing.

However, choosing the right transformer model is crucial. There are several transformer models available, each with unique strengths and weaknesses It is important to consider these differences when selecting a suitable transformer model in the context of network intrusion detection.

Transformers are a diverse class of machine learning models that can perform various tasks. For classification or anomaly detection, which are fundamental tasks in NIDS, the basic transformer architecture comprises three components, as illustrated in Fig. 2. These components include an input encoding module that transforms the input data into a format suitable for use by a transformer, a transformer composed of a number of transformer blocks, and an output head that converts the sequential output of the transformer into a single classification result. Each of these steps involves design choices that can significantly impact the model's overall performance. The best choices of these for flow-based data, in contrast to other domains, remains largely un-evaluated. In this section, we provide a brief explanation of these components within the context of NIDS. Later, we will discuss the options for each of these stages, as implemented in our framework.

### 4.1. Input encoding

Input encoding for classification is a task similar to tokenisation in natural language processing. In NLP, input encoding involves taking written text as input and transforming it into a sequence of fixed-length vectors that can be processed by a transformer model. For example, the GPT series of models used a variant of byte pair encoding to do this (Shibata et al., 1999). In the context of NIDS, the objective is to transform network flows into a format that can be ingested by a transformer model. This is distinct from pre-processing, as it is a part of the model itself, and the encoding is learned during the model training process. Although raw data can be provided to a transformer model directly after pre-processing, most transformer implementations have their own input encoding step within the model (Vaswani et al., 2017). There are various approaches to input encoding discussed in the literature (Wolf et al., 2020), and these will be further explored in Section 5.

### 4.2. Transformer blocks

Transformers consist of a sequence of blocks, each of which performs a single 'transformation' on the input sequences. These can be encoder or decoder blocks. The encoder block in a transformer aims to ingest an input and transform this into a fixed-length feature representation. This representation captures the semantic meaning of the particular input, considering its relation to the other inputs in the sequence. In the NIDS domain, the encoder would transform each flow into a fixed-length feature vector. Decoder blocks are the reverse of an encoder block, and they are more commonly utilised for generational tasks. Decoders take in a sequence of feature representations, and then produce an output sequence. In the NIDS domain, this would take a flow's feature vector, and then produce a raw flow record.

Traditional transformers initially used a stack of encoders, followed by a stack of decoders. Traditional transformers used a stack of encoders followed by decoders (Vaswani et al., 2017), an approach that makes particular sense for sequence to sequence tasks such as translation. But for non-sequential tasks like NIDS classification, the decoder layers can be removed and replaced with a classification head, creating a model made entirely of encoder blocks. BERT is an example of such a model that considers tokens in both directions and solves various tasks. On the other hand, generative transformer models like GPT exclusively use decoder blocks.

### 4.3. Classification head

Transformers are sequence-to-sequence models, but for NIDS, a classification output is needed. The classification head takes the output sequence of tokens from the transformer and converts it into a prediction for one or more classes. However, a dimensionality problem arises when passing the transformer output sequence to a dense neural network for classification, which can exponentially increase the number of parameters as the sequence length increases. Classification heads aim to summarise the information returned from the transformer, so they do not incur the same dimensionality increase. There has been recent work proposing new classification heads such as (Ridnik et al., 2023), which leverage attention mechanisms to process the output of the transformer, however, many traditional approaches used a simple Global Average Pooling approach on the output of the transformer (Ko et al., 2022). Despite the fact that several other basic approaches to classification heads exist, these remain largely unevaluated in the NIDS domain.

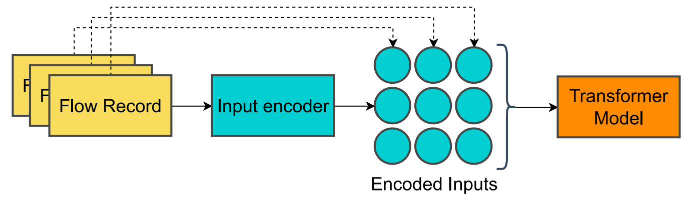


**Fig. 3.** In the NIDS domain, pre-processed flow records are transformed by the input encoder into a fixed length vector, before being passed to the transformer.

## 5. FlowTransformer framework

FlowTransformer is a framework for implementing, testing and evaluating transformer-based NIDSs. The framework provides researchers with several common implementations for each of the transformer components, forming a pipeline for rapid transformer development that can be directly applied to flow-based networking data. One of the key advantages of FlowTransformer is its ability to eliminate the difficulties of choosing input encodings and classification heads, which can significantly speed up the research effort. This is particularly important in the NIDS domain, where the ability to practically test certain architectures against a wide range of datasets is paramount.

In the following sections, we will delve into the various components of FlowTransformer, including pre-processing techniques, input encodings, transformer blocks, and classification heads. We will also discuss how FlowTransformer can be extended to implement new components, enabling researchers to develop and test innovative NIDS architectures with ease.

### 5.1. Input encoding options

In the context of flow-based NIDS, input encoding is a critical process that allows for the transformation of flow data into a fixed-length feature representation. As shown in Fig. 3 the input encoder is responsible for converting each flow into a feature vector that can be effectively processed by the transformer. Flow data is a type of tabular data, where the fields for each flow are known and fixed. This data consists of two types of fields: numerical fields, such as the number of packets sent, where the distance between numbers has contextual meaning, and categorical fields, such as port number, which have a discrete set of values referred to as levels. With categorical fields, there is no guarantee that the value of a level bears any contextual relevance to other levels. For example, port 25 and 22 are very close but represent distinct protocols, while ports 443 and 8080 are often used for the same service despite their numerical distance. The input encoder must be designed to handle these different types of fields appropriately, ensuring that the resulting feature vectors capture the relevant information for the transformer to use.

When it comes to transformer-based NIDSs, there are two primary approaches for handling tabular data. One common method, employed by previous works such as "TabTransformer" (Huang et al., 2020), is to encode categorical fields individually by transforming them into a contextually meaningful continuous vector. These vectors are then concatenated with the numerical fields, resulting in a feature vector that can be used by other machine learning models. While this approach has been successful, it only encodes the categorical features, leaving the numerical fields untreated. An alternative method is to embed all fields simultaneously, allowing for the processing of both numerical and categorical fields together. To achieve this, categorical fields can be one-hot encoded and then concatenated with the numerical values and passed to an encoding layer. However, certain fields may be difficult to encode due to the high dimensionality of the resulting feature space. Previous works in the NIDS domain have demonstrated that state-of-the-art performance can still be achieved using frequency-limited one-hot encoding to overcome this increase in dimensionality.

#### 5.1.1. Categorical only

In terms of approaches to encoding categorical fields as part of the input encoding step, there are three main approaches:

- **Lookup-Based Embedding Layer** (Mikolov et al., 2013): The lookup-based embedding layer maps a categorical input into a continuous feature vector. This is implemented as a lookup table, mapping between the unique values of the input, and the set of feature vectors. During training time, the values of these vectors are learned.
- **Fully Connected Embedding Layer:** This is also referred to as a *Dense Embedding Layer*, and it maps a categorical input to a continuous feature vector, using a non-linear transformation. In essence, this is a single layer neural network with an activation function. This transformation is learned during training time, but unlike the lookup-based embedding layer, embeddings are resolved by applying mathematical operations on the input rather than looking up a pre-learned vector.
- **Projection Layer:** The projection layer or linear projection layer, is similar to the dense embedding layer, however, instead of applying a non-linear transformation, the projection layer applies a linear transformation. This can be seen as a very basic form of dimensionality reduction and is widely used for this purpose (Sorzano et al., 2014).

Previous works have proposed the processing of categorical fields with a separate transformer model before concatenating, as a means of embedding categorical fields in tabular data (Huang et al., 2020). However, in addition to the large increase in model complexity from such an approach, because this was a non-sequential implementation, built to process a single row of tabular data and make a classification, we do not implement it as part of FlowTransformer.

#### 5.1.2. Categorical + numerical – record level

Instead of only embedding the categorical fields, which also requires a transformation to be applied to each categorical field individually, we can instead embed the entire flow record in one pass. This can also be seen as a form of dimensionality reduction, and given that numerical fields in flow records such as number of bytes and number of packets sent often has significant portions of shared information, it is sensible to expect there to be a high level of redundancy in the raw representation of flows that can be removed through this encoding.

The two dense approaches used to embed categorical fields can also be used to embed a flow record. However, lookup-based embedding does not function with (continuous) numerical fields, and thus cannot be applied to the record as a whole which includes both categorical and numerical fields.

- **Record Level Embedding - Dense:** To apply dense embedding, the categorical fields are one-hot encoded and concatenated with the numerical fields before being passed to the embedding layer which then maps the record to a continuous feature vector.
- **Record Level Projection:** The projection layer is implemented in the same manner as the dense record level embedding, omitting the non-linear activation and bias.

#### 5.1.3. Other approaches

- **No encoding:** If categorical fields in pre-processed flow records are one-hot encoded, they can be passed directly to the transformer, at the cost of increasing dimensionality. This is feasible for lightweight flow formats with few features.
- **Handling as text:** The paper does not explore handling flow records as text because flow data has a structured format with distinct features. This structured format is lost when treated as text, making it harder for the model to capture feature-value relationships, in addition to several other disadvantages.

### 5.2. Transformer models

This study compares four transformer architectures, namely ‘shallow’ (low number of layers) encoder-based, shallow decoder-based, deep encoder-based, and deep decoder-based transformers. The shallow models are based on the basic multi-head self-attention transformer architecture and comprise between 2 and 6 encoder or decoder blocks. Two specific deep transformer models, GPT 2.0 and unmasked BERT, are also considered. The difference between shallow and deep models lies primarily in their depth, number of attention heads, and internal size, while their core transformer block structure is the same. Although both GPT and BERT use scaled-dot-product attention, BERT’s attention mechanism is bidirectional, considering tokens in both directions, unlike GPT, which only considers previous tokens.

#### 5.2.1. GPT 2.0 - Deep decoder transformer

GPT 2.0 (OpenAI, 2019) is a generative transformer model that uses a sequence approach to predict the next word. It only uses transformer decoder blocks, repeating them up to 12 times in smaller models. The input sequence is passed through the decoder blocks one by one, with each block using the previously generated words as part of the context to predict the next word. By considering the input sequence and previously generated words, GPT can model natural language more effectively than traditional transformer models. It is an autoregressive model that generates tokens one at a time, considering only the tokens to the left of the current token.

#### 5.2.2. BERT - Deep encoder transformer

BERT (Devlin et al., 2018), in contrast to GPT, is an encoder-only transformer model designed for natural language understanding tasks. It consists of repeated transformer encoder blocks that generate a fixed-length representation of the input sequence. BERT was trained to perform masked language modelling and sentence pair classification, however the fixed-length representation can be fine-tuned for various downstream natural language understanding tasks. Unlike GPT, BERT is not autoregressive and can use the entire context for representation generation. This means the generated representations depend on both the ‘left’ and ‘right’ tokens.

### 5.3. Classification head options

We show the location and function of a classification head in Fig. 4. Because transformers are sequence-to-sequence models, we must transform the sequential output into a classification result for an NIDS. Typically, this involves performing some operation on the output sequence, to prepare it to be passed into a dense Multilayer Perceptron (MLP), which then performs the final classification. While it is possible to feed all the outputs from the transformer directly into the MLP, this causes the parameter count to rise exponentially as the sequence length increases, and therefore this is suboptimal in many applications. We call this approach ‘flattening’, as the feature vector is flattened before being passed to the MLP, and consider it in our evaluation. However, we also consider other, more efficient approaches, which will be discussed in this section.

The most common approach in the NLP domain is to average over the features in the sequence, called Global Average Pooling. For each feature in the resulting feature vector, we simply take the average of this value over the entire sequence. This approach works well for tasks such as sentiment classification, where the average of features over an entire sentence or paragraph can be combined, however, this makes less sense in the NIDS domain. Instead of simply taking a global average across every flow in the sequence, we could use a densely connected neural network layer, for each feature. This allows the model to apply different weights to the different flows in the sequence. This is referred to as Featurewise Embedding or Featurewise Projection in this work.

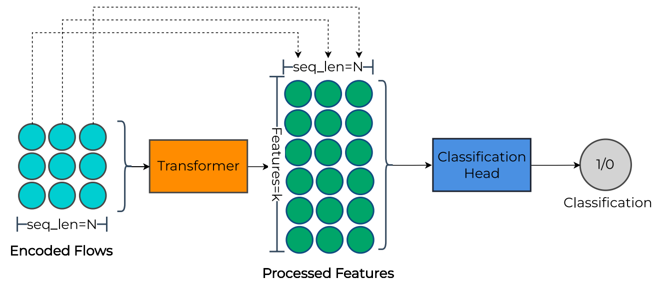


**Fig. 4.** The classification head of a transformer model converts between the sequential output of the transformer, into a classification result.

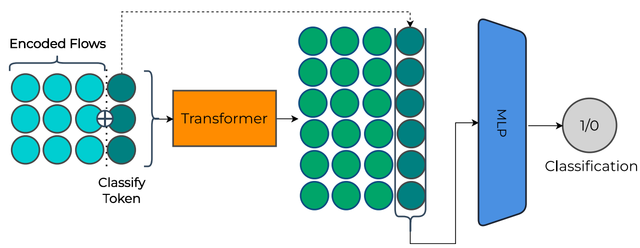


**Fig. 5.** Here the classification head takes the output corresponding to the special 'classification' token, and feeds it directly into a dense network for classification.

These approaches both include information from previous flows that is already considered by the transformer itself. Instead, we can take the last output from the transformer and use this as input to the classification head. This corresponds to the feature vector from the last flow. We refer to this as Last Token in this work. Since typically, the classification task of an NIDS is only concerned with the class (e.g. benign vs malicious) of the last flow, using only the last contextual embedding vector should be sufficient.

Finally, models such as BERT, which are trained to perform multiple tasks, introduce a special token that instructs the model to perform a particular task. We also test the efficacy of this in the NIDS domain, by evaluating the impact of appending a 'classification' token to the end of a sequence of flows, and using the corresponding output of the transformer to perform classification. We give an example of this in Fig. 5.

## 6. Implementation

The FlowTransformer framework implementation shown in Fig. 6 comprises several interchangeable blocks that form a processing pipeline. These blocks facilitate swift customisation, training, and evaluation of transformer models, and other sequential models, against flow-based NIDS datasets. In the sections below, we discuss each of these blocks. We provide the source code on GitHub for public use and more detailed documentation, along with reference implementations for all the transformers used in this work (Manocchio, 2023). In addition, we highlight the use of flow datasets with varying formats, for example, we demonstrate ingesting an MQTT format dataset MQTT-IoT-IDS2020 (Hindy et al., 2020) as well an example with automatic dataset download for the UNSW-NB15 (Moustafa & Slay, 2015) datatset.

### *6.1. Dataset ingestion*

FlowTransformer begins with a dataset ingestion component, which can accommodate any tabular dataset, making it compatible with various flow data formats. FlowTransformer automatically handles out-of-range or missing values. To apply FlowTransformer to a specific dataset, a dataset specification must be provided. This lists the categorical and numerical features of the dataset, for use by the pre-processing and input encoding stages. The dataset specification also identifies the class column to be trained on and the benign traffic label, both of which are used in the evaluation component to identify malicious and legitimate flows. Once the data specification is provided, FlowTransformer can ingest the dataset through the later stages without requiring coding. Because the format definition is flexible, we are able to ingest any form of flow based or tabular data consisting numeric and categorical fields, and after defining the format, the remainder of the FlowTransformer pipeline will automatically adjust. Examples of this are provided in the Github (Manocchio, 2023).

### *6.2. Pre-processing*

Upon receiving a dataset, FlowTransformer loads and divides it into training and evaluation splits for pre-processing. Users can select from three splitting configuration options for added flexibility. In this paper, we split the dataset in a 90% to 10% ratio. The pre-processing is first fit on the training data and then applied to the entire dataset, ensuring no information leakage from the evaluation dataset during pre-processing. The pre-processing layer also takes into account the categorical format expected by the input encoder. Depending on the input encoder's requirements, the pre-processing layer can either one-hot encode or integer encode the categorical variables. The pre-processing approach we employ in this work, is that proposed in (Manocchio et al., 2022). It proved to be the most effective for neural network-based NIDS systems compared to other approaches tested. The pre-processing involves encoding the $N$ most-frequent categorical features using one-hot or integer encoding (depending on the model's requirements) and min–max scaling numerical features after taking the logarithm. Custom pre-processing can be specified by implementing fit and transform methods for numerical and categorical fields. The expected categorical format (one-hot or integer encoded) is provided as a parameter to the fit and transform methods for the categorical fields, allowing the pre-processing to be switched depending on the input encoding.

The pre-processing we have proposed allows FlowTransformer to handle any tabular data format consisting of numerical or categorical fields, with standardisation of arbitrary inputs built into the pre-processing step, meaning data does not need to be standardised before being used with the FlowTransformer framework.

### *6.3. Model pipeline*

Once the pre-processing is complete, the input data's final size is determined, which is then used to construct the actual transformer model. FlowTransformer builds the transformer as a TensorFlow Keras model, which can be compiled and trained independently if required.

The transformer model comprises three interchangeable components, which we have discussed prior:

1. The input encoder transforms the pre-processed data into an encoded format suitable for the transformer. These components can perform any transformation on the raw data. We provide one pre-processing implementation with the base framework, which we use for our evaluation.
2. The transformer itself can be replaced with any transformer model, or any machine learning model that accepts a 3 dimensional input. We include implementations for basic transformer blocks, as well as transformer decoder blocks from GPT-3.0, and transformer encoder blocks from BERT. The number of flows in the sequence provided is fully configurable.
3. The classification head receives the outputs from the transformer blocks and converts them into a classification. Although optimised for binary classification, multi-class classification heads

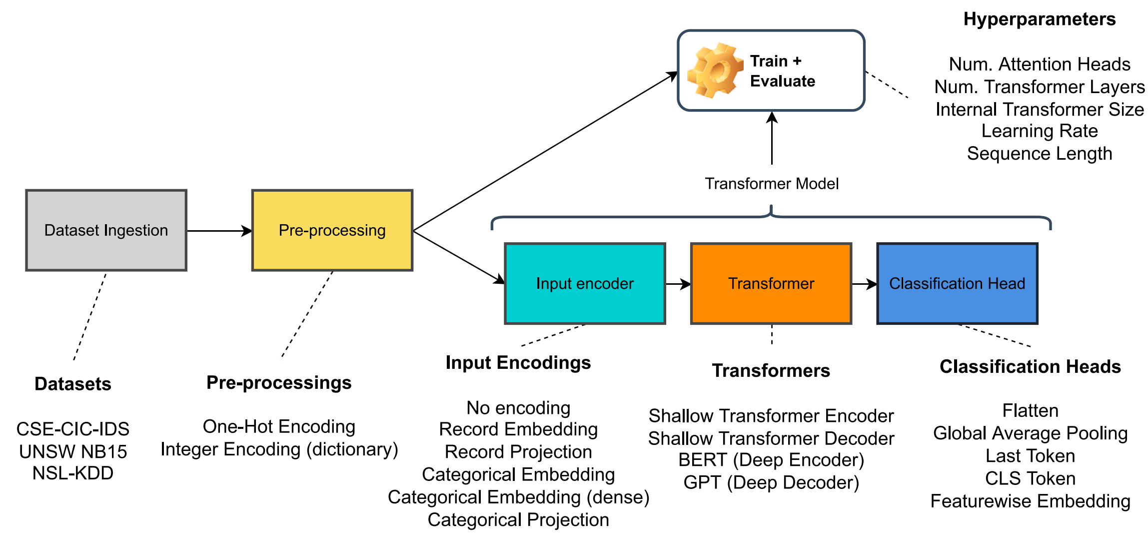


**Fig. 6.** The FlowTransformer implementation allows the three primary aspects of a transformer to be interchanged, as well as different datasets and pre-processing approaches. Once a pipeline is specified, the evaluation is supported automatically, or a TensorFlow model can be exported for standalone training.

are also supported within custom training loops when using FlowTransformer's built-in evaluation functions.
If needed, the classification head can also modify the tensor after input encoding, such as appending or removing tokens before passing them to the transformer, which is necessary for approaches like appending a classification token.

### 6.4. Evaluation support

FlowTransformer provides methods to evaluate a built model, offering several optimisations for faster dataset loads between model runs, enabling quicker hyperparameter searches. It also generates comprehensive outputs from each experiment, detailing parameters such as size, timing, performance results, and incremental results during training. This functionality was utilised by our work for collecting results.

## 7. Experimental methodology

In this paper, we utilise the FlowTransformer framework to test a large number of applicable transformer components, to determine which are the most effective across a three benchmark NIDS datasets. We have previously discussed a number of input encodings, transformer blocks and output encodings, and each of these are tested using the FlowTransformer framework. When collecting results, we performed a grid search. The dimensions explored in this paper are, input encodings, transformer block type, transformer depth, transformer feed forward size, number of attention heads, classification head and learning rate.

### 7.1. Model training and grid search

When performing a grid search over a target space, we performed a minimum of 3 repeats of each experiment. We take the result from the best repeat, which can control for poor model initialisation during the experiments. For the model training, we used early stopping and an epoch limit. The early stopping was set to a patience of 5 epochs, and the maximum number of epochs was limited to 20. We chose 20 epochs as we observed during our initial experimentation that the majority of models had converged to within 1% of their final performance by the 20th epoch. We used the Adam (Kingma & Ba, 2014) optimiser for training with the parameters defined in the paper, with learning rate being specified in each experiment (as this was part of the grid search).

Results were collected on an Intel Core i7-10750H CPU (6 cores, 12 processors), hyperthreading enabled, base clock 2.9 GHz, capped at 3.5 GHz with an NVIDIA GeForce RTX 2070 with Max-Q Design (8 GB GDDR6 GPU memory), and 16 GB VRAM at 2933 Mhz. We used Tensorflow 2.3.0, built for GPU, on a Windows 10 computer running Python 3.9.

Models were built with a value type of float64, with each parameter occupying 64 bits of space.

### 7.2. Datasets

This paper considers three different widely used and highly cited NIDS datasets. Specifically, we use the flow format version of these datasets, which have become prevalent in the NIDS community. The conversion process of these datasets to flow format is detailed in (Sarhan et al., 2022) which was proposed by the authors as a standardised format for flow-based NIDS.

1. **NSL-KDD** (Tavallaee et al., 2009), an NIDS dataset for traditional networks that is one of the most widely used NIDS benchmark datasets. This was developed to replace the KDD cup datasets.
2. **UNSW-NB15** (Moustafa & Slay, 2015), an NIDS dataset for traditional networks featuring 'a hybrid of real modern normal activities and synthetic contemporary attack behaviours' with 'nine types of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms.
3. **CSE-CIC-IDS2018** (Sharafaldin et al., 2018), an NIDS dataset for traditional networks including 'seven different attack scenarios: Brute-force, Heartbleed, Botnet, DoS, DDoS, Web attacks, and infiltration of the network from inside.

### 7.3. Evaluation metrics

To assess the performance of different transformer models, standard metrics were utilised, such as F1 score, false alarm rate and detection rate. The metrics are computed using a combination of True Positives, True Negatives, False Positives, and False Negatives, denoted as $TP$, $TN$, $FP$, and $FN$ respectively. We use both F1 score as the primary metrics to compare approaches, and we are reporting results for the binary classification task.

### 7.4. Transformer hyperparameters

Besides comparing input encoding approaches and classification heads, we also explored various transformer configurations and corresponding hyperparameters. These are detailed in Table 1.

**Table 1**
Hyperparameter values used.

| Hyperparameter | Values |
|---|---|
| Transformer block | Encoder, decoder |
| Layers | 2, 4, 6, 8 |
| Feed Forward (FF) dimensions | 128, 256, 512 |
| Attention heads | 2, 4, 6, 8, 12 |
| Learning rate | 0.01, 0.001, 0.0005, 0.0001, 0.00001 |

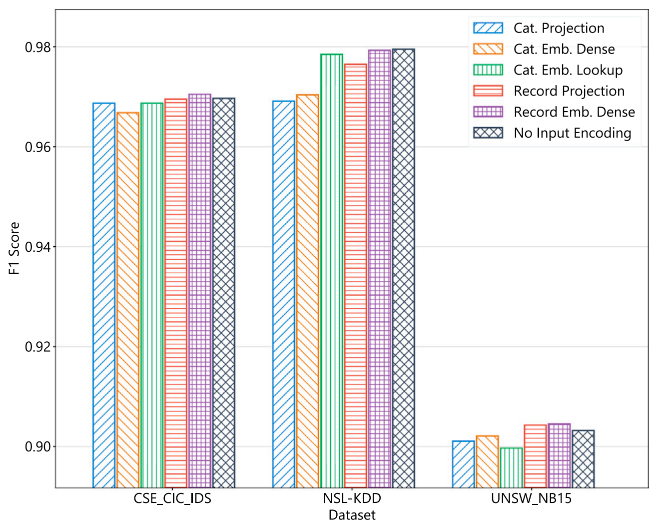


**Fig. 7.** Results for various input encodings, for the basic transformer model (2 layers, 2 heads, 128 ff. dim size), on the CSE-CIC-IDS2018, NSL-KDD and UNSW-NB15 dataset.

### *7.5. Inference & training time*

For inference timing and training timing steps, we ensured that the GPU was cleared and the timings were begun from a warm TensorFlow state. Training time was measured by timing each batch, which encompasses the backpropagation step. We then divided this by the batch size, and averaged over all batches used during training. This was done using TensorFlow's train_on_batch function. We ensured that no extraneous processes were running during model training, and performed outlier testing to ensure that there was no significant drift of batch training times during training.

To measure inference timing, we began by recording the inference time from the start to the end of a single batch. We repeated this process four times on the same batch of data to ensure the results were consistent and not being cached, and took the median time from those four measurements to obtain a stable range. We then repeated this procedure for 50 batches of data, ensuring that the batches were selected randomly, and calculated the mean time over all the batches.

Results are presented in terms of throughput, or the number of flows per second that can be ingested by the model.

## 8. Results

The results of this study are divided into four sections. The first section discusses the choice of input encodings, followed by a discussion of transformer blocks, classification heads, and finally, training and inference times. The study also evaluates five common transformer architectures, including BERT and GPT, by comparing the best choice of input encodings and classification heads for each of these models.

### *8.1. Input encodings*

The impact of different classification heads and input encoding approaches on model size (number of parameters, which are indicators of model size and in turn the overall resource utilisation of the model), detection performance (F1 Score, False Alarm Rate, True Positive Rate) and training and inference speed (throughput in terms of flows per second) are presented in Table 2. The table shows the results for a basic transformer model (details below) trained on the CSE-CIC-IDS2018 dataset. The corresponding results for the NSL-KDD and UNSW-NB15 datasets are largely consistent with the results in Table 2, and are provided in the appendix, in Tables 6 and 7 respectively. The basic transformer model used in the experiment is a two-layer, two-attention head shallow (low number of layers) transformer encoder, which is a sensible starting point for comparison, given its smaller size. Our evaluation found that these trends are consistent across all model depths, and that shallow transformers achieve competitive performance when compared to deeper models. Learning rates were chosen for each model through a grid search over the hyperparameter range listed in Section 7.

The comparison of different input encodings in terms of F1 score in Table 2 indicates that no method significantly outperforms the others. This result is also presented in a graphical form in Fig. 7 across the three datasets. The figure is an aggregate version of Tables 2, 6, and 7, and shows the performance of various input encodings. Each bar represents a single input encoding, while each group of bars represents a dataset. The $y$-axis represents the F1 score from the best choice of classification head for that particular input encoding.

Fig. 7 clearly shows that the change resulting from various input encodings is minimal, and any choice of input encoding is still capable of achieving a reasonable model predictive performance. However, when considering Table 2, it is clear that the choice of input encoding is a determining factor with regards to parameter count. Although categorical embedding is one of the most widely used approaches, record level embedding achieves the same performance with only half the model parameter count. For example, if we consider input encodings using the Last Token classification head in Table 2, categorical feature embedding has a parameter count of $\approx$ 500K, and record level embedding has a parameter count of $\approx$ 200K.

Next, we look at the difference between projection and embedding. We can see that both projection and embedding approaches performed equivalently, with projection sometimes outperforming embedding. Projection is a simpler linear transformation and embedding is non-linear. It could be argued that if a linear transformation of the input is sufficient to achieve the highest accuracy, then we could also use no input encoding and allow the model to learn through weights. However, a projection layer is a very effective form of dimensionality reduction, and can significantly reduce model size. It is also smaller than an embedding layer due to its lack of bias.

The model parameter count was also influenced by the use of 'categorical only' embedding versus 'record level embedding'. Our experiments found that both overall record embedding and individual categorical field embedding performed similarly in accuracy. However, the former approach reduces dimensionality across all features, helping to reduce model size without sacrificing accuracy. This is especially useful for flow formats with many non-categorical numerical features that cannot be categorically embedded. In addition to the size reduction, record-level approaches also showed an improvement in training and inference time versus categorical approaches. While lookup-based embedding is commonly used in Transformers for NLP, it has slower training times compared to record-level approaches for NIDS, and also was slower at inference time, likely as a result of needing to be applied multiple times for each categorical field, rather than once for the record as a whole.

### *8.2. Transformer size*

We will now examine the transformer models themselves, including the number of encoder blocks or layers, the internal feed-forward dimension, and the number of heads. To facilitate these comparisons, we

**Table 2**
Results for a basic transformer model (2 layers, 2 heads, 128 ff. dim size), on the CSE-CIC-IDS2018 dataset.

| Classification head | Input encoding | Model parameters | Train throughput (flows/s) | Inference throughput (flows/s) | F1 score | Detection rate | False alarm rate |
|---|---|---|---|---|---|---|---|
| Last token | Categorical emb. dense | 470,785 | 1,518 | 7,014 | 0.9651 | 95.75% | 0.22% |
| | Categorical emb. lookup | 470,641 | 354 | 7,420 | 0.9674 | 95.53% | 0.26% |
| | Categorical projection | 470,641 | 1,238 | 8,394 | 0.9682 | 95.84% | 0.21% |
| | Record emb. dense | 384,897 | 1,653 | 7,877 | 0.9676 | 95.94% | 0.19% |
| | Record projection | 193,409 | 1,341 | 8,258 | 0.9695 | 95.92% | 0.14% |
| | No input encoding | 785,049 | 1,501 | 8,258 | 0.969 | 95.96% | 0.15% |
| Flatten | Categorical emb. dense | 624,897 | 1,494 | 7,420 | 0.9654 | 95.89% | 0.24% |
| | Categorical emb. lookup | 624,753 | 372 | 7,315 | 0.966 | 95.18% | 0.23% |
| | Categorical projection | 624,753 | 1,227 | 8,828 | 0.9666 | 95.35% | 0.16% |
| | Record emb. dense | 499,585 | 1,254 | 7,642 | 0.9682 | 95.67% | 0.17% |
| | Record projection | 250,753 | 1,322 | 7,877 | 0.9693 | 95.82% | 0.13% |
| | No input encoding | 1,044,889 | 1,469 | 8,533 | 0.9672 | 95.26% | 0.20% |
| Featurewise emb. | Categorical emb. dense | 449,966 | 1,480 | 7,314 | 0.9637 | 95.38% | 0.34% |
| | Categorical emb. lookup | 449,966 | 424 | 7,642 | 0.9675 | 95.77% | 0.20% |
| | Categorical projection | 449,822 | 1,471 | 8,258 | 0.9673 | 94.99% | 0.17% |
| | Record emb. dense | 186,370 | 1,606 | 8,534 | 0.9705 | 95.96% | 0.10% |
| | Record projection | 186,306 | 1,490 | 10,240 | 0.9659 | 95.84% | 0.20% |
| | No input encoding | 749,244 | 1,450 | 8,534 | 0.9675 | 95.06% | 0.17% |
| Featurewise projection | Categorical emb. dense | 449,965 | 1,289 | 8,262 | 0.9662 | 95.19% | 0.22% |
| | Categorical emb. lookup | 449,965 | 353 | 8,982 | 0.9681 | 95.52% | 0.16% |
| | Categorical projection | 449,821 | 1,498 | 8,533 | 0.9664 | 95.24% | 0.20% |
| | Record emb. dense | 186,369 | 1,584 | 8,393 | 0.969 | 95.46% | 0.09% |
| | Record projection | 186,305 | 1,530 | 8,258 | 0.9665 | 95.07% | 0.22% |
| | No input encoding | 749,243 | 1,336 | 8,827 | 0.9651 | 95.75% | 0.35% |
| Global average pooling | Categorical emb. dense | 470,785 | 1,524 | 8,000 | 0.3311 | 95.94% | 33.91% |
| | Categorical emb. lookup | 470,641 | 359 | 7,530 | 0.3466 | 95.75% | 17.19% |
| | Categorical projection | 470,641 | 1,264 | 7,111 | 0.3379 | 93.36% | 25.92% |
| | Record emb. dense | 193,473 | 1,741 | 7,014 | 0.3371 | 92.51% | 29.39% |
| | Record projection | 193,409 | 1,507 | 9,481 | 0.3395 | 89.38% | 24.21% |
| | No input encoding | 785,049 | 1,165 | 8,127 | 0.351 | 96.21% | 26.19% |
| CLS token | Categorical emb. dense | 473,485 | 1,436 | 8,127 | 0.3199 | 95.82% | 42.22% |
| | Categorical emb. lookup | 473,341 | 353 | 7,014 | 0.3615 | 95.74% | 19.07% |
| | Categorical projection | 473,341 | 1,150 | 9,144 | 0.3483 | 95.87% | 26.56% |
| | Record emb. dense | 387,597 | 1,093 | 7,014 | 0.3251 | 95.69% | 33.33% |
| | Record projection | 196,109 | 1,581 | 7,641 | 0.3262 | 93.80% | 20.88% |
| | No input encoding | 787,749 | 1,109 | 8,127 | 0.3436 | 97.55% | 29.01% |

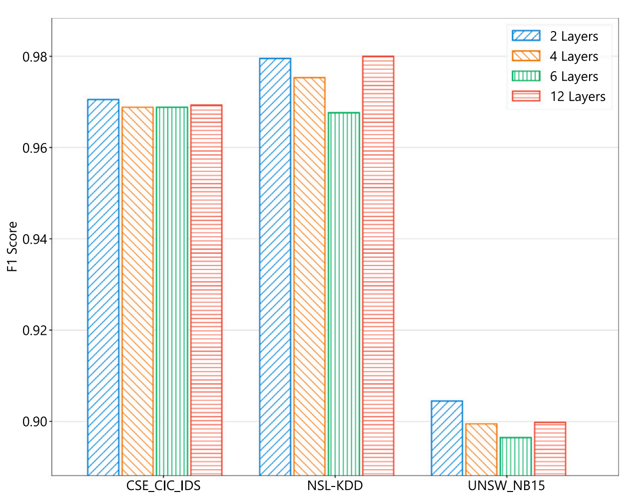


**Fig. 8.** The F1 scores of four depths of transformer models, across the three benchmark datasets.

will use transformer encoder blocks, as we did for the input encoding comparison.

Fig. 8 compares transformer decoder models with different numbers of layers, in terms of their F1 score across the three benchmark datasets. Each group of bars corresponds to one dataset, and each bar corresponds to a particular transformer with that number of layers. The results shown in the figure are for the best-performing models based on the input encoding, classification head, and other hyperparameters used. This ensures a fair comparison between the best models with a particular depth, internal size, and number of heads.

We can see in Fig. 8 that the model depth does not have a very large impact on the model performance. The shallowest 2-layer model outperformed the deeper models across all three datasets. This trend is mirrored in Fig. 9, which compares the number of model heads, where again, the smallest 2-head model outperformed the models with more heads. For the internal feed forward size, shown in Fig. 10, again the smallest internal size performs best for the CSE-CIC-IDS2018 and UNSW-NB15 dataset, but notably for the NSL-KDD dataset, the largest internal size performed the best.

The number of attention heads, internal size and number of layers required for a task are all dependent on the complexity of the task, and the data available, so the variations between datasets are to be expected. However, across the 3 datasets we tested, shallow models showed comparable performance to deeper models. Where shallow models are applicable for a particular task, they can be preferable, as models with fewer parameters require less data to train, and are generally less prone to overfitting. Based on our observed results in the NIDS domain, shallow transformer models should not be ruled out in favour of larger models. Later we will include a specific comparison with two shallow transformer models, and two well-known large transformer models, GPT and BERT, to enable further comparison.

### 8.3. Classification heads

When investigating the impact of classification heads, we refer back to Table 2, which shows the impact of different classification heads and

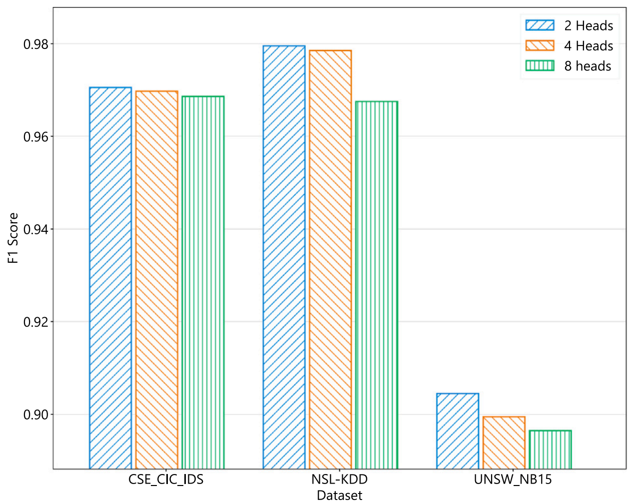


**Fig. 9.** The F1 scores of transformer models with 2, 4 and 8 attention heads respectively, across the three benchmark datasets.

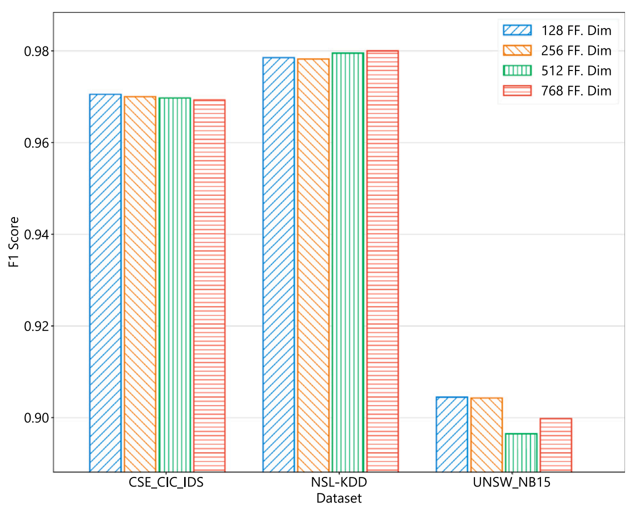


**Fig. 10.** The F1 scores of transformer models with four different internal feed forward sizes, across the three benchmark datasets.

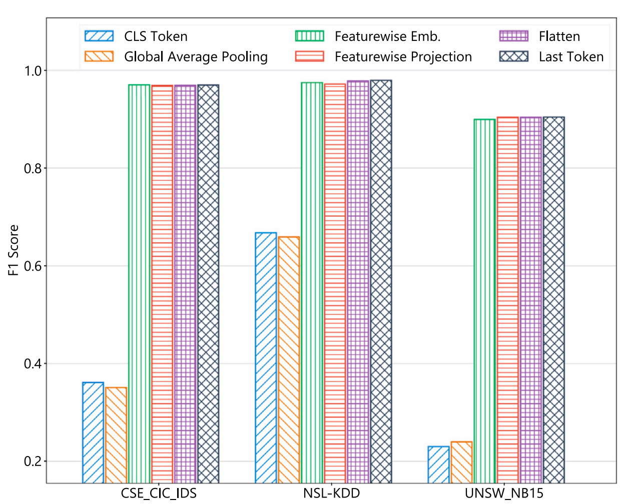


**Fig. 11.** The F1 scores of transformer models with six different classification heads, across the three benchmark datasets.

input encodings on the F1 score, the number of model parameters, as well as and training and testing time. When comparing the F1 scores of different classification heads, it is apparent that classification heads had the largest overall effect on performance, and notably, the effect was consistent across all datasets.

To make this trend clear, we show the graphical version of Tables 2, 6 and 7 in Fig. 11 for each of the three benchmark datasets. Each bar represents a choice of classification head, and the $y$-axis is the F1 score. We choose the F1 score from the model with the best choice of input encoding for this classification head. This graph shows that the consistently best approach is using the last token classification head, followed by flatten, then the featurewise embedding approaches. However, each of these approaches has a relatively high performance, in contrast to CLS and Global Average Pooling, which performed very poorly. This is a significant result. A large number of related works, particularly in the natural language processing domain, use Global Average Pooling when performing classification. However, this was one of the lowest performing approaches in the NIDS domain. Averaging includes information from previous flows that may not be relevant to the classification task of the last flow in the sequence, which may explain why it performed worse.

Among the highest performing classification heads was the Flatten approach, which uses the full outputs from the transformer and passes them directly to the classification MLP. However, we can see in Table 2 that this comes at the cost of a disproportionately large parameter count, which increases exponentially as the sequence length increases, unlike for the other approaches. Although for small sequences of flows this approach can be used, for longer sequences the model size can become too large. A larger number of model parameters requires more training data and increases the likelihood of overfitting.

Finally, featurewise embedding also performed relatively well. This approach allows a weighted combination from all of the features of flows in the sequence to be passed to the classification MLP, rather than just the features from the last flow. However, this is likely not advantageous over simply taking only the last flow's embedding, due to its lower performance and higher complexity.

### *8.4. Comparative evaluation*

Finally, we provide a comparative evaluation of four transformer models. This evaluation covers the following models. Basic Dense Encoder - a two encoder block transformer, Basic Dense Decoder - a two decoder block transformer, GPT Model - a 12 decoder block transformer, with a 768 internal dimension and 12 attention heads, BERT Model - a 12 encoder block transformer, with a 768 internal dimension and 12 attention heads.

The results are shown in Table 3. For each model, we include the two best combinations of input encoding and classification head, and the corresponding parameter count, training and inference times, F1 score, detection rate and false alarm rate.

We can see that both the shallow (low number of layers/depth) and deep models have a similar F1 score. However, the deep models have parameter counts orders of magnitudes larger than the shallow models. This model size difference is also reflected in the training and inference throughput, with the shallow models having a much higher throughput than the deep transformer models. With the CSE dataset, the best shallow encoder has $\approx$ 200K parameters with a throughput of $\approx$ 8.5K flows per second, whereas the deep encoder has $\approx$ 29M parameters and a throughput of just $\approx$ 300 flows per second. Because the accuracy of the deeper models is lower than those of the shallower models, it does not appear that GPT or BERT are optimal choices for network intrusion detection. During our initial experimentation we also evaluated a subset of these larger models with long training time and no early stopping, and we found that regardless of training time, the deep models did not outperform the shallow transformer models.

**Table 3**
For each dataset and the four identified models, we display the results for the two highest performing combinations of parameters. These results are from the best choice of model hyperparameters, input encoding and classification head and learning rate for each dataset.

| Model and dataset | | Best performing components and hyperparameters | | | | | Model results | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | Transformer | Layers | Heads | FF dim | Classification head | Input encoding | Model parameters | Throughput (flows/s) | | F1 score | Detection rate | False alarm rate |
| | | | | | | | | Train | Inference | | | |
| CSE_CIC_IDS | Shallow encoder | 2 | 2 | 128 | Featurewise emb. | Record emb. dense | 186,370 | 1,606 | 8,534 | 97.05% | 95.96% | 0.10% |
| | | 2 | 2 | 256 | Last token | Record emb. dense | 714,369 | 1,281 | 8,394 | 97.00% | 95.77% | 0.14% |
| | Shallow decoder | 2 | 2 | 128 | Last token | No input encoding | 1,538,673 | 961 | 8,127 | 96.23% | 95.21% | 0.34% |
| | | 2 | 2 | 128 | Flatten | No input encoding | 1,798,513 | 945 | 5,626 | 96.05% | 95.02% | 0.35% |
| | GPT model (Deep decoder) | 12 | 12 | 768 | Last token | Record projection | 3,607,425 | 536 | 834 | 96.93% | 95.86% | 0.11% |
| | | 12 | 12 | 768 | Last token | No input encoding | 53,979,633 | 59 | 786 | 96.90% | 95.14% | 0.13% |
| | BERT model (Deep encoder) | 12 | 12 | 768 | Flatten | Record projection | 29,921,153 | 89 | 277 | 95.80% | 95.48% | 0.47% |
| | | 12 | 12 | 768 | Last token | Categorical emb. dense | 79,637,585 | 74 | 149 | 95.41% | 94.73% | 0.43% |
| NSL-KDD | Shallow encoder | 2 | 2 | 512 | Last token | No input encoding | 920,865 | 1,469 | 6,400 | 97.95% | 98.48% | 0.55% |
| | | 2 | 2 | 512 | Last token | Record emb. dense | 1,347,457 | 1,392 | 7,420 | 97.93% | 98.06% | 2.10% |
| | Shallow decoder | 2 | 2 | 128 | Last token | No input encoding | 182,849 | 1,399 | 5,505 | 95.29% | 93.77% | 2.68% |
| | | 2 | 2 | 128 | Flatten | No input encoding | 261,697 | 1,428 | 10,894 | 93.84% | 93.17% | 3.70% |
| | GPT model (Deep decoder) | 12 | 12 | 768 | Last token | Record projection | 3,594,497 | 658 | 3,459 | 98.00% | 97.87% | 1.02% |
| | | 12 | 12 | 768 | Flatten | No input encoding | 6,226,497 | 172 | 2,612 | 97.90% | 98.23% | 1.31% |
| | BERT model (Deep encoder) | 12 | 12 | 768 | Last token | No input encoding | 40,909,249 | 77 | 354 | 94.57% | 91.81% | 0.50% |
| | | 12 | 12 | 768 | Last token | Record projection | 29,850,881 | 77 | 177 | 93.52% | 90.79% | 1.26% |
| UNSW_NB15 | Shallow encoder | 2 | 2 | 128 | Last token | Record emb. dense | 382,081 | 1,630 | 7,877 | 90.45% | 99.89% | 1.03% |
| | | 2 | 2 | 256 | Featurewise projection | Record projection | 350,529 | 1,527 | 6,321 | 90.43% | 98.47% | 0.96% |
| | Shallow decoder | 2 | 2 | 128 | Last token | No input encoding | 1,327,649 | 1,095 | 6,481 | 88.94% | 99.79% | 1.22% |
| | | 2 | 2 | 128 | Flatten | No input encoding | 1,567,777 | 1,088 | 6,321 | 87.91% | 99.61% | 1.32% |
| | GPT model (Deep decoder) | 12 | 12 | 768 | Featurewise projection | Categorical emb. lookup | 20,312,029 | 105 | 287 | 89.98% | 100.00% | 1.07% |
| | | 12 | 12 | 768 | Last token | Record emb. dense | 3,606,081 | 288 | 2,959 | 89.95% | 99.82% | 1.07% |
| | BERT model (Deep encoder) | 12 | 12 | 768 | Last token | No input encoding | 123,889,249 | 48 | 154 | 87.74% | 99.75% | 1.33% |
| | | 12 | 12 | 768 | Last token | Record projection | 29,862,401 | 83 | 752 | 76.23% | 98.90% | 2.98% |

**Table 4**
Results for the FlowTransformer based models with the best choice of components and parameters for all 6 dataset used in our evaluation.

| Dataset | T. layers | Heads | T. internal size | Head | Input encoding | F1 score |
|---|---|---|---|---|---|---|
| MQTT | 2 | 2 | 256 | Last token | No input encoding | 99.90% |
| CSE_CIC_IDS_v2 | 2 | 2 | 128 | Last token | No input encoding | 99.62% |
| TON_IOT | 2 | 2 | 512 | Last token | No input encoding | 98.91% |
| NSL-KDD | 12 | 12 | 768 | Last token | Record projection | 98.00% |
| CSE_CIC_IDS | 2 | 2 | 128 | Last token | Record embedding | 97.05% |
| UNSW_NB15 | 2 | 2 | 128 | Last token | Record embedding | 90.45% |

**Table 5**
Comparison of the best FlowTransformer based model, to state-of-the-art models on the NSL-KDD dataset in terms of F1 score with results reported by (Liu & Wu, 2023) (*), and F1 score and training time for the CSE-CIC-IDS dataset with results from (Wu et al., 2022) and (Zhang & Wang, 2022).

| Model | NSL-KDD* | CSE-CIC-IDS | |
|---|---|---|---|
| | F1 score (%) | F1 score (%) | Training time (s) |
| (Y. Liu & Wu, 2023) | 88.20% | N/A | |
| VIT (Yang et al. 2022) | 85.50% | N/A | |
| RTIDS (Wu et al. 2022) | 86.40% | 99.17% | 195.6 |
| CNN-Transformer (Zhang et al. 2022) | 86.10% | 96.60% | 7,350.0 |
| FlowTransformer (ours) | 98.00% | 97.05% | 25.5 |

Decoder models, including GPT performed similarly to encoder models, which is interesting because decoder models are typically more suitable for generative tasks. However the auto-regressive attention mechanism in GPT considers only previous flows, making it well-suited for flow-based classification tasks, where we aim to detect if the last sequence in the flow is anomalous.

Overall, based on our results, we can observe that the highest performing model uses record level projection, decoder blocks with 2 layers and 2 heads, with a last token classification head. These choices represent a good starting point for exploring transformer models for flow-based network intrusion detection. The input encoding and classification head particularly are good starting choices for a variety of datasets and transformer models, with the transformer itself being a good candidate for modification. The learning rate must be determined for a particular choice of model, and this should be done across repeated iterations of model training, given the training volatility we observed. The FlowTransformer framework provides an efficient platform for this type of experimental exploration.

### *8.5. Additional datasets*

To demonstrate FlowTransformer's capability and flexibility in ingesting datasets of diverse formats, we have applied it to three supplementary datasets: an MQTT dataset using a new format of flow data (Hindy et al., 2020), a newer version of the CSE IDS dataset (Liu et al., 2022) modified to mitigate concerns outlined in (Liu et al., 2022), and the TON-IOT dataset (Moustafa, 2021), which specifically pertains to edge Network Intrusion Detection Systems (NIDS). We present these results in Table 4. The table shows the F1 Score achieved by our FlowTransformer-based model on these additional three dataset. For comparison, we also included the corresponding results for the other three datasets, as already shown in Table 3. We can observe that across all datasets, our FlowTransformer-based model performs very well, with UNSW-NB15 the only dataset with a sub 97% performance. The ease in which we were able to perform experiments on these additional datasets with varying formats and feature sets demonstrates the flexibility of FlowTransformer. Information and examples of how FlowTransformer can be used to ingest and process flow data of different formats is provided on our GitHub page (Manocchio, 2023).

### *8.6. Comparison to other transformer models in the literature*

While FlowTransformer is designed as a versatile framework rather than a definitive solution for NIDS systems, we offer comparisons with various other transformer-based solutions in the NIDS domain to our best FlowTransformer based model for the NSL-KDD dataset. (Liu & Wu, 2023) applied 4 state-of-the-art transformer models to the NSL-KDD dataset, these models were the RTIDS (Wu et al., 2022), VIT (Yang et al., 2022), CNN-Transformer (Zhang & Wang, 2022) as well as their proposed transformer model (Liu & Wu, 2023), which we have discussed in our literature review. We show these results, alongside our FlowTransformer-based model. In the original RTIDS and CNN-Transformer works, the authors of the respective papers applied their models to the CSE-CIC-IDS dataset. In Table 5, we report the respective F1 scores and training times, in comparison to the numbers of our best FlowTransformer model.

As we can see in the table, FlowTransformer outperforms the considered state-of-the-art approaches for the NSL-KDD dataset with an F1 score of 98%. For the CSE dataset, FlowTransformer achieved an F1 score of 97.05%, which comes reasonably close to the best performing model (RTIDS) which achieved 99.17%.

In terms of training time, we can see that our system required only 25.5 s, which is almost 8 times faster than RTIDS, which required 195 s to train. The difference to CNN-Transformer is even greater, with its 7,000 s of training time.

The reduced complexity of our FlowTransformer-based model enables a significantly reduced training time compared to the considered related works. For this comparison, it is obviously important to consider the hardware platforms on which the experiments were conducted. RTIDS utilised an Nvidia RTX 2080 GPU, which is the most powerful of the GPUs in this comparison. In our experiments, we used an Nvidia RTX 2070 Max-Q, which is comparatively weaker than the RTX 2080 (Nvidia, 2022). Finally, CNN-Transformer used a GTX 1050, which is comparatively the oldest and weakest card (Nvidia, 2022). FlowTransformer was nearly an order of magnitude faster than RTIDS, despite RTIDS using a more powerful GPU, and more than two orders of magnitude faster than CNN-Transformer. For the latter case, only a fraction of the difference can be attributed to the relatively minor variation in GPU speeds.

## 9. Discussion and limitations

One important consideration for NIDS is transferability. Performance of an NIDS can degrade considerably when trained on dataset from different networks (Layeghy & Portmann, 2023), as there is often differences in dataset distribution (Layeghy et al., 2023). Extracting domain invariant features using adversarial domain adaptation from multiple network domains has been shown to work on NIDS benchmark

**Table 6**
NSL-KDD dataset, same format as Table 2.

| Classification head | Input encoding | Parameters | Train time | Inference time | F1 score | Detection rate | False alarm rate |
|---|---|---|---|---|---|---|---|
| Last token | Categorical emb. dense | 397,225 | 1,749 | 7,111 | 0.9704 | 95.77% | 1.02% |
| | Categorical emb. lookup | 397,113 | 455 | 8,394 | 0.9785 | 98.06% | 1.31% |
| | Categorical projection | 397,113 | 1,707 | 8,097 | 0.9691 | 96.76% | 1.57% |
| | Record emb. dense | 180,545 | 1,906 | 8,533 | 0.9763 | 97.57% | 1.68% |
| | Record projection | 180,481 | 1,701 | 7,314 | 0.9765 | 97.68% | 1.29% |
| | No input encoding | 239,649 | 1,861 | 8,258 | 0.9761 | 97.23% | 1.23% |
| Flatten | Categorical emb. dense | 528,041 | 1,692 | 8,127 | 0.9676 | 98.45% | 1.05% |
| | Categorical emb. lookup | 527,929 | 458 | 7,211 | 0.9768 | 99.31% | 1.52% |
| | Categorical projection | 527,929 | 1,705 | 6,244 | 0.9662 | 95.85% | 2.05% |
| | Record emb. dense | 237,889 | 1,872 | 8,000 | 0.9772 | 97.98% | 1.36% |
| | Record projection | 237,825 | 1,914 | 7,212 | 0.9744 | 97.98% | 1.34% |
| | No input encoding | 318,497 | 1,850 | 9,660 | 0.9725 | 98.20% | 0.89% |
| Featurewise emb. | Categorical emb. dense | 379,708 | 1,713 | 7,530 | 0.9522 | 93.14% | 2.36% |
| | Categorical emb. lookup | 379,708 | 473 | 7,758 | 0.963 | 94.98% | 1.96% |
| | Categorical projection | 379,596 | 1,696 | 7,211 | 0.9597 | 95.66% | 2.41% |
| | Record emb. dense | 343,810 | 1,828 | 7,758 | 0.9636 | 95.10% | 2.18% |
| | Record projection | 173,378 | 1,873 | 7,758 | 0.9626 | 95.38% | 1.52% |
| | No input encoding | 229,498 | 1,904 | 8,127 | 0.9669 | 95.74% | 1.39% |
| Featurewise projection | Categorical emb. dense | 379,707 | 1,739 | 7,111 | 0.9599 | 95.32% | 0.66% |
| | Categorical emb. lookup | 379,707 | 450 | 9,309 | 0.9718 | 98.25% | 2.12% |
| | Categorical projection | 379,595 | 1,740 | 8,000 | 0.9568 | 93.72% | 1.55% |
| | Record emb. dense | 343,809 | 1,836 | 8,259 | 0.9648 | 96.07% | 1.02% |
| | Record projection | 173,377 | 1,899 | 7,877 | 0.9628 | 95.55% | 2.20% |
| | No input encoding | 229,497 | 1,909 | 8,258 | 0.9568 | 95.02% | 2.68% |
| Global average pooling | Categorical emb. dense | 397,225 | 1,620 | 7,420 | 0.6592 | 78.39% | 35.53% |
| | Categorical emb. lookup | 397,113 | 454 | 7,014 | 0.6591 | 74.50% | 27.95% |
| | Categorical projection | 397,113 | 1,724 | 7,642 | 0.654 | 74.46% | 31.17% |
| | Record emb. dense | 359,041 | 1,726 | 7,014 | 0.6535 | 75.59% | 38.49% |
| | Record projection | 180,481 | 1,904 | 7,211 | 0.6471 | 69.54% | 37.16% |
| | No input encoding | 239,649 | 1,809 | 8,000 | 0.6478 | 70.45% | 23.90% |
| CLS token | Categorical emb. dense | 399,925 | 1,658 | 7,529 | 0.6652 | 84.03% | 29.52% |
| | Categorical emb. lookup | 399,813 | 473 | 8,258 | 0.66 | 76.39% | 21.51% |
| | Categorical projection | 399,813 | 1,648 | 7,877 | 0.667 | 94.08% | 34.03% |
| | Record emb. dense | 361,741 | 1,652 | 8,533 | 0.6677 | 79.39% | 36.58% |
| | Record projection | 183,181 | 1,602 | 8,258 | 0.6612 | 78.89% | 35.90% |
| | No input encoding | 242,349 | 1,525 | 7,877 | 0.6652 | 90.37% | 41.59% |

datasets of NFv2-CIC-2018 and NFv2-UNSW-NB15 with a One-Class SVM (Layeghy et al., 2023). Such an approach with a transformer based model can be readily applied via the FlowTransformer with modification to the training loop, as discussed in (Layeghy et al., 2023).

In the wider context of machine learning, there are several important concepts and challenges such as transferability, resistance to concept drift, explainability, requirements for labelling and training in an unsupervised manner and other model assumptions that must be considered. While these issues are highly relevant in the context of ML-based NIDS, we consider them largely orthogonal to our work, and therefore out of scope of this paper. At this stage, the FlowTransformer framework does not address these issues 'out of the box'.

However, as a flexible framework, we believe that FlowTransformer provides a platform that can facilitate research into these important aspects. In our GitHub repository (Manocchio, 2023), we have provided guidance and examples on how FlowTransformer can be used to explore the concepts of transferability, concept drift, and temporal bias as part of the 'extended_concepts' Jupyter notebook.

It is important to note the limitations of FlowTransformer. For a range of important challenges in the context of ML-based NIDS, such as explainability and open-world applications of models, FlowTransformer does not provide a solution.

## 10. Conclusion

This paper presents FlowTransformer, a modular framework that allows the rapid implementation and evaluation of transformers for NIDS. The framework supports the combination of a variety of common transformer components, including input encodings, classification heads, and transformer constructions, and enables the formulation of training and inference pipelines for flow-based networking data, specifically in the context of NIDS.

We have demonstrated the utility and power of FlowTransformer via an extensive and systematic evaluation of a wide range of transformer model configurations, across 3 widely-used NIDS benchmark datasets.

We also specifically explored the feasibility of GPT 2.0 and GPT 3.0-based technologies for NIDS, together with size and performance trade-offs across three common NIDS benchmark datasets. The key findings from our experimental evaluation are:

- The choice of classification head was the most critical factor in model performance, and the best choice of classification head was 'Last Token'.
- Using record level embedding or projection, can reduced the model size by more than half, without reducing classification performance. Record level approaches were also the fastest in terms of both inference and training time.
- Transformer models with a small number of layers are sufficient for certain NIDS tasks, including those present in the benchmark datasets we tested.
- Transformer encoders and decoder blocks were both effective, but given the smaller size and higher applicability of encoder blocks, these are the best choice.

Finally, we compared the performance of shallow transformer models, with that of the larger GPT and BERT architecture for the task of NIDS classification. We showed that although the larger models were able to reach an equivalent level of performance as the shallower models, because of the increased size and lower model throughput, these are likely not the optimal choices for most NIDS tasks.

**Table 7**
UNSW-NB15 dataset, same format as Table 2.

| Classification head | Input encoding | Parameters | Train time | Inference time | F1 score | Detection rate | False alarm rate |
|---|---|---|---|---|---|---|---|
| Last token | Categorical emb. dense | 470,433 | 1,566 | 8,000 | 0.9021 | 99.82% | 1.06% |
| | Categorical emb. lookup | 470,289 | 400 | 6,919 | 0.8963 | 99.93% | 1.14% |
| | Categorical projection | 470,289 | 1,596 | 7,529 | 0.9011 | 99.64% | 1.06% |
| | Record emb. dense | 382,081 | 1,630 | 7,877 | 0.9045 | 99.89% | 1.03% |
| | Record projection | 192,001 | 1,563 | 8,127 | 0.8957 | 99.86% | 1.11% |
| | No input encoding | 725,649 | 1,587 | 8,827 | 0.9019 | 99.82% | 1.04% |
| Flatten | Categorical emb. dense | 624,545 | 1,498 | 8,258 | 0.8936 | 99.75% | 1.13% |
| | Categorical emb. lookup | 624,401 | 381 | 9,143 | 0.8975 | 99.96% | 1.13% |
| | Categorical projection | 624,401 | 1,573 | 7,014 | 0.891 | 99.89% | 1.17% |
| | Record emb. dense | 249,409 | 1,786 | 9,309 | 0.8997 | 99.72% | 1.07% |
| | Record projection | 249,345 | 1,723 | 8,534 | 0.9043 | 99.89% | 1.04% |
| | No input encoding | 965,777 | 1,517 | 5,885 | 0.9032 | 99.68% | 1.03% |
| Featurewise emb. | Categorical emb. dense | 449,614 | 1,522 | 7,421 | 0.8907 | 99.43% | 1.17% |
| | Categorical emb. lookup | 449,614 | 401 | 7,529 | 0.8936 | 99.01% | 0.02% |
| | Categorical projection | 449,470 | 1,556 | 7,014 | 0.8976 | 97.98% | 0.02% |
| | Record emb. dense | 366,850 | 1,685 | 8,678 | 0.8899 | 99.72% | 1.21% |
| | Record projection | 184,898 | 1,667 | 8,127 | 0.8885 | 99.54% | 1.20% |
| | No input encoding | 692,638 | 1,563 | 6,827 | 0.8963 | 99.61% | 1.11% |
| Featurewise projection | Categorical emb. dense | 449,613 | 1,504 | 7,641 | 0.8957 | 99.75% | 1.13% |
| | Categorical emb. lookup | 449,613 | 405 | 7,758 | 0.8997 | 98.90% | 1.02% |
| | Categorical projection | 449,469 | 1,558 | 7,420 | 0.8982 | 99.36% | 1.05% |
| | Record emb. dense | 366,849 | 1,674 | 8,000 | 0.8877 | 99.79% | 1.23% |
| | Record projection | 184,897 | 1,545 | 8,982 | 0.8794 | 99.33% | 1.31% |
| | No input encoding | 692,637 | 1,575 | 8,393 | 0.8932 | 99.79% | 1.05% |
| Global average pooling | Categorical emb. dense | 470,433 | 1,490 | 8,394 | 0.2307 | 98.90% | 30.36% |
| | Categorical emb. lookup | 470,289 | 400 | 9,309 | 0.2293 | 99.82% | 22.50% |
| | Categorical projection | 470,289 | 1,574 | 8,533 | 0.2401 | 99.47% | 12.27% |
| | Record emb. dense | 192,065 | 1,701 | 9,661 | 0.2294 | 98.93% | 31.54% |
| | Record projection | 192,001 | 1,751 | 8,828 | 0.2356 | 99.72% | 26.91% |
| | No input encoding | 725,649 | 1,581 | 6,563 | 0.2302 | 99.29% | 21.20% |
| CLS token | Categorical emb. dense | 473,133 | 1,436 | 6,481 | 0.2285 | 99.54% | 31.47% |
| | Categorical emb. lookup | 472,989 | 386 | 8,393 | 0.228 | 99.75% | 4.72% |
| | Categorical projection | 472,989 | 1,340 | 8,393 | 0.2301 | 99.61% | 32.24% |
| | Record emb. dense | 384,781 | 1,511 | 7,014 | 0.2286 | 99.75% | 32.14% |
| | Record projection | 194,701 | 1,639 | 8,678 | 0.2297 | 99.86% | 32.63% |
| | No input encoding | 728,349 | 1,513 | 8,000 | 0.2291 | 98.83% | 30.29% |

We believe FlowTransformer can provide a basis for further explorations of the great potential of transformer-based approaches in the specific context of network intrusion detection, and possibly beyond that. For further exploration, it would be exciting to apply FlowTransformer with the consideration of explainability and other extended concepts in mind.

## CRediT authorship contribution statement

**Liam Daly Manocchio:** Conceptualization, Methodology, Software, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Siamak Layeghy:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Wai Weng Lo:** Writing – original draft. **Gayan K. Kulatilleke:** Writing – review & editing. **Mohanad Sarhan:** Writing – review & editing. **Marius Portmann:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

We have used public datasets and have made our code available publicly.

## Appendix A

See Tables 6 and 7.

## Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2023.122564.

## References


Aitken, P., Claise, B., & Trammell, B. (2013). *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*: *RFC No. 7011*, IETF, https://datatracker.ietf.org/doc/rfc7011/.

Delgadillo, K., & Marketing, C. I. P. (1996). Netflow services and applications, Cisco Whitepaper.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference, vol. 1* (pp. 4171–4186). Association for Computational Linguistics (ACL), https://arxiv.org/abs/1810.04805v2.

Han, X., Cui, S., Liu, S., Zhang, C., Jiang, B., & Lu, Z. (2023). Network intrusion detection based on n-gram frequency and time-aware transformer. *Computers & Security*, Article 103171. http://dx.doi.org/10.1016/j.cose.2023.103171, https://www.sciencedirect.com/science/article/pii/S0167404823000810.

Hassan, M., Gumaei, A., Alsanad, A., Alrubaian, M., & Fortino, G. (2019). A hybrid deep learning model for efficient intrusion detection in big data environment. *Information Sciences*, *513*, http://dx.doi.org/10.1016/j.ins.2019.10.069.

Hindy, H., Bayne, E., Bures, M., Atkinson, R. C., Tachtatzis, C., & Bellekens, X. J. A. (2020). Machine learning based IoT intrusion detection system: An MQTT case study. CoRR, https://arxiv.org/abs/2006.15340.

Huang, X., Khetan, A., Cvitkovic, M., & Karnin, Z. (2020). Tabtransformer: Tabular data modeling using contextual embeddings. https://arxiv.org/abs/2012.06678.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980.

Ko, B., Kim, H.-G., Heo, B., Yun, S., Chun, S., Gu, G., & Kim, W. (2022). Group generalized mean pooling for vision transformer. https://arxiv.org/abs/2212.04114.

Kumar, S., Gupta, S., & Arora, S. (2021). Research trends in network-based intrusion detection systems: A review. *IEEE Access*, *9*, 157761–157779.

Layeghy, S., Baktashmotlagh, M., & Portmann, M. (2023). DI-NIDS: Domain invariant network intrusion detection system. *Knowledge-Based Systems*, *273*, Article 110626.

Layeghy, S., & Portmann, M. (2023). Explainable cross-domain evaluation of ML-based network intrusion detection systems. *Computers & Electrical Engineering*, *108*, Article 108692.

Li, Y., Yuan, X., & Li, W. (2022). An extreme semi-supervised framework based on transformer for network intrusion detection. In *Proceedings of the 31st ACM international conference on information & knowledge management* (pp. 4204–4208).

Liu, L., Engelen, G., Lynar, T., Essam, D., & Joosen, W. (2022). Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018. In *2022 IEEE conference on communications and network security* (pp. 254–262). IEEE.

Liu, Y., & Wu, L. (2023). Intrusion detection model based on improved transformer. *Applied Sciences*, *13*(10), 6251.

Manocchio, L. D. (2023). FlowTransformer, the framework for transformer based NIDS development. GitHub repository, https://github.com/liamdm/FlowTransformer.

Manocchio, L. D., Layeghy, S., & Portmann, M. (2022). Network intrusion detection system in a light bulb. In *2022 32nd international telecommunication networks and applications conference* (pp. 1–8). http://dx.doi.org/10.1109/ITNAC55475.2022.9998371.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, *26*.

Moustafa, N. (2021). A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustainable Cities and Society*, *72*, Article 102994.

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems. In *2015 military communications and information systems conference* (pp. 1–6). IEEE.

Nam, M., Park, S., & Kim, D. S. (2021). Intrusion detection method using bi-directional GPT for in-vehicle controller area networks. *IEEE Access*, *9*, 124931–124944.

Nguyen, L. G., & Watabe, K. (2022). Flow-based network intrusion detection based on BERT masked language model. In *Proceedings of the 3rd international CoNEXT student workshop* (pp. 7–8).

Nvidia (2022). *NVIDIA TURING GPU ARCHITECTURE graphics reinvented NVIDIA turing GPU architecture*: *(tech. rep.)*, https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

OpenAI (2019). GPT-2 1.5B release. https://openai.com/research/gpt-2-1-5b-release/. (Accessed 20 April 2023).

OpenAI (2022). Introducing ChatGPT. https://openai.com/blog/chatgpt. (Accessed 16 March 2023).

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., & Tran, D. (2018). Image transformer. In *International conference on machine learning* (pp. 4055–4064). PMLR.

Phaal, S., Panchen, S., & McKee, N. (2001). InMon Corporation's sFlow: A method for monitoring traffic in switched and routed networks. *RFC No. 3176*, IETF, https://www.ietf.org/rfc/rfc3176.txt.

Ridnik, T., Sharir, G., Ben-Cohen, A., Ben-Baruch, E., & Noy, A. (2023). ML-decoder: Scalable and versatile classification head. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (pp. 32–41).

Sarhan, M., Layeghy, S., & Portmann, M. (2022). Towards a standard feature set for network intrusion detection system datasets. *Mobile Networks and Applications*, 1–14. http://dx.doi.org/10.1007/s11036-021-01843-0.

Schultz, M. J., & Crowley, P. (2012). Performance analysis of packet capture methods in a 10 gbps virtualized environment. In *2012 21st international conference on computer communications and networks* (pp. 1–8). IEEE.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSp*: *vol. 1*, (pp. 108–116).

Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., & Arikawa, S. (1999). *Byte Pair encoding: A text compression scheme that accelerates pattern matching*: *Technical Report DOI-TR-161*, Department of Informatics, Kyushu University.

Sikora, M., Gerlich, T., & Malina, L. (2019). On detection and mitigation of slow rate denial of service attacks. In *2019 11th international congress on ultra modern telecommunications and control systems and workshops* (pp. 1–5). IEEE.

Singh, G., & Khare, N. (2022). A survey of intrusion detection from the perspective of intrusion datasets and machine learning techniques. *Int. J. Comput. Appl.*, *44*(7), 659–669.

Sorzano, C. O. S., Vargas, J., & Montano, A. P. (2014). A survey of dimensionality reduction techniques. arXiv:1403.2877.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1–6). http://dx.doi.org/10.1109/CISDA.2009.5356528.

Ullah, F., Ullah, S., Srivastava, G., & Lin, J. C.-W. (2023). IDS-INT: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic. *Digital Communications and Networks*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Walling, S., & Lodh, S. (2022). A survey on intrusion detection systems: Types, datasets, machine learning methods for NIDS and challenges. In *2022 13th international conference on computing communication and networking technologies* (pp. 1–7). IEEE.

Wang, W., Jian, S., Tan, Y., Wu, Q., & Huang, C. (2023). Robust unsupervised network intrusion detection with self-supervised masked context reconstruction. *Computers & Security*, Article 103131.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., .... Rush, A. M. (2020). HuggingFace's transformers: State-of-the-art natural language processing. arXiv:1910.03771.

Wu, Z., Zhang, H., Wang, P., & Sun, Z. (2022). RTIDS: A robust transformer-based approach for intrusion detection system. *IEEE Access*, *10*, 64375–64387.

Yang, Y.-G., Fu, H.-M., Gao, S., Zhou, Y.-H., & Shi, W.-M. (2022). Intrusion detection: A model based on the improved vision transformer. *Transactions on Emerging Telecommunications Technologies*, *33*(9), Article e4522.

Yun, S., Jeong, M., Kim, R., Kang, J., & Kim, H. J. (2019). Graph transformer networks. *Advances in Neural Information Processing Systems*, *32*.

Zhang, Z., & Wang, L. (2022). An efficient intrusion detection model based on convolutional neural network and transformer. In *2021 ninth international conference on advanced cloud and big data* (pp. 248–254). IEEE.