3RD WORKSHOP ON MACHINE LEARNING IN NETWORKING (MaLeNe) PROCEEDINGS

SEPTEMBER 1, 2025

CO-LOCATED WITH
THE 6TH INTERNATIONAL CONFERENCE ON NETWORKED SYSTEMS (NETSYS 2025)
ILMENAU, GERMANY

3rd Workshop on Machine Learning in Networking (MaLeNe)

The Third International Workshop on Machine Learning in Networking (MaLeNe, https://www.uni-augsburg.de/en/fakultaet/fai/informatik/prof/netcom/events/malene2025/) was a successful half day event held at Technical University of Ilmenau on September 1, 2025, where it was co-located with the Conference on Networked Systems (NetSys 2025). It follows the history of MaLeNe KuVS workshops (Fachgespräch) and MaLeNe NetSys workshops as the 6th edition of the MaLeNe series, and was organized by workshop co-chairs Michael Seufert (University of Augsburg, Germany), Andreas Blenk (Siemens AG, Germany), and Björn Richerzhagen (Siemens AG, Germany). The workshop accepted 7 full papers for presentation.

On the day of the workshop, the co-chairs welcomed 34 registered participants. The workshop started with an industry keynote given by Philippe Buschmann (Siemens AG, Germany) who elaborated on "Recent AI Trends in Industrial Network Environments". He explored industrial AI and its application in smart mobility, smart infrastructure, and manufacturing. He especially highlighted the challenges and opportunities of AI adoption in these settings through the perspective of Siemens.

Afterwards, the first technical session started. Katharina Dietz (University of Würzburg, Germany) evaluated the consensus of XAI -based network intrusion detection and how to improve consensus with feature selection methods. Nasim Nezhadsistani (University of Zurich, Switzerland) presented an asynchronous consensus-driven multi-agent approach to decentralized federated learning for intrusion detection in 5G networks. Zineddine Bettouche (Deggendorf Institute of Technology, Germany) discussed spatiotemporal machine learning techniques for cellular traffic forecasting with a particular focus on Mamba deep learning architectures.

After the coffee break, Christian Maier (Salzburg Research, Austria) elaborated on predicting performance metrics in edge-cloud networks using GNNs. Timothy Harrison (University of Hagen, Germany) presented work on state cloning for high order Markov chains with the GraphLearner. Yanakorn Ruamsuk (University of Hagen, Germany) talked about emotion-controlled communication in agent networks. Finally, Alexander Niedermayer (Karlsruhe University of Applied Sciences, Germany) presented an approach for client-agnostic continuous authentication via keystroke-induced traffic patterns.

The workshop co-chairs closed the day with a short recap and thanked all speakers and participants who engaged in the fruitful discussions. As the workshop has proven to foster active collaborations in the research community, another edition will be considered in the future.

We would like to thank all the authors, reviewers, and attendants for their precious contributions towards the successful organization of the workshop!

Michael Seufert, Andreas Blenk, Björn Richerzhagen
MaLeNe 2025 Workshop Co-Chairs

Program

Welcome and Workshop Opening

Keynote

Recent AI Trends in Industrial Network Environments, Philippe Buschmann (Siemens AG)

Session 1: Machine Learning for Intrusion Detection and Traffic Forecasting

1. *I Choose You: Evaluating the Impact of Feature Selection on XAI Consensus for ML-NIDS*
Katharina Dietz (University of Würzburg), Johannes Schleicher (University of Augsburg), Nikolas Wehner (University of Würzburg), Mehrdad Hajizadeh (Technical University of Chemnitz), Pedro Casas (AIT Austrian Institute of Technology), Stefan Geißler (University of Würzburg), Michael Seufert (University of Augsburg) and Tobias Hossfeld (University of Würzburg)

2. *Decentralized Federated Learning for Intrusion Detection in 5G Networks: An Asynchronous Consensus driven Multi-Agent Approach*
Nasim Nezhadsistani (University of Zurich), Francisco Enguix (Polytechnic University of Valencia), Carlos Carrascosa (Polytechnic University of Valencia) and Burkhard Stiller (University of Zurich)

3. *HiSTM: Hierarchical Spatiotemporal Mamba for Cellular Traffic Forecasting*
Zineddine Bettouche, Khalid Ali, Andreas Fischer and Andreas Kassler (Deggendorf Institute of Technology)

Session 2: Novel Machine Learning Approaches and Applications in Networking

4. *Predicting Performance Metrics in Edge-Cloud Networks using Graph Neural Networks*
Christian Maier, Nina Großegesse and Felix Strohmeier (Salzburg Research)

5. *State Cloning with the GraphLearner*
Timothy Harrison and Herwig Unger (University of Hagen)

6. *Emotion-Controlled Communication in Agent Networks*
Yanakorn Ruamsuk and Herwig Unger (University of Hagen)

7. *Client-Agnostic Continuous Authentication via Keystroke-Induced Traffic Patterns*
Alexander Niedermayer, David Monschein and Oliver Waldhorst (Karlsruhe University of Applied Sciences)

Wrap-up and Closing Remarks

I Choose You: Evaluating the Impact of Feature Selection on XAI Consensus for ML-NIDS

Katharina Dietz*, Johannes Schleicher§, Nikolas Wehner*, Mehrdad Hajizadeh†,
Pedro Casas‡, Stefan Geißler*, Michael Seufert§, Tobias Hoßfeld*
*University of Würzburg, Germany, †Technical University of Chemnitz, Germany
‡AIT Austrian Institute of Technology, Vienna, Austria, §University of Augsburg, Germany
*{katharina.dietz, nikolas.wehner, stefan.geissler, tobias.hossfeld}@uni-wuerzburg.de,
†mehrdad.hajizadeh@etit.tu-chemnitz.de, ‡pedro.casas@ait.ac.at, §{johannes.schleicher, michael.seufert}@uni-a.de

***Abstract*—Machine learning-based network intrusion detection systems (ML-NIDS) are increasingly enhanced with explainable AI (XAI) techniques to support transparency and trust in automated security decisions. However, recent studies have shown that different post-hoc XAI methods often yield inconsistent explanations. These variations depended on the dataset and underlying model, and were possibly caused by training the ML models on correlated features. In this work, we investigate the hypothesis that feature selection prior to model training can influence the level of consensus among XAI methods. Through a comprehensive evaluation across multiple datasets, we analyze the impact of different feature selection strategies on explanation agreement. While we found that feature selection can improve XAI consistency in controlled synthetic settings, its effects on real-world NIDS data are mixed: occasionally enhancing, but sometimes reducing consensus, while offering only modest gains over using all features. These insights highlight the importance of thoughtful feature selection to improve interpretability and consistency in XAI-driven network intrusion detection systems.**

***Index Terms*—Machine Learning, Intrusion Detection, Explainable AI, Disagreement Problem, Feature Selection.**

I. Introduction

The rapid evolution of data networks has revolutionized modern life by enabling seamless communication, automation, and large-scale data exchange. However, this increased connectivity has also expanded the attack surface, offering more opportunities for cyber adversaries. According to the European Union Agency for Cybersecurity (ENISA), there has been a marked increase in the frequency, diversity, and impact of cyber attacks [1]. Adversaries are now exploiting automation and artificial intelligence (AI) to design more evasive attack strategies [2], [3]. These developments have exposed the limitations of traditional security mechanisms, especially signature-based detection methods, which depend on predefined patterns and struggle with evolving threats [4], thereby prompting the emergence of machine learning (ML)-based network intrusion detection systems (NIDS) as promising tools to identify malware and network attacks [3], [5]–[7].

Despite these advances, the integration of ML into computer security still often encounters mistrust and skepticism [8], [9], not least due to the lack of explainability [10]. The opaque nature of many AI models limits their practical deployment, as security analysts must be able to interpret and trust the decisions made by automated systems [3], [7], [10]. In cyber-security, errors or blind trust in automated decisions can have severe consequences, potentially endangering infrastructure, privacy, and even human safety [3], [11]. While explainable AI (XAI) has emerged to demystify ML behavior, current XAI methods often produce divergent explanations, which creates confusion rather than clarity [10], [12], [13]. Thus, to ensure actionable insights for security professionals, it is essential to establish consensus among different explanation techniques. Furthermore, the European Union's General Data Protection Regulation (GDPR) reinforces this through its "right to explanation" for decisions made by algorithms [14].

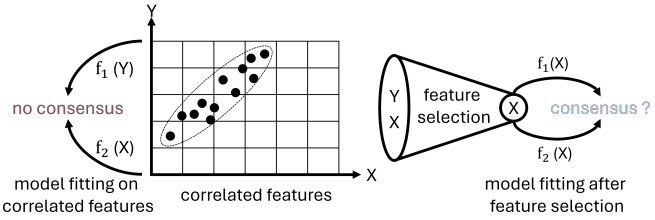


Fig. 1: Feature selection for decorrelation.

A detailed analysis on the consensus of XAI methods for ML-NIDS [13] revealed that the level of agreement between different post-hoc XAI approaches, varied significantly depending on the dataset and, in some cases, the underlying model. Disagreements might stem from the selection of related or correlated features, indicating that multiple, seemingly divergent explanations could still be valid. Figure 1 exemplifies this phenomenon, where two explainers, f_1 and f_2, select different features, X and Y, to explain the same decision. Although these features encode similar characteristics, their disagreement results in a lack of consensus. When applying feature selection (FS), both explainers may agree on the same feature. This suggests that FS before model training might play a crucial role w.r.t. consensus, and that we can potentially improve the consensus by using the right FS method.

In this work, we conduct extensive experiments on both synthetic and real-world NIDS datasets, using six different FS strategies and two widely used post-hoc XAI methods. To the best of our knowledge, this is the first systematic analysis of the relationship between FS and XAI disagreement by bringing the disagreement problem into the context of NIDS. We highlight both the potential and limitations of improving

explanation consistency, and more broadly, the challenges of relying on post-hoc XAI for trustworthy interpretations.

The remainder of this paper is structured as follows: Section II provides information on intrusion detection, XAI, FS, and related works. Section III outlines the used datasets and ML workflow for analyzing consensus. Section IV presents the obtained results, and finally, Section V summarizes the key findings and contributions of this study.

II. Background and Related Work

A. ML-NIDS

In network security, an intrusion implies that the *confidentiality*, *integrity*, or *availability* of network resources (e. g., devices, data) is being compromised [15]. Thus, intrusion detection systems (IDS) are employed to detect such attacks and enable further steps for their containment. IDS can be deployed at different vantage points, e. g., in global locations to monitor the network traffic on a large scale (NIDS) or directly on the host (HIDS) to locally investigate malicious programs, files, etc. The former is often based on features extracted from coarse-grained flows (i. e., aggregates based on the 5-tuple of IPs, ports, and protocol such as NetFlow) or directly on the packets for more fine-grained monitoring.

Research has shifted towards ML-NIDS, since recent trends such as 5G/6G, Internet of Things (IoT), and Industry 4.0 have increased attack surfaces, necessitating more sophisticated countermeasures. Though, ML-based solutions are often perceived as more complex than traditional solutions (e. g., based on signatures or rules) [16] and deemed less trustworthy, particularly in sensitive areas such as cybersecurity. So, techniques that provide model insights should be adopted, especially regarding the GDPR [14] and European AI Act [17].

B. Explainable AI

The aforementioned challenges have led to the development of XAI to provide insights into the decisions of ML models, either by utilizing white-box models that are interpretable by design or by utilizing post-hoc explainers, which explain the output of black-box models [10]. XAI methods can be classified in various ways, e. g., w.r.t. model compatibility or algorithm type [18]. In this work, we focus on two of the most prominent post-hoc XAI representatives: LIME (Local Interpretable Model-agnostic Explanations) [19] and SHAP (SHapley Additive exPlanations) [20]. Generally, both of them are compatible with any ML model and work via *perturbation*, i. e., masking or obfuscating input features of a data sample to determine their influence on the model's decision. That is, LIME builds a linear (therefore interpretable) surrogate model by learning the output of the original model by adding noise to each input feature. SHAP, on the other hand, is based on Shapley values [21] from game theory. This concept assigns feature contributions by evaluating all possible feature subsets.

Even though XAI has gained increasing popularity, new challenges arise, e. g., in form of the *disagreement problem* [12]. This problem stems from the fact that produced explanations by the various explainers often differ, sometimes even contradict. This has been observed for use cases like network security [10], [13], [22]–[24], but also areas outside of communication networks [12], [25], [26]. Feature interactions, such as correlation or other relationships, are often cited to be a contributing factor to this phenomenon [13], [25]–[30]. Naturally, redundant features enable multiple valid explanations.

C. Feature Selection

Selecting only a relevant feature subset is a standard process in many ML workflows, since fewer features not only improve training and inference time, but also reduce overfitting [18]. They also improve interpretability, since less features have to be interpreted. Here, reducing the number of features may help improve XAI consensus twofold. One, the explainers have less choices to choose from in general, and two, depending on the FS mechanism, feature interactions may be reduced.

Similar to XAI, FS aims to identify the top features. This is a preprocessing step (i. e., before model training), while post-hoc XAI is applied after model training. Stańczyk [31] (as well as Khani et al. [18]) groups FS into three groups: *filters*, *wrappers*, and *embedded* techniques, many of which are implemented in scikit-learn [32]. One of the most well-known selection techniques is impurity-based FS that leverages the structure of tree-based models. This is an *embedded* technique, since it makes use of the internals of a pretrained model. It quantifies how much a feature reduces the "impurity" at each split across all trees w.r.t. the mix of class labels.

Instead of making use of model internals, *wrappers* utilize a classifier to evaluate different feature (sub)sets and their "usefulness in classification" [31, p. 32]. That is, they observe how the model's performance (e. g., accuracy) changes under different conditions. For example, permutation importance shuffles the value of a feature. Alternatively, recursive feature elimination starts with a full feature set and iteratively prunes the feature with the least impact. Similarly, backward sequential FS works accordingly, while the forward variant starts from an empty feature set.

Lastly, *filters* work separately from any ML model and observe the relationship between a feature and the class label, e. g., via ANOVA F-values [18], as implemented by default by sklearn's *SelectKBest()*. Another approach clusters correlated features together by treating correlation as a similarity measure, before simply choosing one feature from each group [33].

D. Related Work

Many works on XAI-driven NIDS have already shifted their focus to a quantitative comparison of different explainers instead of merely using XAI to explain a decision, e. g., regarding an explanation's robustness or faithfulness to the ground truth [22], [34]–[38], and/or conduct qualitative studies with security admins to gain insights from practitioners [24]. Some works leverage XAI itself for FS (e. g., [18], [39]), whereas we view this as two separate steps in the ML pipeline. The disagreement problem is sometimes a partial factor in these works, but rarely a focal point. In contrast, our work focuses on investigating the XAI consensus in a more detailed manner.

Our goal goes beyond stating the existence of the disagreement problem, which we explored ourselves previously [13].

Besides works on XAI-NIDS, other research areas have put a more in-depth emphasis on the disagreement problem by investigating the impact of varying model parameters or steps in the ML workflow. One approach is limiting the scope of the explanation to *regional areas* [40], [41], i. e., adjusting the background datasets of explainers to be more locally relevant to the instance to explain. Instead of restricting the reference data, other works limit the actual input features via dimensionality reduction to reduce multicollinearity [13], [26], [42]. Lastly, other works analyze the reasons of the disagreement problem by controlling dataset parameters (e. g., features, samples, labels, noise, redundancy) [27], [28], while others explore the impact of different preprocessing techniques (e. g., scaling, encoding) [43], or influence of model parameters (e. g., training duration and loss functions) [30], [44]. Often, these works also make use of synthetic data, which is easier to configure. While there exist some works that make use of FS, as well as investigating feature interactions, our work specifically zeroes in on the differences between various selection methods. To the best of our knowledge, we are also the first to bring the disagreement problem into the NIDS domain in-depth (or monitoring in general).

III. Methodology

A. NIDS Datasets

In this work, we use three NIDS datasets of varying complexity and feature granularity: CICIDS2017 [45], CIDDS-001 [46], and Edge-IIoTset [47]. CICIDS2017, one of the most popular NIDS datasets in state-of-the-art literature [7], provides 77 flow-based features[1], e. g., statistical moments of packet sizes and IATs. We use the Wednesday subset with almost 700k samples and DoS/DDoS attacks. CIDDS-001 offers 14 features based on NetFlow, which is one of the most commonly used protocols in practice for traffic monitoring. We use the first week of the dataset (over 8M samples), which includes Pingscan, Portscan, Bruteforce, and DoS attacks. Note that we additionally derived flow IATs and number of parallel flows to enrich the feature set. Edge-IIoTset covers diverse IoT/IIoT protocols (e. g., TCP, MQTT, MODBUS) with 35 features for over 2M samples, including DDoS, Portscan, and other attacks. While we generally follow the authors' proposed preprocessing steps, we remove further features with limited generalizability (e. g., IPs, checksums, ACK numbers). Our code is available for reproducibility[2].

B. XAI Workflow

To ensure temporally coherent splits, we use sklearn's *StratifiedGroupKFold()*, where groups are defined as 30s time intervals w.r.t. each dataset's timestamp column[3], instead of simply shuffling the entire dataset randomly before splitting. We use three folds, ensuring that each sample appears during testing. Categorical features are one-hot-encoded, zero-variance features are filtered out, and features are minmax-scaled. The top ten features are selected via the six selection methods described in Section II-C: impurity-based, permutation-based, recursive, (forward) sequential, and correlation-based FS, as well as *SelectKBest()*. Similar subsets have been found useful in related work on NIDS [18], [39]. For FS methods that require a classifier, we utilize a lightweight Random Forest (RF; 10 trees, max. depth 10). After encoding and filtering, we balance the training data by selecting 250k samples of each class (benign, malign). For the actual classification task, we also use an RF (50 trees, max. depth 20) and a Multi-Layer-Perceptron with two layers (MLP; 64 neurons per layer, followed by ReLU). Both are commonly used in recent XAI [12], [30] and NIDS literature [7], giving insights for shallow ML and Deep Learning (DL).

For our explainers, we utilize the aforementioned LIME and SHAP. For the latter, we use the more efficient, model-specific implementations (TreeSHAP, DeepSHAP). To calculate the consensus between pairs of explanations (i. e., SHAP vs. LIME-based explanations), we use metrics similar to Krishna et al. [12]. We focus on two types: unordered (UC) and ordered consensus (OC) of the top 5 features. For the UC, we simply calculate the intersection of features. For the OC, we take the actual order of importance into account. In detail, we compute how many of the top features match in order until the first mismatch. Consequently, we are not interested if, e. g., only the fifth feature matches if previous features do not. We only count features as matching if their sign also matches.

C. Synthetic Data

We also utilize synthetic data to analyze the effect of FS in a configurable manner, for which we make use of sklearn's *make_classification()*, adapted from a benchmark data generator for a FS competition [48]. The algorithm has four feature types: *informative*, *redundant*, *repeated*, and *useless*. The *informative* features are the ones actually relevant to the prediction target, and the informativeness is split among all of them. The *redundant* features are linear combinations of other features, while *repeated* features are simple duplications. Last, *useless* features are just noise. For all feature types, we add new features (up to 50 extra, in increments of 10), select the top 10, and calculate the XAI consensus. Each experiment starts with five informative features. For each combo of type and number, we generate 250 balanced synthetic datasets of 1k samples, which we split in a 80:20 ratio.

IV. Evaluation

A. Preliminary Experiments on Synthetic Data

Before diving into the experiments of the NIDS datasets, we first want to analyze the impact of FS in a controllable fashion. Figure 2 illustrates the results of various experiments described previously. Each row of subfigures illustrates the four different feature types, while the columns of subfigures represent the two consensus metrics for all 200 test set samples, as well as the accuracy. Since the synthesized datasets are balanced,

[1] Before encoding, filtering etc. (for all datasets).

[2] https://github.com/lsinfo3/malene2025-xai-nids-feature-selection

[3] Edge-IIoTset has >100k samples with invalid timestamps, which we drop.

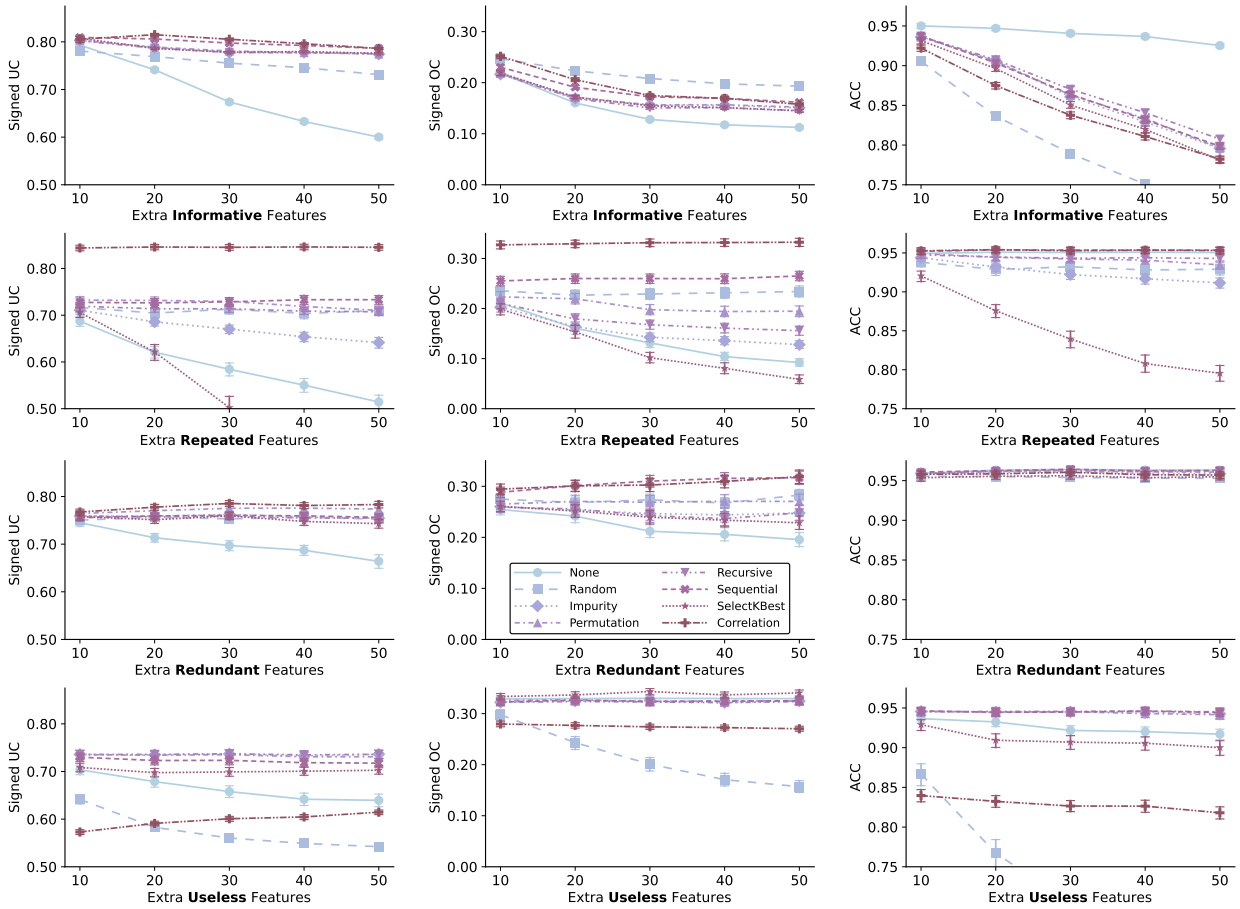
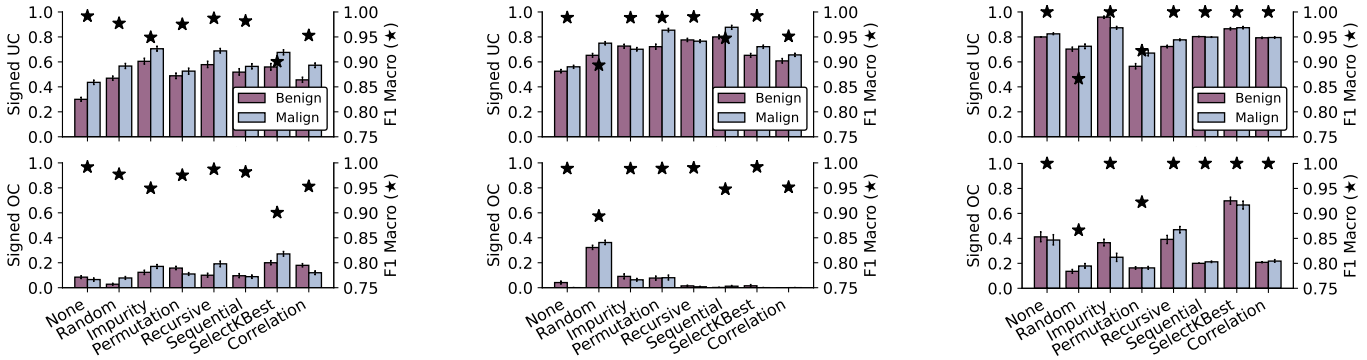


Fig. 2: Comparison of FS methods across feature types (rows) and (un)ordered consensus+accuracy (columns; UC/OC, ACC).

using accuracy is adequate here. The x-axis represents the extra added features (in addition to the five starting features), and the y-axis the respective metric. The different linestyles depict the different FS methods as well as results with all features ("None") and a random FS. Errorbars depict the 95% confidence intervals of the 250 runs. For the sake of brevity, results are shown for the RF as underlying model.

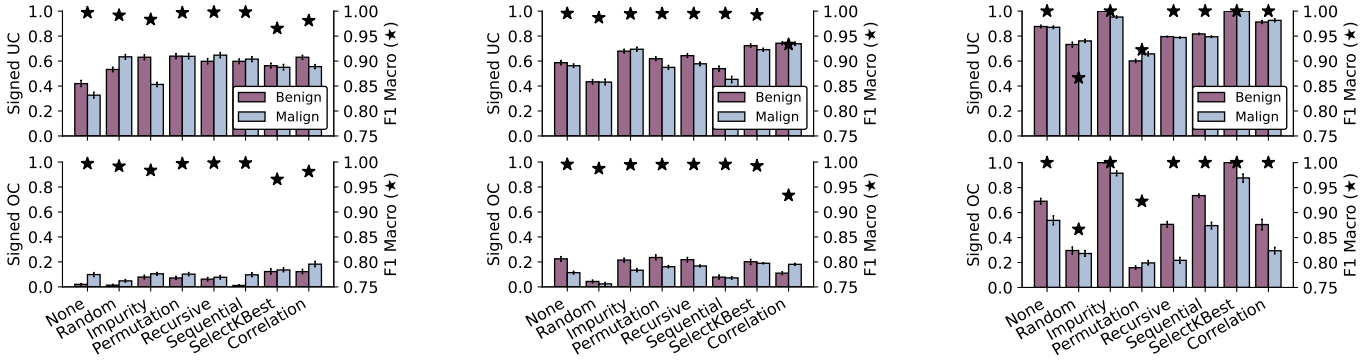
For the *informative* features in the first row, the UC stays almost constant for all FS methods, but drops significantly when no FS is applied. We can also observe a slight edge of the correlation-based and the sequential approach over the others. Since informative features are generated in a way so they do indeed contain covariance, taking this into account may help slightly. For the OC, we see a similar trend, though the difference between no FS and the rest is less severe and the consensus is generally lower. Interestingly, the random baseline comes out on top here. For the actual accuracy, however, we see that only keeping all features maintains model performance. This is expected, as we increase the number of informative features, which are all relevant to the classification, retaining only the top 10 is not sufficient anymore. We also see that while the correlation-based approach has a slight edge for the consensus, it slightly underperforms. We also see that, despite being best for the OC, the random baseline drastically underperforms, too. Our hypothesis is, that the OC may be increased since the random approach might have chosen one feature that may be most import, while the rest is not as descriptive (as reflected by the accuracy), making the explainers agree on that feature. In other words, the FS is so suboptimal, that it makes explainers agree on the top feature.

For the *repeated* features in the second row, we see more distinct trends. For the UC, the correlation-based selection outshines the rest, since it is able to determine that in total only five features are relevant, since it can only establish five clusters, because all features are perfect duplicates. This is followed by the three wrapper-based methods. Interestingly, even the permutation-based approach is able to keep the UC constant. Although permutation is prone to correlation, this effect is negated here since all initial features have the same chance to be duplicated. In other words, while correlated features dampen each other's importance, this effect happens for all features uniformly. This is also the reason why the random baseline performs decent, as on average, it will choose each original feature at least once. The impurity-based approach and *SelectKBest()*, however, are both biased towards the top feature. Especially the latter even drops its consensus below the baseline. In the worst-case scenario, the selected features only contain a single feature and its replicates, thus making it hard for SHAP and LIME to determine which is most important. For the OC, we generally see a similar trend to



(a) CICIDS (b) CIDDS (c) Edge-IIoT

Fig. 3: Signed UC (top) and Signed OC (bottom) on NIDS data with **MLP** as underlying model.



(a) CICIDS (b) CIDDS (c) Edge-IIoT

Fig. 4: Signed UC (top) and Signed OC (bottom) on NIDS data with **RF** as underlying model.

the UC. Though, since it is generally lower, the baseline and *SelectKBest()* cannot drop as steep numerically. The accuracy also shows the suboptimal FS of *SelectKBest()*.

The *redundant* features in the third row generally show a more "noisy" trend compared to the previous feature types. That is, while this feature type still contains redundancies, they are more obfuscated. In other words, choosing a different feature set other than the initial features still holds value. For both the UC and OC we still see the correlation-based approach having an edge over the others, though far less significant than for the repeated features. Additionally, *SelectKBest()* and the impurity-based approach also do not diverge as strongly. For the accuracy, all methods are able to keep up the performance, since there is no issue of choosing (perfect) duplications.

Lastly, the *useless* features in the fourth row show a contrasting trend to the previous features. For the UC, we now see that the correlation-based approach performs significantly worse then the others, as well as illustrating why random FS is a bad idea. Even no FS is better here. The same can be observed for the OC, and is also reflected in the accuracy dropping. Since the correlation-based approach clusters the features first, it may choose a handful of noise. Similarly, the random FS will choose noise. Since this noise actually has no meaning for the explanations, it makes it increasingly hard for the explainers to decide which of these noisy features are actually more relevant, similar to the perfectly correlated features. Contrary to the previous feature types, the baseline without FS performs worse since it also uses the noise to train.

Discussion: The synthetic experiments validate our core assumptions: FS can improve consensus, particularly when feature interactions such as correlation or redundancy are present. Selection methods that respect such interactions outperform simpler approaches like *SelectKBest()*. However, when the added features are purely noisy, correlation-based selection may actually reduce consensus and accuracy by mistakenly prioritizing irrelevant features. Overall, sequential selection proved to be a balanced choice across all feature types.

B. Final Experiments on NIDS Datasets

We now shift our focus to more realistic NIDS data, which may not contain perfectly (un)correlated features, but more complex feature relationships. For this, Figures 3 and 4 show the consensus for all three previously described NIDS datasets as a barplot, for MLP and RF, respectively. The top row depicts the UC and the bottom row shows the OC. The x-axis depicts the selection method, while the left y-axis holds the consensus. The consensus is the average of 100 random samples from the test set for all three splits, i. e., each bar is made up by $3 \cdot 100$ values. We divide the consensus analysis between our two classes, as the datasets are imbalanced. The errorbars contain the 95% confidence intervals. The right y-axis marks the average macro F1 to take into account imbalances via stars. We also utilize the macro F1 for FS for performance scoring here where applicable, as the default is accuracy.

Starting with the MLP as underlying model in Figure 3 and the UC (top row), we see that no selection method really stands out, whether it is w.r.t. consensus or performance. The only method that consistently has similar performance to the baseline with no filtering is the recursive selection. For CICIDS, the majority of the methods are able to at least slightly boost the consensus, though only marginally in most cases. Features are reduced from around 70 features to only 10, but the consensus increase is nowhere near drastic. This also comes at a performance loss in some cases. For example, the impurity-based approach increases the consensus the most, but performance drops by roughly 5%. Interestingly, even a random FS performs better accuracy-wise. For CIDDS, results are similar concerning the UC boost. For Edge-IIoT, the consensus is already quite high, so any further filtering does not actually have a great impact. Interestingly, *SelectKBest()* and the impurity-based approach are the only ones that increase the UC here, which may seem counterintuitive compared to the synthetic data. For the OC (bottom row), we see similar observations. While the FS is able to establish some consensus in some cases, the improvement is not as drastic as desired. Again, for Edge-IIoT, consensus is higher and filtering can actually have a significant negative effect.

The results for the RF as underlying model are depicted in Figure 4. How the selection methods perform w.r.t. F1 follows similar trends as for the MLP, though the score is consistently higher. While the consensus is in a similar range compared to before, there are some nuanced differences. Which method improves the consensus most for the datasets is partially inconsistent with the results from the MLP, i. e., not following a noticeable pattern. For the UC, filtering features may have a positive effect for CICIDS and CIDDS, while for Edge-IIoT the baseline already reaches near perfect consensus, despite using much more features. For the OC, the consensus of CICIDS is only slightly improved by filtering and for CIDDS filtering has no huge impact either. For Edge-IIoT, *SelectKBest()* and impurity-based FS are the only methods that increases the consensus meaningfully here.

Discussion: Results on real-world NIDS datasets show that the impact of FS on consensus is less predictable. While consensus can improve over using all features, the gains are generally modest. CICIDS generally responded the best to FS. Some of its features are various statistical moments of packet sizes, IATs etc., which are naturally more correlated. CIDDS is also contains flow-based data, but already has very few features, thus FS will not have as much of an impact. For Edge-IIoT, *SelectKBest()* and impurity-based FS performed best, which is in stark contrast to the previous analyses. We hypothesize this is due to a few highly indicative (and likely related) features. More sophisticated methods may discard these in favor of spreading importance across weaker (thus noisier) features, reducing consensus. This is supported by the fact that random FS dropped performance the most drastically here for both MLP and RF. In other words, the other datasets contain more related features in terms of informativeness, i. e., random FS is partially feasible. Overall, the impact of FS is harder to assign for NIDS data, since it is composed of a more mixed set of features w.r.t. noise and correlation, so effects supposedly counteract each other.

TABLE I: MSE $\pm$ 95% CI for the toy example, 2.5k trials.

	X_1	X_2	**Both**
LR	0.53 ± 0.01	69.8 ± 0.61	0.00 ± 0.00
RF	0.60 ± 0.01	0.62 ± 0.01	0.56 ± 0.01

To illustrate the impact of removing correlated features on the actual model performance, we trained a linear regression (LR) and RF on two highly correlated inputs (one linear and one exponential feature, X_1 and X_2), and defined the regression target as their sum. Using both features yielded lower errors than only one (see Table I). Thus, unless a feature is an identical copy of another, correlated features might still carry information. These findings emphasize that reducing features to minimize potential correlations does not always help, both in terms of consensus as well as performance, and is highly dependent on dataset and model.

To summarize the findings of this work in a more practical context, post-hoc explanations and their resulting consensus are highly sensitive to slight changes in the ML pipeline. Filtering *can* help by removing totally irrelevant noise or avoiding actual duplicates (e. g., analogous to radius vs. diameter of a circle), but it is not a cure for all problems by simple application and still requires critical thinking. It also raises the need for interpretable consensus metrics, which can be misleading when inflated by highly indicative, but correlated features (as potentially seen with the Edge-IIoT dataset).

V. Conclusion

In this work, we analyzed how feature selection influences the disagreement problem between post-hoc XAI methods in ML-NIDS. On synthetic datasets, where we could control redundancy and noise, feature selection behaved as expected by improving explanation consistency in a predictable manner. However, results on real-world NIDS datasets were less consistent. While in some cases feature selection modestly improved agreement between explainers, it sometimes actually reduced consensus, particularly when selection methods emphasized noisy features. Overall, the improvement over the full-feature baseline was often smaller than maybe desired. If aligning explainers remains unreliable, it hinders the practical adoption of these methods, and they risk becoming "rebranded" standard feature importance metrics. Our results suggest a need for stronger emphasis on inherently interpretable models, rather than reliance on post-hoc techniques that struggle to produce consistent and trustworthy explanations.

Acknowledgment

This work has been partly funded by the Bavarian Ministry of Economics, Regional Development and Energy (StMWI) as part of the project VIPNANO (DIK-2307-0006), by Deutsche Forschungsgemeinschaft (DFG) under grant SE 3163/3-1, project nr.: 500105691 (UserNet), and by the German Federal Ministry of Research, Technology and Space (BMFTR) under grant 18KIS2282 (SUSTAINET-Advance). The authors alone are responsible for the content.

REFERENCES

[1] C. Ardagna, S. Corbiaux, K. Van Impe, and R. Ostadal, "ENISA threat landscape 2023," *European Union Agency for Cybersecurity*, 2023.

[2] F. N. Motlagh *et al.*, "Large language models in cybersecurity: State-of-the-art," *arXiv:2402.00891*, 2024.

[3] Z. Zhang *et al.*, "Explainable artificial intelligence applications in cyber security: State-of-the-art in research," *IEEE Access*, vol. 10, 2022.

[4] L. Caviglione *et al.*, "Tight arms race: Overview of current malware threats and trends in their detection," *IEEE Access*, vol. 9, 2020.

[5] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Comput. Secur.*, vol. 81, 2019.

[6] M. A. Talib *et al.*, "APT beaconing detection: A systematic review," *Comput. Secur.*, vol. 122, 2022.

[7] K. Dietz *et al.*, "The missing link in network intrusion detection: Taking AI/ML research efforts to users," *IEEE Access*, vol. 12, 2024.

[8] G. Apruzzese, P. Laskov, and J. Schneider, "SoK: Pragmatic assessment of machine learning for network intrusion detection," in *IEEE Eur. Symp. Secur. Priv. (EuroS&P)*, 2023.

[9] S. Oesch *et al.*, "An assessment of the usability of machine learning based tools for the security operations center," in *Proc. Int. Conf. Internet Things (iThings)*, 2020.

[10] A. Nadeem *et al.*, "SoK: Explainable machine learning for computer security applications," in *IEEE Eur. Symp. Secur. Priv. (EuroS&P)*, 2023.

[11] G. Jaswal, V. Kanhangad, and R. Ramachandra, *AI and Deep Learning in Biometric Security: Trends, Potential, and Challenges*. CRC, 2021.

[12] S. Krishna *et al.*, "The disagreement problem in explainable machine learning: A practitioner's perspective," *Trans. Mach. Learn. Res. (TMLR)*, 2024.

[13] K. Dietz *et al.*, "Agree to disagree: Exploring consensus of XAI methods for ML-based NIDS," in *Workshop Netw. Secur. Oper. (NeSecOr)*, 2024.

[14] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a 'right to explanation'," *AI Mag.*, vol. 38, no. 3, 2017.

[15] T. Grance *et al.*, "Guide to information technology security services," *NIST Special Publication 800-35*, 2003.

[16] J. Mink *et al.*, "Everybody's got ML, tell me what else you have: Practitioners' perception of ML-based security tools and explanations," in *IEEE Symp. Secur. Priv. (S&P)*, 2023.

[17] https://eur-lex.europa.eu/eli/reg/2024/1689, accessed: 2025-05-03.

[18] P. Khani, E. Moeinaddini, N. D. Abnavi, and A. Shahraki, "Explainable artificial intelligence for feature selection in network traffic classification: A comparative study," *Trans. Emerg. Telecommun. Technol. (ETT)*, vol. 35, no. 4, 2024.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, 2016.

[20] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017.

[21] L. Shapley, "A value for n-person games," *Contrib. Theory Games*, vol. 2, 1953.

[22] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, "Evaluating explanation methods for deep learning in security," in *IEEE Eur. Symp. Secur. Priv. (EuroS&P)*, 2020.

[23] L. Rui and O. Gadyatskaya, "Position: The explainability paradox-challenges for XAI in malware detection and analysis," in *IEEE Eur. Symp. Secur. Priv. Workshops (EuroS&PW)*, 2024.

[24] D. Bhusal *et al.*, "SoK: Modeling explainability in security analytics for interpretability, trustworthiness, and usability," in *Proc. 18th Int. Conf. Availab. Reliab. Secur. (ARES)*, 2023.

[25] M. Flora, C. Potvin, A. McGovern, and S. Handler, "Comparing explanation methods for traditional machine learning models part 1: an overview of current methods and quantifying their disagreement," *arXiv:2211.08943*, 2022.

[26] ——, "Comparing explanation methods for traditional machine learning models part 2: Quantifying model explainability faithfulness and improvements with dimensionality reduction," *arXiv:2211.10378*, 2022.

[27] Z. Carmichael and W. Scheirer, "How well do feature-additive explainers explain feature-additive predictors?" in *Workshop XAI Action: Past Present Future Appl. (NeurIPS XAIA)*, 2023.

[28] A. F. Markus *et al.*, "Understanding the size of the feature importance disagreement problem in real-world data," in *ICML 3rd Workshop Interpret. Mach. Learn. Healthc. (IMLH)*, 2023.

[29] F. Fumagalli *et al.*, "Unifying feature-based explanations with functional ANOVA and cooperative game theory," in *Proc. 28th Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2025.

[30] A. Schwarzschild *et al.*, "Reckoning with the disagreement problem: Explanation consensus as a training objective," in *Proc. 2023 AAAI/ACM Conf. AI Ethics Soc. (AIES)*, 2023.

[31] U. Stańczyk, "Feature evaluation by filter, wrapper, and embedded approaches," in *Feature Sel. Data Pattern Recognit.*, 2014.

[32] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res. (JMLR)*, vol. 12, 2011.

[33] https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance_multicollinear.html, accessed: 2025-05-29.

[34] O. Arreche, T. R. Guntur, J. W. Roberts, and M. Abdallah, "E-XAI: Evaluating black-box explainable AI frameworks for network intrusion detection," *IEEE Access*, vol. 12, 2024.

[35] A. N. Gummadi, O. Arreche, and M. Abdallah, "A systematic evaluation of white-box explainable AI methods for anomaly detection in IoT systems," *Internet of Things*, vol. 30, 2025.

[36] J. Tritscher *et al.*, "Evaluation of post-hoc XAI approaches through synthetic tabular data," in *Found. Intell. Syst.: 25th Int. Symp. (ISMIS)*, 2020.

[37] J. Tritscher, M. Wolf, A. Hotho, and D. Schlör, "Evaluating feature relevance XAI in network intrusion detection," in *World Conf. Explain. Artif. Intell. (xAI)*, 2023.

[38] O. Lukás and S. García, "Bridging the explanation gap in AI security: A task-driven approach to XAI methods evaluation." in *Proc. 16th Int. Conf. Agents Artif. Intell. (ICAART)*, 2024.

[39] O. Arreche, T. Guntur, and M. Abdallah, "XAI-based feature selection for improved network intrusion detection systems," *arXiv:2410.10050*, 2024.

[40] G. Laberge, Y. B. Pequignot, M. Marchand, and F. Khomh, "Tackling the XAI disagreement problem with regional explanations," in *Proc. 27th Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2024.

[41] S. Aswani and S. D. Shetty, "Explainable news summarization–analysis and mitigation of disagreement problem," *arXiv:2410.18560*, 2024.

[42] P. Alves *et al.*, "Comparing LIME and SHAP global explanations for human activity recognition," in *Intell. Syst.: 34th Braz. Conf. (BRACIS)*, 2024.

[43] N. Koenen and M. N. Wright, "Toward understanding the disagreement problem in neural network feature attribution," in *World Conf. Explain. Artif. Intell. (xAI)*, 2024.

[44] P. Silva, C. T. Silva, and L. G. Nonato, "Exploring the relationship between feature attribution methods and model performance," in *Proc. 2024 AAAI Conf. Artif. Intell.*, 2024.

[45] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *Proc. 4th Int. Conf. Inf. Syst. Secur. Priv. (ICISSP)*, 2018.

[46] M. Ring *et al.*, "Flow-based benchmark data sets for intrusion detection," in *Proc. 16th Eur. Conf. Cyber Warf. Secur. (ECCWS)*, 2017.

[47] M. A. Ferrag *et al.*, "Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning," *IEEE Access*, vol. 10, 2022.

[48] I. Guyon, "Design of experiments of the NIPS 2003 variable selection benchmark," in *NIPS Workshop Feature Extr.*, 2003.

Decentralized Federated Learning for Intrusion Detection in 5G Networks: An Asynchronous Consensus-driven Multi-Agent Approach

Nasim Nezhadsistani[1], Francisco Enguix[2], Carlos Carrascosa[2], Burkhard Stiller[1]
[1]Communication Systems Group, Department of Informatics, University of Zürich, Switzerland
Email: {nezhadsistani, stiller}@ifi.uzh.ch
[2]Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València (UPV),
Camino de Vera s/n, 46022, Valencia, Spain
Email: fraenan@upv.es, carrasco@dsic.upv.es

***Abstract*—5G networks expand the wireless infrastructure attack surface while concurrently limiting data streams through in-your-face privacy regulations. Centralized traffic aggregation-based traditional intrusion-detection pipelines, thus, are plagued with bottlenecks, sole points of failure, and regulatory insurrection. This paper uses PA-CoL, a Parallel Asynchronous Consensus-based Learning framework in which multi-agent base stations train a shared deep neural model without ever exporting raw traffic records. Each agent performs local mini-batches on 5G-NIDD flow features and intermittently averages parameters with a randomly chosen neighbor; no parameter server or global synchrony is required. Experiments across three overlay graphs (Complete, Ring, Small-World), two data Types (IID and non-IID), and three federation sizes (5, 8, 10 agents) show that the newly developed scheme reaches F1 ≥ 0.99 under IID data and F1 ≥ 0.93 under severe non-IID skew. These results indicate that a lightweight, peer-to-peer consensus can deliver carrier-grade intrusion detection for privacy-sensitive 5G edge clouds and can pave the way toward 6G self-defending networks.**

***Index Terms*—5G Network, Federated Learning (FL) , Multi-Agent Systems (MAS), Intrusion Detection.**

I. Introduction

Fifth-generation (5G) mobile networks are a wireless communications paradigm shift that provides greater bandwidth, ultra-reliable low-latency (URLLC), and massive machine-type communications (mMTC). This results in new capabilities like autonomous vehicles, smart manufacturing, and remote medicine. But with these capabilities comes the price tag of an exponentially increased attack surface. Use of encrypted network traffic, pervasive use of network slicing, and the inherent dynamic nature of 5G topologies introduce high degrees of complexity for threat detection and security monitoring. Conventional security analytics, having a dependency on centralized data aggregation and correlation, become less effective in such cases [1]. Centralized solutions have critical drawbacks, including bottlenecks, single points of failure, and lack of end-to-end visibility in highly segmented virtualized environments [4], [10].

An additional level of intricacy arises from stricter data sovereignty and privacy regulations, such as the General Data Protection Regulation (GDPR) within the European Union. These regulations mandate that organizations maintain sensitive user data within local or national borders, effectively excluding unregulated trafficking of raw data to distant cloud servers for processing. Security products, therefore, relying on packet capture and aggregation of all traffic across the network, are not only technologically infeasible but also most likely to be non-compliant with global privacy standards. There is therefore a pressing need for threat detection and response capabilities without losing data locality in the security architectures [28], [29], [31].

Federated Learning (FL) [2] proves to be a viable paradigm to overcome privacy and locality of data concerns. FL facilitates training machine learning models in a decentralized manner through enabling local nodes or edge devices to hold their data locally and send model updates (e.g., parameters or gradients) to a coordination agent. This paradigm reduces risk to privacy and restricts exposure of sensitive data to an absolute minimum. But the traditional FL architecture usually depends on a parameter server in the middle to collect model updates and scatter the updated global model. This server-based architecture creates new threats, such as communication bottlenecks and single points of failure, which are undesirable in the case of large-scale, mission-critical 5G networks. In addition, centralized FL presumes synchronized and consistent participation of all the nodes that, in reality, cannot be ensured due to heterogeneity and dynamic connectivity of mobile devices [9], [16], [18].

Decentralized Federated Learning (DFL) has been suggested to bypass the downsides of centralized servers. Peer-to-peer cooperation in DFL substitutes the central aggregation node such that nodes are able to train models jointly in an unsupervised way without a coordinator node. The distributed approach improves robustness and scalability but typically comes with the cost of rigorous synchronization among cooperating nodes. 5G real-world scenarios are proven to be challenging in device capability, network state, and participation rates shifting unpredictably [5]. Multi-Agent Systems (MAS) [3] offer the natural paradigm for distributed decision-making and intelligence for advanced environments. Autonomous agents

and local sensing and knowledge-directed agents interact with each other in MAS to achieve common goals. Translated to 5G security, MAS enable localized discovery and quick response to emerging threats, allowing the requirements of adaptive, distributed defenses [32].

To take advantage of the strong points of DFL and MAS and handle synchrony and heterogeneity problems, this paper uses Parallel Asynchronous Consensus-based Learning (PA-CoL) [8]. PA-CoL facilitates the convergence of the models of decentralized agents without the need for global synchronization. Agents instead update and exchange their models asynchronously with neighbors in such a way that the network converges as an aggregate under delays, network partition, or stragglers. This asynchrony is particularly suited for variable conditions under 5G networks. The key contributions of this paper are as follows:

1) This paper constructs an entirely distributed intrusion detection system (IDS) using PA-CoL over a MAS overlay specially designed for the distinctive nature and needs of 5G networks.
2) This paper applies the developed framework using the *PyTorch* deep learning framework and performs an extensive experiment on the 5G-NIDD dataset [1], with specific emphasis on non-IID (non-identically and independently distributed) data scenarios.
3) This paper compares our detection performance, communication cost, and scalability with conventional FL benchmarks and shows its performance benefits in real-world 5G scenarios.

The remainder of the paper is organized as follows. Section II provides related work on intrusion detection for 5G networks, FL, decentralized training algorithms, and multi-agent systems. Section III offers the proposed framework architecture, such as system model, dataset preprocessing, learning algorithm, and comparisons with baseline methods. Section IV provides a description of the experimental setup and results focusing on performance measures, communication efficiency, and resilience with non-IID scenarios. Finally, Section V summarizes the paper and presents future directions of work.

II. Related Work

A. Federated Learning for Network Security

FL has become an attractive paradigm for cooperative, privacy-aware intrusion detection. The initial *FedAvg* algorithm [2] proved that distributed devices could perform local model learning on their own data in parallel and then exchange model updates with a central server to collect. This work motivated diverse research on privacy-preserving machine learning for network security. The latest developments also involve the use of advanced models like BERT, which has made remarkable discoveries of up to 97.8% accuracy for intrusion detection based on datasets [6]. However, most recent FL solutions still rely on a central server to handle the learning process. This reliance is accompanied by a cluster of weaknesses: the master server is a point of contention, a target for denial-of-service or poisoning attacks, and a single point of failure, which compromises the resilience and scalability needed for 5G network security.

B. Multi-Agent Systems in 5G Security

MAS provides an adaptive and agile answer to distributed security in 5G networks. In MAS, autonomous heterogeneous agents with local knowledge and goals interact and collaborate to detect, respond to, and pursue threats. This distributed approach is most applicable to the dynamic, partitioned, and large-scale environment of 5G. The combination of MAS with FL was even investigated recently to further improve system resilience and fault tolerance to allow agents the capability to collaboratively train and improve security models in the absence of a central controller [7]. Although these are promising directions, existing research in these directions has been mainly theoretical models or simulations, with extremely few being based on real 5G datasets like 5G-NIDD. Therefore, there is an urgent need for experimental investigations that assess the efficiency and usability of MAS-FL hybrids in real-world 5G deployment environments.

C. From Centralized to Decentralized FL in 5G IDS

Early 5G intrusion detection research used the standard *FedAvg* framework, where edge devices provide gradient updates to a parameter server; the framework attains high accuracy on IIoT data but suffers from a single point of failure and is still susceptible to model-poisoning attacks [12], [14]. Server-side bandwidth constraints also reduce scalability when thousands of gNBs transmit traffic features every few seconds. A number of studies find that server-centrism is antithetical to the URLLC aspirations of 5G and support peer-to-peer aggregation instead [11], [13]. There exists a taxonomy of decentralized forms with security in focus as well [5]. They collectively induce a turn towards topologically fully decentralized or at least hierarchically organized training structures.

D. Asynchronous Decentralized FL

Weighted-average consensus on an overlay graph eliminates the server bottleneck, while the synchronous one continues to assume uniform compute speed [27]. Multilayer consensus and dynamic-average consensus enable all agents to update whenever resources become available and thus eliminate stragglers [16], [18]. The pull-based protocol generalizes the concept further by allowing nodes to pull the latest neighbor model on demand [9], while the latency-compensated scheduler rewinds stale gradients [17]. All such systems, though, broadcast each update across the network and thereby flood low-power radios.

E. Security, Privacy, and Verifiability

Consensus ledger verifiable computation prevents tampering but adds additional latency [15]; end-to-end correctness proofs add even more integrity [21]. Update-level trust scoring and the proof-of-data system, which is cryptographic, prevents decentralized FL poisoning but has not yet been tested with

real 5G traffic [19], [20]. Communication-saving methods like locally differentially private updates indicate privacy and efficiency can be pursued in tandem [25].

F. Parallel Asynchronous Consensus-based Learning

Consensus-Based Learning approaches make fusion into an iterated averaging problem with no kind of global coordinator [30]. The most recent evolution, PA-CoL, features an explicit coalition layer [8]. Agents sharing similar semantics or geographically co-located agents in PA-CoL organize intra-coalition clusters that reach consensus in frequent rounds; models aggregated from each coalition leader engage in the slower inter-coalition exchange. Experiments across a non-IID image data set reduce total bytes transferred by about 35 % without affecting accuracy compared to single-coalition baselines, and message-direction analysis validates that most traffic is local between coalitions [8]. Since 5G slices natively map to trust domains, we generalize the coalition concept to slice-aware security monitoring [8], [33].

G. Graph and Self-Supervised Models for Network Data

Graph neural networks improve representation of host-to-host relationships and protocol hierarchies: Edge-based GraphSAGE in [26], DIGNN-A in [22], and feature-rearranged (GNNs in [23] all see double-digit F1 gains over multilayer perceptrons. Self-supervised pretext tasks like TS-IDS in [24] see comparable large gains when labelled data is limited. Nevertheless, parameter counts are an order of magnitude greater than for lightweight MLPs, increasing the communication overhead decentralized FL already needs to cope with. Compression methods like the autoencoder pipeline in [29] and hierarchical clustering in [30] mitigate this expense in part.

III. Experimental Design

A. Data Set and Pre-processing

The evaluation relies on the public 5G-NIDD dataset [1], which captures network traffic across nine behaviour classes. Each record carries forty-eight normalised numerical features. Rows with missing values less than $0.1\,\%$ of the file are discarded to avoid introducing noise. A stratified hold-out split reserves $20\,\%$ of the cleaned data for testing so every class preserves its prior. Feature scales are already aligned; therefore, no extra standardisation is applied.

B. Neural Architecture and Local Optimisation

Both agents have the same feed-forward MLP depicted in Fig. 1. There are two hidden widths 128 layers with ReLU activations and dropout rate 0.3 to prevent over-fitting. The output layer provides nine logits. Training uses the Adam optimiser with a step size of 0.001 and categorical cross-entropy loss. A training iteration is a single forward-and-backward pass over a mini-batch of 32 samples.

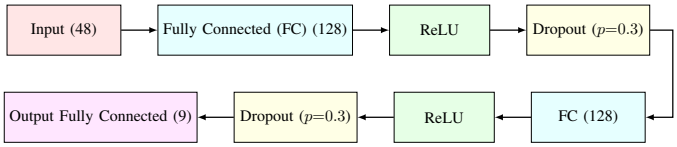


Fig. 1: Architecture of the MLP model with two hidden layers of size 128 and dropout.

C. Data Partitioning Strategy

To examine the impact of statistical bias, two placement modes are considered. *IID mode*: The fold employed for training is shuffled and divided equally into shards, and each agent gets an equal amount of samples and class distribution. *Non-IID mode*: sampling indices are sampled from Dirichlet concentration $\alpha = 0.3$. Biases samples class proportions such that some agents have labels unobservable elsewhere, simulating true edge drift.

D. Round Structure and Consensus Rule

Agents transfer information along static undirected graphs of five, eight, or ten nodes. The considered topologies are complete, small-world (rewiring probability 0.3), and ring. The largest node degree $\deg_{\max}$ fixes the mixing rate

$$\epsilon = \frac{1}{\deg_{\max}} \tag{1}$$

During each round, an agent first performs one local epoch, then selects one neighbor at random and updates its weights via

$$\theta_i \leftarrow (1-\epsilon)\,\theta_i + \epsilon\,\theta_j, \tag{2}$$

where θ_i and θ_j are the agent's parameter vector and the selected neighbor's parameter vector, respectively. There is one such exchange in each round. The protocol is executed for 500 rounds for each $\{$*graph, partition mode*$\}$.

E. Hardware and Software stack

All experiments were conducted on a computing environment equipped with an Intel Core i7 13650HX CPU, featuring 20 logical processors, and 32 GB of RAM. The graphics processing was managed by an NVIDIA GeForce RTX 4070 Laptop GPU. The system used an NVMe SSD storage solution, ensuring rapid data read and write speeds. The operating system for this setup was Windows 11 Pro 64-bit.

IV. Experimental Results

Table I reports the maximum mean-over-agent performance metrics obtained by the parallel Asynchronous Consensus-based Learning framework. This paper evaluates three communication topologies (Complete, Ring, and Small-World), two data distributions (IID and non-IID), and three network sizes (5, 8, and 10 agents). Under IID conditions, different topologies perform almost the well as they can and show very few differences in all metrics and network sizes. This is because, with IID, each agent can access data that is

equally representative, which leads to steady and consistent convergence no matter what the network structure is. For example, accuracies and F1 scores often get close to 0.995, which confirms that topology does not matter much in this ideal situation. On the other hand, the non-IID setting (with Dirichlet skew) is more realistic and difficult, as agents have very different and biased local data distributions. In this case, the topology choice becomes a very important design consideration that impacts convergence and the final result.

Under non-IID conditions, the *Complete* topology exhibits significant performance degradation. For example, with 8 agents, the accuracy drops dramatically to 0.6973, and the F1 score decreases to 0.5908. This highlights the vulnerability of fully connected graphs to bias amplification, as global averaging may allow agents with extreme local distributions to disproportionately influence the consensus model. Increasing the number of agents from 8 to 10 slightly improves performance (accuracy rising to 0.8558), suggesting that increased diversity among agents can partially counteract individual biases, but overall, the Complete topology remains the most sensitive to non-IID data.

The ring topology shows greater strength in non-IID setups. By restricting each agent's interactions to close neighbors, it lessens the spread of strong local biases. As shown in Table I, the ring topology keeps high accuracy and F1 scores (for example, an accuracy of 0.9677 and an F1 score of 0.9370 with eight agents), which points to more consistent and dependable model behavior. Also, the ring topology's behavior stays even and less jumpy across varied network sizes, showing it can scale with data differences.

The small-world topology has middle-of-the-road behavior in non-IID settings. While it gains from shortcuts that allow quicker information spread than a simple ring, it also struggles with the impact of highly linked hub nodes, which can make local biases worse if these hubs have slanted data. This is clear in the fairly low precision and F1 scores (like a precision of 0.8341 and an F1 score of 0.7919 with eight agents), which shows a give-and-take between quicker mixing and flexibility to differences.

In considering how scaling impacts performance, merely adding more agents does not assure better results in non-IID setups. For instance, the Complete topology improves somewhat with ten agents. But the Ring and Small-World topologies are more stable, without big drops in performance. This suggests that just increasing the number of agents isn't enough to fix data differences.

The network structure must be carefully thought out. From a real-world use point, these results point to using sparse, structured overlays like the Ring topology when data is inherently non-IID. Situations include edge networks in 5G and distributed intrusion detection systems. Keeping high F1 scores (above 0.93) even with few communication links per agent gives a good balance. This balance is between detection quality and communication costs, which is key for privacy and places with limited bandwidth.

Fig.2 (a)-(i) illustrates the per-round evolution of *accuracy*, *F1 score*, and *precision* for networks consisting of 10, 8, and 5 agents, each evaluated across six configurations. These configurations combine three communication topologies (Complete, Ring, and Small-World) with two data distributions (IID and Dirichlet-skewed, $\alpha = 0.3$).

TABLE I: Maximum of the mean-over-agents metrics.

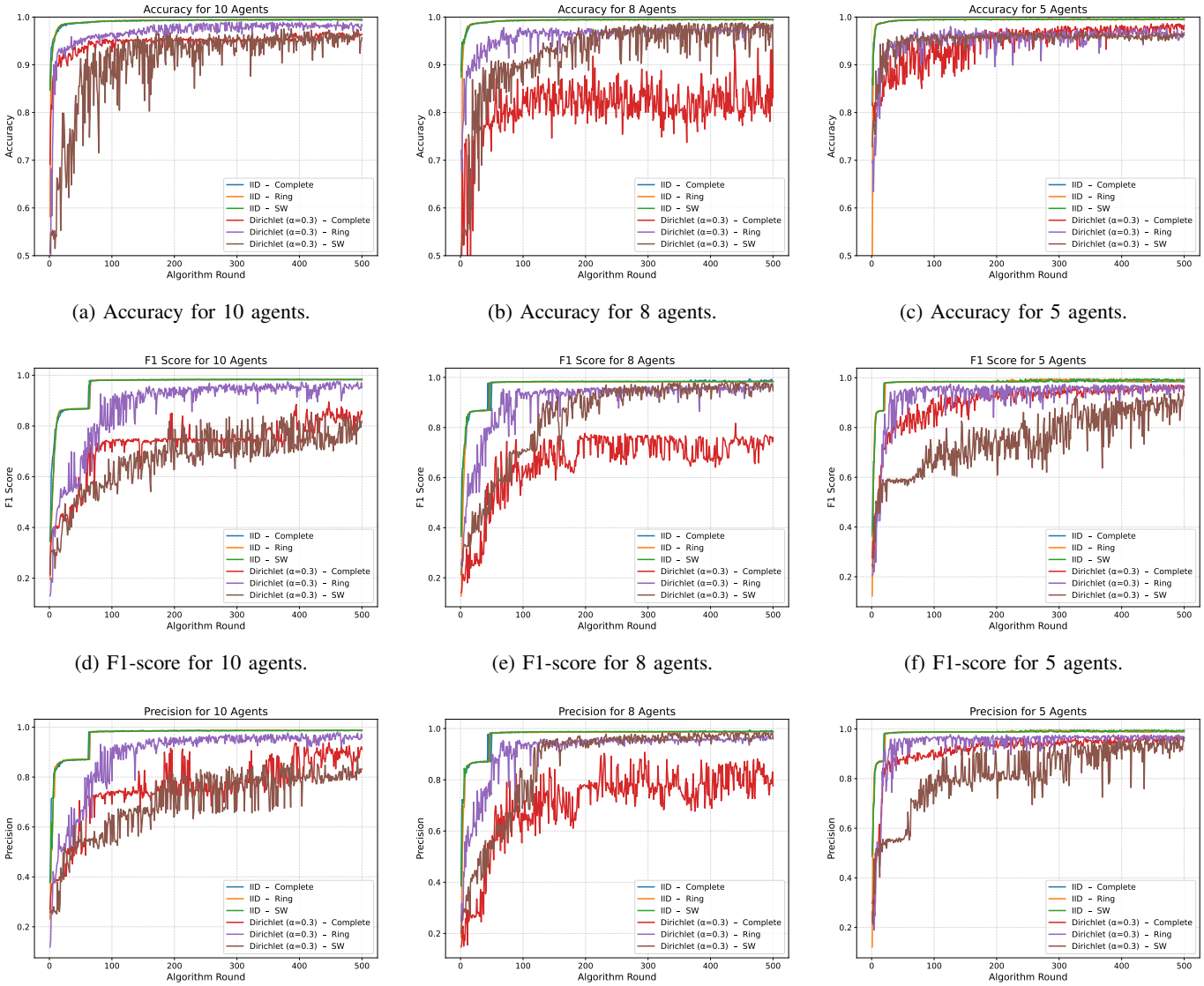
Nodes	Topology	Dirichlet (non-IID)			IID		
		Accuracy	Precision	F1	Accuracy	Precision	F1
5	Complete	0.9687	0.9571	0.9545	0.9959	0.9929	0.9916
	Ring	0.9621	0.9591	0.9383	**0.9963**	**0.9943**	**0.9937**
	Small-World	0.9410	0.8642	0.7990	0.9959	0.9929	0.9918
8	Complete	0.6973	0.6647	0.5908	0.9943	**0.9883**	**0.9845**
	Ring	0.9677	0.9519	0.9370	**0.9949**	0.9875	0.9833
	Small-World	0.7191	0.8341	0.7919	0.9945	0.9882	0.9838
10	Complete	0.8558	0.6964	0.6406	0.9934	0.9859	0.9822
	Ring	0.9547	0.9300	0.9066	0.9944	**0.9869**	**0.9831**
	Small-World	0.8938	0.7209	0.6881	**0.9948**	0.9862	0.9828

In the IID setting (indicated by blue, orange, and green curves), all topologies achieve rapid and smooth convergence. Accuracy, precision, and F1 score all exceed 0.95 within approximately 10 to 20 communication rounds, regardless of the number of agents. The Complete topology exhibits the steepest initial improvement, typically saturating in as few as 5 rounds. Ring and Small-World topologies follow slightly slower but still very close trajectories. This behavior confirms that when data is balanced and fully representative at each agent, the effect of communication topology becomes negligible. Thus, these IID results act merely as an upper bound on possible performance.

Under non-IID conditions (Dirichlet, $\alpha = 0.3$), the convergence patterns change dramatically and reveal the true impact of topology. The Ring topology (purple curves) shows the most robust and stable performance. For example, in the 8-agent configuration, accuracy surpasses 0.95 by around round 40, and F1 score exceeds 0.90 by about round 60, with minimal oscillations in subsequent rounds. This stability arises because each agent exchanges information only with local neighbors, which helps prevent global consensus from being dominated by agents with highly skewed local data.

The Complete topology (red curves) shows high initial instability with non-IID data. Initial F1 scores may fall below 0.60 in the first 50 rounds, and the trajectory has clear oscillations before gradually stabilizing. Even after 500 rounds, final F1 scores only stabilize around 0.75 to 0.78, highlighting the vulnerability of global averaging to local bias amplification. Nevertheless, as the number of agents increases, the added diversity among peers slightly mitigates these effects, as observed by a final accuracy increase from approximately 0.70 (8 agents) to 0.86 (10 agents).

The Small-World topology (brown curves) presents an intermediate behavior. Its shortcut edges enable a faster initial rise, for instance, with 10 agents, accuracy approaches 0.90 already by round 30. However, these same shortcut links can allow certain highly connected "hub" agents to propagate biased or noisy updates more broadly. This leads to convergence settling at intermediate final scores (e.g., around 0.78 accuracy for 8



(a) Accuracy for 10 agents.

(b) Accuracy for 8 agents.

(c) Accuracy for 5 agents.

(d) F1-score for 10 agents.

(e) F1-score for 8 agents.

(f) F1-score for 5 agents.

(g) Precision for 10 agents.

(h) Precision for 8 agents.

(i) Precision for 5 agents.

Fig. 2: Accuracy, F1-score, and precision; across all agent configurations.

agents), reflecting a trade-off between accelerated mixing and robustness against local skew. When considering the effect of network size, increasing the number of agents from 5 to 10 slightly increases early-stage variance, especially in the Complete topology, where more potential sources of skew exist. However, in the Complete graph, this added diversity can also support moderate recovery in later stages. Meanwhile, the Ring topology stays very consistent no matter the network size, which points to strong resilience and scalability. Small-world topologies only gain a little from more agents.

From an engineering perspective, these observations highlight that in practical distributed environments such as 5G edge networks, where non-IID data is common and agents often operate with limited communication bandwidth, sparse and structured overlays like the Ring topology provide the most favorable balance between convergence stability and communication overhead. The Ring topology achieves high final F1 scores (above 0.93), stable learning curves, and minimal susceptibility to local data skew, even with minimal connectivity. In contrast, higher-density topologies like Complete or Small-World can be justified only under ultra-tight convergence requirements of application-level latency demands at favorable (IID) conditions. In addition, the insensitivity of terminal performance to the number of agents under IID conditions indicates that neighborhood edge clusters can scale relatively adaptively with minimal hyperparameter readjustment, a desirable aspect in dynamic and fast-expanding 5G FL scenarios.

V. Summary and Future Work

This paper proposes a distributed intrusion detection system for 5G networks, based on the synergetic combination

of FL and MAS with the Asynchronous Consensus-based Learning protocol. The proposed solution meets the new requirements of 5G environments in terms of data heterogeneity, latency, and privacy-preserving and scalable threat detection. Our system proves excellent performance in various aspects through rigorous experimentation on the 5G-NIDD dataset. It obtains outstanding detection performance, lowers communication overhead, and retains robustness even in the case of non-IID data distribution and asynchronous update conditions. These results verify the feasibility of employing decentralized collaborative intelligence at the edge of the network to protect next-generation 5G infrastructures. Future work will explore integrating graph neural networks for richer modelling of inter-slice and inter-node traffic relationships and adding trusted execution environments to harden local training and update integrity.

VI. Acknowledgment

This work was supported partially by (a) the University of Zürich UZH, Switzerland, (b) the Horizon Europe Framework Program's project NATWORK, Grant Agreement No. 101139285, funded by the Swiss State Secretariat for Education, Research, and Innovation SERI, under Contract No. 23.00642, and (c) the grant PID2021-123673OB-C31, PRE2022-101563, funded by MICIU/AEI/10.13039/501100011033 and by "ERDF A way of making Europe".

References

[1] S. Samarakoon, Y. Siriwardhana, P. Porambage, M. Liyanage, S.-Y. Chang, J. Kim, J. Kim, and M. Ylianttila, "5G-NIDD: A Comprehensive Network Intrusion Detection Dataset Generated over 5G Wireless Network," *arXiv preprint arXiv:2212.01298*, 2022.

[2] H. B. McMahan and D. Ramage, "Federated Learning: Collaborative Machine Learning without Centralized Training Data," *Google AI Blog*, 6 Apr. 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[3] M. Wooldridge and N. R. Jennings, "Intelligent Agents: Theory and Practice," *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.

[4] C. Carrascosa, A. Pico, M. M. Matagne, M. Rebollo, and J. A. Rincón, "Asynchronous Consensus for Multi-Agent Systems and Its Application to Federated Learning," *Engineering Applications of Artificial Intelligence*, vol. 135, Art. 108840, 2024.

[5] E. Hallaji, R. Razavi-Far, M. Saif, B. Wang, and Q. Yang, "Decentralized Federated Learning: A Survey on Security and Privacy," *arXiv preprint arXiv:2401.17319*, 2024.

[6] F. Adjewa, M. Esseghir, and L. Merghem-Boulahia, "Efficient Federated Intrusion Detection in 5G Ecosystem Using Optimized BERT-Based Model," *arXiv preprint arXiv:2409.19390*, 2024.

[7] N. Latif, W. Ma, and H. B. Ahmad, "Advancements in Securing Federated Learning with IDS: A Comprehensive Review of Neural Networks and Feature Engineering Techniques for Malicious Client Detection," *Artificial Intelligence Review*, vol. 58, Art. 91, Jan. 2025.

[8] F. Enguix, J. A. Rincón, and C. Carrascosa, "Introducing Coalitions to Improve the Performance of Federated Learning Consensus-Based Algorithms (ACoL)," in *Proc. Int. Conf. Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, CCIS 2149, pp. 28–39, 2025.

[9] B. Wang, Z. Tian, J. Ma, W. Zhang, W. She, and W. Liu, "A decentralized asynchronous federated learning framework for edge devices," *Future Generation Computer Systems*, vol. 166, Art. 107683, 2025.

[10] S. Chennoufi, G. Blanc, H. Jmila, and C. Kiennert, "SoK: Federated Learning based Network Intrusion Detection in 5G: Context, State of the Art and Challenges," in *Proc. 19th Int. Conf. Availability, Reliability and Security (ARES)*, Vienna, Austria, 2024.

[11] J. Wu, F. Dong, H. Leung, Z. Zhu, and J. Zhou, "Topology-aware Federated Learning in Edge Computing: A Comprehensive Survey," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–41, 2024.

[12] A. Karunamurthy, K. Vijayan, P. R. Kshirsagar, and K. T. Tan, "An optimal federated learning-based intrusion detection for IoT environment," *Scientific Reports*, vol. 15, Art. 8696, 2025.

[13] A. Belenguer, J. A. Pascual, and J. Navaridas, "A review of federated learning applications in intrusion detection systems," *Computer Networks*, vol. 258, Art. 111023, 2025.

[14] S. M. S. Bukhari *et al.*, "Enhancing cybersecurity in Edge IIoT networks: An asynchronous federated learning approach with a deep hybrid detection model," *Internet of Things*, vol. 27, Art. 101252, 2024.

[15] F. Zhang, Y. Zhang, S. Ji, and Z. Han, "Secure and decentralized federated learning framework with non-IID data based on blockchain," *Heliyon*, vol. 10, no. 5, e27176, 2024.

[16] M. Rebollo and C. Carrascosa, "Multilayered Asynchronous Consensus-Based Federated Learning (MACoFL)," in *Intelligent Data Engineering and Automated Learning – IDEAL 2024*, LNCS 14452, pp. 386–396.

[17] Y. Xu, Z. Ma, H. Xu, S. Chen, and J. Liu, "FedLC: Accelerating asynchronous federated learning in edge computing," *IEEE Trans. Mobile Computing*, vol. 23, pp. 5327–5343, 2024.

[18] Z. Chen, D. Li, J. Zhu, and S. Zhang, "DACFL: Dynamic average consensus-based federated learning in decentralized sensor networks," *Engineering Applications of Artificial Intelligence*, vol. 135, Art. 108840, 2024.

[19] Z. Alsulaimawi, "Enhancing security in federated learning through adaptive consensus-based model update validation," *arXiv:2403.04803*, 2024.

[20] H. Liu, F. Zhu, and L. Cheng, "Proof-of-Data: A consensus protocol for collaborative intelligence," *arXiv:2501.02971*, 2025.

[21] X. Zhao, A. Wu, Y. Pei, Y.-C. Liang, and D. Niyato, "End-to-end verifiable decentralized federated learning," *arXiv:2404.12623*, 2024.

[22] J. Liu and M. Guo, "DIGNN-A: Real-time network intrusion detection with integrated neural networks based on dynamic graph," *Computers, Materials & Continua*, vol. 82, no. 1, pp. 817–842, 2025.

[23] H.-D. Le and M. Park, "Enhancing multi-class attack detection in graph neural networks through feature rearrangement," *Electronics*, vol. 13, no. 12, Art. 2404, 2024.

[24] H. Nguyen and R. Kashef, "TS-IDS: Traffic-aware self-supervised learning for IoT network intrusion detection," *Knowledge-Based Systems*, vol. 279, Art. 110966, 2023.

[25] L. Li, X. Zhang, Y. Wang, and C. Chen, "Locally differentially private online federated learning with limited communication," *arXiv:2411.18752*, 2024.

[26] W. W. Lo *et al.*, "E-GraphSAGE: A graph neural network-based intrusion detection system for IoT," in *Proc. IEEE/IFIP NOMS*, 2022, pp. 1–9.

[27] A. Giuseppi, S. Manfredi, and A. Pietrabissa, "A weighted average consensus approach for decentralized federated learning," *Machine Intelligence Research*, vol. 19, no. 4, pp. 319–330, 2022.

[28] M. Chahoud, S. Otoum, and A. Mourad, "On the feasibility of federated learning towards on-demand client deployment at the edge," *Information Processing & Management*, vol. 60, no. 1, Art. 103150, 2023.

[29] A. S. M. Tayeen *et al.*, "CAFNet: Compressed autoencoder-based federated network for anomaly detection," in *Proc. IEEE MILCOM*, 2023, pp. 325–330.

[30] X. Sáez-de-Cámara *et al.*, "Clustered federated learning architecture for network anomaly detection in large-scale heterogeneous IoT networks," *arXiv:2303.15986*, 2023.

[31] M. Nivaashini *et al.*, "FEDDBN-IDS: Federated deep belief network-based wireless network intrusion detection system," *EURASIP J. Information Security*, vol. 2024, Art. 22.

[32] E. Gelenbe, B. C. Gül, and M. Nakıp, "DISFIDA: Distributed self-supervised federated intrusion detection algorithm with online learning for health Internet of Things and Internet of Vehicles," *Internet of Things*, vol. 28, Art. 101340, 2024.

[33] P. Kumar, J. Liu, A. S. Md Tayeen, S. Misra, H. Cao, J. Harikumar, and O. Perez, "FLNET2023: Realistic Network Intrusion Detection Dataset for Federated Learning," in *Proc. IEEE Military Communications Conference (MILCOM)*, 2023, pp. 345–350.

HiSTM: Hierarchical Spatiotemporal Mamba for Cellular Traffic Forecasting

Zineddine Bettouche, Khalid Ali, Andreas Fischer, Andreas Kassler
Deggendorf Institute of Technology
Dieter-Görlitz-Platz 1, 94469 Deggendorf
{zineddine.bettouche, khalid.ali, andreas.fischer, andreas.kassler}@th-deg.de

***Abstract*—Cellular traffic forecasting is essential for network planning, resource allocation, or load-balancing traffic across cells. However, accurate forecasting is difficult due to intricate spatial and temporal patterns that exist due to the mobility of users. Existing AI-based traffic forecasting models often trade-off accuracy and computational efficiency. We present Hierarchical SpatioTemporal Mamba (HiSTM), which combines a dual spatial encoder with a Mamba-based temporal module and attention mechanism. HiSTM employs selective state space methods to capture spatial and temporal patterns in network traffic. In our evaluation, we use a real-world dataset to compare HiSTM against several baselines, showing a 29.4% MAE improvement over the STN baseline while using 94% fewer parameters. We show that the HiSTM generalizes well across different datasets and improves in accuracy over longer time-horizons.**

***Index Terms*—Time series forecasting, spatiotemporal modeling, 5G network traffic prediction, deep learning, state space models, Mamba architecture, attention mechanisms, convolutional neural networks (CNNs), hierarchical modeling, AI for telecommunications.**

I. Introduction

Accurate traffic forecasting plays a critical role in enabling predictive network resource allocation, network planning and network optimization, directly influencing the operational expenditure of telecom providers [1]. Traditional time series forecasting methods such as ARIMA [2] and AI-based models such as LSTM [3] often treat each base station independently and fail to account for spatial dependencies among neighboring cells. To address this challenge, recent work focused on employing AI-based models jointly with data preprocessing and feature engineering techniques to incorporate spatial knowledge in the training procedure [4], [5].

A key challenge in spatiotemporal modeling is the trade-off between using a single global model and multiple cell-specific models. While cell-specific models can capture local dynamics with higher fidelity, they are costly to train (as they require one model per cell), validate, deploy, and maintain, especially at the scale of modern cellular networks. Conversely, global models may benefit from exploiting shared patterns across cells but often underperform due to distributional heterogeneity [1].

Alternatively, a global model can be trained to cover the entire spatial grid while incorporating the spatial dependencies as features, formulating the forecasting task as a multivariate timeseries forecasting problem. However, such global models often marginally improve performance due to their limited ability to capture the intricate spatiotemporal dynamics effectively. In 5G networks, user mobility and overlapping coverage areas introduce non-trivial spatial correlations [6], making standalone temporal modeling insufficient as they may introduce large forecasting errors.

Consequently, a single spatiotemporal model capturing both temporal dynamics and latent spatial interactions may exploit correlations among neighboring cells better, leading to increased forecasting accuracy. Indeed, recent advancements using graph neural networks (GNNs) and transformers have shown promise in learning complex spatiotemporal dependencies [7]. However, these architectures typically suffer from high computational cost, making them less viable for real-time or large-scale deployments.

To address these challenges, we propose HiSTM, a novel spatiotemporal forecasting model based on the Mamba architecture [8], leveraging a hierarchical structure that captures both local variability and global patterns. Unlike graph-based or attention-heavy models, HiSTM demonstrates promising performance with reduced resource requirements, offering a favorable trade-off between accuracy and computational efficiency. Our contributions are summarized as follows:

- We propose HiSTM, a novel spatiotemporal forecasting architecture that leverages the standard Mamba model combined with optimized data processing—applying spatial CNNs per frame, temporal modeling over spatial patches via Mamba, and temporal attention to aggregate time-step features for improved representation.
- We apply HiSTM on real-world open-source cellular traffic datasets, demonstrating improved forecasting accuracy (Milan dataset) and consistent generalization ability when evaluated on unseen dataset of a different city (Trentino dataset).
- We show that HiSTM achieves a 29.4% MAE reduction compared to the STN baseline and competitive performance against other methods.
- HiSTM achieves better accuracy over longer forecasting horizons, with a 58% slower error accumulation rate than STN, highlighting its ability to capture long-term temporal dependencies.

The rest of this paper is structured as follows: Section II reviews 5G traffic prediction methods. Section III formalizes the forecasting problem, introduces HiSTM and baselines

we compare against. Section IV describes the datasets and experimental setup. Section V presents results and analysis. Section VI concludes the study and outlines directions for future work.

II. Related Work

Traditionally, 5G traffic forecasting is treated as a purely temporal forecasting task, relying on recurrent neural networks. AI-based models such as vanilla RNNs, LSTMs [3], and GRUs [9] were applied to learn sequential traffic patterns, often achieving gains over classical time-series methods such as ARIMA [2]. However, these approaches inherently ignore spatial correlations. Prior studies note that while RNN-based methods can capture long-term dependencies, they neglect spatial context, reducing their effectiveness [10] [11]. For example, Chen et al. proposed a multivariate LSTM (MuLSTM) using dual streams for traffic and handover sequences, but the model remains fundamentally temporal [12]. Temporal models employ hybrid or preprocessing strategies for improved accuracy. For instance, Hachemi et al. [4] applied an FFT filter before LSTM to separate periodic signals, while Wang and Zhang [5] used Gaussian Process Regression alongside LSTM to manage bursty patterns. These methods often operate at hourly aggregation levels and improve accuracy in long-term forecasts, though they still lack spatial awareness and incur higher computational cost.

Recently, spatial models have become popular as they can exploit spatial correlations within the data. For example, grid-based methods apply ConvLSTM [13] or 3D-CNNs to traffic maps [14], though these require regular cell layouts. Graph-based architectures, such as STCNet and its multi-component variant A-MCSTCNet, combine CNNs or GRUs with attention mechanisms [15]. STGCN-HO uses graph convolutions based on handover-derived adjacency to model spatial links across base stations, with Gated Linear Units (GLUs) for time dependencies [16]. Many of these models supplement deep networks with auxiliary features, such as weekday labels or slice-specific parameters, to improve generalization. More advanced spatiotemporal architectures, such as DSTL, adopt a dual-step transfer learning scheme that clusters gNodeBs and fine-tunes shared RNN models per group [17]. It forecasts at a 10-minute interval while reducing training overhead. To the best of our knowledge, Mehrabian et al. [18] published the first model to incorporate the Mamba framework into a spatiotemporal graph-based predictor for 5G traffic. It adapts a dynamic graph structure within a bidirectional Mamba block to model complex spatial and temporal dependencies.

In contrast, our model, HiSTM, uses the Mamba framework and adapts data input and output processing to work directly on spatial grids with temporal attention layers. This design improves accuracy and efficiency, especially for long-term forecasts and generalization to unseen data.

III. Methodology

This section formalizes the spatiotemporal forecasting problem, describes the proposed model architectures, and outlines the baseline models that we compare against.

A. Problem Formulation

Cellular network traffic forecasting aims to predict future traffic volume based on historical observations. We formulate this task as a spatiotemporal sequence prediction problem. Given a sequence of traffic measurements $M = \{M_1, M_2, ..., M_T\}$ where each $M_t \in \mathbb{R}^{H \times W}$ represents a spatial grid of $H \times W$ cells at time step t, our goal is to predict the next traffic volume grid M_{T+1}. Each grid cell contains a scalar value representing the traffic volume. This volume correlates with resource demands in the cellular network.

We define our input tensor as $X_t^{(i,j)} \in \mathbb{R}^{T \times K \times K}$, where T is the sequence length (window of observation time steps), $K \times K$ represents the spatial dimensions of the input kernel, and (i, j) are the spatial coordinate in the grid M_t. The prediction target is $x_{t+1}^{(i,j)} \in \mathbb{R}$, representing the traffic volume at the center cell of the kernel for the next time step. Formally, we aim to learn a function $f : \mathbb{R}^{T \times K \times K} \to \mathbb{R}$ that minimizes the prediction error $\min_\theta L(f_\theta(X), x_{t+1}^{(i,j)})$, where L is the loss function (i.e., Mean Absolute Error), and θ represents the learnable parameters of the model. The spatiotemporal nature of the data introduces a unique challenge: capturing both spatial correlations between neighboring cells and temporal dependencies across the sequence.

B. Proposed Architecture: HiSTM

We propose the Hierarchical SpatioTemporal Mamba (HiSTM), an architecture that combines hierarchical spatiotemporal processing with attention-based temporal aggregation (Figure 1). Given an input tensor $\mathbf{X} \in \mathbb{R}^{T \times K \times K}$ (where T is the number of time steps and $K \times K$ is the spatial kernel), the model predicts target values through three key components:

1) Hierarchical Spatiotemporal Encoding: The input $\mathbf{X}$ is first augmented with an initial channel dimension (i.e., $D_{in} = 1$), then passes through N stacked Encoder Layers. Each Encoder Layer l transforms its input $\mathbf{X}^{(l-1)}$ (where $\mathbf{X}^{(0)}$ is the initial augmented input) into an output $\mathbf{X}^{(l)} \in \mathbb{R}^{T \times K \times K \times C}$. The operations within each layer are:

- **Spatial Convolution**: Input features are first reshaped appropriately (e.g., to $T \times D'_{in} \times K \times K$). A 2D convolution followed by a ReLU activation is applied. The first layer up-projects D'_{in} to C channels. Subsequent layers take $D'_{in} = C$ channels and output C channels.
- **Temporal Mamba Processing**: The C-channel output from the convolution is reshaped to $\mathbf{X}_{\text{flat}} \in \mathbb{R}^{(K^2) \times T \times C}$. A Mamba SSM [8] then models the temporal dependencies for each of the K^2 spatial locations treated as sequences of length T, with $d_{\text{mamba}} = C$.

We use Mamba for its state space foundation, which models sequences through continuous-time dynamics rather than attention. This enables precise control over temporal structure and inductive bias, making it suitable for forecasting tasks where long-range temporal dependencies interact with fine-grained spatial patterns. Its ability to selectively retain and

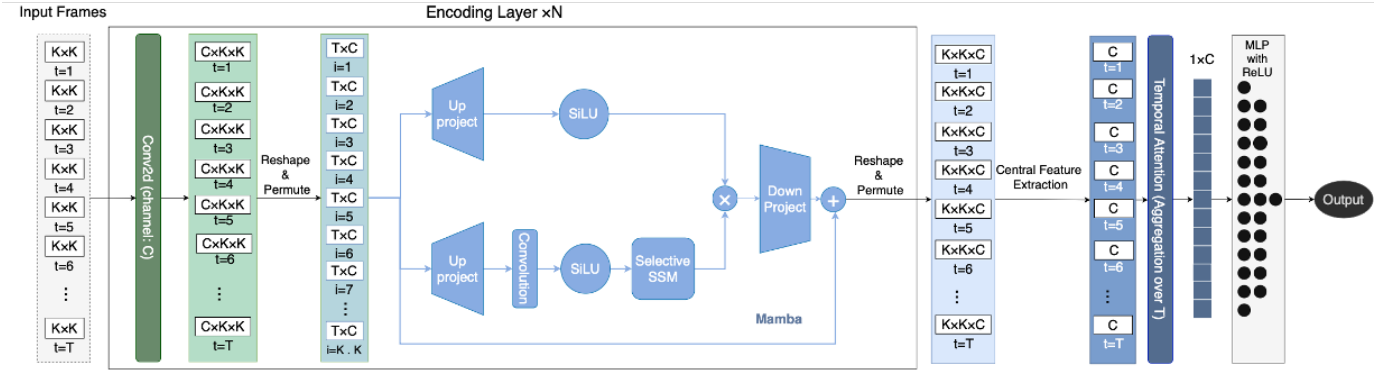


Figure 1. HiSTM Architecture

propagate information aligns with the demands of spatiotemporal modeling.

The Mamba output is reshaped back to $\mathbb{R}^{T\times K\times K\times C}$, forming $\mathbf{X}^{(l)}$. The output of this encoding stage is $\mathbf{X}_{\text{encoded}} = \mathbf{X}^{(N)}$.

2) Temporal Attention-Based Aggregation: From the encoded features $\mathbf{X}_{\text{encoded}} \in \mathbb{R}^{T\times K\times K\times C}$, features corresponding to the center spatial cell are extracted across all T time steps. This yields a sequence $\mathbf{X}_{\text{center}} \in \mathbb{R}^{T\times C}$. An attention mechanism then computes an aggregated context vector $\mathbf{c} \in \mathbb{R}^{B\times C}$:

$$e_t = \text{Linear}_{\text{att}}(\mathbf{h}_t), \quad \alpha_t = \frac{\exp(e_t)}{\sum_{j=1}^{T}\exp(e_j)}, \quad \mathbf{c} = \sum_{t=1}^{T}\alpha_t\mathbf{h}_t \tag{1}$$

where $\mathbf{h}_t \in \mathbb{R}^C$ is the feature vector from $\mathbf{X}_{\text{center}}$ (for a given batch instance) at time step t. The $\text{Linear}_{\text{att}}$ layer maps from $\mathbb{R}^C \to \mathbb{R}$, producing a scalar energy e_t. The softmax function normalizes these energies across all T time steps to obtain attention weights α_t.

3) Prediction Head: Finally, the aggregated context vector $\mathbf{c} \in \mathbb{R}^C$ is fed into a Multilayer Perceptron (MLP) head. This MLP consists of two linear layers with a ReLU activation in between, passing the dimensionality from C to MLP_{in}, and then to 1, to produce the final prediction $\hat{y} \in \mathbb{R}^1$.

C. Baseline Models

We benchmark HiSTM against the following baselines:

- **STN** [14]: A deep neural network capturing spatiotemporal correlations for long-term mobile traffic forecasting.
- **xLSTM** [19]: A scalable LSTM variant with exponential gating and novel memory structures (sLSTM/mLSTM) designed as Transformer alternatives. For fair comparison, our implementation uses only one mLSTM layer (denoted as `xLSTM[1:0]` in their paper) to prioritize computational efficiency while retaining its parallelizable architecture.
- **STTRE** [20]: A Transformer-based architecture leveraging relative embeddings to model dependencies in multivariate time series.

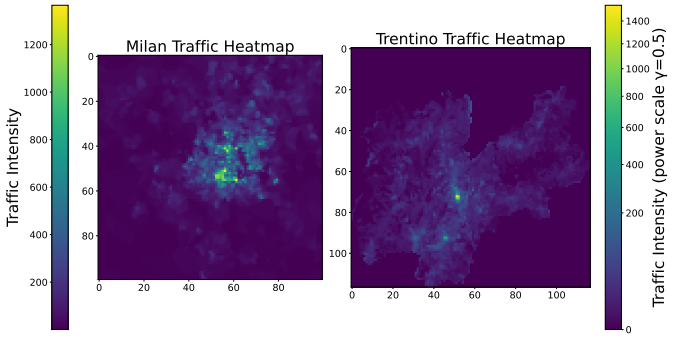


Figure 2. Spatial distribution of traffic flow intensity across Milan and Trentino regions.

- **VMRNN-B & VMRNN-D** [21]: Vision Mamba-LSTM hybrids addressing CNN's limited receptive fields and ViT's computational costs. VMRNN-B (basic) and VMRNN-D (deep) use Mamba's selective state-space mechanisms for compact yet competitive performance.

IV. Datasets & Experimental Setup

This section outlines the datasets, preprocessing steps, training configuration, and evaluation process used to assess model performance in spatiotemporal data forecasting.

A. Datasets

We use the dataset introduced by Barlacchi et al. [22], which contains two sub-datasets from Milan and Trentino with 10,000 and 11,466 spatial cells, respectively. Both datasets record SMS-in/out, Call-in/out, and Internet Traffic Activity in 10-minute intervals. The data exhibits spatial heterogeneity (see Figure 2), with higher activity in central regions and localized clusters elsewhere, adding complexity to forecasting tasks. To better visualize these differences, the Milan heatmap uses a linear scale, while the Trentino heatmap employs a logarithmic scale to highlight the dense urban center against its sparsely populated surroundings.

Lag plot analysis (Figure 3) reveals that both datasets maintain correlation at higher lag values. Individual cell traffic demonstrates higher volatility (Approximate Entropy [23] of

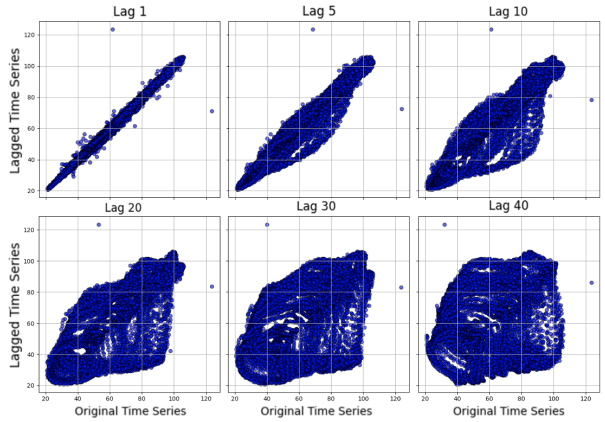


Figure 3. Lag plot for the autocorrelation of the dataset for the entire grid.

1.386 for a single cell) compared to spatially aggregated traffic (0.196), with the latter exhibiting enhanced cyclical patterns and passing the Augmented Dickey Fuller test [24] for stationarity. It is evident that the aggregated series is more correlated with itself for different lags; hence, it is easier to predict. Consequently, incorporating the spatial element in the prediction can help the model to capture the distribution more effectively, reduce the influence of individual events and magnify predictable cyclical patterns. This makes the series more predictable in a spatiotemporal context compared to when using individual per-cell series.

B. Data Preprocessing

To ensure robust spatiotemporal feature extraction, chronological integrity, and leakage-free normalization for model training and evaluation, we preprocess the raw 100×100 spatial grid as follows. We set $K = 11$, with boundary effects mitigated by cropping. Temporal sequences are constructed by aggregating six consecutive time steps. For the training set, input sequences are generated using a temporal stride of 6 between consecutive samples, minimizing temporal overlap from a single grid to promote feature diversity. During testing, a stride of 1 is used to ensure exhaustive evaluation across all temporal segments. The model is trained to predict the central value of the 11×11 kernel at the seventh (subsequent) time step. The preprocessed dataset is chronologically partitioned into training (70%), validation (15%), and test (15%) sets to preserve temporal order and prevent information leakage from future to past. Input features and target values are normalized to the `[0, 1]` range via Min-Max scaling, where scaling parameters are derived from the training data. Validation and test sets are transformed using these parameters, and out-of-range values are clipped to maintain the bounds.

C. Implementation and Training Configuration

We implement the HiSTM model in PyTorch, leveraging GPU-optimized operations for its `Mamba` and `Conv2D` modules to ensure computational efficiency. For reproducibility, we release the complete source code for our HiSTM implementation in a public repository [25]. For training and inference, we use an AI-server with a single NVIDIA A100 80GB GPU with 64 CPU cores and 512 GB RAM, using CUDA 12.4 and PyTorch 2.6.0+cu124. HiSTM and baseline models were trained with a batch size of 128 for up to 40 epochs, using early stopping with a patience of 15 and saving the best model based on validation loss. We use Adam optimizer with a learning rate of 10^{-4} and `ReduceLROnPlateau` scheduler (patience: 7, factor: 0.5).

D. Evaluation Metrics

We evaluate prediction accuracy using four complementary metrics. Mean Absolute Error (MAE) measures the average absolute difference between predictions and ground truth. Root Mean Squared Error (RMSE) penalizes large errors more heavily. The coefficient of determination (R^2) quantifies the proportion of variance explained by the model. Structural Similarity Index (SSIM) assesses the visual quality of spatial predictions. All metrics are computed after reversing the normalization to original scale.

V. Results and Analysis

This section evaluates forecast accuracy, cross-dataset generalization, and computational efficiency, comparing our proposed HiSTM model against baselines.

A. Prediction Accuracy

1) Single-step Prediction Results: HiSTM achieves the best single-step forecasting performance on the Milan dataset, with an MAE of 5.2196 and SSIM of 0.9925 (see Table I). Numbers are averaged across all spatial cells. This corresponds to a 29.4% MAE reduction over STN and a 2.3% SSIM gain, while outperforming other architectures such as STTRE (MAE: 5.5558) and parameter-intensive models like VMRNN-D (MAE: 6.4151). HiSTM also reports the lowest RMSE (11.2476) and the highest R^2 score (0.9799), indicating both lower large-error impact and improved variance explanation.

In addition to numerical metrics, Figure 4 provides a density-based comparison of predicted vs. actual values for all models. HiSTM shows a tighter concentration of points along the diagonal, particularly at higher traffic volumes, indicating improved predictive fidelity across the full dynamic range. It also exhibits fewer high-error outliers and lower dispersion compared to other models. STN and VMRNN-B, by contrast, display broader scatter, especially in the high-traffic regime, with more frequent large deviations. The distributional compactness in HiSTM suggests better generalization and robustness across heterogeneous spatial patterns.

2) Multi-step Autoregressive Forecasting: HiSTM demonstrates improved stability over extended forecasting horizons (Table II). While all models accumulate error with each additional step, HiSTM maintains the lowest MAE and RMSE across all six steps. At step 6, HiSTM reports an MAE of 6.69, which is 36.8% lower than STN (10.59) and 11.3% below STTRE (7.54). The slope of MAE progression is 58% lower

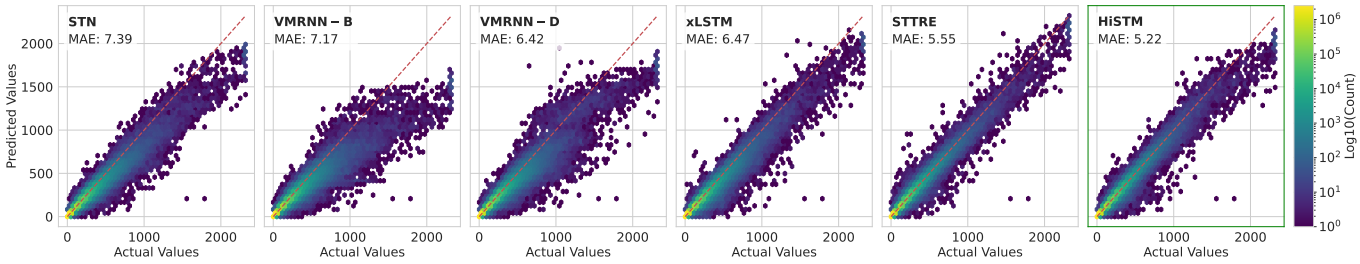


Figure 4. Density comparison of predicted vs. actual values over the entire grid for HiSTM and baseline models, with associated MAE

Table I
SINGLE-STEP PREDICTION PERFORMANCE ON MILAN DATASET. BEST MODEL INDICATED THROUGH BOLD FONT.

Model	MAE ↓	RMSE ↓	R^2 Score ↑	SSIM ↑
STN	7.3908	16.8824	0.9546	0.9853
VMRNN-B	7.1659	19.0876	0.9420	0.9843
xLSTM	6.4672	15.0901	0.9637	0.9870
VMRNN-D	6.4151	16.3284	0.9575	0.9873
STTRE	5.5558	11.4426	0.9791	0.9917
HiSTM	**5.2196**	**11.2476**	**0.9799**	**0.9925**

than that of STN, indicating reduced error propagation. SSIM also degrades more gradually in HiSTM, remaining above 0.95 even at step 6. This suggests that HiSTM better preserves temporal dependencies and structural consistency across time.

3) Cross-dataset Generalization on Trentino dataset: On the unseen Trentino dataset, HiSTM achieves a 47.3% MAE reduction (1.3870 vs. 2.6344) and a 36.9% RMSE improvement (4.8134 vs. 7.6370) over STN (see Table III). It also yields the highest SSIM (0.9916) and R^2 score (0.9649), confirming strong structural preservation. HiSTM outperforms all baselines across all evaluation metrics, including STTRE (MAE: 1.8132), VMRNN-D (MAE: 1.5870), and VMRNN-B (MAE: 1.9751). This demonstrates HiSTM's capacity to generalize to new spatial environments with distinct activity distributions and scales.

B. Cell-specific Modeling and Spatially-aware Accuracy

To evaluate the spatially-aware accuracy of the model, we use Milan dataset and select a random 7-day time period (1008 time steps) from the test dataset. We use a 6-step memory window and an 11-step kernel on a 100×100 sensor grid to forecast a single next timestep. To analyze spatial performance across varying traffic conditions, we select four representative cells, corresponding to (a) urban (b) suburban (c) rural and (d) the cell with the maximum temporal variance (Figure 5). The predicted versus actual traffic trajectories for all four cells are shown in Figure 6.

The model achieved its lowest MAPE (8.49%) for the urban cell, demonstrating strong performance in dense traffic zones. Prediction accuracy remained stable for the suburban cell (11.62% MAPE) with only minor deviations during peak periods. The rural cell showed higher relative error (24.86%

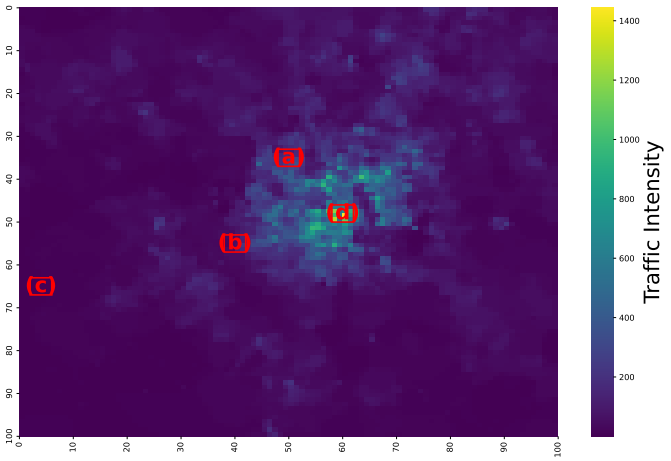


Figure 5. Selected cells from the Milan's traffic network. The cells represent different traffic patterns: (a) urban, (b) suburban, (c) rural, and (d) maximum variance cell.

MAPE), where low absolute traffic volumes magnified percentage errors. Notably, the model maintained robust performance (15.30% MAPE) for the high-variance cell despite its extreme fluctuations, highlighting HiSTM's capacity to handle volatile temporal patterns while showing expected limitations in sparse rural conditions where signal-to-noise ratios are challenging. These performance differences reflect the interaction between model complexity and regional traffic characteristics: urban and suburban areas offer rich temporal patterns and denser signals that align well with the model's spatiotemporal structure, whereas rural areas pose challenges due to sparse activity and fewer recurring patterns. The high-variance cell underscores the model's ability to generalize to non-stationary behavior, though peak underestimation suggests further headroom for improvement in handling extremes.

C. Computational Efficiency Analysis

HiSTM achieves a better performance while requiring fewer model parameters (Table IV). At 33.8K parameters (0.13 MB), it requires 18× fewer parameters than VMRNN-D and 5.4× less than xLSTM while delivering faster inference (1.19 ms) than all baselines, except xLSTM. Though its multiply-accumulate operations (MACs, a standard metric for computational workload) total 1.36×10^7—higher than

Table II
PERFORMANCE COMPARISON OF MODELS OVER MULTIPLE STEPS (AUTOREGRESSIVE FORECASTING). BEST MODEL INDICATED THROUGH BOLD FONT.

Step	HiSTM			STN			STTRE			xLSTM			VMRNN-B			VMRNN-D		
	MAE	RMSE	SSIM	MAE	RMSE	SSIM	MAE	RMSE	SSIM	MAE	RMSE	SSIM	MAE	RMSE	SSIM	MAE	RMSE	SSIM
1	**3.87**	**9.54**	**0.9833**	5.28	13.05	0.9742	4.21	9.83	0.9808	4.64	11.86	0.9759	5.11	13.75	0.9745	4.62	12.25	0.9777
2	**4.38**	**10.95**	**0.9785**	6.44	16.29	0.9615	4.88	11.45	0.9739	5.62	14.63	0.9639	5.08	13.85	0.9740	4.61	12.36	0.9772
3	**4.85**	**12.06**	**0.9735**	7.39	18.54	0.9490	5.47	12.73	0.9668	6.47	16.78	0.9509	6.06	16.33	0.9637	5.35	14.30	0.9698
4	**5.56**	**13.42**	**0.9680**	8.59	21.05	0.9335	6.27	14.12	0.9586	7.50	18.95	0.9356	6.83	17.64	0.9561	6.08	15.49	0.9641
5	**6.09**	**14.62**	**0.9633**	9.55	22.94	0.9196	6.88	15.28	0.9513	8.34	20.58	0.9217	7.45	18.91	0.9492	6.59	16.54	0.9591
6	**6.69**	**16.02**	**0.9578**	10.59	25.01	0.9033	7.54	16.60	0.9430	9.23	22.38	0.9064	8.24	20.38	0.9398	7.30	17.87	0.9520

Table III
SINGLE-STEP GENERALIZATION PERFORMANCE ON TRENTINO DATASET. BEST MODEL INDICATED THROUGH BOLD FONT.

Model	MAE ↓	RMSE ↓	R^2 Score ↑	SSIM ↑
STN	2.6344	7.6370	0.9116	0.9762
xLSTM	2.5974	8.9235	0.8793	0.9615
VMRNN-B	1.9751	6.6270	0.9334	0.9839
STTRE	1.8132	5.0050	0.9620	0.9903
VMRNN-D	1.5870	5.5754	0.9529	0.9885
HiSTM	**1.3870**	**4.8134**	**0.9649**	**0.9916**

Table IV
MODEL COMPARISON

Model	Parameter Count	Size (MB)	GPU (MB)	Inference (ms)	MACs
xLSTM	607,753	2.32	13.69	1.01	5.96×10^5
VMRNN-B	137,282	0.52	**9.77**	8.16	2.06×10^7
VMRNN-D	1,506,498	5.75	15.47	18.58	5.52×10^7
STTRE	165,380	0.63	58.07	4.54	2.83×10^7
STN	576,755	2.20	11.34	2.46	2.31×10^6
HiSTM	**33,794**	**0.13**	10.63	**1.19**	1.36×10^7

STN's—HiSTM balances computational cost with accuracy, achieving a MAC/MAE ratio 2.6× better than STTRE's. This efficiency-profile positions HiSTM as a practical solution for resource-constrained deployment scenarios.

VI. CONCLUSION

In this paper, we presented HiSTM, a hierarchical spatiotemporal model for efficient and accurate cellular traffic forecasting. By combining dual spatial encoders with a Mamba-based temporal module and an attention mechanism, HiSTM captures complex spatiotemporal patterns with minimal overhead. Experiments on real-world datasets show that HiSTM demonstrates competitive performance, reducing MAE by 29.4% compared to STN baseline while maintaining computational efficiency. It generalizes well to the unseen Trentino dataset and sustains lower errors over longer forecast horizons. These results suggest HiSTM's potential as an efficient approach for cellular traffic prediction, though broader validation across diverse network conditions would strengthen deployment recommendations.

Future work will explore mixture-of-experts approaches to better model spatially-clustered traffic patterns and kernel-to-kernel forecasting to capture finer-grained temporal dynamics across service types. We aim to investigate diffusion-based decoding strategies to enhance long-range predictive capabilities and extend the model's scope through different aggregation strategies.

To strengthen generalizability, we will evaluate HiSTM on diverse geographical datasets beyond Italy and implement attention visualization for improved interpretability. We also plan to address practical deployment challenges including

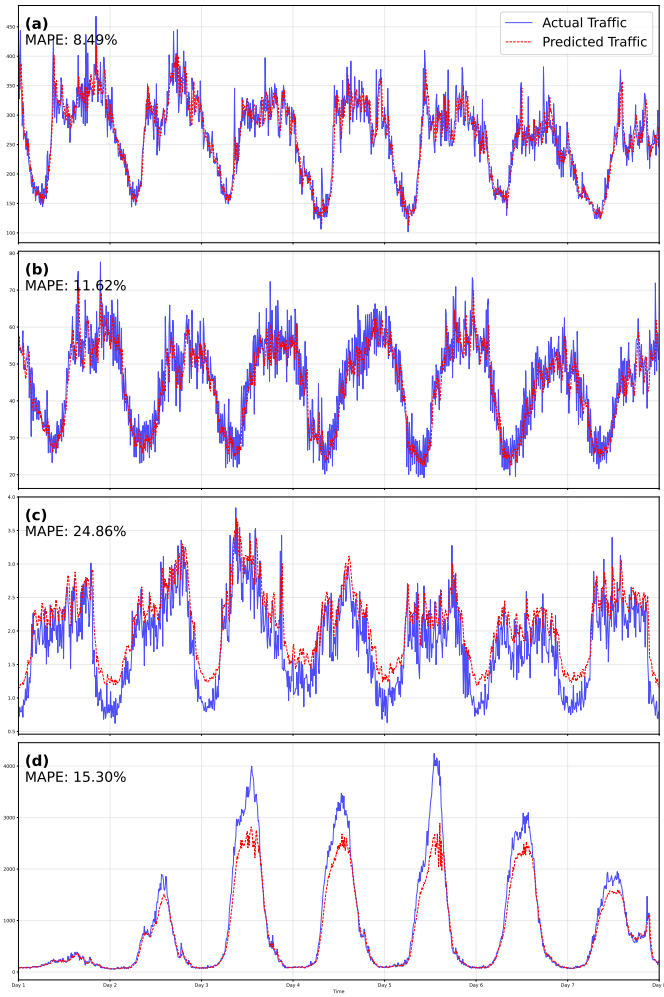


Figure 6. Week-long traffic predictions from the HiSTM model across four representative cells in Milan's traffic network. The cells were selected to represent different traffic patterns: (a) urban, (b) suburban, (c) rural, and (d) maximum variance cell. Time is shown in days, with each day containing 144 readings (10-minute intervals)

missing data handling, concept drift adaptation, and spatial heterogeneity through adaptive weighting mechanisms.

ACKNOWLEDGEMENT

This work was partly funded by the Bavarian Government by the Ministry of Science and Art through the HighTech Agenda (HTA).

REFERENCES

[1] W. Shao, J. Wang, Z. Guo, M. Xu, and Y. Wang, "Spatial-temporal neural network for wireless network traffic prediction," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2104–2113, 2020.

[2] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, New Jersey: Wiley, 2015.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] H. Hachemi, V. Vidal, and K. Boussetta, "Towards real-time mobile traffic prediction with fft-lstm," in *2020 IEEE GLOBECOM*, 2020.

[5] J. Wang and T. Zhang, "Gaussian process assisted lstm for 5g traffic prediction," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2472–2485, 2020.

[6] B. Yu, H. Yin, Z. Zhu, and Q. Zhang, "Spatiotemporal graph neural network for urban traffic flow prediction," *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[7] R. Gu, Q. Liu, X. Li, and Q. Zhu, "Glsttn: A global-local spatial-temporal transformer network for traffic prediction," *IEEE Access*, vol. 9, pp. 152 323–152 334, 2021.

[8] A. Gu, T. Dao, A. Rudra, B. Recht, and T. B. Hashimoto, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2024.

[9] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[10] X. Ma, J. Zhang, and X. S. Shen, "Traffic prediction for mobile networks using machine learning: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2141–2169, 2019.

[11] Y. Li, L. Duan, H. Liu, and X. Wang, "A survey on deep learning techniques in wireless resource allocation for 5g and beyond," *IEEE Wireless Communications*, vol. 28, no. 5, pp. 152–159, 2021.

[12] L. Chen, T.-M.-T. Nguyen, D. Yang, M. Nogueira, C. Wang, and D. Zhang, "Data-driven c-ran optimization exploiting traffic and mobility dynamics of mobile users," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 1773–1788, 2021.

[13] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[14] C. Zhang and P. Patras, "Long-term mobile traffic forecasting using deep spatio-temporal neural networks," in *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–15.

[15] I. Narmanlioglu and A. Cicek, "Multi-component spatio-temporal modeling for cellular traffic forecasting," *IEEE Access*, vol. 10, pp. 12 771–12 784, 2022.

[16] C. Borcea, J.-M. Gorce, and M. Duflot, "Stgcn-ho: Handover-aware spatiotemporal graph convolutional network for mobile traffic forecasting," *IEEE Transactions on Mobile Computing*, 2023.

[17] S. Aziz, L. Hu, and Z. Han, "Dstl: Dual-step transfer learning for spatiotemporal 5g traffic forecasting," *IEEE Transactions on Network and Service Management*, 2025, to appear.

[18] S. Mehrabian, L. Jiang, and S. Lee, "A-gamba: Adaptive graph mamba network for 5g spatiotemporal traffic forecasting," *IEEE Transactions on Mobile Computing*, 2025, to appear.

[19] M. Beck, K. Pöppel, M. Spanring, A. Auer, O. Prudnikova, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter, "xlstm: Extended long short-term memory," 2024. [Online]. Available: https://arxiv.org/abs/2405.04517

[20] A. Deihim, E. Alonso, and D. Apostolopoulou, "Sttre: A spatio-temporal transformer with relative embeddings for multivariate time series forecasting," *Neural Networks*, vol. 168, pp. 549–559, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608023005361

[21] Y. Tang, P. Dong, Z. Tang, X. Chu, and J. Liang, "Vmrnn: Integrating vision mamba and lstm for efficient and accurate spatiotemporal forecasting," 2024. [Online]. Available: https://arxiv.org/abs/2403.16536

[22] G. Barlacchi, M. D. Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland, and B. Lepri, "A multi-source dataset of urban life in the city of milan and the province of trentino," *Scientific Data*, vol. 2, p. 150055, 2015. [Online]. Available: https://www.nature.com/articles/sdata201555

[23] S. M. Pincus, "Approximate entropy as a measure of system complexity," *Proceedings of the National Academy of Sciences*, vol. 88, no. 6, pp. 2297–2301, 1991.

[24] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American Statistical Association*, vol. 74, no. 366a, pp. 427–431, 1979.

[25] Z. Bettouche and K. Ali, "Histm: Hierarchical spatiotemporal mamba," https://github.com/ZineddineBtc/HiSTM-Hierarchical-Spatiotemporal-Mamba, 2025.

Predicting Performance Metrics in Edge-Cloud Networks using Graph Neural Networks

Christian Maier, Nina Großegesse, Felix Strohmeier
Salzburg Research Forschungsgesellschaft mbH
<firstname.lastname@salzburgresearch.at>

***Abstract*—This paper explores the application of Graph Neural Networks (GNNs) for predicting performance metrics in edge-cloud networks. By modeling the edge-cloud network as a graph, where nodes represent devices, and edges represent communication links, GNNs effectively capture the complex interdependencies and interactions within the network. We demonstrate that GNNs can accurately predict key performance metrics such as latency and jitter, using data from real network conditions. Our findings highlight the potential of GNNs to enhance performance monitoring and optimization in edge-cloud environments, paving the way for more efficient resource management and energy-efficiency.**

***Index Terms*—Graph Neural Networks, Edge-Cloud-Networks, Performance Prediction**

I. Introduction

Machine learning models as digital twins in network management take the state of the network as input and produce predictions of performance metrics in the network as output. In the context of edge-cloud networks, important metrics are end-to-end latency, jitter and packet-loss of packet flows. This performance prediction is intended to replace measurements or estimation by simulations, as these are usually associated with considerable effort (in terms of network overhead or computing time).

The prediction of performance metrics can be done with a variety of different machine learning models, like decision trees or artificial neural networks. A common approach in the literature is the prediction of network performance metrics using Graph Neural Networks. GNNs have emerged as a powerful tool for predicting performance metrics in communication networks due to their ability to model complex relationships and dependencies among network components. By representing the network as a graph, where nodes correspond to devices or network elements and edges represent connections or interactions, GNNs can effectively capture the topological structure and dynamic behaviour of the network.

Recent studies have demonstrated that GNNs can predict various performance metrics, such as latency, throughput, and packet loss, by learning from historical data and network configurations. Their capacity to incorporate both node features (e.g., bandwidth, processing power) and edge features (e.g., link quality, distance) allows for a more nuanced understanding of how different factors influence overall network performance.

Additionally, GNNs can generalize well to unseen network topologies, making them suitable for real-time applications where network conditions may change frequently. This adaptability, combined with their ability to process large-scale data efficiently, positions GNNs as a promising approach for enhancing the performance prediction capabilities in modern communication networks, particularly in scenarios involving dynamic and heterogeneous environments (which is particularly the case for edge cloud networks).

Training a GNN to predict performance metrics in communication networks typically involves several steps. First, the network is represented as a graph, where nodes correspond to network elements (such as routers and switches) and edges represent the connections between them. Each node and edge is associated with features that capture relevant information, such as bandwidth and traffic load. For GNNs, it is not relevant to have a complete picture of the underlying network including all components, but only for those nodes and links that shall be analysed.

The training process begins with the collection of historical performance data, which serves as the ground truth for the metrics to be predicted. This data is used to create labelled training samples, where the input consists of the graph structure and associated features, while the output corresponds to the performance metrics (e.g., latency, throughput).

During training, the GNN learns to aggregate information from neighbouring nodes and edges through multiple layers of message passing. This process allows the model to capture both local and global patterns in the graph. The GNN is typically trained using supervised learning techniques, where a loss function (such as mean squared error) measures the difference between the predicted metrics and the actual values from the training data. Optimization algorithms, such as Adam or stochastic gradient descent, are employed to minimize this loss function by adjusting the model parameters.

Once trained, the GNN can be evaluated on a separate validation dataset to assess its predictive performance. Fine-tuning may be performed to improve accuracy, and the model can be deployed for real-time predictions in dynamic communication environments, adapting to changes in network conditions as new data becomes available. Overall, the training of GNNs for performance metric prediction leverages the unique graph structure of communication networks to enhance prediction

This work has been partly funded by the Austrian Federal Ministry of Innovation, Mobility, and Infrastructure (BMIMI) and the National Austrian Funding Agency (FFG).

accuracy and efficiency.

The remainder of the paper is structured as follows. Section II surveys related work on performance prediction with machine learning approaches. Section III gives an overview of the background (both on the used measurement framework and on GNNs), Section IV provides a detailed description of the methodology and Section V presents the results of our evaluation. Finally, Section VI concludes the paper and provides an outlook to future work.

II. Related work

Performance metrics in edge cloud networks have attracted a lot of attention in recent years due to the increasing demand for low latency applications and the proliferation of Internet of Things (IoT) devices. Various studies have investigated different aspects of performance measurement, focusing on latency, bandwidth, reliability and resource utilization.

One of the pioneering works in the domain of predicting performance metrics of communication networks by a machine learning model which takes the graph structure of the network into account is by Rusek et al. [12]. They introduced a GNN-based framework called RouteNet for network performance prediction and demonstrated that GNNs could effectively capture the spatial and temporal dependencies in network data, leading to improved accuracy in predicting latency and throughput compared to traditional machine learning approaches. Their experiments showed that GNNs outperformed baseline models by leveraging the graph structure to learn from both node features and connectivity patterns. Building on this, there is a lot of further work that refine the RouteNet model (e.g. [5], [4]) and thus further improve accuracy of the predictions. In another significant contribution, Dai et al. [3] explored the use of GNNs for multi-task learning in communication networks. However, all these studies use simulation data and do not consider edge cloud networks in particular.

Furthermore, recent advancements have focused on integrating GNNs with reinforcement learning techniques to optimize network performance dynamically. For instance, Li et al. [8] developed a GNN-based reinforcement learning model that predicts performance metrics while simultaneously optimizing routing decisions in real-time. Their approach showcased the potential of combining predictive modelling with decision-making processes, leading to enhanced network efficiency and reduced latency. Li et al. [9] propose a distributed scheduler based on GNNs an reinforcement learning for edge clusters that minimizes the total completion time of ML tasks through co-optimizing task placement and fine-grained multi-resource allocation

III. Background

This section provides background information about the used performance measurement framework and about GNNs.

A. Performance Measurement Framework

Measuring performance on the network (IP) and transport (TCP/UDP) level can be done by various available tools (e.g. iperf3 [7], wireshark [6], ping [10]) and according to various metrics, such as IP performance metrics (IPPM), round-trip time (RTT), round-trips per minute (RPM), throughput or topology. Our approach to performance measurement is to use the MINER infrastructure as programmable orchestration framework [2]. MINER is a distributed Java application, where so-called "Toolproxies" are passively waiting for execution orders from a centralised measurement application. Communication between the nodes may be secured by a VPN. All configuration parameters are defined by the measurement application, which schedules the execution as soon as all the involved measurement tools are ready to run. Measurement tools can be standard tools integrated by the Toolproxy, or Miner-specific tools, such as the "IPPMTool". During and/or after the measurement execution, the results are collected, so that follow-up measurement analytics can take place centralised after the run.

After investigating different performance metrics and studying the related work in Section II, we considered latency, jitter, packet loss and throughput as the most relevant metrics in the edge-cloud continuum. For the analyses performed in this paper we measured latency, jitter on application and packet level, as well as packet loss by generating active UDP measurement flows with constant bitrates, using the Miner IPPMTool [2]. To receive a proper variation of measurement results, different packet sizes and rates are selected before starting the measurement flows.

B. Graph Neural Networks for Homogeneous Graphs

GNNs are machine learning models, which are made to process data in the form of graphs appropriately. Here, a graph is a mathematical object, consisting of nodes and (directed or undirected) edges between them. A frequently occurring task is the determination of quantities $y_v \in \mathbb{R}$ which are assigned to individual nodes v. In the context of a communication network, the nodes of the graph are typically the devices of the network, the edges represent communication links and a common quantity of a node is for example the utilization of a queue of the represented device. The task of a *generic* GNN can then be subdivided into two subtasks: First, a *hidden state vector* h_v needs to be computed for each node v. This vector h_v lies in some chosen state space $\mathbb{R}^m$ of dimension m and consists of information on the state of v at some level of granularity. It is computed by an iterative *message passing* scheme: After an initialisation of h_v with node-level features related to v, each node sends its state to all of its neighbours. Hence, each node v receives a certain number of states $h_{v_1}, \ldots, h_{v_k}$, where k is the number of neighbours of v. These states, together with the state h_v of v itself, are converted into an *aggregated message* m_v:

$$m_v = \sum_{i=1}^{k} M(h_v, h_{v_i}) \tag{1}$$

by a *message function* $M : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$. The dimension n of the codomain of M is again a chosen value. Using this aggregated message, an *update function* $U : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$ computes a new hidden state $U(h_v, m_v)$ for each node v. This message passing is repeated T times until the states of all nodes have (approximately) reached stationary values. Then, secondly, a *readout function* $R : \mathbb{R}^m \to \mathbb{R}$ computes y_v for each node v by applying R to h_v.

A generic GNN thus essentially consists of three functions: M, U and R. The mapping rules of these functions are given by the application of certain independent neural networks (like feed forward or recurrent neural networks). This justifies the name GNN and allows the execution of a training process: The internal parameters of the neural networks are updated via supervised learning in order to compute the target quantities y_v accurately. To achieve this, instances of graphs together with the target quantities y_v for all nodes have to be provided. Note that the way in which such a generic GNN is modelled enables an application to graphs of different sizes and structures (both during training and predicting).

Further details on this generic GNN architecture can be found in the literature [1, 11] and in the references provided there. Many GNN models deviate from the architecture of standard GNN models. Particularly, they consider heterogeneous graphs as input (i.e. with various types of nodes), and build a double message-passing phase to exchange the information between the different element types. In the next subsection we recall this architecture.

C. Graph Neural Networks for Heterogeneous Graphs

Heterogeneous graphs are graphs which consist of nodes of different types. For simplicity, we only consider heterogeneous graphs with nodes of two types A and B. The message passing explained in the previous subsection is then adapted to a so-called 2-stage message passing phase: In the first stage, nodes of type A send their messages to all nodes of type B to which they are connected. This is done in the same way as explained above. This results in updates of the hidden state vectors for all nodes of type B. After that, all nodes of type B send their messages to the nodes of type A, which the updates the states of the type A nodes. Notice that the message functions and update functions can be different for the two stages in the message passing phase. The readout function is then usually only applied to the nodes either of type A or of type B. In our approach (which we outline below), the nodes of the graphs correspond to links and flows in the physical network. Thus we will have two types of nodes. The readout function will then only be applied to nodes which correspond to flows, since we are eventually interested in calculating performance metrics of flows (mean latencies, mean jitter and packet loss).

IV. Methodology

This section describes our hardware testbed, the dataset which we created from this testbed, the architecture of our ML model and the approach taken in the training process.

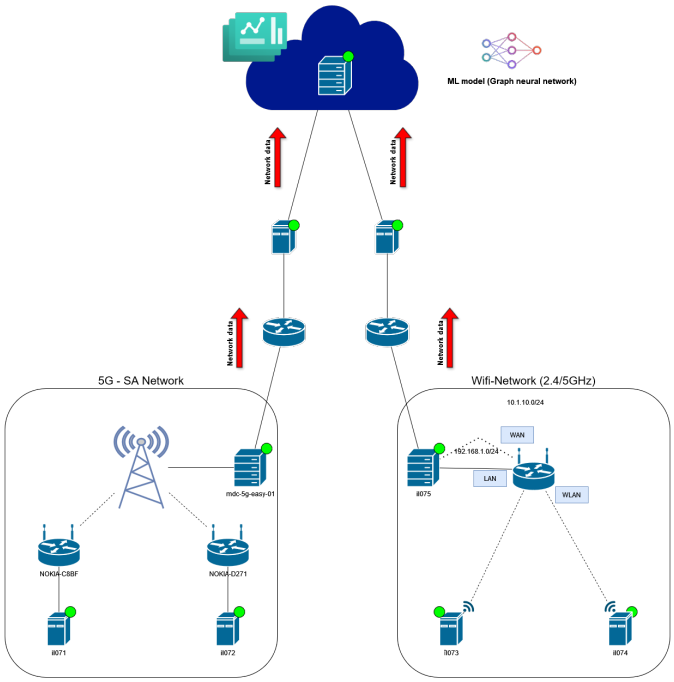


Fig. 1. Hardware testbed

A. Hardware Testbed

An overview diagram of our hardware setup is shown in Fig. 1. It consists of two local edge networks, one using 5G, the other one using Wi-Fi equipment. The 5G edge network is built using a 5G indoor base station located in our laboratory, connected to its 5G core located at the telecom provider premises. Two measurement endpoints are connected via wireless access routers to the indoor base station, a third measurement endpoint is installed in the 5G core. Measurement traffic flows are generated between all three nodes in both directions. More details on the measurements itself are available in Section IV-B.

The other edge network built on 5GHz-Wi-Fi-Technology (”Wi-Fi 5”), which also has three MINER measurement points installed. All measurement nodes are located in our laboratory, two on wireless nodes, and one node connected by wire to the wireless router. Also in this part of the network, measurement traffic flows are generated between all three nodes in both directions.

To be more precise, for each ordered pair of measurement points from the same local network, a packet flow with packets of constant size and constant inter packet time between two packets is generated. Each of the local edge networks is connected to the cloud node via an intermediate on-premise node.

B. Dataset

To create a dataset in our hardware testbed, we use the following approach: We randomly choose packet sizes and inter packet times for each packet flow in our local edge networks. Since there are six packet flows in each local network, this

```
directed: true
multigraph: false
graph: {}
nodes:
  0:
    entity: "node"
    f_v: 0
    id: "il072"
  1:
    entity: "node"
    f_v: 0
    id: "il071"
  2:
    entity: "node"
    f_v: 0
    id: "mdc-5g-easy-01"
links:
  0:
    f_e:
      0: "1046"
      1: "17.317"
    owdApp: "23.084415584415584"
    owdPcap: "23004.324675324675"
    jitterApp: "-0.03246753246753246"
    jitterPcap: "7.307359307359308"
    source: "il072"
    target: "il071"
  1:
    f_e:
      0: "1368"
      1: "91.773"
    owdApp: "22.25"
    owdPcap: "22264.476744186046"
    jitterApp: "-0.011363636363636364"
    jitterPcap: "-72.38823529411765"
    source: "il072"
    target: "mdc-5g-easy-01"
  2:
    f_e:
      0: "805"
      1: "34.413"
    owdApp: "20.689655172413794"
    owdPcap: "20607.82327586207"
    jitterApp: "-0.03879310344827586"
    jitterPcap: "29.745689655172413"
    source: "il071"
    target: "il072"
  3:
    f_e:
      0: "1433"
      1: "64.365"
    owdApp: "20.830645161290324"
    owdPcap: "20629.120967741936"
    jitterApp: "-0.04838709677419355"
    jitterPcap: "-51.016129032258064"
    source: "il071"
    target: "mdc-5g-easy-01"
  4:
    f_e:
      0: "683"
      1: "63.453"
    owdApp: "0.9920634920634921"
    owdPcap: "998.436507936508"
    jitterApp: "-0.007936507936507936"
    jitterPcap: "-8.976190476190476"
    source: "mdc-5g-easy-01"
    target: "il072"
  5:
    f_e:
      0: "1463"
      1: "42.156"
    owdApp: "1.6421052631578947"
    owdPcap: "1548.1736842105263"
    jitterApp: "-0.02631578947368421"
    jitterPcap: "-27.189473684210526"
    source: "mdc-5g-easy-01"
    target: "il071"
```


Fig. 2. An example data sample from a measurement in the local 5G-network. The nodes of the communication network are the nodes of the graph. The links of the graph are the packet flows in the communication network. For each packet flow, the parameters (packet size s and inter-packet-time t) are summarized in the list $f_e = [s, t]$.

results in twelve parameters for each measurement. The packet sizes are varied from 300 to 1500 Bytes and the packet rates between 10 and 100 packets per second resulting in bitrates of 20 to 1200 kbps for the measurement flows. The chosen packet sizes and inter packet times define the configuration of the network traffic. For each configuration, we measure the performance metrics explained in Sec. III (i.e. latency, jitter and packet loss, where latency and jitter are measured both on application and on packet level) for a measurement interval of 10 seconds, where we then use the mean value for latency and jitter. The data set is then saved as a JSON file. An example data sample is shown in Fig. 2.

Our data set in the 5G network consists of measurements from February 26, 2024 to December 31, 2024. Every day between 9 p.m. and 5 a.m., a ten-second measurement was taken every 4 minutes. We only took measurements at night and only every four minutes, because other measurements are taken in this network during the day and in other time slots (which are not relevant for this paper). As some of the measurements failed (for various reasons, e.g. power outages or disruptions in the 5G network), the pre-processed data does not contain a measurement result for every resulting measurement time.

For organizational reasons, the measurements in the Wifi-network ran from 22 May, 2024 to 31 December, 2024. The time between two measurements was also 4 minutes here (to be consistent with the 5G measurements), but measurements were taken throughout the whole day (to obtain a data set similar in size to that in the 5G network).

For each measurement and each transmitted packet, we saved the latency and jitter (both at Pcap and application level) and whether the packet was successfully transmitted. We thus obtained five time series for each packet flow. From these time series, we calculate statistical mean values for the latency and jitter. Fig. 3 and Fig. 4 shows an example of a histogram of the measurement results both for the 5G network and the WiFi network. For this calculation, however, we have neglected the first and last second of the respective time series (in order not to take into account any transient behaviour)

Since we did not have access to the 5G core, there is a synchronization issue in our latency measurements in the 5G network. However, this is irrelevant for our investigations. Synchronization of the computers in the laboratory takes place via their own network to a local, GPS-synchronized time server, not via the "network under test". The core was synchronized to an external time server. In general, synchronization is only important for one-way-delay measurements.

C. Architecture

We use a graph neural network model to predict performance metrics of packet flows in edge-cloud networks, cf. Fig. 5. As already explained, the packet flows consist of packets of constant size with constant inter-packet times between two packets.

The structure of the used 2-stage Graph neural network model is as follows: The nodes of the graph correspond to links and flows in the data set. This means that the graphs which we consider consist of nodes of two types: links and flows. For each link, there is a directed edge to each flow which uses this link. For each flow, there is a directed edge to each link which is used by this flow. Observe that the links of the first type are unordered, while the links of the second type are ordered (in the order in which the flow passes the individual links).

The architecture of the graph neural network is based on a two-stage message passing as described in Sec. III. The dimension of the hidden state vector space is 16. The message function is given by $M(h_v, h_w) = h_w$ for both stages, i.e. nodes simply send their hidden state to the nodes in their neighbourhood. The update function for updating states of links is a feed forward neural network with two hidden layers, each one consisting of 16 neurons with RELU activation functions. The update function for flow updates is a recurrent neural network consisting of GRU cells. The message passing is repeated for $T = 4$ times. The readout function is a feed

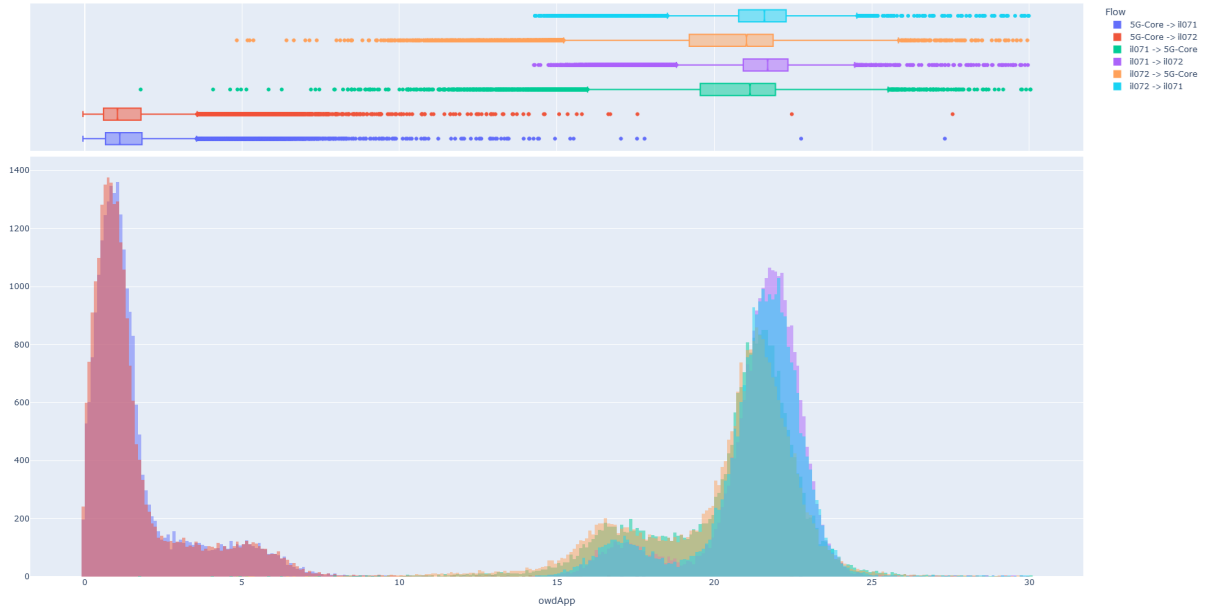


Fig. 3. Histograms of the measurement data of latency on application level in the 5G local network

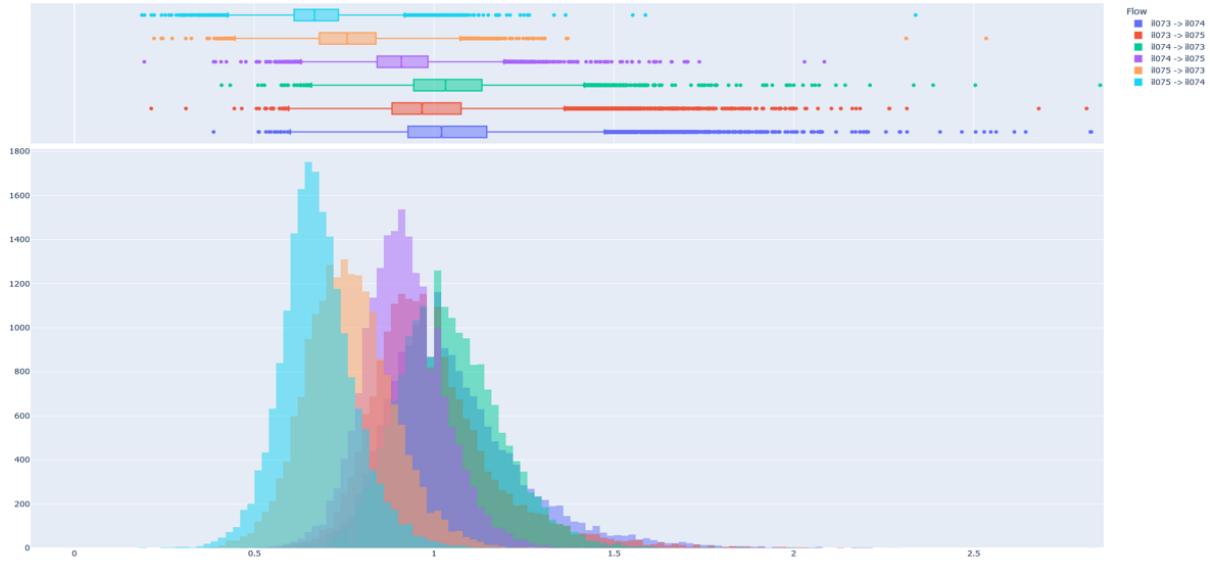


Fig. 4. Histograms of the measurement data of latency on application level in the WiFi local network

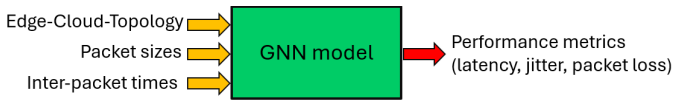


Fig. 5. Overview of the approach

forward neural network as well and with the same structure as the update function for links.

D. GNN Training

For the implementation of the GNN model, we used the iGNNition framework. iGNNITION is a TensorFlow-based framework for fast prototyping of GNNs. It provides a codeless programming interface, where users can implement their own GNN models in a YAML file. IGNNITION also incorporates a set of tools and functionalities that guide users during the design and implementation process of the GNN.

V. Evaluation

We evaluated the following scenarios: (A) One GNN model for each local edge network and (B) a common GNN model for both edge networks. For scenario (A), each model is only trained with the data from the corresponding edge network. For scenario (B), we use the whole data (both from the 5G network

and from the Wifi-Network) to train the model. In both scenarios, data was scaled in a preprocessing. The training of the two separate GNN models for each local network reached a satisfying performance for both latency and jitter prediction. For example, Fig. 6 shows the training process where the GNN model is trained to predict the latency on application level in the 5G network. For the other metrics we achieved similar results (both in the 5G and in the Wi-Fi network).

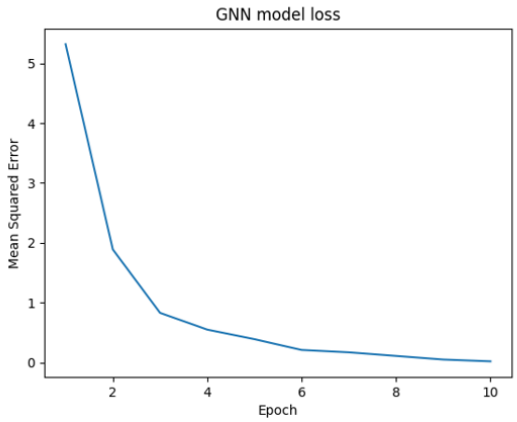


Fig. 6. Loss function of the GNN-model trained for predicting latency on application level in the local 5G network. The GNN-model for the local Wifi network achieved a similar performance.

However, as soon as we tried to train a joint model for both networks, we did not achieve sufficient performance (even with various attempts to tune the hyperparameters). Our explanation for this is that the distributions of the data in the two networks are too different (as can be observed for example from the graphs Fig. 3 and Fig. 4). Even scaling the data has not brought any improvement so far.

VI. Conclusion

This paper illustrates the significant potential of Graph Neural Networks as a powerful tool for performance prediction and network management in edge-cloud systems. By leveraging the inherent graph structure of these networks, GNNs provide accurate insights into critical metrics like latency and jitter, enabling more informed decision-making for resource allocation and optimization. The results underscore the promise of GNN-based approaches to advance the efficiency, reliability, and sustainability of edge-cloud infrastructure, ultimately contributing to more responsive and energy-efficient network environments. However, so far we have only achieved sufficient performance in a simple scenario (one machine learning model for each local network). We will continue to try to train a joint model for both local networks in the future. It would also be interesting to consider other metrics (such as energy efficiency) as well, or to consider more complex network scenarios.

References

[1] Guillermo Bernárdez, José Suárez-Varela, Albert López, Bo Wu, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Is machine learning ready for traffic engineering optimization? *IEEE International Conference on Network Protocols (ICNP)*, 2021.

[2] Christof Brandauer and Thomas Fichtel. Miner-a measurement infrastructure for network research. In *2009 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops*, pages 1–9. IEEE, 2009.

[3] Yueyue Dai, Xiaoyang Rao, Bruce Gu, Youyang Qu, Huiran Yang, and Yunlong Lu. Graph learning-based multi-user multi-task offloading in wireless computing power networks. *IEEE Internet of Things Journal*, 2025.

[4] Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet-fermi: Network modeling with graph neural networks. *IEEE/ACM transactions on networking*, 31(6):3080–3095, 2023.

[5] Miquel Ferriol-Galmés, Krzysztof Rusek, José Suárez-Varela, Shihan Xiao, Xiang Shi, Xiangle Cheng, Bo Wu, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet-erlang: A graph neural network for network performance evaluation. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 2018–2027. IEEE, 2022.

[6] Wireshark Foundation. Wireshark: The world's foremost network protocol analyzer. https://www.wireshark.org/, 1998. Accessed: 2025-05-28.

[7] The iPerf Development Team. iperf3: A tcp, udp, and sctp network bandwidth measurement tool. https://iperf.fr/, 2014. Accessed: 2025-05-28.

[8] Kai Li, Wei Ni, Xin Yuan, Alam Noor, and Abbas Jamalipour. Deep-graph-based reinforcement learning for joint cruise control and task offloading for aerial edge internet of things (edgeiot). *IEEE Internet of Things Journal*, 9(21):21676–21686, 2022.

[9] Yihong Li, Xiaoxi Zhang, Tianyu Zeng, Jingpu Duan, Chuan Wu, Di Wu, and Xu Chen. Task placement and resource allocation for edge machine learning: A gnn-based multi-agent reinforcement learning paradigm. *IEEE Transactions on Parallel and Distributed Systems*, 34(12):3073–3089, 2023.

[10] Mike Muuss. The PING program. https://ftp.arl.army.mil/~mike/ping.html, 1983. Accessed: 2025-05-28.

[11] David Pujol-Perich, José Suárez-Varela, Miquel Ferriol, Shihan Xiao, Bo Wu, Albert Cabellos-Aparicio, and Pere Barlet-Ros. IGNNITION: Bridging the gap between graph neural networks and networking systems. *IEEE Network, in press*, 2021.

[12] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet: Leveraging graph neural networks for network modeling and optimization in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.

State Cloning with the GraphLearner

1st Timothy Harrison
Lehrgebiet Kommunikationsnetze
University of Hagen
Hagen, Germany
timothy.harrison@fernuni-hagen.de

2nd Herwig Unger
Lehrgebiet Kommunikationsnetze
University of Hagen
Hagen, Germany
0000-0002-8818-3600

***Abstract*—This paper introduces the Cloned GraphLearner, a neuromorphic sequence generation model that mitigates state aliasing in high-order Markov chains through a lightweight, iterative state cloning procedure. Starting from the original GraphLearner, which stores variable length histories in a Graph Structured Bloom Filter, the algorithm successively creates layers of cloned states whose identity and inter-clone edges index increasingly long context windows while an oblivion rule bounds growth. When trained with action-observation sequences the resulting Cloned GraphLearner acts as a topographic schema with individual clones firing in context specific patterns that resemble hippocampal place cell activity.**

***Index Terms*—Bloom Filters, Neuromorphic Computing, Place Cells, Markov Models, Transfer Learning**

I. Introduction

The GraphLearner is a neuromorphic machine learning algorithm for sequence generation inspired by Hawkins and Mountcastle's studies of the neocortex [1] [12]. Its behavior is readily human-explainable and it can be trained in an online manner. The GraphLearner works by estimating high order Markov Chains, using Counting Bloom Filters to dynamically weight first order edges of a sequence graph with higher order sequences [5] [6]. However it suffers from issues of state aliasing when observed states are ambiguous. To alleviate this a process of state cloning is introduced, where nodes of the GraphLearner are cloned based on context provided by the state sequences stored on their edges. The resulting Cloned GraphLearner is capable of learning spacial and network topologies, with individual clones mapping to consistent spacial ranges even in novel environments.

This behavior is notably neuromorphic, with clone behavior resembling that of place cells in the brain. To demonstrate this the 1996 rodent place cell experiments of O'Keefe and Burgess [14] are simulated with the Cloned GraphLearner. These simulations demonstrate the topographic learning capabilities of the Cloned GraphLearner.

II. Related Work

A. Bloom Filters

Bloom Filters are designed to store sets which are too large to store in memory, such as usernames on social media websites [3] [4]. They can test for set occupancy with a guarantee of no false negatives and a controllable false positive rate. A Bloom Filter consists of a bit array of m bits, initially all set to 0, and k unique hash functions, which map to locations on the array [3] [4]. This array can be stored on the harddrive and still accessed in O(1) time. When an element is added to the filter it is passed through the k hash functions mapping it to k locations on the array all of which are set to 1. To test if an element exists in a set this process is reversed with an element existing in the set only if all k of its hash mappings return a 1. It is possible for the k hash mappings of an element to be assigned a 1 from other elements. In such a case a false positive occurs where the element is incorrectly considered part of the set. The chance of a false positive occurring depends on m, k, and the number of elements stored into the filter n.

If the Bloom Filter stores integer values instead of bits and iterates hashmappings by 1 every time an element is added then it becomes a Counting Bloom Filter [5]. Such filters provide an estimated count of each item added to the filter with a risk of false counts similar to the false positive risk.

B. The GraphLearner

The GraphLearner can be characterized as a Graph Structured Counting Bloom Filter where the first order edges of a sequence graph structure the filter. In other words each edge of the sequence graph has its own associated filter. This is depicted in Figure 1. These edge filters store training histories which then provide a dynamic edge weights based on the stored count of a given sequence. This means a high order Markov chain probability can be calculated by searching only the first order edges of the last element of an input sequence. In other words the probability of choosing an edge e for an input sequence S is:

$$P(e) = \frac{B_c^e(S)}{\sum_{e' \in E} B_c^{e'}(S)} \tag{1}$$

Where $B_c^e(S)$ is the count returned by the Bloom Counter associated with edge e and E is the set of all edges of the last element of S.

During training sequences of length h, the maximum history length, are stored in appropriate edge filter. For added flexibility padded or shortened sequences are also stored. Thus the GraphLearner can estimate a Markov Chain distribution for any input sequence of h or shorter, with a sequence of length 1 defaulting to a first order probability. When generating new elements for an input sequence the GraphLearner initially searches with the h most recent elements of the input and

iteratively shortens the search until a non-zero $P(e)$ can be calculated, i.e. when at least one edge matches the search.

If stored with traditional transition matrices these High Order Markov Chains would require $O(n^{h+1})$ worst case space complexity, where n is the number of unique states. This becomes prohibitively large for higher values of h. By contrast the GraphLearner can estimate the same Markov Chains with just $O(n)$ references in RAM.

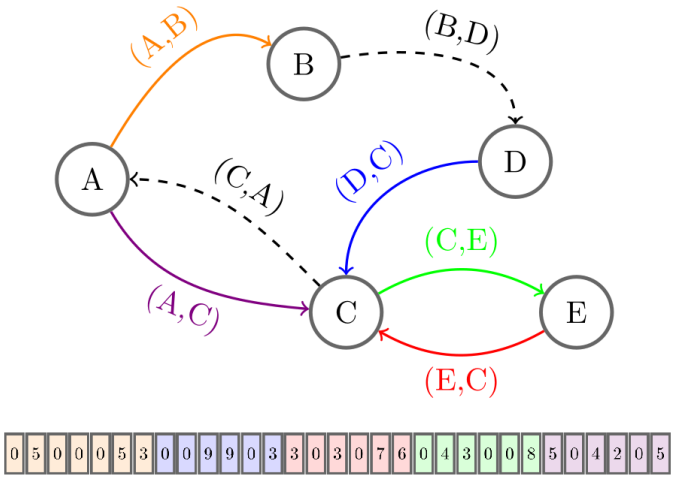


Fig. 1. The original GraphLearner: a Graph Structured Bloom Filter. Edge colors correspond to sections of the structured filter, equivalent to having a filter on each edge. Padded histories of max length h are stored in these filters. The counts returned by the filters for a given h or shorter input then provide dynamic weights which can estimate an h-order Markovian $P(e)$.

C. State Aliasing and Dynamic Markov Modeling

Markov Models like the GraphLearner suffer from problems of state aliasing where a single observable state results from multiple distinct states. For example, in natural language homonyms like ”mouse” or ”bark” have multiple meanings which can only be distinguished with further context.

Dynamic Markov Modeling [11] resolves this issue by creating clones of states from first order information. However the greedy, first order nature of Dynamic Markov Modeling often results in an unsustainable explosion of clones. To resolve this Cloned Hidden Markov Models (CHMM) [18] dynamically merge clones of a given state. In the context of action-state sequence modeling CHMMs are know as Clone Structured Cognitive Graphs (CSCG) [10] due to their similarity to schema networks [17]. The clones of CSCGs are accessed or fired in patterns resembling those of place cells in the brain. However to achieve efficiency CHMMs and CSCGs must fix an upper bound on the number of possible clones of each state.

D. Place Cells and Cognitive Mapping

To perform complex tasks the brain forms cognitive maps [16]. One of the key components of these maps are place cells [13]. Place cells are hippocampal cells which trigger at specific locations in an environment given appropriate observations. As such the firing patterns of place cells encode for spacial topography. Importantly they can fire ahead of time in anticipation of future observations, enabling hippocampal replay. As such, they are important in the planning of actions and complex behaviors.

Place cells specifically occur in the hippocampus but similar structures have been discovered across the brain, including in the neocortex [15].

O’Keefe and Burgess’ expanding box experiment [14] demonstrated that place cell firing patterns corresponded to environmental topographic features, even in novel environments. In this experiment rats were placed in a small square box and given time to learn this environment. The sides of the box were then extended to form three additional environments, a vertical and horizontal rectangle, and a larger square. Place cells which fired at given locations in the small square continued to fire at similar locations in the morphed environments, however in some cases their firing fields were stretched or distorted. However even with these distortions they continue to provide meaningful spacial information, enabling generalization across common environments.

III. The Cloned GraphLearner

The GraphLearner was originally developed for natural language, generating new words and characters from inputs. However it can also be used to control agents when used with action-observation sequences. In this case the GraphLearner is trained on an alternating series of actions and their corresponding observations from a chosen environment. Observations and actions are both treated as states of this sequence. As a result the trained GraphLearner can take an action-observation input ending in an observation and chose an appropriate next action for the agent it controls. Similarly if fed a sequence ending in an action the GraphLearner can predict the next observation, providing a feedback measure of how well it understands its environment.

If trained correctly the GraphLearner forms a simple, transferable schema describing its environment. Unfortunately if that environment has multiple unaliased states the GraphLearner will treat them all identically. In theory a large enough h value can distinguish these states but this method fails when exploring modified environments. Even when those modifications are minor they force the GraphLearner to match shortened sequence histories. In other words the GraphLearner struggles to generalize across common environments because it can no longer rely on precise training histories in these cases. This problem inspired the creation of the Cloned GraphLearner, one layer of which is depicted in Figure 2.

Ideally clones are formed for each distinct state of the training environment, however even when they are not so precise the path taken through successive clones preserves contextual information which might otherwise be lost, such as when the exploration environment is not a precise copy of the training environment.

Creating these clones follows a simple process:

1) Train the GraphLearner as before with a given training sequence T and history length h_0.

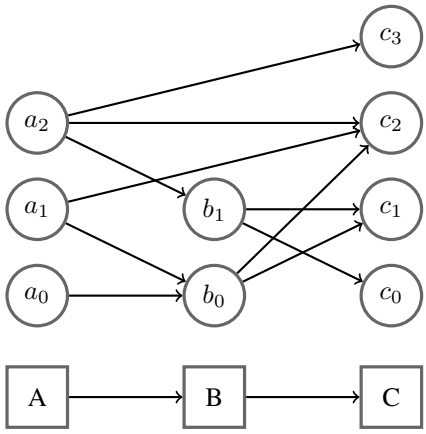


Fig. 2. Part of a Cloned GraphLearner with 1 layer of clones. In (Squares) the original GraphLearner nodes. In (Circles) their clones. Intuitively the paths through these clones can represent more complex historical context than the first order edges of the original nodes. Note: edge (A, C) of the original GraphLearner is not depicted but is implied by the edges connecting clones of A to clones of C. The number of clones implies the existence of other edges.

2) Iterate through T a second time, creating clones with identity edges linking them to original node storing h_0 length sequences.
3) Optionally remove clones which fall under some oblivion threshold.
4) Iterate through T once more connecting clones to clones, creating clone edges which store longer padded histories of max length h_1.
5) Repeat steps 2 through 4 with increasing history lengths, h_n and h_{n+1}, creating new layers of clones from previous layers.

Alternatively it is possible for steps 2 and 4 to be merged, with clones identified and connected to their successors during the same pass of T, followed by oblivion thresholding. However this complicates the oblivion process as all clone to clone edges connecting a deleted clone must also be removed.

Once clones are created and connected the new Cloned GraphLearner generates new sequence elements by following clone to clone edges in the highest possible layer. The cloning and generation processes are described in detail in the following sections.

A. The First Layer of Clones

The following section details the first round of cloning and connecting where clones are created from the nodes of the original GraphLearner and then connected. These layer 1 clones will be indexed with h_0-length sequences, stored on identity edges, and the edges between these clones will store h_1 max length padded histories. Since this process is repeated at each round of cloning Figure 4 refers to h_n and h_{n-1} the history and indexing lengths of the n-th round of cloning.

1) Creating Clones: During cloning each h_0-length subsequence S from the training sequence T is matched to an appropriate clone. For a given S ending in state A the subset of A's edges which match S defines the appropriate clone of A: a_S. This is depicted in Figure 3. An identity edge, with associated Bloom Filter, links A to a_S. Sequence S is stored on this identity edge (A, a_S), along with all other sequences which match a_S's edge subset. Subsequences which match on all edges of A will also have a clone.

In theory T can be a different sequence from the original training sequence so long as it comes from the same underlying distribution, e.g. a separate random walk through the same environment.

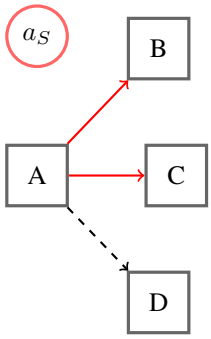


Fig. 3. Clone a_S (Red) is created when a given input sequence only occurs on a subset of A's edges (Red), in this case (A, B) and (A, C). A will have an identity edge from it to a_S which stores all h_0-length sequences matching this edge subset.

2) Connecting Clones: Once clones have been created they must be connected. This process requires two history lengths, h_0 the length of sequences stored on identity edges, and h_1 the maximum length of sequences to be stored on clone to clone edges. The cloning sequence T is traversed again, connected clones are identified from successive h_0-length sequences and h_1-length histories are trained onto their edges. As with the original GraphLearner these h_1 histories are padded so appropriate sequences of length $h_1 - 1$, $h_1 - 2$, ... ,1 are also stored. Figure 4 depicts the clone connecting process for a section of T.

B. Clones of Clones

Once one layer of clones has been created and connected a new layer can be created. This process can be repeated as desired, following the same cloning rules outlined the previous sections, now using subsets of clone to clone edges to define new clones of clones. The edges between these clones of clones then store new longer sequence histories with each successive layer increasing the stored history length. Importantly identity edges will always link from the original, observed states of the GraphLearner to their respective clone.

If clone layer n was accessed with identity edges storing sequences of length h_{n-1} and contained clone to clone edges storing max histories of length h_n then layer $n + 1$ will be indexed with identity edges containing sequences of length h_n and the edges in layer $n+1$ will store max histories of length h_{n+1}, and so on.

The only major difference from the first round of clone creation is that clones of layer $n - 1$ must first be identified

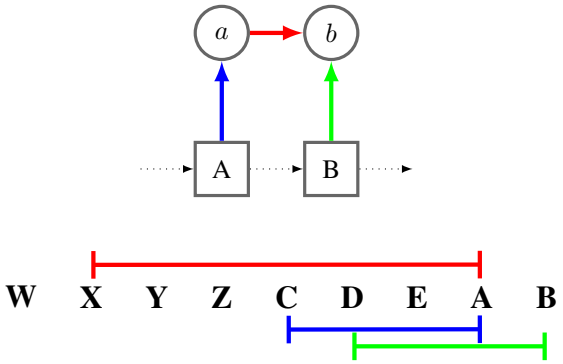


Fig. 4. The connecting process for two layer n clones of A and B: a and b, depicting edges and their corresponding state subsequences for a section of the training sequence. In (Blue) the h_{n-1}-length sequence which matches on the identity edge (A, a) also in (Blue). In (Green) the h_{n-1}-length sequence which matches on the identity edge (B, b) again in (Green). In (Red) the h_n-length sequence which is trained onto edge (a, b) in (Red). Here the clones a and b are identified from their respective subsequences $CDEA$ and $DEAB$ and the longer history $XYZCDEA$ is stored on the edge between them, along with padded or shortened histories $YZCDEA$, $ZCDEA$, etc.

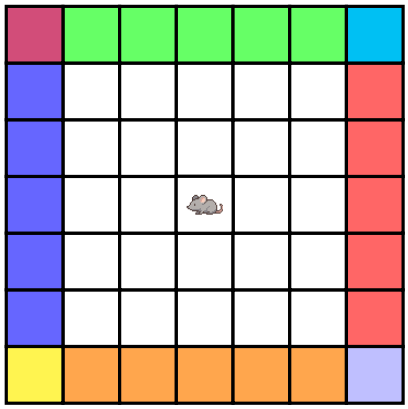


Fig. 5. The 9 observable states of the small square training environment obscure 49 aliased states. This coloration process expedites the formation of meaningful clones by increasing the number of initial edges. The other environments are colored in a similar manner. The cartoon mouse denotes the starting point of the agent. Note that the clones in Figures 6, 7, and 8 are all clones of the white internal state.

by searching the appropriate identity edges before they can be used to create the clones of layer n.

To avoid a potential explosion in the number of clones an appropriate oblivion threshold must be used when creating each new layer.

Once completed the Cloned GraphLearner will have n layers of clones. The clones of each layer are indexed by identity edges containing h_{n-1} length sequences with the edges between clones in that layer storing padded histories of max length h_n. These identity h values are stored to allow faster indexing of clones. In this framework the original GraphLearner effectively forms a zeroth layer, $n = 0$, which is indexed by identity sequences of length 1, i.e. the unique first order states of the training sequence.

C. Oblivion

With each successive round of cloning the number of clones grows. To avoid explosive growth in the number of clones of clones a simple oblivion function is introduced [8]. Once the clone creation process is completed, clones that were indexed less than some threshold value are deleted. This is done before clone connecting occurs to avoid creating edges with deleted clones. The impact of the process can be seen in Figure 9 where a threshold of 100 keeps the number of clones manageable.

D. Generation

At generation time a sequence, S, is input into the Cloned GraphLearner and an appropriate new element is generated for this sequence. If a clone is currently being tracked then the edges of this clone are searched to create a $P(e)$ in the same manner as the original GraphLearner: by recursively searching shortened or padded slices of S until at least one edge match is found. A new clone is selected from this $P(e)$, its associated observation is returned, i.e. the original node-state associated with that clone, and the clone tracker updates to track said clone.

If no clone is currently tracked the last element of S is used to identify a zeroth layer node and the identity edges of this node are searched with h-length slices of the most recent elements of S, starting with the largest possible identity length h_{n-1}. If a matching identity edge is found then its clone is used for generating $P(e)$.

If the input is an action-observation sequence, where new actions are being generated from inputs, then it is possible the most recent observed state, i.e. the last element of S, will not match with the currently tracked clone. In these cases the tracked clone is discarded and a new one must be found.

Given the similarity to place cells, a clone being accessed to generate new sequence elements is referred to as clone firing.

IV. Experiments and Results

A. Experimental Setup

O'Keefe's place cell experiment is replicated with the Cloned GraphLearner. The small square room (SS) is represented by a 7x7 grid, the horizontal rectangle room (HR) with a 14x7 grid, the verticle rectangle room (VR) with 7x14, and the large square room (LS) with a 14x14 grid. All of which are surrounded by impenetrable walls. These rooms are depicted in Figures 6 7 and 8. A simulated rodent agent performs a random walk within the SS environment recording a sequence of actions and observations, T. The GraphLearner is then trained on this sequence and successive rounds of cloning occur, repeatedly using T.

Next the agent performs a new random walk in each of the four environments with the Cloned GraphLearner generating new actions and states from the recorded walk. Each time a clone is accessed during this process the agent's location in the environment is recorded, along with the corresponding clone.

Finally these access locations are used to create a heatmap of where each clone is utilized in the four environments.

The agent's action space is limited to the four cardinal directions. Walls and corners of the environment are treated as distinct states based on the restricted set of actions in each. This results in 9 unique observational states as depicted in Figure 5. The Cloned GraphLearner is trained with initial history length h_0 = 6, and contains 4 layers of clones, not including the zeroth layer nodes of the original GraphLearner. Layer 1 is indexed with a sequences of length h_0 = 6 and stores padded histories of maximum length h_1 = 9. History values continue growing by size 3 until layer 4 which is indexed with h_3 = 15 length sequences and stories padded histories up to length h_4 = 18. An oblivion threshold of 100 is used at each round of cloning. The training sequence, T, is a 150,000 step random walk in the small square environment, see Figure 5.

To measure a process known as place cell remapping the original O'Keefe experiments also considered the directions in which walls were expanded when creating the 3 addition environments. That is not considered in this simulation.

B. Results

The following heatmaps depict the firing locations of selected clones across each of the four environments. In Figure 6 a clone behaves like the place cells from O'Keefe. It has become attached to the left vertical wall and when that wall is extended its firing field also expands.

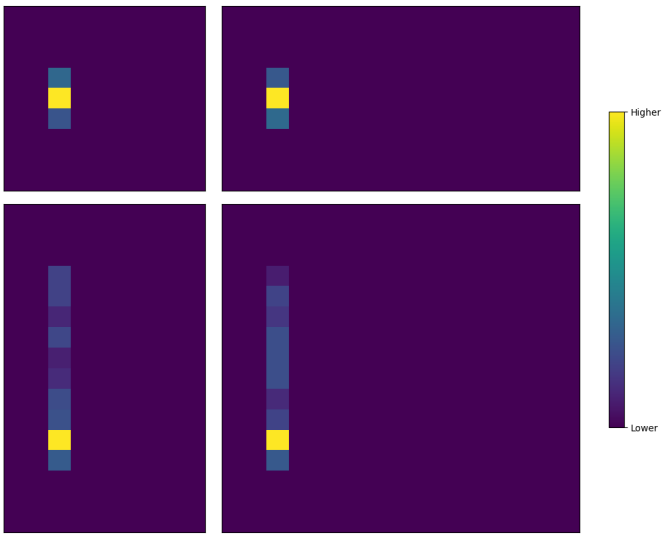


Fig. 6. A stretched clone. Firing like a place cell. In the top left, the original small square room (SS) in which the agent was trained. In the top right: the horizontal rectangle room (HR), bottom left: vertical rectangle room (VR), and bottom right: the large square room (LS). Grid locations are colored according to how often this clone fires in that position. Brighter, yellower coloration representing more frequent firing at a given location. The clone's firing pattern in HR matches its pattern in the original SS environment, yet in the VR and LS environments it becomes extended. This tracks with the observations of O'Keefe. Note: This heatmap includes the barrier wall surrounding each environment.

The ideal case is depicted in Figure 7. Here the clone has fixed onto a single location in the small square environment (SS) and continues to fire at that location even in the other environments.

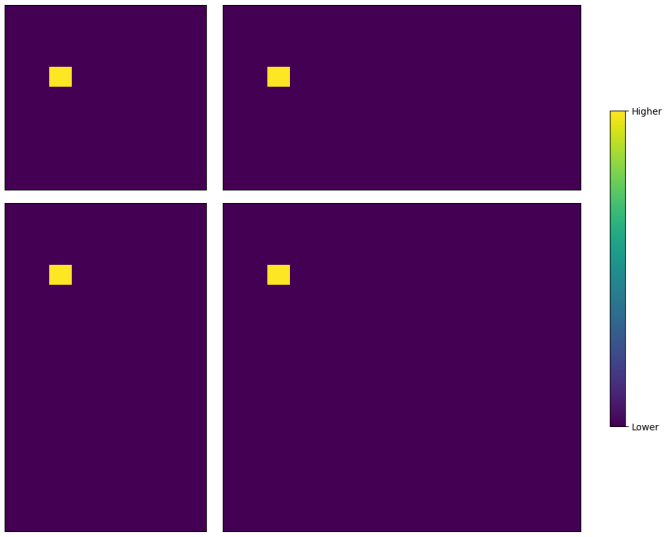


Fig. 7. A clone which has fixed to a specific location in the environment. This clone transfers precise topographic knowledge across environments. Rooms arranged and displayed in the same manner as Figure 6.

A less informative clone is depicted in Figure 8. The clone fires through much of the interior of each environment. However it prefers firing at the bottom of its range and strongly avoids the lower and left most walls.

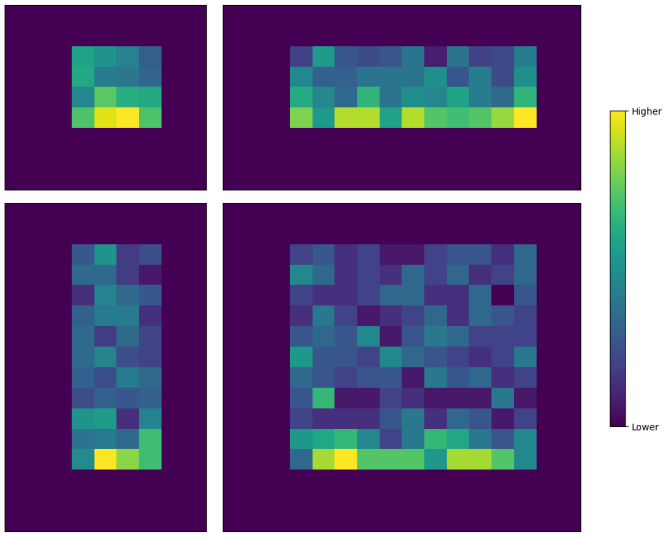


Fig. 8. An emerging clone. Rooms are again arranged in the same manner as Figure 6. A few more rounds of cloning and this clone may encode a unique location. However it already provides partial topographic information by preferring to fire at the bottom center of its range and away from barrier walls. This firing pattern is repeated in the novel environments.

To demonstrate the impact of oblivion thresholding the same 4-layer cloning process from the previous section is repeated without thresholding and compared to the thresholded process in Figure 9.

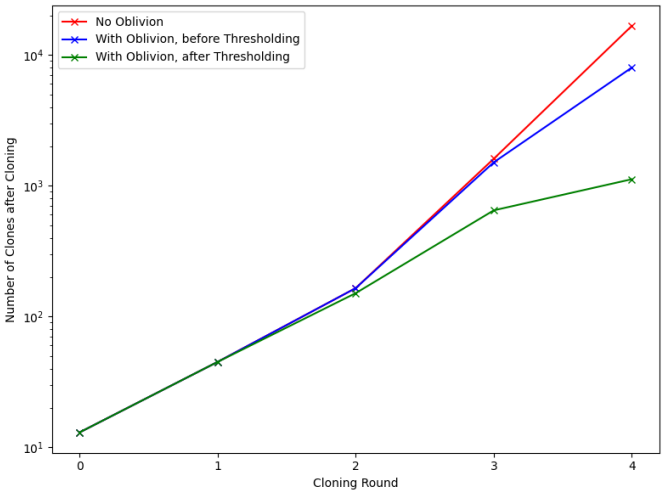


Fig. 9. The impact of oblivion. The average clones at each round of cloning before (Blue) and after (Green) an oblivion threshold is applied. Compared to the number of clones if no thresholding occurs (Red). From the same training and cloning process as described in Section 4A. Note the logarithmic scale.

V. Discussion

State cloning allows the GraphLearner to learn spacial topographies of explored environments. In other words the Cloned GraphLearner builds a schema representation of its learned environment. The topographic knowledge from this schema can be readily transferred to similar environments as demonstrated by replicating the O'Keefe experiments. This place cell like behavior is best seen in Figure 6, which closely follows O'Keefe's results.

The clones of the Cloned GraphLearner all reflect meaningful spacial information. Some like the clone in Figure 7 correspond to an exact location. Any complex behavior linked to this clone can be safely expected to always happen at this location, even in novel environments which resemble the original learned environment. In other words these location specific behaviors can be transferred across common environments.

While the clone in Figure 8 does not provide exact location its bias is clear. In fact this clone represents a snapshot of the knowledge gained from the cloning process. Clones of this clone will have more restricted firing fields, providing increasingly precise spacial information.

Other algorithmic techniques have already achieved similar results, such as Clone Structured Cognitive Graphs. However to be efficiently calculated CSCGs require a predetermined limit on the number of clones of each observed state. By contrast the Cloned GraphLearner maintains efficiency by limiting the depth of its sequence history h. This value can always be increased with further rounds of cloning. The Cloned GraphLearner is highly modifiable. Layers of clones can be deleted and rebuilt as desired, without impacting lower layers.

While the experiments in Section 4 reused the same training sequence at all stages of the cloning process this is not strictly necessary. It is possible for the Cloned GraphLearner to perform cloning in an online manner, with each successive round of cloning relying on a new sequence provided that sequence comes from the same underlying distribution.

As previously noted, the processes of clone creation and clone connection could be merged to occur during the same pass of the training sequence. The oblivion thresholding process would then need to delete not just clones but also all edges containing those clones. This is particularly problematic as the oblivion thresholding used in this paper is only a simple, greedy method of minimizing clone numbers. More advanced methods, such as the clone merging technique employed by CSCGs, must be tested. Keeping clone creation and connection separate leaves open more developmental paths.

VI. Conclusion

The Cloned GraphLearner builds a topographic schema which can be transferred across environments. It is neuromorphic with the firing patterns of clones resembling those of place cells in the hippocampus. This is demonstrated by replicating the rodent place cell experiments of O'Keefe and Burgess. Importantly it does all this while retaining the online-learning, memory efficient, and explainable characteristics of the original GraphLearner. It has untapped potential in real world applications, where topological structures may be too complex to sample and learn in offline manners or store in memory. Hopefully the Cloned GraphLearner can aid in advances both in Neuroscience and in Artificial Intelligence.

References

[1] J. Hawkins and S. Blakeslee, *On Intelligence*. New York, NY, USA: Macmillan, 2004.

[2] V. B. Mountcastle, "The columnar organization of the neocortex," *Brain*, vol. 120, no. 4, pp. 701–722, Apr. 1997, doi: 10.1093/brain/120.4.701.

[3] B. H. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[4] J. Blustein and A. El-Maazawi, "Bloom filters: a tutorial, analysis, and survey," Dalhousie Univ., Halifax, NS, Canada, Dec. 2002, pp. 1–31.

[5] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000, doi: 10.1109/90.851975.

[6] T. Harrison and H. Unger, "The GraphLearner as a high order Markov chain simulator," in *Proc. 16th Int. Conf. Autonomous Syst.*, 2024.

[7] A. A. Markov, *Extension of the Law of Large Numbers to Quantities Depending on One Another*. St. Petersburg, Russia: Imperial Acad. Sci., 1906 (in Russian).

[8] R. D. Fields, "Making memories stick," *Sci. Am.*, vol. 292, no. 2, pp. 74–81, 2005.

[9] T. Harrison and H. Unger, "GraphLearner: An approach to sequence recognition and generation," in *Proc. Mallorca Workshop Autonomous Syst.*, 2023.

[10] D. George *et al.*, "Clone-structured graph representations enable flexible learning and vicarious evaluation of cognitive maps," *Nat. Commun.*, vol. 12, Art. no. 2392, 2021, doi: 10.1038/s41467-021-22559-5.

[11] G. V. Cormack and R. N. S. Horspool, "Data compression using dynamic Markov modelling," *Comput. J.*, vol. 30, no. 6, pp. 541–550, Dec. 1987, doi: 10.1093/comjnl/30.6.541.

[12] J. Hawkins, D. George, and J. Niemasik, "Sequence memory for prediction, inference and behaviour," *Philos. Trans. R. Soc. B*, vol. 364, no. 1521, pp. 1203–1209, May 2009, doi: 10.1098/rstb.2008.0322.

[13] R. U. Muller and J. L. Kubie, "The effects of changes in the environment on the spatial firing of hippocampal complex-spike cells," *J. Neurosci.*, vol. 7, no. 7, pp. 1951–1968, Jul. 1987, doi: 10.1523/JNEUROSCI.07-07-01951.1987.

[14] J. O'Keefe and N. Burgess, "Geometric determinants of the place fields of hippocampal neurons," *Nature*, vol. 381, no. 6581, pp. 425–428, May 1996, doi: 10.1038/381425a0.
[15] A. Bubic, D. Y. von Cramon, and R. I. Schubotz, "Prediction, cognition and the brain," *Front. Hum. Neurosci.*, vol. 4, 2010, doi: 10.3389/fnhum.2010.00025.
[16] J. O'Keefe and L. Nadel, *The Hippocampus as a Cognitive Map*. Oxford, U.K.: Clarendon Press, 1978.
[17] K. Kansky *et al.*, "Schema networks: zero-shot transfer with a generative causal model of intuitive physics," in *Int. Conf. Mach. Learn.* (ICML), 2017, pp. 1809–1818.
[18] A. Dedieu *et al.*, "Learning higher-order sequential structure with cloned HMMs," *arXiv* preprint arXiv:1905.00507, 2019. [Online]. Available: https://arxiv.org/abs/1905.00507

Emotion-Controlled Communication in Agent Networks

1st Yanakorn Ruamsuk
Faculty of Mathematics and Computer Science
FernUniversität in Hagen
Hagen, Germany
yanakorn.ruamsuk@gmail.com

2nd Herwig Unger
Faculty of Mathematics and Computer Science
FernUniversität in Hagen
Hagen, Germany
herwig.unger@gmail.com

***Abstract*—Communication within agent networks involves multiple input and output parameters, making adaptive control and interpretation difficult. Existing systems typically rely on fixed rules or reactive behavior, lacking mechanisms for self-regulation or proactive engagement. This paper proposes an emotion-controlled communication framework, where internal emotional states act as an intermediate layer between input stimuli and output generation. Emotions are modeled using analog circuits to simulate continuous accumulation, decay, and inter-emotional feedback. Simulations with two agents—one active, one passive—demonstrate how internal states evolve in response to input and influence communication behavior over time.**

***Index Terms*—agent communication, emotion modeling, analog circuits, affective systems, internal regulation**

I. Introduction

Modern agent networks—such as those built from cooperating platform interfaces or decentralized service agents—require robust mechanisms to manage communication across multiple participants [1], [2]. These agents often operate through rule-based systems, fixed turn-taking logic, or event-driven APIs, limiting their ability to adapt, self-regulate, or behave meaningfully in dynamic environments. This constraint becomes particularly pronounced when communication needs to reflect varying levels of urgency, engagement, or response depth over time. Despite an increasing reliance on natural language interfaces, most chatbot systems remain reactive—responding only to direct input without internal modulation or context-aware initiative [4].

Recent developments in large language models (LLMs) [3], such as ChatGPT, offer a promising solution for platform integration. By enabling communication through natural language, LLM-based interfaces simplify interaction across heterogeneous systems, bypassing the need for rigid APIs or centralized control hubs. However, while LLMs improve interoperability and usability, they do not inherently address the problem of communication regulation within multi-agent networks. Dialogue remains reactive and externally driven, lacking an internal mechanism for adjusting output based on sustained engagement or shifting priorities. Thus, a higher-level control structure is still needed to modulate agent behavior contextually.

In human networks, internal emotional states often govern how individuals initiate, sustain, or inhibit communication. These states evolve in response to both external input and internal regulation, enabling humans to adapt their behavior fluidly across social contexts [5] [7]. Inspired by this mechanism, the proposed framework introduces emotion as an intermediate control layer in artificial agents. Rather than mapping input metrics—such as message frequency, delay, or length—directly to output, incoming signals are first encoded into abstract emotional states. These states evolve continuously over time through analog circuit [8] [9] [10] dynamics such as accumulation, decay, and mutual influence. Output behavior is then generated in response to these internal states, allowing agents to modulate their actions based on internal readiness or inhibition. This emotion-driven structure simplifies coordination, enhances interpretability, and supports adaptive, context-sensitive communication. The framework is validated through simulations in which two agents—alternating between active and passive roles—demonstrate how emotional dynamics shape message behavior and internal regulation over time.

II. Model Design

This section presents the computational framework of the agent's emotional system, illustrated in Figure 1, which is structured into four interconnected modules: *External Input*, *Sensory Processing*, *Internal Emotional State Modeling*, and *External Output*. These components form a closed-loop architecture that continuously transforms external communication stimuli into internal emotional states, which in turn drive adaptive behavioral expressions. The figure outlines the information flow between modules, emphasizing how emotion acts as an intermediary layer between perception and action.

The framework operates by first interpreting environmental signals X_i, where each X_i represents a normalized and encoded feature derived from raw communication input. These signals are routed through a set of temporal filters $F(X)$ in the sensory processing stage, producing processed stimuli Z. The filtered outputs Z drive changes in internal emotional states S, which evolve continuously over time based on both external stimuli and internal feedback mechanisms $G(S)$. The resulting emotional states $y = f(S)$ ultimately determine how the agent behaves through observable outputs.

At the heart of the framework lies the *Internal Emotional State Modeling* module, where three primary emotional sig-

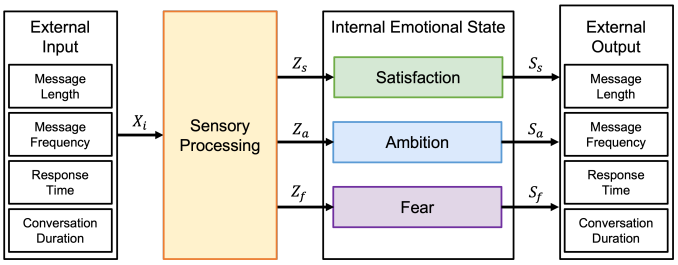


Fig. 1. Overall Framework

nals—**satisfaction**, **ambition**, and **fear**—are maintained as analog voltage states. These emotions interact dynamically to reflect both the agent's current affective experience and its longer-term motivational tendencies.

A. External Input

The External Input module captures raw signals from the agent's communication environment, such as *message length* or *message frequency*. These signals are continuously sampled and normalized into a bounded range to produce stable, comparable values suitable for emotional interpretation. The normalization follows a standard form:

$$\hat{X}(t) = \text{clip}\left(\frac{X(t) - X_{\min}}{X_{\max} - X_{\min} + \varepsilon},\ 0,\ 1\right), \tag{1}$$

where $X(t)$ is the raw input at time t, $X_{\min}$ and $X_{\max}$ define expected feature bounds, and ε prevents division by zero.

Once normalized, each signal is passed through an *Emotion Encoding Function* that transforms it into three emotion-specific analog values: satisfaction, ambition, and fear. These mappings are based on heuristic interpretations of how different input magnitudes influence each emotion—for example, high satisfaction may correlate with longer messages, while fear may rise in response to unusually rapid or large input changes. The output is an emotional stimulus vector:

$$X(t) = \{X_{\text{s}}(t),\ X_{\text{a}}(t),\ X_{\text{f}}(t)\},$$

which represents the raw emotional relevance of the external environment and serves as input to the subsequent *Sensory Processing* module.

B. Sensory Processing

The Sensory Processing module is responsible for converting raw external input signals—such as message length or frequency—into structured, emotion-specific stimuli [11]. These raw inputs are first normalized and mapped to corresponding emotional channels, resulting in intermediate signals $X = \{X_{\text{s}}, X_{\text{a}}, X_{\text{f}}\}$, where each X_i represents the perceptual input related to satisfaction, ambition, or fear. To capture both the persistent intensity and transient dynamics of these signals [6], the module applies a dual-pathway structure consisting of tonic and phasic receptors. These pathways operate in parallel to extract complementary temporal features, ultimately producing the processed outputs $Z = \{Z_{\text{s}}, Z_{\text{a}}, Z_{\text{f}}\}$, which drive changes in the agent's emotional state. The circuit implementation of this dual-pathway architecture is illustrated in Figure 2.

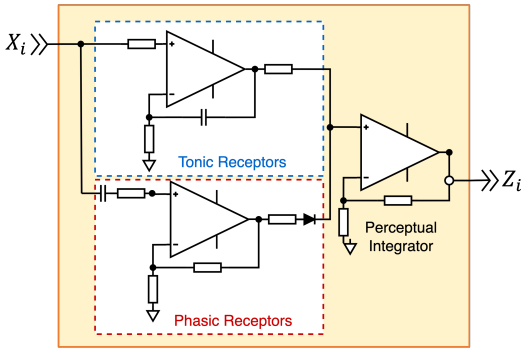


Fig. 2. Sensory Processing Circuit: each input X_i is filtered to produce a time-sensitive output Z_i.

- **Tonic Receptors** simulate the system's sustained attention to environmental input. In the circuit model, they are implemented as *low-pass filters* using summing integrator op-amp circuits. These circuits allow continuous accumulation of input over time, producing outputs that rise gradually with sustained input and decay slowly when input ceases. Mathematically, this behavior can be expressed as a first-order low-pass filter:

$$U_i^{\text{slow}}(t) = \frac{1}{\tau_i}\int_0^t \left(X_i(\tau) - U_i^{\text{slow}}(\tau)\right) d\tau, \tag{2}$$

  where τ_i is the tonic time constant controlling the rate of integration and decay.
- **Phasic Receptors** are responsible for detecting rapid transitions in the input signal. They are realized in the circuit as *high-pass filters*, specifically using leaky differentiator op-amp configurations. These circuits produce sharp, transient outputs in response to sudden increases or decreases in input and decay rapidly to baseline. Their dynamic behavior is captured by the high-pass filter equation:

$$U_i^{\text{fast}}(t) = \tau_d \frac{dX_i(t)}{dt} - U_i^{\text{fast}}(t), \tag{3}$$

  where τ_d is the phasic time constant that controls the responsiveness to fast changes.
  To ensure biologically plausible, excitatory-only responses, the phasic signal is passed through a rectifier circuit that clips negative values. This operation can be modeled mathematically as:

$$\hat{U}_i^{\text{fast}}(t) = \max\left(0, U_i^{\text{fast}}(t)\right). \tag{4}$$

  The rectifier mimics neural mechanisms that primarily transmit excitatory transients while suppressing inhibitory or negative responses.

Finally, the tonic and rectified phasic signals are combined using a weighted summation, forming the final processed signal:

$$Z_i(t) = w_{\text{slow}} \cdot U_i^{\text{slow}}(t) + w_{\text{fast}} \cdot \hat{U}_i^{\text{fast}}(t), \tag{5}$$

where w_{slow} and w_{fast} are tunable weights that determine the influence of tonic and phasic pathways. This fused signal $Z_i(t)$ encodes both the sustained intensity and temporal dynamics of the stimulus, ensuring temporally aware emotional reactions—capturing both how strong and how suddenly a stimulus occurs.

C. Internal Emotional State Modeling

The Internal Emotional State Modeling module governs the evolution of emotional states S based on the filtered stimuli Z. Each emotional variable—*satisfaction* (S_s), *ambition* (S_a), and *fear* (S_f)—is implemented as an analog signal in an independent circuit, and each evolves continuously according to its input and internal feedback. The structure of these emotional circuits, including inter-emotional connections and analog implementations, is shown in Figure 3.

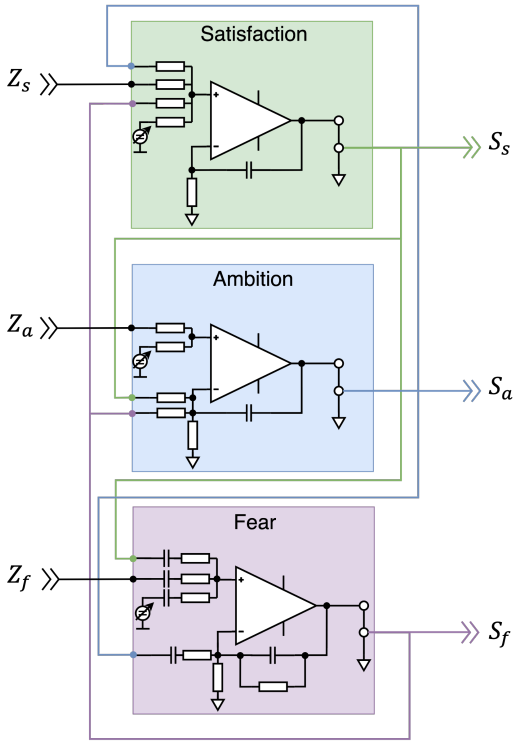


Fig. 3. Emotion Circuit

Satisfaction and **ambition** are realized using summing integrator circuits, which accumulate their respective input signals Z_s and Z_a over time. Their voltage dynamics follow:

$$\frac{dS_i(t)}{dt} = \frac{1}{C_i}\left(-\sum_j \frac{Z_j(t)}{R_{ij}}\right) + G_i(S), \tag{6}$$

where $S_i \in \{S_\text{s}, S_\text{a}\}$, C_i is the integrator capacitor for emotion i, R_{ij} is the resistance from input Z_j, and $G_i(S)$ represents inter-emotional feedback.

Fear, in contrast, is modeled using a leaky differentiator circuit. It reacts sharply to sudden input changes in Z_f, with its output governed by:

$$S_\text{f}(t) = \alpha S_\text{f}(t-1) + \sum_k \frac{C_k}{R_k}\left(Z_k(t) - Z_k(t-1)\right), \tag{7}$$

where $\alpha = e^{-dt/(R_f C_f)}$ is the exponential decay factor, and C_k, R_k are the differentiator's capacitors and resistors for each input Z_k.

The emotional system is further modulated by inter-emotional feedback terms $G(S)$, where outputs from one emotion affect others. For example:

- An increase in fear S_f may suppress ambition: $G_\text{a}(S) = -k_1 S_\text{f}(t)$
- A drop in satisfaction S_s may increase ambition: $G_\text{a}(S) = k_2(1 - S_\text{s}(t))$

The final internal emotional state vector $y = f(S)$ captures the momentary affective configuration of the agent and serves as the basis for behavioral output modulation.

D. External Output

The External Output module translates the internal emotional state vector $y = \{y_\text{s}, y_\text{a}, y_\text{f}\}$ into a single behavioral parameter: *message length*. This parameter represents the richness or expressiveness of the agent's communication and is shaped by the interaction of emotional drives.

Message length is computed as a weighted linear combination of the internal emotional states:

$$L_\text{msg} = w_0 + w_\text{s} \cdot y_\text{s} + w_\text{a} \cdot y_\text{a} + w_\text{f} \cdot y_\text{f}, \tag{8}$$

where w_0 is the baseline message length, and $w_\text{s}, w_\text{a}, w_\text{f}$ are weights associated with satisfaction, ambition, and fear, respectively. These coefficients are selected to reflect behavioral tendencies—for example:

- High **ambition** increases message length.
- High **fear** suppresses message length.
- **Satisfaction** may either reduce or slightly regulate length based on contentment.

To ensure behavioral realism, the final output is clipped to a valid operational range (e.g., $L_\text{msg} \in [0, 100]$):

$$L_\text{msg}^\text{final} = \text{clip}(L_\text{msg},\ 0,\ 100). \tag{9}$$

This output signal defines how much information the agent expresses in a single message and serves as the primary behavioral channel for emotional expression in the current model.

III. Experiments and Results

This section presents simulation-based experiments conducted to evaluate the effectiveness and dynamics of the proposed emotion-driven communication framework. The experiments are divided into progressive layers of analysis, ranging from subsystem validation to full-agent interaction in a communication loop.

A. Experiment Setup

The experimental design is structured around three core modules:

1) **Sensory Processing:** Tests how raw environmental stimuli (e.g., message length) are encoded into emotional signals using the dual-pathway sensory architecture (tonic and phasic circuits).
2) **Internal Emotional Circuits:** Simulates how satisfaction, ambition, and fear evolve over time when stimulated individually. It also examines how inter-emotional feedback shapes internal dynamics, such as inhibition and compensation.
3) **Emotion-Driven Communication:** Integrates all modules into a two-agent interaction framework, where one agent is designated as active and the other as passive in alternating time windows. Agents exchange emotional signals through generated message lengths to evaluate behavioral influence and internal adaptation.

In the communication simulation, roles are pre-assigned and manually switched halfway through the experiment. That is, the role switch is not triggered autonomously by agent behavior but defined explicitly in the simulation configuration.

B. Results

1) Sensory Processing Output: Figure 4 shows how a normalized message-length signal is converted into an emotion-specific stimulus through the sensory processing pipeline. Three distinct phases—spike, adaptation, and decay—are clearly visible, confirming that the dual-pathway processing captures both immediate and sustained components of the signal.

Three characteristic phases emerge in the processed signal:

- **Spike:** A sharp and brief increase at the onset of stimulation, primarily generated by the phasic receptor (modeled as a differentiator). This mirrors biological phasic receptors in humans, such as those found in mechanoreception, which are known to respond quickly to changes but not to constant stimuli.
- **Adaptation:** A gradual rise during the sustained stimulus interval, governed by the tonic receptor's low-pass behavior. The system maintains a smoothed signal while suppressing transient fluctuations. This behavior is comparable to slowly adapting sensory receptors, which continue to respond as long as the stimulus is present but with reduced sensitivity over time.
- **Decay:** After the input returns to zero, the signal drops gradually rather than instantly. This is consistent with the discharge behavior of capacitive integrators and reflects how biological systems slowly return to homeostasis following stimulation.

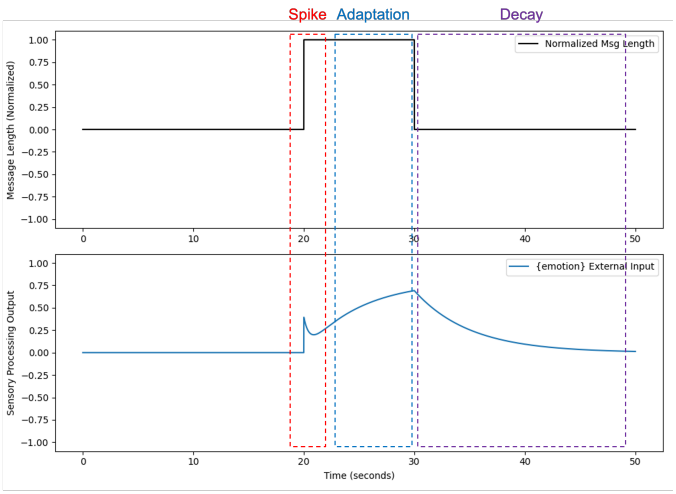


Fig. 4. Sensory processing response to message-length stimulus. Three distinct temporal phases are marked: spike (rapid onset), adaptation (plateau during sustained input), and decay (exponential return to baseline).

From the observation, the combined signal reflects both immediate reactivity and ongoing awareness, capturing the dual nature of human sensory experience: rapid detection followed by gradual internalization and recovery. The shape of the response highlights a dynamic balance between temporal sensitivity (via phasic channels) and cumulative assessment (via tonic channels), aligning with how the human nervous system modulates attention and emotional readiness in response to environmental changes.

2) Internal Emotional Circuit Response: To evaluate the intrinsic behavior of each emotional circuit independently, this section presents controlled stimulation scenarios in which external emotional inputs are directly injected. The aim is to examine how satisfaction, ambition, and fear respond in isolation and interact through feedback mechanisms.

In Figure 5, a sudden drop in external satisfaction input induces a gradual decay in the satisfaction voltage. In response, the ambition circuit shows a compensatory increase due to the absence of positive feedback from satisfaction. The fear signal remains steady, as it is unaffected in this specific case. This behavior reflects the motivational interplay between satisfaction and ambition, where lower fulfillment can provoke increased goal-seeking drive.

Figure 6 depicts the effects of a fear spike. The fear circuit immediately reacts with a sharp transient peak, characteristic of differentiator dynamics. As fear increases, both satisfaction and ambition decrease, demonstrating inhibitory interconnections. This pattern is consistent with emotion theories where fear overrides exploratory or goal-driven behavior, prioritizing inhibition and caution.

Together, these simulations validate the theoretical design of the emotional circuits, confirming that their output aligns with

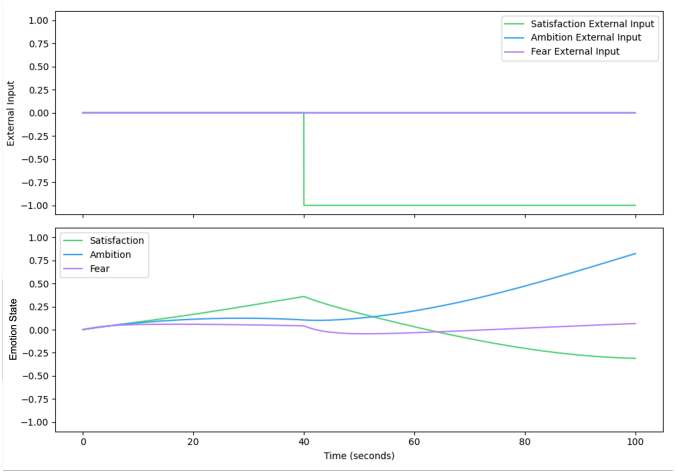


Fig. 5. Response to a decrease in satisfaction input at $t = 40$. Ambition rises gradually as satisfaction decays, and fear remains neutral.

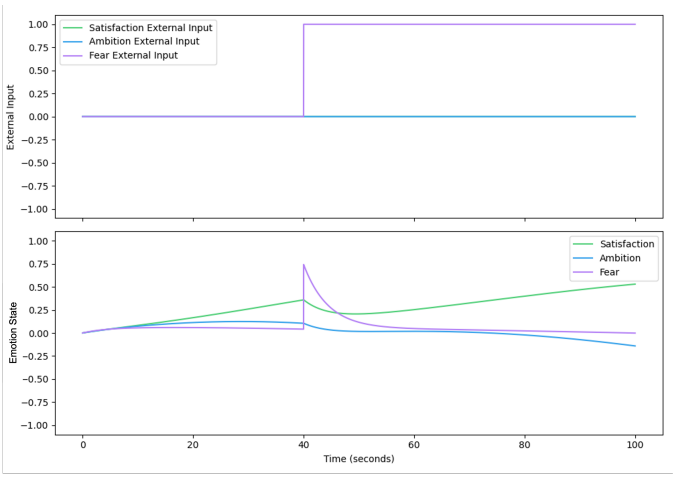


Fig. 6. Response to a spike in fear input at $t = 40$. The fear circuit exhibits a sharp peak followed by decay. Satisfaction and ambition are both suppressed.

both the intended analog behavior and psychological intuition.

3) Emotion-Driven Agent Communication: Figure 7 presents the results from the two-agent simulation. The agents alternate roles in two fixed phases:

1) **Phase 1 (0–50s):** Agent 1 is active and generates message outputs based on its internal emotional state. Agent 2 remains passive and only receives input.
2) **Phase 2 (50–100s):** Agent 2 becomes active while Agent 1 switches to passive mode.

During the first phase (0–50s), Agent 1 is active and generates message-length output modulated by its internal emotional state. At the start, ambition (S_a) is slightly elevated, initiating moderate message production. As the phase progresses, ambition gradually decreases, leading to a corresponding decline in message length. Satisfaction (S_s) remains relatively low due to the lack of incoming input, while fear (S_f) stays minimal. Meanwhile, Agent 2—acting as the passive recipient—receives the message stream as external input.

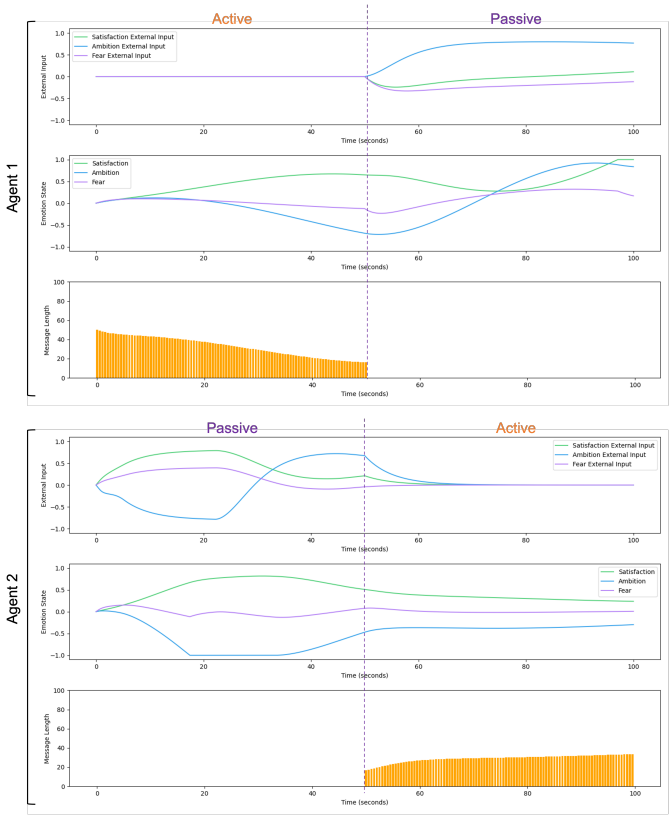


Fig. 7. Bidirectional simulation of emotion-driven agent communication. Top: External inputs, emotional states, and message length output of Agent 1. Bottom: External inputs, emotional states, and message length output of Agent 2.

These signals are processed through tonic and phasic sensory pathways, resulting in a gradual increase in satisfaction and a mild rise in fear in response to sustained stimulation. Ambition in Agent 2 decreases over time, following the increase in satisfaction due to inverse emotional coupling.

At the 50-second mark, roles reverse: Agent 2 becomes active, and Agent 1 becomes passive. A similar emotional pattern unfolds. Agent 2 begins with low ambition and produces moderate-length messages, but as ambition continues to decrease, message length steadily drops. Agent 1, now receiving messages, shows a delayed increase in satisfaction through tonic accumulation, and a moderate rise in fear due to continuous stimulation. Its ambition also decreases as satisfaction builds, replicating the emotional progression seen earlier in Agent 2.

Across both phases, no sharp spikes in fear are observed. This is attributed to the smooth, uninterrupted nature of the input signals—there are no sudden onsets or cessations that would trigger transient phasic fear responses. As a result, fear remains stable and subdued throughout the simulation, while satisfaction and ambition exhibit gradual, inversely coupled trends.

The simulation reveals a consistent emotional-behavioral loop: active agents begin output with low ambition, which

decreases further as messages are emitted. Message length follows this decline closely, reflecting the influence of ambition on output intensity. Passive agents, on the other hand, accumulate satisfaction in response to received messages, which in turn suppresses ambition. Fear rises modestly in both agents but remains stable due to the absence of abrupt emotional stimuli.

This interaction results in:

- Emotion-dependent message generation that self-adjusts based on internal ambition.
- A consistent inverse dynamic between satisfaction and ambition within each agent.
- Emotional adaptation in passive agents driven by external input, even without active participation.
- Smooth transitions supported by the tonic-phasic design, enabling both stability and sensitivity to changes.

Although the switch between active and passive roles is externally configured, the simulation demonstrates how internal emotional states evolve naturally based on message flow. This structure provides a foundation for future implementations where role transitions may be driven autonomously by emotional thresholds or interaction dynamics.

IV. Conclusion

This paper presented an analog emotion-based framework for AI agents, incorporating biologically inspired sensory processing and internal emotional dynamics to modulate communication behavior. Simulations showed that the emotional states—satisfaction, ambition, and fear—respond appropriately to input stimuli, and that these states influence message generation in a continuous and interpretable manner.

In the current setup, however, agents follow predefined active and passive roles to facilitate controlled observation, rather than exhibiting emergent behaviors such as autonomous turn-taking. Moreover, each agent's emotional state is treated as internally driven, responding only to external message stimuli without recognizing or adapting to the emotional state of the other agent. As a result, true emotional interdependence—where one agent's emotional state influences the other's message generation in a feedback loop—is absent.

Future development should address this limitation by integrating mutual emotional awareness. This would allow agents not only to react to message patterns, but to sense and respond to the emotional states of others, enabling richer, more socially intelligent interactions in multi-agent systems.

References

[1] T. Chen, Z. Liu, J. Tang, et al., "AgentVerse: Facilitating Multi-Agent Collaboration through Large Language Models," arXiv preprint arXiv:2210.02199, 2022.

[2] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. Chawla, O. Wiest, and X. Zhang, "Large Language Model Based Multi-agents: A Survey of Progress and Challenges," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, Aug. 2024, pp. 8048–8057. doi: 10.24963/ijcai.2024/890.

[3] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2025.

[4] A. Ho, J. Hancock, and A. Miner, "Psychological, relational, and emotional effects of self-disclosure after conversations with a chatbot," *Journal of Communication*, vol. 68, no. 4, pp. 712–733, 2018.

[5] A. Moors, P. Ellsworth, K. Scherer, and N. Frijda, "Appraisal theories of emotion: State of the art and future development," *Emotion Review*, vol. 5, no. 2, pp. 119–124, 2013.

[6] I. Dozmorov and D. Dresser, "Immune system as a sensory system," *Int. J. Biomed. Sci.*, vol. 6, no. 3, pp. 167–175, 2011.

[7] M. R. Roxo, P. R. Franceschini, C. Zubaran, F. D. Kleber, and J. W. Sander, "The limbic system conception and its historical evolution," *Scientific World Journal*, vol. 11, pp. 2428–2441, Dec. 2011.

[8] J. Sun, P. Gao, P. Liu, and Y. Wang, "Memristor-based emotion regulation circuit and its application in faulty robot monitoring," *IEEE Internet of Things Journal*, vol. 11, no. 19, pp. 31633–31645, Oct. 1, 2024.

[9] J. Sun, Y. Zhai, P. Liu, and Y. Wang, "Memristor-based neural network circuit of associative memory with overshadowing and emotion congruent effect," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 2, pp. 3618–3630, Feb. 2025.

[10] M. Zhang, C. Wang, Y. Sun, and T. Li, "Memristive PAD three-dimensional emotion generation system based on D–S evidence theory," *Nonlinear Dynamics*, vol. 112, pp. 1–21, 2024.

[11] D. Julius and J. Nathans, "Signaling by sensory receptors," *Cold Spring Harbor Perspectives in Biology*, vol. 4, no. 1, p. a005991, Jan. 2012.

Client–Agnostic Continuous Authentication via Keystroke–Induced Traffic Patterns

Alexander Niedermayer, David Monschein, Oliver P. Waldhorst
Institute of Data–Centric Software Systems (IDSS), Karlsruhe University of Applied Sciences
{nial1016, david.monschein, oliver.waldhorst}@h-ka.de

***Abstract*—How can we continuously verify the identity of users without modifying their devices? We introduce a client–agnostic method that leverages keystroke–induced network traffic patterns to passively authenticate users. It can be deployed on infrastructure already common in network environments. By applying contrastive learning to Web–Socket packet traces, we compare new traffic against previously seen patterns from the same user. In an experiment with 75 users, our method achieved 87 % accuracy–improving over a statistical baseline by 27.7 percentage points (pp). These results demonstrate that network traffic captures meaningful behavioral signatures and can serve as a foundation for practical, continuous user authentication.**

***Index Terms*—Continuous Authentication, Network Monitoring, Network Security.**

I. Introduction

Nowadays, many people use web–systems ranging from streaming services, over online games, to highly sensitive online banking applications. Even though authentication is such an important aspect, many applications solely rely on knowledge-based approaches like passwords. These have been found susceptible to different kinds of attack vectors. Some examples are dictionary attacks and heat analysis [14].

The impact of these attacks can be reduced by continuously authenticating the user's identity. An easy approach would be to request the user's credentials in short intervals. While being effective, this strongly decreases the usability of the application. Based on this issue, the field of behavioral authentication tries to offer a solution and provide an additional layer of security without sacrificing usability.

Recent behavioral authentication approaches are based on metrics derived from the user interacting with the device [20]. These have been proven to be an effective way of matching users to their behavior, but rely heavily on client–sided modifications [22, 26]. These modifications are needed to capture keyboard or touch–screen inputs. This can be an obstacle for two main reasons. First, the client needs to be modified to collect the necessary user behavior. Second, the collection and sending of the user behavior might have negative impact on the battery life of the device and use additional mobile data.

We propose a new approach, relying on the continuous authentication of users based on their network traffic generated by using an application. The system takes a time series of the exchanged packets as input for a machine learning model. This eliminates the need for client–sided modifications and thus does not have any impact on battery life or used mobile data. Our approach can be injected inside an existing system with minimal changes, as seen in Fig. 1. The contributions of this paper are, therefore:

- An approach for continuous authentication without application changes on the client side.
- An evaluation of the proposed approach using a dataset, generated from keyboard inputs.
- Empirical evidence that network traffic captures non–trivial human behavior and can be used for continuous authentication.

We evaluated our approach in an experiment with 75 users. The results show an improvement in accuracy of 27.7 pp compared to a statistical baseline approach and 37 pp compared to a random guess. Another finding is the strong association between the user's typing behavior and the generated web–socket packets, which further supports the claim of using network traffic for user authentication.

This paper is structured as follows. In Section II we provide an overview of the foundation and related work in the field of behavioral authentication and network traffic analysis. Section III describes our approach for continuous authentication based on network traffic. The experiment setup and results are presented in Section IV-B. Finally, we discuss possible limitations of our approach in Section V and conclude our work in Section VI.

II. Related Work

In this section, we will give an overview of the two main fields, that are the foundation for our work. The first one is a brief summary of behavioral authentication. The second one covers the basics of network traffic analysis using machine learning approaches.

A. Behavioral Authentication

Behavioral authentication offers a supporting way of authenticating users. The main goal is, to make knowledge-based approaches more secure, by collecting and analyzing user behavior. Currently a lot of focus lies on the analysis of the users physical interaction with the device.

The device interaction can be captured in different ways [14]. One of the most common ways is to capture the users typing behavior. This approach has been shown by multiple publications [1, 4], to be a reliable method of using the user behavior for additional authentication information.

The rise of the Internet of Things (IoT) devices has also brought up new methods of capturing user behavior. For

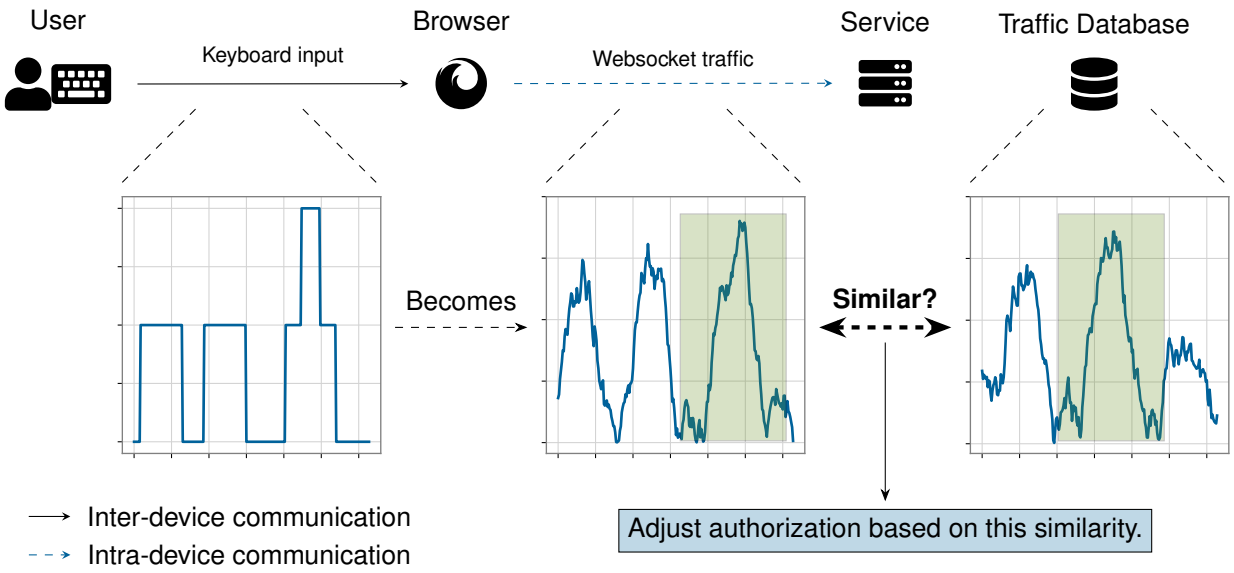


Fig. 1. Overview of our network–traffic–based continuous authentication approach.

example, a smartphone offers many different sensors like touchscreens, accelerometers, gyroscopes and more [3]. Additionally smartwatches even offer the possibility of capturing the user's heartbeat, breathing rhythm or other health related data [14]. These behavior metrics have also been shown, to be usable for user authentication [12].

Another approach would be, to combine user behavior and device attributes, by correlating user behavior to internal device timing characteristics. This has been done in [19]. The key insight is, that when users interact with a device—through typing, mouse movements, or gestures—the resulting input events are not only shaped by the user's behavior but also by the device's internal processing and sampling mechanisms. Specifically, many human-interface devices (like keyboards and touchpads) sample inputs at fixed intervals determined by hardware clocks. These clocks introduce subtle, device-specific timing patterns into the recorded input stream. When analyzing the timing of these events, the fixed sampling rates can create distinct frequency peaks in the input data, which then can be used to identify the device and, by extension, the user. In the mentioned work, the authors demonstrate a rank–1 accuracy of 84.6 % for 10,000 devices when using this approach.

B. Network Traffic Analysis with Machine Learning

Currently, the most common use–case for network traffic analysis is the detection of anomalies and attacks [2]. This is often done by implementing so called Intrusion Detection Systems (IDS). A frequent use–case for machine learning in the field of IDS, is the recognition of Distributed Denial of Service (DDoS) attacks [11, 5]. Overall the scope of IDS extends far beyond DDoS attacks and can also focus on common exploits and malware [10, 29].

For this task different algorithms ranging from simple rule–based approaches to deep learning models have been used [6]. In recent deep learning publications, contrastive learning has shown to be a promising approach for network traffic analysis. Although results exceeded 90 % balanced accuracy, the classification tasks were limited to binary classification (normal/malicious traffic) [13, 16, 28] or application identification (video streaming, file download...) [17].

C. Contrastive Learning

The main idea behind contrastive learning is to learn a vector representation of the input data, that has a short distance for similar inputs and a long distance for dissimilar inputs [9, 27]. A loss function for this optimization problem can be defined as:

$$\mathcal{L} = \sum_{i \in I} \mathcal{L}_i = -\log \left(\frac{e^{z_i \cdot z_{j(i)}}}{\sum_{k \in S(i)} e^{z_i \cdot z_k}} \right) \tag{1}$$

With x_i being an example input and $z_i = f(x_i)$, the function $j(i)$ delivering a index where $x_{j(i)}$ is from the same class as x_i. The function $K(i)$ delivers a set of indices, where $x_{k \in K(i)}$ is from a different class than x_i. Furthermore $S(i) \subseteq K(i)$. There exist different strategies to sample $S(i)$, for example random sampling or hard negative sampling [8].

This formulation makes contrastive learning particularly well–suited for our user recognition task, as it does not require a fixed number of classes. Unlike traditional n–class classification, contrastive learning operates on similarity between samples, allowing the model to scale to an arbitrary number of users.

III. Methodology

The following section describes the methodology of our continuous authentication system. We start by introducing the main idea behind our approach. Afterwards, we describe the implementation of our system.

A. Main Idea

The main idea is, that during event–based communications between client and server, the characteristics of user interaction are causally connected to the network traffic. This offers the possibility for behavior being represented in network metrics, especially packet frequency, packet inter–arrival time and packet size.

For example, consider a web application with a typing input. If every typing input triggers a packet exchange, the packet frequency can be correlated to the typing speed. Also different patterns, when typing key combinations might be represented in the data stream. Stragapede et al. [25] have already successfully identified users based on typing events, so the transfer to causally connected network events is a logical step.

B. User Recognition with Web–Socket Packets

The architecture of our approach is shown in Fig. 2. As input, we receive a list of web–socket packets that were sent between the client and the server. The web–socket packets are timestamped on arrival and contain the payload of the message. We calculate the difference between the receive time of two consecutive packets, giving us the packet inter–arrival time as first time series. The second time series in our approach is the effective change a packet has on the text input. Having access to the payload of the message, we can extract, whether the packet adds a sequence of characters to the text input or deletes a sequence of characters. Both of these time series are normalized, using the minimum and maximum value, derived from the respective training data.

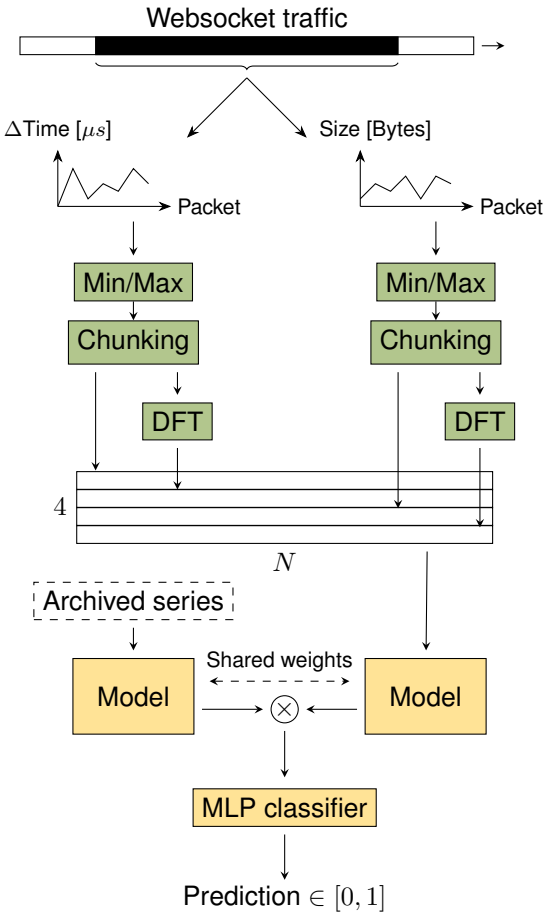


Fig. 2. Architecture of the user recognition on the packet level.

For better processing, the resulting time series are split into chunks of a fixed size $N = 2^k \in \mathbb{N}$. For each chunk we also calculate the discrete fourier transform (DFT) of the packet inter–arrival times and the number of character changes per package. This leaves us with a final 2–dimensional array with the shape $N \times 4$.

For the contrastive learning method, these chunks are then combined into matching and non–matching pairs. Every chunk for a user is randomly matched with another chunk of the same user. The same is done for non–matching pairs. Every chunk of a user is randomly matched with a chunk of another user. A dataset with $S \in \mathbb{N}$ samples will then produce S matching and S non-matching pairs, resulting in a dataset with $2S$ samples.

Our system is based on a siamese neural network architecture [18]. Siamese networks are composed of two identical sub–networks, that share the same parameters and weights. They are used for comparing two inputs. Each input is being forwarded through a sub–network, that extracts important features, represented in the embedding space. Afterwards, a distance metric is being applied to the embeddings, which will output a similarity score.

Since a multivariate time series can be seen as a grayscale image, we take a similar approach to image recognition tasks. We use a deep learning model as backbone, that is trained to generate an embedding of the input data. This is done for each of the two input time series. The embeddings are then multiplied and the result is passed through a multi–layer perceptron (MLP), that outputs the similarity score of the two input time series.

IV. Evaluation

This section starts by presenting the experimental setup, developed for creating a dataset of network traffic, generated by user inputs. We then evaluate our method from the previous section, using this dataset. We also compare our approach to a baseline method, as evidence for substantial user behavior being captured in the network traffic.

A. Generation of User–behavior driven Network Traffic

To generate a dataset of network traffic, we are using the popular web–service *Overleaf*[1]. Overleaf is an online LaTeX editor that allows users to collaborate on documents in real–time. The service uses a web–socket connection, to send updates to the server, whenever a user interacts with the document. This results in network traffic that is heavily influenced by the users typing behavior.

[1]https://github.com/overleaf/overleaf

For capturing the exchanged network traffic, we use the open–source tool *mitmproxy*[2]. Mitmproxy is modified, to store every forwarded web–socket packet in a log file. The log file contains the timestamp of the packet and the respective payload. The payload of the packet is then used to extract the transmitted character changes in the text. Furthermore, we also remove the acknowledgements (ACKs) from the packet stream, since they do not contain any user behavior information.

The dataset is generated, by using recorded keystrokes from the *Keystroke Verification Challenge* dataset [23]. We take every user data that spans over a time period of 15 minutes for desktop and one hour for mobile devices. Afterwards, a script is used, to control the browser and simulate the typing behavior of the given user. The script inputs the keystrokes into the Overleaf editor, which then generates the network traffic. The keystrokes are replayed in real–time, to get an accurate representation of the corresponding network traffic. An overview of the experiment can be seen in Fig. 3.

For our evaluation, we are using a ResNet18 [7] model that is trained on the generated dataset. The dataset consists of traffic generated from 75 different users. The dataset is balanced, so that every user maps to the same amount of traffic. This leaves us with a total time of 18 hours and 45 minutes of recorded network traffic.

The optimization problem is solved using the AdamW [15] optimizer with the standard learning rate of 0.001 and standard weight decay of 0.01. The model is trained for 100 epochs with a batch size of 256. Our loss function for the classification task is the binary cross–entropy loss.

B. Results

We use a 5–fold cross–validation to evaluate the model with specific dataset splits. Metrics are always given as mean over the best score from the 5–folds with a standard deviation. We select the best score as epoch with the highest accuracy. Precision, Recall and F1–Score are also given from this epoch.

As additional metric, we provide a baseline using a decision tree classifier. The classifier is trained using the mean, standard deviation, maximum and minimum value of each feature time series. The evaluation for the decision tree is done using the same 5–fold cross–validation.

[2]https://mitmproxy.org/

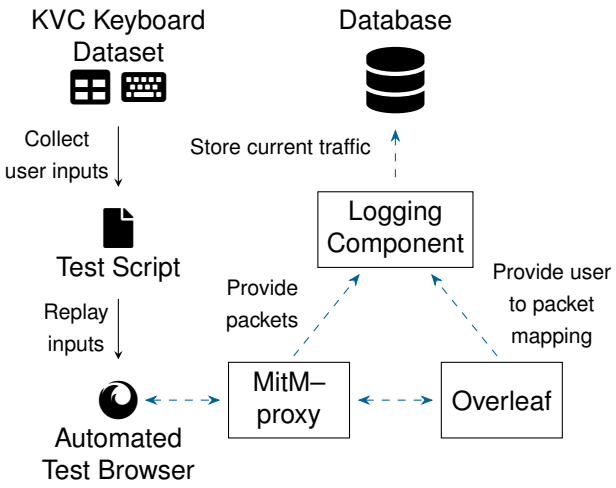


Fig. 3. Overview of our dataset generation setup we used for evaluating our continuous authentication approach.

In Table I we present the results of our evaluation for different chunk sizes N. With a chunk size of 64 records, we achieve an average accuracy of 87 % with a standard deviation of 1 pp. Furthermore, the results show, as expected, that a longer observation time results in a higher accuracy. For both models, we observed, that this difference can lie in the range of up to 11 pp for classification accuracy, when going from a chunk size of 16 to 64. The area under the curve (AUC) of the receiver operating characteristics (ROC) curve gives a value of 93.19 % $\pm$ 1.74%. This high score should be taken with a grain of salt, since as seen in Table I, the measured precision value is at 82.5 %. The ROC curve is displayed in Fig. 4, representing the average performance over a 5–fold cross-validation. The shaded area around the curve indicates the standard deviation, providing a measure of variability across the folds.

Figure 5 evaluates the impact of different preprocessing methods on the accuracy of our approach. The x–axis represents different chunk sizes N and the y–axis shows the classification accuracy in percent. The best results were achieved, when having full access to the packet body. Removing the

TABLE I
METRICS OF THE USER RECOGNITION BASED ON NETWORK TRAFFIC. RELATIVE TO CHUNK SIZE N.

Metric	Model	$N = 16$	$N = 32$	$N = 64$
Accuracy	Our approach	75.9 % $\pm$ 4.3 %	82.8 % $\pm$ 3.0 %	**87.00 %** $\pm$ 1.0 %
	Baseline	61.8 % $\pm$ 7.3 %	64.1 % $\pm$ 6.8 %	59.3 % $\pm$ 3.6 %
Precision	Our approach	71.2 % $\pm$ 2.4 %	83.2 % $\pm$ 5.6 %	**82.5 %** $\pm$ 1.8 %
	Baseline	58.9 % $\pm$ 5.5 %	61.6 % $\pm$ 5.7 %	58.1 % $\pm$ 4.9 %
Recall	Our approach	87.1 % $\pm$ 5.3 %	83.0 % $\pm$ 8.2 %	**94.0 %** $\pm$ 2.0 %
	Baseline	79.7 % $\pm$ 10.9 %	80.8 % $\pm$ 8.1 %	76.0 % $\pm$ 12.0 %
F1–Score	Our approach	78.3 % $\pm$ 1.6 %	82.6 % $\pm$ 3.1 %	**87.85 %** $\pm$ 0.8 %
	Baseline	67.4 % $\pm$ 6.3 %	69.4 % $\pm$ 3.3 %	64.9 % $\pm$ 1.8 %

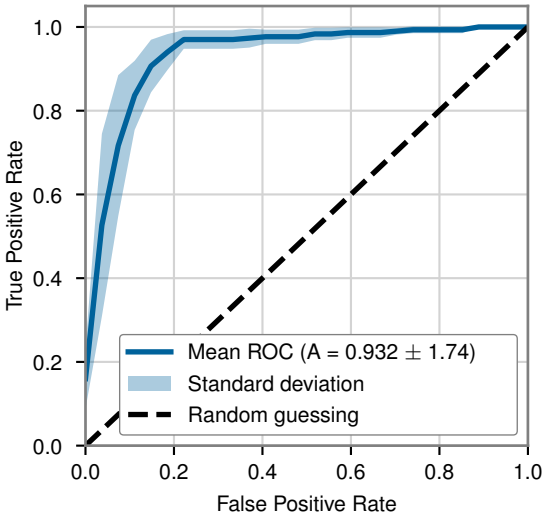


Fig. 4. Average ROC curve derived from the folds of our siamese–network approach with $N = 64$.

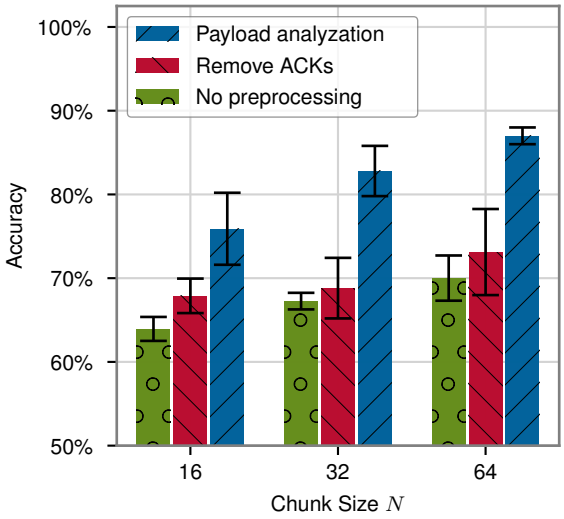


Fig. 5. Accuracy comparison of the siamese network using different preprocessing methods for different chunk sizes N.

ACKs from the packet stream yields a small improvement compared to working on the raw packet stream. The accuracy improvement lies in the range of around 3 pp for $N \in \{16, 64\}$ and 1.5 pp for $N = 32$. The analysis of the character changes transmitted in the packet body and removal of ACKs shows an increased accuracy of 14 pp for $N \in \{32, 64\}$ and 8 pp for $N = 16$, when comparing to just removing the ACKs.

V. Discussion

Compared to the baseline, our approach shows an improvement in the range of 14 pp to 27 pp in accuracy. We hereby argue that due to the improvement of our approach over the baseline, we are able to capture more user behavior inside the network traffic, than just the average typing speed.

We come to this conclusion, since the baseline model is trained on the average, standard deviation, maximum and minimum value of the two time series. The average value of the character changes per packet and the average typing speed of the user are correlated. This is also shown in Fig. 6, where the correlation coefficient $r = 0.78$ indicates a strong association between these two variables. This association is also causally linked, since more keyboard inputs lead to more transmitted characters. The average typing speed of a user should therefore be present in the average transmitted characters. Going further, this metric is also available to the baseline model. Due to the improvement of our approach over the baseline, there must be more user behavior captured in the network traffic, than just the average typing speed.

Nonetheless, when compared to methods that rely on direct keystroke data, there is still room for improvement. Recent publications on the same dataset yield results of around 92 % accuracy [24]. Therefore there might be a loss of information, when transferring the keystrokes to network traffic.

Besides our contributions there are also two main limitations in our approach. First, our current method needs access to the packet body, to extract the required features. This is not always possible, for example when using Transport Layer Security (TLS). A common solution would be to employ a TLS proxy, which decrypts the traffic for inspection before re-encrypting it and forwarding it to the destination. The latter is a common practice in many organizations, with little to no impact on the user experience [21].

The second limitation of our approach is that it currently requires a time window of approximately 15 minutes for effective analysis. This constraint arises from the lower frequency of network traffic events compared to direct keyboard inputs, which limits the granularity and immediacy of the captured behavioral signals. Reducing the required observation time

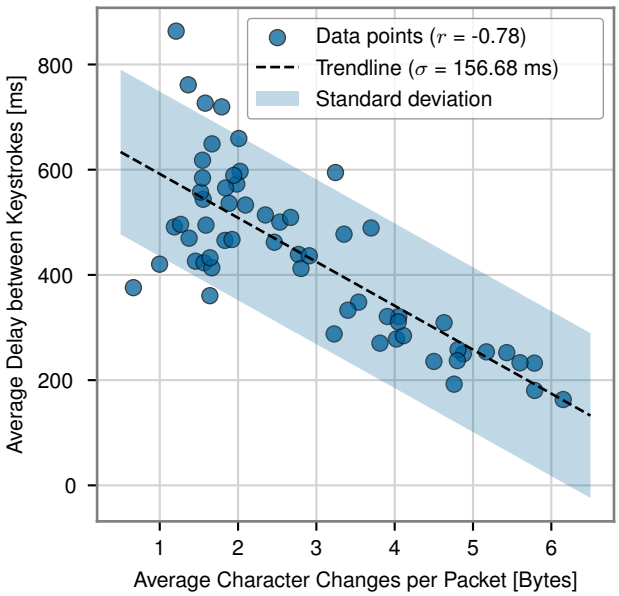


Fig. 6. Correlation between the average delay between keystrokes from the KVC dataset [23] and the average character changes per packet from our dataset.

could significantly enhance the practicality of the method, especially for real-time applications. This may be achievable with access to a larger and more diverse dataset containing longer and more varied user sessions, which would enable better generalization and potentially allow the model to detect patterns in shorter time spans.

VI. Conclusion

Our work showed, that user recognition based on network traffic can be possible. While achieving good results of 87 % accuracy, methods using the direct keystroke data still yield better results. Additionally our work offers evidence, that more user behavior is captured in the network traffic, than just the average transmitted characters. Therefore user–centric network traffic analysis is a promising approach for continuous user authentication, without the need for client–sided modifications.

In respect of future work, improvements regarding the machine learning setup are possible. Our proof of concept used a siamese–network with a convolutional network. While achieving good results on comparison to the needed computing power, we are still curious about the potential of more complex approaches. This might improve the accuracy and may also reduce the observation window.

At last there might also be the possibility to achieve similar results with a more secure and privacy–preserving approach. Maybe there is enough information hidden in TLS encrypted network traffic, that can be used for user recognition. Another common sight in large–scale network is the use of so called aggregated flows for network traffic analysis. These flows are a summary of the network traffic and can be used for anomaly detection. Recognizing users based on these flows might be another privacy–centric approach for continuous user authentication.

Code and Data Availability

The repository https://doi.org/10.5281/zenodo.15579140 contains the code and data used for this work.

Acknowledgments

This work was supported by the project bwNET2.0 funded by the Ministry of Science, Research and the Arts Baden–Württemberg (MWK).

References

[1] Alejandro Acien et al. “TypeNet: Deep learning keystroke biometrics”. In: *IEEE Transactions on Biometrics, Behavior, and Identity Science* 4.1 (2021), pp. 57–70.

[2] Ijaz Ahmad et al. “Machine learning meets communication networks: Current trends and future challenges”. In: *IEEE access* 8 (2020), pp. 223418–223460.

[3] Abdulaziz Alzubaidi and Jugal Kalita. “Authentication of smartphone users using behavioral biometrics”. In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 1998–2026.

[4] Salil P Banerjee and Damon L Woodard. “Biometric authentication and identification using keystroke dynamics: A survey”. In: *Journal of Pattern recognition research* 7.1 (2012), pp. 116–139.

[5] Narmeen Zakaria Bawany, Jawwad A Shamsi, and Khaled Salah. “DDoS attack detection and mitigation using SDN: methods, practices, and solutions”. In: *Arabian Journal for Science and Engineering* 42 (2017), pp. 425–441.

[6] Anna L Buczak and Erhan Guven. “A survey of data mining and machine learning methods for cyber security intrusion detection”. In: *IEEE Communications surveys & tutorials* 18.2 (2015), pp. 1153–1176.

[7] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[8] Yannis Kalantidis et al. “Hard negative mixing for contrastive learning”. In: *Advances in neural information processing systems* 33 (2020), pp. 21798–21809.

[9] Prannay Khosla et al. “Supervised contrastive learning”. In: *Advances in neural information processing systems* 33 (2020), pp. 18661–18673.

[10] Ansam Khraisat et al. “Survey of intrusion detection systems: techniques, datasets and challenges”. In: *Cybersecurity* 2.1 (2019), pp. 1–22.

[11] Samuel Kopmann, Hauke Heseding, and Martina Zitterbart. “HollywooDDoS: Detecting Volumetric Attacks in Moving Images of Network Traffic”. In: *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. IEEE. 2022, pp. 90–97.

[12] Antwane Lewis, Yanyan Li, and Mengjun Xie. “Real time motion-based authentication for smartwatch”. In: *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2016, pp. 380–381.

[13] Longlong Li et al. “End-to-end network intrusion detection based on contrastive learning”. In: *Sensors* 24.7 (2024), p. 2122.

[14] Yunji Liang et al. “Behavioral Biometrics for Continuous Authentication in the Internet-of-Things Era: An Artificial Intelligence Perspective”. In: *IEEE Internet of Things Journal* 7.9 (2020), pp. 9128–9143. DOI: 10.1109/JIOT.2020.3004077.

[15] Ilya Loshchilov, Frank Hutter, et al. “Fixing weight decay regularization in adam”. In: *arXiv preprint arXiv:1711.05101* 5 (2017), p. 5.

[16] Jian Luo et al. “A multi-channel contrastive learning network based intrusion detection method”. In: *Electronics* 12.4 (2023), p. 949.

[17] Yuxiang Ma et al. “A balanced supervised contrastive learning-based method for encrypted network traffic classification”. In: *Computers & Security* 145 (2024), pp. 104023/1–12.

[18] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. “Siamese network features for image matching”. In:

2016 23rd international conference on pattern recognition (ICPR). IEEE. 2016, pp. 378–383.

[19] John V Monaco. "Device fingerprinting with peripheral timestamps". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 1018–1033.

[20] David Monschein and Oliver P. Waldhorst. *mPSAuth: Privacy-Preserving and Scalable Authentication for Mobile Web Applications*. 2022. arXiv: 2210.04777. URL: https://arxiv.org/abs/2210.04777.

[21] Mark O'Neill et al. "TLS inspection: how often and who cares?" In: *IEEE Internet Computing* 21.3 (2017), pp. 22–29.

[22] Praveen Kumar Rayani and Suvamoy Changder. "Continuous user authentication on smartphone via behavioral biometrics: a survey". In: *Multimedia Tools and Applications* 82.2 (2023), pp. 1633–1667.

[23] Giuseppe Stragapede et al. "IEEE BigData 2023 Keystroke Verification Challenge (KVC)". In: *2023 IEEE International Conference on Big Data (BigData)*. IEEE. 2023, pp. 6092–6100.

[24] Giuseppe Stragapede et al. "Keystroke verification challenge (KVC): biometric and fairness benchmark evaluation". In: *IEEE access* 12 (2023), pp. 1102–1116.

[25] Giuseppe Stragapede et al. "KVC-onGoing: Keystroke Verification Challenge". In: *Pattern Recognition* 161 (2025), pp. 111287/1–14.

[26] Cheng Wang et al. "Behavioral authentication for security and safety". In: *Security and Safety* 3 (2024), pp. 2024003/1–36.

[27] Feng Wang and Huaping Liu. "Understanding the behaviour of contrastive loss". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 2495–2504.

[28] Yawei Yue et al. "Contrastive learning enhanced intrusion detection". In: *IEEE Transactions on Network and Service Management* 19.4 (2022), pp. 4232–4247.

[29] Bruno Bogaz Zarpelão et al. "A survey of intrusion detection in Internet of Things". In: *Journal of Network and Computer Applications* 84 (2017), pp. 25–37.