Data Pipeline System Designs for In-network Learning

Patient Ntumba[1], Nour-El-Houda Yellas[2], Salah Bin-Ruba[1], Fehmi Ben Abdesslem[3], Stefano Secci[1]
[1] Cnam, Paris, France. firstname.lastname@cnam.fr.
[2] Orange Innovation, France. nourelhouda.yellas@orange.com. [3] RISE, Stockholm, Sweden. fehmi.ben.abdesslem@ri.se

***Abstract*—This paper introduces the design of a data pipeline system (DPS) integrated with artificial intelligence (AIF) functions to support continuous AI learning and operations for network automation in 5G/6G systems. We design the DPS as a chain of functions, namely ingress and egress Network Data Broker Function (iNDBF and eNDBF) and Network Data Preprocessing Function (NDPPF), to support in-network learning operations. To take into account the distributed nature of the network architecture of 5G systems and beyond, we conceive the DPS to be integrated seamlessly with a distributed learning frameworks such as the federated learning (FL). We performed a realistic evaluation, employing a real dataset from a national mobile operator to simulate the network architecture. Additionally, a FL framework for anomaly detection is integrated with the DPS to assess the effectiveness of our proposal. Evaluation results show that delays in end-to-end data transmission and preprocessing to the AIF locations can cause distributed learning AIFs to work with stale data. The results also highlight how the DPS can counterbalance these delays leading to desynchronisation of the distributed learning process, bringing to AIFs with higher accuracy.**

***Index Terms*—data pipeline, 5G/6G networks, distributed learning, online learning**

I. Introduction

The integration of Artificial Intelligence Functions (AIF) for network automation is one of the key innovations expected for 6G systems [21]. AIFs are meant to perform different analytic tasks, including learning, anomaly detection [18], feature clustering, traffic load prediction, and network resource allocation [3]. In particular, telecom and edge cloud operators are looking for solutions to support the deployment and orchestration of distributed AIFs across the cellular xHaul and core segments [21]. The challenge is to collect and deliver network system-level metrics to AIF locations in real-time with practical delivery rate and redundancy.

In this work, we evaluate different Data Pipeline System (DPS) designs for in-network learning, able to collect data at distributed operator network nodes (e.g. base stations or virtualisation servers), and deliver them to AIFs where in-network learning services such as Anomaly Detection (AD), or NetWork Data Analytic Function (NWDAF) are running. In such settings, a DPS for in-network learning must go beyond the basic requirements of existing telecom network inventory systems [20]: it must be able to perform real-time data processing, aggregation and cleaning operations along the way from data sources to AIFs. Indeed, the actual delivery time of data at AIF locations is critical to achieving high AI accuracy and overall infrastructure operational performance: late data delivery can reduce AIF efficiency, hence their usefulness. AIFs can also become overloaded at higher data delivery rates, leading to longer learning times. For instance, while lower data delivery rates can put AIFs in a starvation state, and prevent them from delivering an accurate AI model: redundancy and higher data rate can slow the convergence of learning algorithms, adding significant overhead and becoming problematic at the scale of distributed network nodes.

A general conceptual DPS model is proposed in [17]; it includes several processing stages, such as storing data in a data lake for later use, which could be useless for an in-network learning that requires real-time data processing. To acquire data in real-time, a conventional approach is to combine real-time data acquisition with a publish/subscribe model and a data stream processing model. Several engines are proposed in the literature for either publish/subscribe systems (e.g. Apache Kafka, Rabbit-MQ) [6], and data stream processing systems (e.g. Apache Flink, Apache Kafka) [5]. Another challenge lies in the distributed nature of the network operator architecture, where the DPS architecture component and AIF placement should both take into account operational constraints in terms of link delays, node processing capacities, as well as the set of available Data Sources (DS) [2], [8]. The main contributions of this paper are summarised as follows:

- We propose a DPS network architecture to enable in-network learning. Two key functions are identified: (i) the Network Data Broker Function (NDBF), which buffers and routes data from sources to AIFs, and (ii) the Network Data Preprocessing Function (NDPPF), which processes data in real-time before AI exploitation.
- Using a real-world dataset from a mobile operator, we emulate propagation delays in a mobile backhauling network. We deploy DPS functions with three designs: (i) *Border-deployment*, functions are placed at the network's borders with other Internet Autonomous Systems; (ii) *CN (Core Network)-deployment*, functions are deployed in the operator's CNs; and (iii) *Edge-deployment*, where functions are closer to Data Sources (DS).
- We evaluate the DPS designs using as AIF system reference a federated learning application for anomaly detection in mobile access network operations. It runs in both synchronous and asynchronous modes to measure how delayed data arrival at AIF locations affects the

978-3-903176-66-9

accuracy of the global trained model. We make the evaluation environment, along with the dataset, available to the community [13].
- We employ two fitness metrics: (i) end-to-end time, to evaluate the total travel time of data from the data sources to the AIF locations; to which we associate a target time to take into account the real-time constraint; (ii) F1-score to assess the accuracy of the global trained model, updated continuously as the learning process evolves.

The rest of the paper is organized as follows: Section II covers related work. Section III outlines the DPS design for in-network learning and deployment designs. Section IV addresses the challenges and solutions for integrating the DPS with AIFs in the operator network. Section V covers the evaluation methodology, and Section VI presents the results. Finally, Section VII provides the conclusion.

II. Related work

A data pipeline system transports data from sources to consumers through transformations for analytics. Its core design follows the Extract, Load, and Transform (ELT) method, where data is extracted, loaded into storage (e.g., data lake), transformed, and then processed in the destination system [19]

Several works design DPS using the ELT approach for operational networks with stricter constraints than basic IT systems. Helu et al. [12] propose a scalable DPS for IIoT, handling large, high-velocity data. Goodhope et al. [11] describe LinkedIns real-time DPS using Apache Kafka for high throughput and low-latency processing, emphasizing fault tolerance and scalability. Poojara et al. [15] propose a serverless DPS for IoT in fog and cloud computing, improving latency.

In [17], a general DPS architecture using the ELT approach is proposed. Data is first generated from various sources (human or machine), collected through connectors. Data is then stored in a data lake, processed into a single format, and moved to a data warehouse. Before training AI models, data is labeled and preprocessed. The AI models act as sinks, with their output data collected continuously by the DPS.

For in-network learning, unlike for the conventional ELT approach, we need to prune out the data lake step in order to meet real-time requirements: data does not need to be stored for the learning purpose (the amount of required storage would be cumbersome also for large operators), but rather consumed on the fly by the AIFs. Indeed, data collection can occur at high frequencies, such as with milliseconds intervals, instead of seconds or minutes for conventional IT systems, leading to large data volumes over time. Furthermore, unlike in [17], data should not be re-used or stored since AIF-based network automation requires real-time data to adjust network configuration based on the current state of the network, to capture zero-day events and vulnerabilities. Storing all data would demand excessive and unnecessary storage capacity.

Federated Learning (FL) is possible AI distributed technique where nodes (AIFs) collaboratively train an AI model while keeping data local. The FL for Anomaly Detection (FLAD) framework in [18] uses multiple AIF clients that communicate with a central AIF server. The server aggregates local models and distributes the global model to clients, giving them a global view of the system. This framework does not include a DPS; instead, the authors create a data collection layer with a single repository, or data lake, to store collected metrics. Each AIF client retrieves a data subset from this lake in rounds, leading to varied views of the system state for model training

In our work, we enhance data delivery by integrating FLAD into an operational DPS. This setup collects and preprocesses data, feeding it to AIF clients in real-time. The FLAD framework uses a Long-Short-Term Memory (LSTM) neural network with an auto-encoder. The auto-encoder processes input data to reconstruct it, aiming to match the original as closely as possible by minimizing the Mean Squared Error (MSE) loss function. To classify an input sequence as normal or an anomaly, a threshold is set based on the MSE from training. During inference, high MSE values above this threshold indicate an anomaly. Additionally, our LSTM model requires feature normalization for training. Unlike [18], we implement an online normalization technique to adapt the FLAD framework for real-time data.

III. In-network Data Pipeline System

A. Design of the Data Pipeline System

To design a DPS for in-network learning, we need to consider the specific requirements of the network operator: (i) the DPS must perform real-time data collection and preprocessing without the need for any data saving, that is, to enable online learning; (ii) the distributed nature of network operator infrastructure where the AIFs are deployed: RAN functions and hardware as well as 5G core function clusters are geographically distributed. In this respect, the DPS should be designed in a decoupled manner giving the opportunity for searching the optimal placement of the DPS across the network. To address these requirements, we leverage on the conceptual DPS model from [17].

Our DPS adopts as well the *publish/subscribe* [7] and the *stream processing* [4] technologies; a publish/subscribe approach enables to decouple the DPS functions in space, time, and synchronisation scales. For the spacial decoupling, the interacting (communicating) functions are not required to know each other: wherever DPS functions are deployed, communication sequence should be satisfied. As for time decoupling, it removes the requirement on the DPS functions to be actively participating in the communication at the same time. The synchronisation decoupling allows non-blocking communication when a component is sending/receiving data from/to the component it is interacting with. Eventually, stream processing enables a specific DPS function to preprocess data in real-time. The DPS is therefore a chain of functions as depicted in Figure 1. These DPS functions are interconnected with each other: the output of a function is fed as the input of the next function.

1) Network Data Broker Function (NDBF): The NDBF queues data for a short period until they are fetched to be ingested for either the network data preprocessing function or

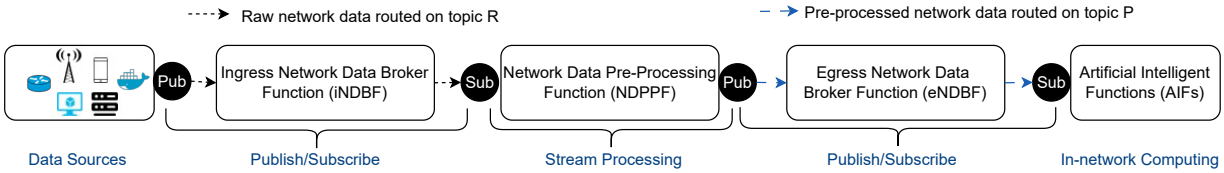


Fig. 1: Conceptual model of data pipeline system for in-network learning.

the AIFs. To do so, we use the topic-based publish/subscribe pattern for routing data in the DPS [7]. With topic-based, the function publishes data to the so-called topics, initially created in the NDBF. The interacting functions will receive all the data published to the topics to which they subscribe. In this respect, we distinguish the ingress NDBF (iNDBF) for queuing the raw network data, i.e. the ingress network data to the NPPF, the egress NDBF (eNDBF) for queuing the preprocessed network data, i.e. the egress network data from the NDPPF.

2) Network Data preprocessing Function (NDPPF): Depending on the schema of the network data generated by the sources, the NDPPF is responsible for aggregating, reducing, parsing, normalising, reshaping, or transforming the ingress raw network data. We leverage the stream processing pattern in order to preprocess the data on the fly (near real-time) as they arrive at the NDPPF. To do so, the preprocessing tasks performed by the function forms a workflow that constantly processes each cycle new data tuple delivery. Because of the infinite nature of real-time data, a *window* [1] mechanism is used for handling data with flexible time bounds, in order to process a finite, yet ever changing sequence of data.

As depicted in Figure 1, we consider that the sources publish the raw network data to iNDBF and the NDPPF publish the preprocessed network data to eNDBF. Both (sources and NDPPF) include the connector Pub which enables to create a connection with both NDBFs and to publish data on a specific topic, i.e., topic R for raw network data, and P for preprocessed network data. Moreover, we consider the NDPPF and the AIFs capable of gathering data published to both the iNDBF and the eNDBF. Both the NDPPF and the AIFs include the connector Sub in order to establish a connection with the NDBFs by subscribing to a specific topic. Here, the NDPPF subscribes to topic T and the AIF subscribes to topic T' for gathering respectively the raw data and the preprocessed data.

B. DPS Integration with AIFs for federated learning

The DPS can be seamlessly integrated with AIFs, whether for inference (e.g. prediction), centralised learning, or distributed learning. Due to the distributed nature of the operator network, we focus on the distributed learning where the raw network data generated by different sources are distributed among the AIF clients that perform training on a subset of the raw network data. Figure 2 depicts the integration of DPS with distributed AIFs for federated learning application. For each AIF client, we consider a chain of functions iNDBF, NDPPF and eNDBF. As concrete example, we use the FLAD framework (see Section II) as AIFs to be deployed together with

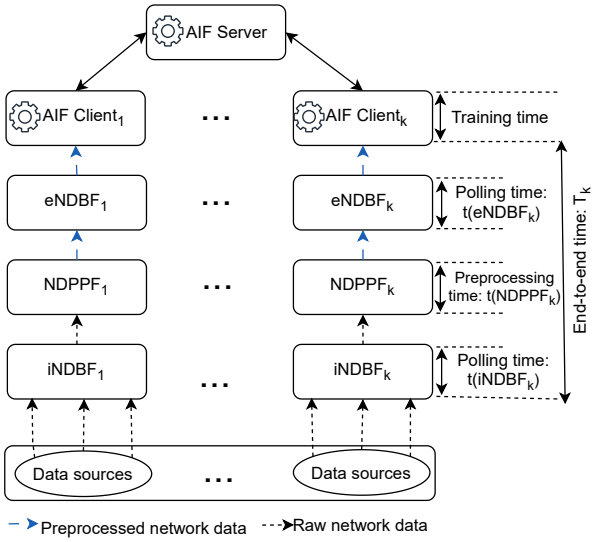


Fig. 2: DPS integrated with AIFs for federated learning.

Algorithm 1: Incremental normalisation.

```
Global: gMax ← 0, gMin ← +∞
while next batch is available do
    max ← Maximum value for the batch;
    min ← Minimum value for the batch ;
    if max > gMax then
        gMax ← max;
    end
    if min < gMin then
        gMin ← min;
    end
    Return MinMax(batch, gMax, gMin);
end
```

the DPS functions. The following challenges were overcome to achieve this integration.

1) Real time network data preprocessing: We develop the NDPPF as an ad-hoc Python application that preprocesses data in real-time. The NDPPF performs two steps to prepare the raw network data into the format expected by the LSTM model used in the FLAD framework:

a) Step 1: Set a window to collect the data to form batch , B, of raw network data. The data of each B are normalised to standardise the scales of different data features. Specifically, we use the min-max normalisation, which linearly transforms

the original data feature values x into normalised values:

$$x' = MinMax(x, gMin, gMax) = \frac{x - gMin}{gMax - gMin} \quad (1)$$

In the context of real-time data, the full set of data features is not available and neither minimum $gMin$ nor maximum $gMax$ can be calculated globally. Therefore, we use the Algorithm 1 proposed in [9], where the global $gMin$ and global $gMax$ for the data features are tracked over time per data batch, and then the standard min-max normalisation is applied to each data feature of the batch with these global $gMin$ and $gMax$ values.

b) Step 2: Before feeding the network data to the LSTM, the normalised network data are reshaped optimally for the model to understand. Generally, the input of a LSTM takes the shape of a 3D array in the form (X,Y,Z) where X is the number of samples, Y is the number of timestep the samples were collected, Z is number of features of each sample.

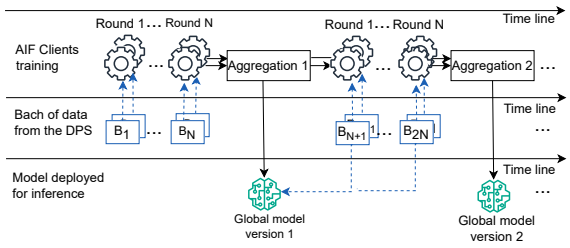


Fig. 3: Federated learning and inference in real-time

2) Learning and Inference processes: Figure 3 shows the learning and inference processes as the DPS continuously delivers to each individual AIF clients different batch of the preprocessed network data ($B_1 \cdots B_N \cdots$).

For the learning, the AIF clients perform the training in parallel on the incoming batch in N rounds. After each N rounds, the AIF server aggregates the results of the AIF client training to generate the global model.

For the aggregation of the trained results, we extend the FLAD framework to use not only the synchronous but also asynchronous methods [16].

a) The synchronous method: requires the AIF server to aggregate the trained results only if all the AIF clients involved in the learning process have transmitted the trained results.

b) The asynchronous method: considers a global waiting time W after which the AIF server triggers the aggregation if at least two AIF clients have transmitted the trained results.

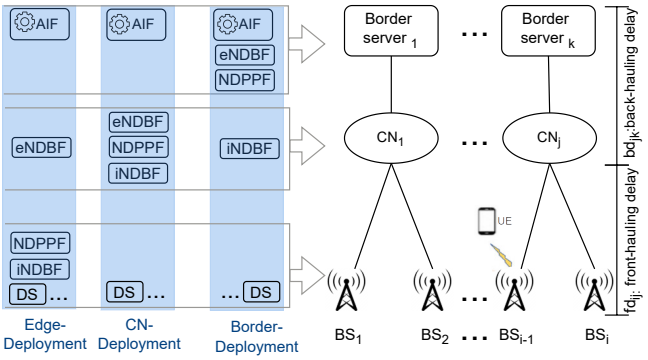
For inference, the global model is deployed as new model version for inference beside each AIF client. The inference is performed on recent network data batch unseen by the current deployed global model version. To classify an incoming batch as normal or anomaly, the global model version uses a MSE threshold computed from the MSE computed from composite features-level MSEs obtained during the learning of the AIF client beside which it is deployed. This enables to consider the difference in the traffic patterns.

IV. Deployment of the Data Pipeline System

A. Operator Network Architecture

Figure 4 (b) shows the adopted the operator network architecture [14]. In this figure, we add the Border server as a node located at the border of a CN used by some service provider, such as Google or Akamai, in order to distribute the application at the border of the CN for load balancing, or serving content faster to UEs [10]. We also highlight the delay on the different links of network topology: front-hauling delay, fd_{ij}: the network delay from a base station (BS) BS_i (i.e., source) to a CN CN_j; and back-hauling delay, bd_{jk}: the network delay from a CN CN_j to its Border server k.

B. Possible Deployment designs of the DPS



(a) Deployment Designs of DPS functions (b) Operator Network Architecture

Fig. 4: Possible DPS functions deployment in operator network.

We propose to deploy the AIFs at the border of the CNs. For clarity, we call AIF (client/node) k ($k = 1, 2, \cdots, K$), the AIF deployed on border server k. Futhermore, let C be a set of DPS function chains c_k, where a DPS function chain c_k includes an instance of $NDPPF$, $iNDBF$ and $eNDBF$. We associate k to indicate that the c_k is deployed to serve the AIF client k i.e. each c_k is deployed to bring the network data from the data source to the AIF client k, and therefore $|C| = K$. We map the data source to the corresponding network equipment being monitored, such as the base stations (BS) [14]. We then consider three possible DPS deployments (see Figure 4).

1) Border-deployment: for each AIF client deployed on a Border k, we deploy one instance of $iNDBF_k$ in the CN, and both $NDPPF_k$ and $eNDBF_k$ at the border server k. As a result, the raw network data are transmitted all the way from the data source to the Border k and get preprocessed there.

2) CN-deployment: for each AIF client k, each of the $iNDBF_k$, $NDPPF_k$ and $eNDBF_k$ are deployed at a CN. In this design the raw network data are preprocessed at the CN before arriving at the Border k.

3) Edge-deployment: for each AIF client k, both $iNDBF_k$ and $NDPPF_k$ are deployed closer to the data source, hence the raw network data are preprocessed closer to the sources. The $eNDBF_k$ is then deployed in a CN.

C. Deployment timeliness

To assess the deployment timeliness, we first introduce the end-to-end time T_k as the time it takes for the network data produced at the data source to reach the AIF client k:

$$T_k = \underset{\forall BS_i}{max}(fd_{ij}) + bd_{jk} + t(iNDBF_k) + t(eNDBF_k) + t(NDPPF_k) \quad (2)$$

where, the max function gives the maximum fraunt-hauling delay fd_{ij} among the delays from all the base stations BS_i connected to CN CN_j, bd_{jk} is the delay (i.e., back-hauling delay) from the CN CN_j to the AIF client k deployed on the Border server k (see Figure 4). $t(iNDBF_k)$ is the polling time of the raw network data at the $iNDBF_k$ and $t(eNDBF_k)$ is the polling time of the preprocessed network data at $eNDBF_k$. Note that by polling time, we refer to the time needed to retrieve data from either $iNDBF_k$ or $eNDBF_k$. Furthermore, $t(NDPPF_k)$ is the time that takes $NDPPF_k$ to preprocess the raw network data.

A lower T_k allows the AIF client k to perform learning or prediction on recent (real-time) network data that matches the current state of the monitored system. In the case of federated learning application, the out-of-sync end-to-end time at AIF clients can lead to a synchronisation problem of the global learning process with the AIF server, resulting in straggler nodes being detected as too out-of-sync and thus excluded from the aggregation round. In this respect, we assume that the end-to-end time T_k of data arrival at an AIF client k should satisfy the constraint in the equation (3). This is to ensure that the AIF clients perform training on the monitored network data that matches the actual state of the monitored system.

$$T_k \leq \text{Target} \quad (3)$$

where $Target$ is the user-defined target time value that can be in the order of second, minutes or hours depending on the network operator requirement.

Should the training time tt_k of an AIF client k exceed the end-to-end time of the network data, this may result in a significant increase in the queuing time of the processed network data at $eNDBF_k$. Consequently the end-to-end time T_k may increase, which could potentially impact the target time constraint. We propose the equation (4) which gives stable deployment of the DPS integrated with AIF nodes.

$$(1+\alpha) \cdot \frac{1}{T_k} \leq \frac{1}{tt_k} \quad (4)$$

The parameter $\alpha \in [0, 1]$ serves to prevent potential equality or tiny difference between the data arrival rate to the AIF client and the training rate. In the long run, this can lead the deployment to an unstable state, mainly due to the stochastic behaviour of the network and computing resources.

In the case the stability constraint (3) is not satisfied, we propose the Algorithm 2 to identify the best batch size $|B|$ for each DPS function chains c_k that brings the network data at the AIF client k. In this algorithm, we propose to iteratively increase the batch size of the raw network data to be processed by $NDPPF_k$, consequently the preprocessing time (i.e. $t(NDPPF_k)$) will increase as well. Increasing $t(NDPPF_k)$ will increase also the end-to-end time T_k, therefore, we consider as best batch size the one that satisfy the stability constraint and the end-to-end time constraint.

Algorithm 2: Deployment stability

```
Input : x: Batch increment value
Input : C: DPS function chains set
for c_k ∈ C do
    stability ← False
    while stability = False do
        ||B|| ← ||B|| + x
        Measure : T_k, tt_k
        if Equations (4) & (3) then
            stability ← True
            Set||B|| the new batch size
        end
    end
end
```

We use $N >> 1$ to reduce the high communication cost between the AIF clients and the AIF server as well as a high waiting time of the network data at the eNDBF, due to the frequent blocking time of the AIF client waiting during the aggregation process of the AIF server.

Given that the AIF client is blocked during aggregation, waiting for the aggregated results to continue the next N training rounds, the processed data from $NDPPF_k$ may also be blocked at $eNDBF_k$. This happens while waiting for the AIF client k to process it. As a result, $t(eNDBF_k)$ may increase exponentially, raising the overall end-to-end time T_k. To solve this, we propose a continuous polling mechanism to retrieve processed data from $eNDBF_k$ and store it in a queue, ready for processing. If the time constraint from equation (3)is not met, the dataset is marked as outdated. Therefore, we introduce the metric δ_k to calculate the rate of outdated data for each DPS function chain c_k. $\delta_k = \frac{O_k}{N}$ where N is the number of processed network data, representing the training rounds before aggregation by the AIF server. O_k refers to the number of processed network data identified as outdated.

V. Evaluation Methodology

This section presents the inference of a realistic operator network, the setting of the DPS integrated with the FLAD framework on the defined topology, and the learning setup.

A. Mobile xHaul Network Topology

To deploy the operator network topology, we use a dataset of traffic traces extracted from a French network operator to emulate a real network infrastructure.

1) Dataset: The dataset was collected using passive measurement probes tapping the interface between PGWs and external public data networks. This approach captures all traffic passing through the operator network across France.

The probes use dedicated classifiers to associate each TCP and UDP traffic flow to the corresponding mobile applications of UE generating the traffic.

For the evaluation testbed, we use Apache Kafka as the NDBF due to its efficiency in balancing throughput regardless of data delivery rates [6]. Apache Kafka uses the TCP protocol to interact with Pub and Sub connectors. Therefore, we focus only on TCP traffic flows for two different days in June 2023: $1,905,990,117$ requests from a weekday and $1,711,327,873$ requests from a weekend. To preserve user privacy, the TCP traffic flows are aggregated into 1036 TAC zones, which are large-scale cellular network areas composed of a certain number of BSs and UEs (typically function of the population in the area). We consider the following fields: (i) $starttime$ the startime of the TCP session; (ii) $stoptime$ the stop time of the TCP session; (iii) the delay from a BS_i to a CN_j that we map as the front-hauling delay fd_{ij}; (iv) the delay from a BS_i to a mobile application server k located at CN border or further on the Internet that we map as the internet edge delay d_{ik}; (v) Service provider; and (vi) tac the id of the TAC area in which the TCP flows are grouped.

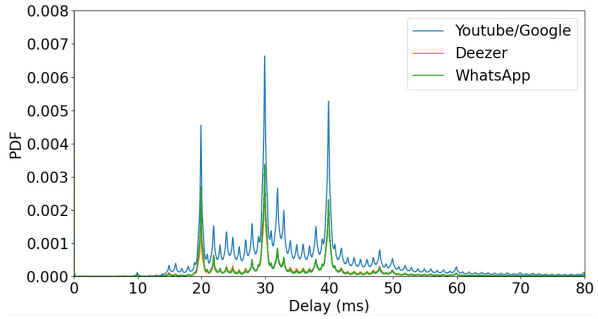


Fig. 5: PDF of the back-hauling delays.

2) Inference process for network topology: To emulate the operator's network topology, we first derive the number of CNs from the measurement. To do this, we filter the measurement based on service providers, then we select the TCP sessions belonging to a service provider known to be highly distributed (Google, Facebook, Akamai, etc.). Based on the measurement belonging to a single service provider, we observe that the delay difference is equal to the back-hauling delay:

$$bd_{jk} = d_{ik} - fd_{ij} \tag{5}$$

We then plot the probability density function (PDF) of the back-hauling delays bd_{jk} for all measurements belonging to the same service provider. From the same CN, the back-hauling delays bd_{jk} follow a normal distribution, from different CNs, we can expect several distributions with different mean values of the back-hauling delays bd_{jk}, so we let $\overline{bd_{jk}}$ take this mean value per distribution. We assume that each distribution corresponds to delays from the same CN, so the number of distributions is equal to the number of CNs.

In order to apply the above methodology for inferring the number of CNs from the traces, we analyse the variance of the delays bd_{jk} for 7 different service providers (i.e. Google/Youtube, Apple, Facebook, Microsoft, Twitter, WhatsApp, Deezer and LeBonCoin). The aim was to select the service provider with the highest dispersion of bd_{jk} delays.

We find that Facebook experiences higher delay variation on weekdays and lower on weekends. For other providers, delay variation is either consistent across days or lower on weekdays and higher on weekends. Specifically, Google/YouTube has the highest delay variation on weekends. This leads us to focus on weekend measurements. We consider three service providers: Google/YouTube, WhatsApp and Deezer for respectively highest, medium and lowest traffics. Figure 5 shows that delays have similar distributions but different probability densities, indicating the presence of multiple CNs. Based on this, we focus on distributions with high probability densities around mean delays of 20 ms, 30 ms, and 40 ms.

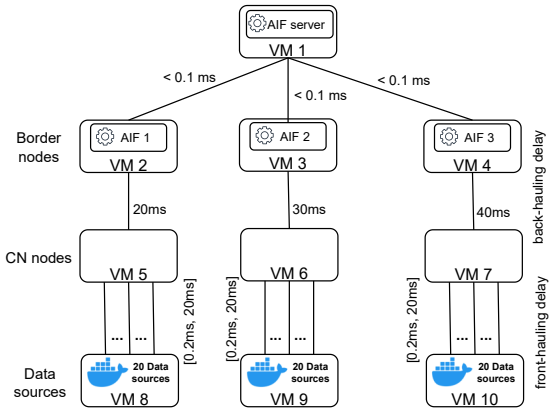


Fig. 6: Emulated Operator Network from an operator data set.

	AIF1	AIF2	AIF3	Server
Data Sources (DS)	20	20	20	-
Total data per DS	5000	5000	5000	-
Abnormal data per DS	0	0	1250	-
DS sampling rate (ms)	100	100	100	-
Batch size for training	200	200	200	-
Batch size for inference	200	200	200	-
Async waiting time (s)	-	-	-	70
Target time (s)	5	5	5	-
Model type	LSTM	LSTM	LSTM	-
Learning rate	0.01	0.01	0.01	-
Aggregation function	-	-	-	FedAvg
Loss function	MSE	MSE	MSE	MSE

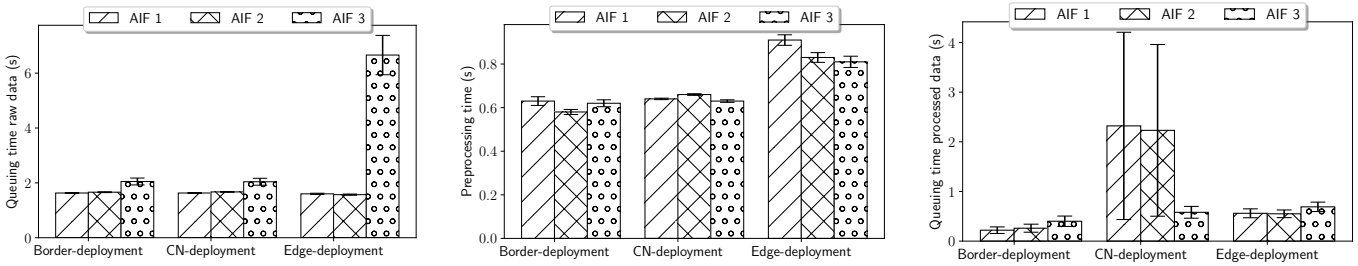
TABLE I: Evaluation setting parameters

We use this information to emulate the operator network topology, as shown in Figure 6. This setup includes three CNs (VMs 5, 6, and 7) and three border nodes (VMs 2, 3, and 4). We deploy TAC areas as Docker containers in VMs 8, 9, and 10. The VMs have 15 GB RAM and 4 CPUs at 2.127 GHz, running Ubuntu 20.04 LTS.

To set the delays in the emulated operator network topology, the back-hauling delays are set to the mean delay values ($\overline{bd_{jk}}$) i.e., 20 ms, 30 ms, and 40 ms respectively, obtained from the inference methodology. We consider 20 data sources (DS) for each CN. Given Formula (5), for each CN we select the front-hauling delays fd_{ij} for which the bd_{jk} values are closer to or

	Synchronous			Asynchronous		
	End-to-end time (s)					
	AIF1	AIF2	AIF3	AIF1	AIF2	AIF3
Border-deployment	2.48 ± 0.32	2.5 ± 0.36	3.07 ± 0.64	2.47 ± 0.3	2.44 ± 0.3	2.75 ± 0.44
CN-deployment	4.59 ± 8.14	4.56 ± 7.46	3.25 ± 0.74	2.55 ± 0.19	2.84 ± 0.46	2.45 ± 0.38
Edge-deployment	3.07 ± 0.44	2.95 ± 0.39	8.16 ± 3.15	2.87 ± 0.21	2.84 ± 0.32	10.82 ± 4.73
	Training time (s)					
Border-deployment	1.63 ± 0.07	1.66 ± 0.07	2.05 ± 0.54	1.62 ± 0.07	1.66 ± 0.06	1.91 ± 0.38
CN-deployment	1.63 ± 0.08	1.67 ± 0.07	2.04 ± 0.55	1.63 ± 0.07	1.65 ± 0.06	1.59 ± 0.35
Edge-deployment	1.6 ± 0.1	1.57 ± 0.1	6.66 ± 3.1	1.58 ± 0.09	1.53 ± 0.09	9.34 ± 4.74

TABLE II: Results of the training and end-to-end times per AIF client, deployment designs and synchronisation methods.



(a) Average queuing time of raw data. (b) Average Preprocessing. (c) Average queuing time of proc. data.

Fig. 7: The values of the components that constitute the end-to-end time in synchronous method.

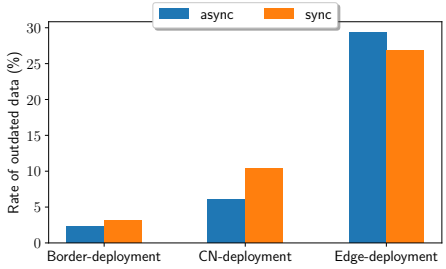


Fig. 8: Average rates of outdated data per deployments

equal to the mean delay $\overline{bd_{jk}}$. As the result, from the dataset, the front-hauling delays fd_{ij} are in the range [0.2 ms, 20 ms] which we set randomly between the DS and the CN nodes.

B. Set up of the DPS integrated with AIFs

We consider three AIF clients deployed on the three VMs representing the border servers (VM 2,3, and 4) of the adopted operator network topology and the AIF server deployed on another border node VM 1. For each AIF client, a DPS function chain is deployed in the emulated topology according the proposed deployment designs.

The DS use the python tool `psutil` to collect and transmit the CPU and memory related metric with 26 features as raw network data. For factual analysis, each DS collects and produces 5000 raw network data samples. We inject the anomaly only in 20 DS of the VM 10, to do so, for the last 1250 raw network data samples, we increase the memory usage by simulating a matrix transformation in each docker container DS. We set the batch size $|B| = 200$, thanks to Algorithm 2 with $\alpha = 0.02$ and $\text{Target} = 5\text{s}$.

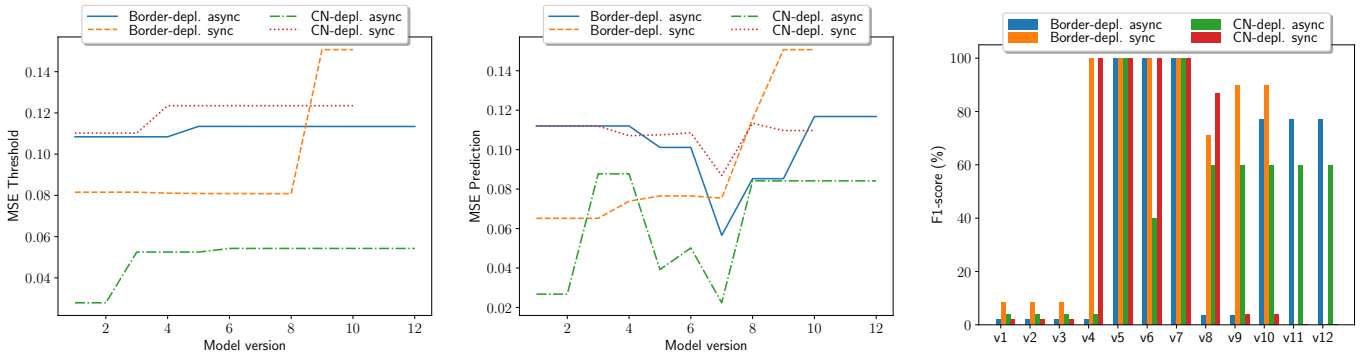
We run the FLAD framework using synchronous and asynchronous methods. For the latter, we set $W = 70s$ inferred as the average of the times elapsed between the aggregations processes of each deployment design executed in synchronous aggregation method. Table I summarises the setting parameters of the DPS integration with the improved FLAD framework

VI. Evaluation Results

In this section, we evaluate the DPS across three deployment designs, tested with two AIF server aggregation methods: synchronous (sync) and asynchronous (async). We analyze the impact of data transmission and preprocessing on end-to-end time to the AIFs. Additionally, we assess the DPS's ability to deliver network data within the target time and examine the quality of real-time anomaly detection from the model.

A. End-to-end time analysis

In synchronous aggregation of the trained results, Table II depicts the average and variance values of the end-to-end time. On the other hand, figure 7 represents the average values of each of the end-to-end time components. We can notice that the Border-deployment has the lowest end-to-end time. By deploying the NDPPF, the eNDBF and the AIF client in the same Border node, the network delay is negligible between these functions, consequently the queuing time of the preprocessed data at eNDBF is reduced as depicted in Figure 7c. The back-hauling delay has in fact an impact on the queuing time of the preprocessed data at eNDBF, this is the first cause of the observed high end-to-end times in CN-deployment and Edge-deployment. The second cause is the preprocessing time, shown in Figure 7b. We see the longest preprocessing times in Edge-deployment. This is because the edge VM experiences the highest memory and CPU consumption, as it hosts the



(a) MSE Threshold. (b) Average MSE prediction (c) F1-score.

Fig. 9: Quality of the generated model version for anomaly detection as the learning process evolves

iNDBF, NDPPF, and 20 docker containers as DS. In Edge-deployment, network delays between the DS, iNDBF, and NDPPF are minimal since they share the same VM. However, the reduced queuing time for raw network data, shown in Figure 7a, does not offset the long preprocessing time.

In asynchronous aggregation, Table II shows that Border deployment has the shortest end-to-end time, compared to CN and Edge deployments. The explanation for the end-to-end time and its components in the synchronous case also applies here, so the component figure is omitted.

Comparing the synchronous and asynchronous methods shows that the latter tends to provide the shortest end-to-end time for Border and CN deployments, but not always, as seen in the variances of the end-to-end time results in Table II. Asynchronous achieves end-to-end lower times due to reduced queuing time at eNDBF, since the AIF server can aggregate once at least two AIF clients send results. In contrast, synchronous aggregation causes longer waits at eNDBF, as the server must wait for all three AIF clients. Edge deployment performs poorly in both methods due to the high load imposed on it.

Table II also shows the average training times. Using Algorithm 2, we ensure a stable DPS deployment where the end-to-end data arrival rate is lower than the AIF training rate. Although network and computing resource fluctuations can increase end-to-end delay, the algorithm minimizes stale data and maintains stable deployment. In this respect, Figure 8 shows that asynchronous deployment designs best meet the end-to-end time constraint. Border-deployment and CN-deployment result in 2.31 % and 5.99 % outdated data, respectively. In the synchronous method, these values rise to 3.2 % and 10.3 %. Border-deployment consistently produces the least outdated data in both methods. In contrast, Edge-deployment, hindered by a heavily loaded VM, has the highest amount of outdated data, failing to meet time constraint.

B. Analysis of the model accuracy

Figures 9 evaluate the global model's quality using MSE and F1-score. To ensure a fair comparison, we only assess anomaly detection for Border and CN deployments. In Edge deployment, network data from the DS shows high values due to the heavy load on the edge VM. In the synchronous method, all three AIF clients join each aggregation, while in the asynchronous method, at most two join. This results in 10 global model versions for the synchronous method and up to 12 for the asynchronous. For each model version, the MSE threshold is set as the 90th percentile of MSE over 50 training rounds by AIF client k, as shown in Figure 9a for $AIF3$.

In the experiment, the DS generate and transmit normal data (low CPU and memory usage) during the first three quarters, and abnormal data (high CPU and memory usage) in the last quarter (see Table I). In the synchronous method, Figure 9b shows that MSE predictions align with this pattern, especially in Border-deployment. Model versions 1 to 7 predict lower MSE values, while versions 8 to 10 show higher MSE. CN-deployment's MSE predictions do not steadily match the ground truth. In the asynchronous method (for CN and Border), MSE predictions do not converge to the ground truth. The synchronous method's accuracy benefits from aggregating results from all three AIF clients, unlike the asynchronous method, which usually aggregates data from mainly two.

Figure 9c shows the F1-score for evaluating anomaly detection performance across different model versions. For Border-deployment: (a) in the synchronous method, model versions 1 to 3 correctly identify normal network data 8 % of the time, with detection quality improving to 100% for model versions 4 to 7. Model versions 8 to 10 achieve 70% and 89.89% accuracy in detecting abnormal network data; (b) in the asynchronous method, model versions 5 to 7 detect normal data 100% of the time, while other versions range from 2.2 % to 77 %. For CN-deployment: (a) in the synchronous method, performance varies widely from 11.9 % and 100 %; and (b) in the asynchronous method, only versions 4 and 6 achieve 100 % accuracy in detecting normal data, with other versions ranging from 3.84 % to 54 %.

Overall, in Border-deployment with the asynchronous setting, the AIF server often aggregates results from only two AIF clients due to imbalanced end-to-end times. This prevents the model version from converging quickly, In synchronous method, all model versions converge more efficiently because

they use aggregated results from all three AIF clients.

C. Discussion and Limitation

1) Lower model performance in asynchronous method: To improve model performance, the aggregation approach should be updated to include training results from stale data. These results should be weighted by a decreasing factor. The weight should decrease as the gap between data staleness and the aggregation time threshold increases.

2) DPS scalability in large operator network setting: The DPS is designed as chains of decoupled functions, each chain assigned to an AIF to gather network data. The framework scales easily for larger networks by adding more AIFs and DPS function chains, with simple management as only forwarding states need updating. The functions can be duplicated without state synchronization, avoiding scalability issues.

3) High resource consumption in Edge-deployment: High resource consumption results from deploying both data sources and DPS functions on the same VM. To reduce the high queuing and processing times, deploying DPS functions on an additional VM within the same physical server as the data source VMs may help. This will be explored in future work.

VII. Conclusion

We proposed a data pipeline system for in-network learning: we defined a generic conceptual model as a chain of ingress Network Data Broker, Network Data preprocessing, and egress Network Data Broker Functions, and evaluated three possible designs differing in how the function embedding is done: Edge, Core Network and Border nodes deployment strategies. As a benchmark AI application, we use a federated learning for anomaly detection framework at the state of the art, comparing synchronous and asynchronous aggregation methods of the trained results. We deploy both the DPS functions and the AIF instances on an operator network where we emulate xHaul link delays leveraging real-world traces from a telecom operator.

Through evaluation, we show that the Border-deployment in both synchronous and asynchronous aggregation methods is the only one able to deliver the network data samples to the AIF clients, within the target time constraint, with small outdated network data. We also show that the AIF clients can perform the training with real-time data, and that we can deploy different global model versions to perform the anomaly detection in real-time. Furthermore, we show in the asynchronous method that imbalanced (asynchronous) end-to-end time of data arrival to the AIF client has a negative impact on performance of the global model versions.

Future work will focus on automated optimal deployment of DPS functions taking into account data delivery targets, possibly coupled with the AIF orchestration, in a large scale experiment setup involving more than a dozen of AIF clients.

Acknowledgement

This work was partly funded by the ANR CoCo5G (contract nb: ANR-22-CE25-0016; https://coco5G.roc.cnam.fr), the H2020 AI@EDGE (https://aiatedge.eu; grant nb. 101015922) and ANR TREES (ANR-24-TSIA-0004) projects.

References

[1] T. Akidau, R. Bradshaw, C. Chambers, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *the VLDB Endowment*, 8(12):1792–1803, 2015.

[2] R. Banno, J. Sun, M. Fujita, et al. Dissemination of edge-heavy data on heterogeneous mqtt brokers. In *the 6th International Conference on Cloud Networking*, pages 1–7. IEEE, 2017.

[3] L. Bonati, S. D'Oro, S. Basagni, et al. Scope: An open and softwarized prototyping platform for nextg systems. In *the 19th International ACM MobiSys*, pages 415–426, 2021.

[4] Paris Carbone, Marios Fragkoulis, Vasiliki Kalavri, et al. Beyond analytics: The evolution of stream processing systems. In *the ACM SIGMOD*, pages 2651–2658, 2020.

[5] M. Čermák, D. Tovarňák, M. Laštovička, et al. A performance benchmark for netflow data analysis on distributed stream processing systems. In *IFIP NOMS*, pages 919–924. IEEE, 2016.

[6] P. Dobbelaere and K. S. Esmaili. Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations. In *the 11th ACM DEBS*, pages 227–238, 2017.

[7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131, 2003.

[8] D. Z. Fawwaz, S. Chung, C. Ahn, et al. Optimal distributed mqtt broker and services placement for sdn-edge based smart city architecture. *Sensors*, 22(9):3431, 2022.

[9] M. N. Fekri, H. Patel, K. Grolinger, et al. Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network. *Applied Energy*, 282:116177, 2021.

[10] P. Gigis, M. Calder, et al. Seven years in the life of hypergiants' off-nets. In *ACM SIGCOMM*, pages 516–533, 2021.

[11] K. Goodhope, J. Koshy, et al. Building linkedin's real-time activity data pipeline. *IEEE Data Eng. Bull.*, 35(2):33–45, 2012.

[12] M. Helu, T. Sprock, D. Hartenstine, et al. Scalable data pipeline architecture to support the industrial internet of things. *CIRP Annals*, 69(1):385–388, 2020.

[13] P. Ntumba. Evaluation testbed of the data pipeline system designs for in-network learning. https://gitlab.roc.cnam.fr/data-pipeline-system-designs-for-in-network-learning [Accessed: Sept. 12, 2024].

[14] P. Parastar, A. Lutu, Ö. Alay, et al. Spotlight on 5g: Performance, device evolution and challenges from a mobile operator perspective. In *the IEEE INFOCOM*, pages 1–10. IEEE, 2023.

[15] S. R. Poojara, C. K. Dehury, P. Jakovits, et al. Serverless data pipeline approaches for iot data in fog and cloud computing. *FGCS*, 130:91–105, 2022.

[16] C. Prigent, A. Costan, G. Antoniu, et al. Enabling federated learning across the computing continuum: Systems, challenges and future directions. *FGCS*, 2024.

[17] A. Raj, J. Bosch, H. H. Olsson, et al. Modelling data pipelines. In *the 46th Euromicro conference on software engineering and advanced applications*, pages 13–20. IEEE, 2020.

[18] S. Ruba, N. E. Yellas, and S. Secci. Anomaly detection for 5g softwarized infrastructures with federated learning. In *the 1st International Conference on 6GNet*, pages 1–4. IEEE, 2022.

[19] F. Waas, R. Wrembel, T. Freudenreich, et al. On-demand elt architecture for right-time bi: extending the vision. *International Journal of Data Warehousing and Mining*, 9(2):21–38, 2013.

[20] Z. Wang, G. Wei, Y. Zhan, and Y. Sun. Big data in telecommunication operators: data, platform and practices. *Journal of Communications and Information Networks*, 2(3):78–91, 2017.

[21] Nour-El-Houda Yellas, Bernardetta Addis, Roberto Riggio, and Stefano Secci. Function placement and acceleration for in-network federated learning services. In *the 18th International Conference on Network and Service Management (CNSM)*, pages 212–218, 2022.