Energy-Efficient Function Chaining and Assignment for In-Network Learning

Garance Gérard*
Patient Ntumba*
Safia Kedad-Sidhoum*
Amélie Lambert*
garance.gerard@lecnam.net
patient.ntumba@cnam.fr
safia.kedad-sidhoum@cnam.fr
amelie.lambert@cnam.fr
Cedric-Cnam
Paris, France

Nancy Perrot†
Orange Research
Paris, France
nancy.perrot@orange.com

Abstract

Next-generation networks enable the deployment of Artificial Intelligence Functions (AIF) directly in the network. As massive volumes of data are generated across geographically distributed network nodes, Federated Learning (FL) emerges as a key paradigm for distributed model training through local updates and aggregation, without requiring centralized data collection. To support timely in-network learning, we leverage the Data Pipeline System (DPS) to collect, preprocess, and route network data toward AIF clients. However, the joint chaining and assignment of DPS functions and AIF clients onto the physical network remains largely unexplored. We study the joint chaining and assignment of DPS functions and AIF clients with the objective of minimizing energy consumption while satisfying data end-to-end latency constraints. This problem, denoted as DPS-AIF-CA, is NP-hard. We propose a mathematical formulation and evaluate its performance on benchmark network topologies.

Keywords

Assignment problem, Federated learning, Energy minimization

1 Introduction

The emergence of 5G and the ongoing evolution toward beyond-5G networks are driving a paradigm shift in the design and operation of communication infrastructures. Modern networks are no longer static systems but highly dynamic, heterogeneous, and densely distributed ecosystems composed of diverse elements such as base stations, routers, switches, or edge servers. These components continuously generate vast amounts of statistical information, namely network data. Handling and processing these data is becoming essential for enabling intelligent network automation and supporting advanced tasks such as traffic optimization, anomaly detection, predictive maintenance, fault localization, or resource management [2]. To address these challenges, there is a growing trend toward embedding AIF directly into the network fabric, where they can process data closer to its point of origin. This concept, commonly referred to as *in-network intelligence*, allows learning and decision-making processes to operate with significantly reduced latency, improved reactivity, and enhanced scalability. By analyzing live network conditions and adapting control operations in real time, in-network intelligence is expected to play a central role in the management of next-generation communication systems.

At the same time, the rapid expansion and densification of network infrastructures have dramatically increased the structural complexity of communication networks. Next-generation architectures are becoming inherently distributed and hierarchical, with intelligence expected to be deployed across multiple layers, from centralized core network nodes to decentralized multi-access edge computing nodes, and even at base stations. This decentralization is motivated by the need to avoid the bottlenecks, overhead, and reliability concerns associated with purely centralized management. Consequently, distributed AI frameworks, most notably FL, are being actively explored as they enable training models from distributed data and sharing only model updates for aggregation. This reduces communication overhead and allows learning from local data and context-specific network observation. However, effectively supporting FL in operator networks requires more than local training alone: data must be collected, preprocessed, and routed in a timely and energy-efficient manner. The Data Pipeline System (DPS) plays a key role in enabling such in-network intelligence by structuring data flows from sources toward AIF clients [8].

As depicted in Figure 1, the DPS consists of a virtual chain of three functions: network data flows from the sources to the *ingress broker function (ib)*, it is then processed in batch by the *preprocessing function (p)*, and it is finally delivered by the *egress broker function (eb)* to the corresponding AIF client.

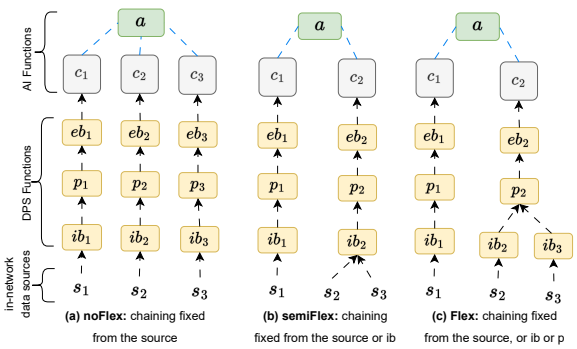

Figure 1: Three possible settings of DPS functions (*ib*, *p*, and *eb*) connecting data sources *s* to AIF client *c*

*Authors contributed equally to this research.

INOC '26, Liège (Belgium)

10.48786/inoc.2026.19

In this work, the DPS functions and the AIF for FL are modelled as virtual network nodes, and the data flows between them are represented as virtual links, where merging data at *s*, at *ib* or at *p* is enforced to preserve data locality. Mapping this virtual network onto the physical network infrastructure results in an assignment problem. The aim is then to jointly optimize DPS and AIF Chaining and their Assignment within the operator network to enable efficient in-network FL. This problem will be denoted (DPS-AIF-CA) in the sequel. To the best of our knowledge, no prior work directly addresses this problem, even though this problem can be naturally related to the virtual network embedding (VNE) problems [7, 12]. VNE problems have been extensively studied and are often modelled as mixed-integer linear or quadratic optimization problem [6, 11]. Some heuristics or decomposition-based approaches were also proposed [3, 4]. These works typically consider node resource constraints and, in some cases, bandwidth and end-to-end latency constraints, together with structural constraints such as function precedence, placement conflicts, or function sharing across virtual function chains. Existing objectives include installation and operational cost minimization [4], energy consumption minimization [3, 13], or joint energy–traffic-aware optimization [10].

Unlike standard VNE problems, achieving FL in large-scale networks introduces several fundamental challenges. First the chaining and assignment of the DPS functions strongly influence the latency, freshness, and quality of the data delivered to the AIF clients. Suboptimal configurations may lead to excessive delays, stale data, and degraded model performance, which can introduce bias or reduce training stability in FL scenarios. Moreover, the vast amount of data to handle raises significant concerns regarding network-wide energy consumption, as both communication and computation contribute to overall energy usage. Efficient assignment of DPS functions and AIF is therefore crucial not only for performance but also for sustainable and energy-efficient network operation. These issues highlight the need for a holistic approach that jointly considers the chaining and assignment of the DPS functions and AIF within an integrated framework which is the scope of our work.

The key contributions of this paper can be summarized as follows. We first prove that the DPS-AIF-CA problem is NP-hard (Section 2), then we introduce a mathematical model for the DPS-AIF-CA problem (Sections 3). Finally, we perform a thorough evaluation on benchmark network topologies from the literature to assess the efficiency of our formulation (Section 4).

2 Problem Statement

Mapping DPS functions and AIF (clients and aggregator) onto nodes of the operator network and determining the routing of data flows through the network reduces to an assignment problem. The routing decisions specify the paths as well as the data between successive functions and the associated traffic volumes. The solution must satisfy a set of constraints. Namely, resource constraints must be satisfied as memory and processing capacities are limited on nodes. In addition, bandwidth capacities on the links must be observed. Finally, latency constraints are imposed. Indeed, AIF clients require sufficiently recent data to ensure up-to-date training. Therefore, the age of data, which we define as the end-to-end latency along the data paths to each AIF client, must not exceed the target threshold, in order to prevent training on stale data. In addition to resource, bandwidth, and latency constraints, the virtual chaining and network assignment affects the overall energy consumption of the operator's network. The total energy consumption includes a fixed energy part for configuring nodes and a variable part induced by deploying functions, processing or routing data. In addition, energy consumption is incurred for transporting data across the network.

Enforcing strict end-to-end latency constraints to guarantee data freshness may require deploying, for each source, a dedicated DPS function chain and an AIF client, and placing both as close as possible to the associated data source. While this strategy reduces latency, it increases the number of active nodes and the overall network utilization, which in turn raises energy consumption. Conversely, increasing the number of data sources assigned to the same *ib*, as well as the number of *ib* assigned to a *p* function, can reduce the number of active nodes and the number of links involved in data transport. However, this comes at the cost of increased processing and pulling times, which can lead to violations of strict end-to-end latency constraints. Therefore, the DPS-AIF-CA problem inherently entails a trade-off among data freshness, resource usage, and energy efficiency. The parameters of the problem are formally defined in the following.

For every FL round, all data must be collected, processed by DPS functions, and delivered to the AIF clients within a given time limit $T_{\max}$, which specifies the maximum allowable age of the data delivered to the AIF clients in each training round. This per-round latency constraint is enforced independently for all rounds, thereby ensuring temporally consistent and up-to-date inputs across successive FL iterations.

The operator network infrastructure is modeled as an undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of links. Each node $v \in \mathcal{V}$ has an available computing resource A_v, characterized by the amount of memory and the number of CPU cores. For example, a node v may have computing resources of 32 GB of memory and 16 CPU cores. An available bandwidth A_{uv} is defined for each link $(u, v) \in \mathcal{E}$. The parameter D_{uv} provides the time required to transmit data from node u to v, namely the propagation delay. The set of data sources, which may include base stations, VNF embedded within nodes, or any other nodes in the operator network from which network data are collected for AI task purposes, is denoted $\mathcal{S}$. We will consider that each data source $s \in \mathcal{S}$ produces N_s data samples per training round.

As mentioned earlier, the DPS is composed of a set of virtual chains of three functions, the *ib*, which queues data sent by the associated data sources, the *p* function, which fetches data from the associated *ib* for batch processing, and the *eb*, which queues preprocessed data and makes them available for consumption by the associated AIF client. The *p* function produces data batches that are indivisible and of size between $B_{\min}$ and $B_{\max}$.

We denote $\mathcal{I}$, $\mathcal{P}$ and $\mathcal{O}$ as the set of *ib*, *p* and *eb* functions respectively. A maximum number of instances allowed for all functions in the network $M_{\max}$ is imposed. We consider a set $\mathcal{C}$ of AIF clients, which train local models using data received from the associated DPS function chain. We adopt the classical FL setup, where an aggregator, denoted by a, periodically performs the aggregation after receiving the model updates of the locally trained models from the set of connected AIF clients $c \in \mathcal{C}$.

We assume that each function $f \in \mathcal{F}$, where $\mathcal{F} = \mathcal{I} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{C} \cup \{a\}$, is deployed on a node as a Docker container and therefore it requires a computational resource U_f, which captures both the amount of memory needed to load the function onto the node, and the number of CPU cores required to process incoming data. We

also define $\mathcal{L} = \{(\mathcal{S}\times\mathcal{I})\cup(\mathcal{I}\times\mathcal{P})\cup(\mathcal{P}\times\mathcal{O})\cup(\mathcal{O}\times\mathcal{C})\cup(\mathcal{C}\times\{a\})\}$ as the set of allowable function virtual links.

Parameter E_{fv} denotes the energy consumed to process one byte of data when function f is executed on node v. The energy consumed to transmit one byte of data over the network link (u, v) is given by E_{uv}. Installing and launching the function induces a fixed energy consumption F_{fv} when deploying function f on node v. Similarly, configuring node v to host one or more functions f incurs a fixed energy consumption F_v.

A per-sample processing time T_f is defined for each function $f \in \mathcal{F} \setminus \{a\}$.

Finally, the number of bytes per sample for a function f is defined by α_f. Before processing at p, all functions operate on raw data of a given size. Once processed by p, the data size is reduced by a factor $\rho \in (0, 1)$, accounting for filtering, normalization, and other transformations required to generate AIF-ready data.

The virtual chaining problem aims to determine both the number of functions $f \in \mathcal{F}$ to deploy in the network and their chaining structure, which can be represented as a sequence (virtual chain) (s, i, j, k, c, a) where $s \in \mathcal{S}$ is the data source, $i \in \mathcal{I}$ the *ib* function, $j \in \mathcal{P}$ the *p* function, $k \in \mathcal{O}$ the *eb* function, $c \in \mathcal{C}$ the AIF client and a the aggregator. The chaining allows at least one source to be assigned to the same *ib*, as well as at least one *ib* to be assigned to the same *p* function. In addition, the chaining must follow a tree structure as the ones depicted in 1, and the clients and the aggregator a cannot share the same physical node.

The following theorem establishes the complexity of the problem.

Theorem 1. *The DPS-AIF-CA problem is NP-hard.*

Proof. We consider a restricted version of our problem in which there is a single source and no capacity constraints on either links or nodes. We set $T_{\max} = +\infty$ (i.e., no latency constraints). Precedence constraints (i.e., no ordering among DPS functions and AIF) and batch size constraints are relaxed. Furthermore, unit energy consumption on arcs E_{uv} and on nodes E_{fv} are set to zero, and the number of data samples at the source is fixed to $N_s = 1$. The aggregator function is considered as fixed. Under these assumptions, the resulting restricted instance is identical to the instance used to establish the NP-hardness of the VNF placement and routing problem addressed in [4]. This problem reduces to an uncapacitated facility location problem, which is known to be NP-hard. Therefore, our problem is also NP-hard. □

3 DPS-AIF-CA Problem Formulation

In this section, we present the mathematical formulation of the DPS-AIF-CA problem. The set of candidate functions that can be placed right after (resp. before) function $f \in \mathcal{F}$ in the sequence representing the virtual chain is denoted $\delta^+(f) = \{g \mid (f, g) \in \mathcal{L}\}$ (resp. $\delta^-(f) = \{h \mid (h, f) \in \mathcal{L}\}$), they are also called downstream (resp. upstream) functions.

Decision variables. We introduce binary variables x_{fg} ($(f, g) \in \mathcal{L}$) that indicate if the pair of functions (f, g) corresponds to an active virtual link. For the assignment of virtual chains to the physical network, we use binary assignment variable y_{fv} ($(f, v) \in \{(\mathcal{S}\times\mathcal{V})\cup(\mathcal{F}\times\mathcal{V})\}$) that is equal to 1 if function f is deployed at node v. Moreover, variables w_v ($v \in \mathcal{V}$) indicate if node v is active. To model the data paths in the physical network, we decompose the path into active links related to each (f, g). These links will be defined by binary variables z^{uv}_{fg} ($(f, g) \in \mathcal{L}, (u, v) \in \mathcal{E}$), that indicate whether the physical link (u, v) is used to route data from f to g. The number of data samples from f to g are given by positive variables $n_{fg}((f, g) \in \mathcal{L})$. The number of data sample reaching function g is defined by positive variables $n_g(g \in \mathcal{F})$. Finally, we use positive variables $t_f(f \in \mathcal{I} \cup \mathcal{P} \cup \mathcal{O})$ to compute the time at which the data ends being processed by function f. Since the age of data samples from its sources to the AIF clients are bounded by T_{max}, we introduce a positive variable $t_c(c \in C)$ representing the end-to-end latency to client c.

In what follows, we describe more formally the constraints of the DPS-AIF-CA problem.

Virtual chaining constraints.

Source chaining. Each source s is linked to exactly one *ib* i:

$$\sum_{i\in\mathcal{I}} x_{si} = 1, \quad \forall s \in \mathcal{S}. \tag{1}$$

Single downstream function. Each function f can send its output to at most one downstream function g, enabling a tree structure for the function chaining:

$$\sum_{g\in\delta^+(f)} x_{fg} \leq 1, \quad \forall f \in \mathcal{F} \setminus \{a\}. \tag{2}$$

Single upstream function for eb and AIF clients. Since the *p* function merges data sample into an indivisible batch, we enforce that the processed batch follows the same path in the rest of the virtual chain, (i.e. same *eb* and same AIF client) to satisfy the data locality constraint of FL:

$$\sum_{h\in\delta^-(f)} x_{hf} \leq 1 \quad \forall f \in \mathcal{O} \cup C. \tag{3}$$

Activation of virtual chains from p functions. If an *ib* i sends data to a *p* function j, the downstream virtual chain must be activated where n is the associated index:

$$x_{j_n k_n} \geq x_{ij_n},\ x_{k_n c_n} \geq x_{ij_n},\ x_{c_n a} \geq x_{ij_n}, \forall i \in \mathcal{I}, \forall n \in [\![1, M_{\max}]\!]. \tag{4}$$

Virtual chaining consistency. To ensure that each virtual chain starts from a source and ends at the aggregator, we activate a virtual link (f, g) only if f has both upstream and downstream functions:

$$x_{fg} \leq \sum_{h\in\delta^+(g)} x_{gh}, \quad \forall (f, g) \in \mathcal{L} \setminus C \times \{a\}, \tag{5}$$

$$x_{fg} \leq \sum_{h\in\delta^-(f)} x_{hf}, \quad \forall (f, g) \in \mathcal{L} \setminus \mathcal{S} \times \mathcal{I}. \tag{6}$$

Number of data samples in the virtual chains. The number of data of a source s is N_s, and the maximum number of data between f and g is bounded by the maximum batch size $B_{\max}$:

$$n_{si} = N_s x_{si}, \quad \forall s \in \mathcal{S},\ \forall i \in \mathcal{I}. \tag{7}$$

$$n_{fg} \leq B_{\max} x_{fg}, \quad \forall (f, g) \in \mathcal{L} \setminus (\mathcal{S} \times \mathcal{I}). \tag{8}$$

Number of data samples reaching function g.

$$n_g = \sum_{f\in\delta^-(g)} n_{fg}, \quad \forall g \in \mathcal{F}. \tag{9}$$

Batch size limits at p.

$$B_{min} \leq n_j \leq B_{max}, \quad \forall j \in \mathcal{P}. \tag{10}$$

Flow conservation at each function of the virtual chain.

$$\sum_{h\in\delta^-(f)} n_{hf} - \sum_{g\in\delta^+(f)} n_{fg} = 0, \quad \forall f \in \mathcal{F} \setminus \{a\}. \tag{11}$$

Constraints for assigning virtual chains to physical network.

Assignment of virtual functions to physical nodes. We enforce function f to be assigned to at most one physical node v, only if f is activated within a virtual function chain:

$$\sum_{v \in V} y_{fv} \leq 1, \qquad \forall f \in \mathcal{F}, \tag{12}$$

$$\sum_{v \in V} y_{fv} \geq x_{hf}, \qquad \forall f \in \mathcal{F}, \quad \forall h \in \delta^-(f). \tag{13}$$

Node activation. We can assign function f to node v only if v is active:

$$y_{fv} \leq w_v, \quad \forall f \in \mathcal{F}, \quad \forall v \in \mathcal{V}. \tag{14}$$

AIF clients and aggregator conflict constraint. The aggregator cannot be assigned to the same node than any client:

$$y_{cv} + y_{av} \leq 1, \quad \forall c \in C, \ \forall v \in \mathcal{V}. \tag{15}$$

Data paths in the physical network. Any active virtual link in $\mathcal{L}$ between f and g must follow a path in the physical network:

$$\sum_{w:(v,w)\in\mathcal{E}} z_{fg}^{vw} - \sum_{u:(u,v)\in\mathcal{E}} z_{fg}^{uv} = x_{fg}\,(y_{fv} - y_{gv}), \forall v \in \mathcal{V}, \ \forall (f,g), \tag{16}$$

$$\sum_{u:(u,v)\in\mathcal{E}} z_{fg}^{uv} \leq 1 - y_{fv}, \quad \forall v \in \mathcal{V}, \ \forall (f,g), \tag{17}$$

$$\sum_{w:(v,w)\in\mathcal{E}} z_{fg}^{vw} \leq 1 - y_{gv}, \quad v \in \mathcal{V}, \ \forall (f,g). \tag{18}$$

Chaining-routing consistency.

If there is no virtual link between f and g, or if both functions are assigned to the same node v, there exist no path in the physical network between f and g:

$$z_{fg}^{uv} \leq x_{fg}, \quad \forall (f,g) \in \mathcal{L}, \ \forall (u,v) \in \mathcal{E}, \tag{19}$$

$$z_{fg}^{uv} \leq 2 - (y_{fv} + y_{gv}), \quad \forall (f,g) \in \mathcal{L}, \ \forall (u,v) \in \mathcal{E}. \tag{20}$$

Bandwidth capacity. Each edge $(u,v) \in \mathcal{E}$ has a bandwidth capacity A_{uv} to be satisfied:

$$\sum_{(f,g)\in\mathcal{L}} \alpha_g n_{fg} z_{fg}^{uv} \leq A_{uv}, \quad \forall (u,v) \in \mathcal{E}. \tag{21}$$

Node resources capacity. The total required resources (CPU, memory) of all functions assigned to a node must not exceed A_v:

$$\sum_{f \in \mathcal{F}} U_f y_{fv} \leq A_v, \quad \forall v \in \mathcal{V}. \tag{22}$$

Computation of the end-to-end latency.

The time t_i at which *ib* function i ends its processing is equal to the sum of the latest arrival time of data from its upstreamed sources and of the processing time of *ib*:

$$t_i = \max_{s \in \mathcal{S}} \Big\{ \sum_{(u,v)\in\mathcal{E}} D_{uv} z_{si}^{uv} \Big\} + T_i n_i, \quad \forall i \in \mathcal{I}, \tag{23}$$

Similarly, the time t_j at which function p ends its processing is:

$$t_j = \max_{i \in \mathcal{I}} \Big\{ t_i x_{ij} + \sum_{(u,v)\in\mathcal{E}} D_{uv} z_{ij}^{uv} \Big\} + T_j n_j, \quad \forall j \in \mathcal{P}. \tag{24}$$

We classically linearize the max operators of Constraints (23) and (24) as follows:

$$t_i \geq \sum_{(u,v)\in\mathcal{E}} D_{uv} z_{si}^{uv} + T_i n_i, \qquad \forall s \in \mathcal{S}, \forall i \in \mathcal{I}, \tag{25}$$

$$t_j \geq t_i x_{ij} + \sum_{(u,v)\in\mathcal{E}} D_{uv} z_{ij}^{uv} + T_j n_j, \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{P}. \tag{26}$$

Recall that there is no data merging in the virtual chains after p, therefore the time t_{k_n} at which *eb* function k_n ends its processing is:

$$t_{k_n} = t_{j_n} + \sum_{(u,v)\in\mathcal{E}} D_{uv} z_{j_n k_n}^{uv} + T_{k_n} n_{j_n k_n}, \quad \forall n \in [\![1, M_{\max}]\!]. \tag{27}$$

The end-to-end latency between the sources and a client c_n is:

$$t_{c_n} = t_{k_n} + \sum_{(u,v)\in\mathcal{E}} D_{uv} z_{k_n c_n}^{uv}, \quad \forall n \in [\![1, M_{\max}]\!]. \tag{28}$$

We finally enforce that the allowable end-to-end latency at client c (age of data) is at most T_{max}:

$$t_c \leq T_{\max}, \qquad \forall c \in C. \tag{29}$$

Objective function. We aim to minimize the total energy consumed by the network for handling the DPS and AIF, which includes fixed energy consumption related to activation of nodes and assignment of functions, as well as variable energy consumption induced by data processing and propagation on the physical network.

The fixed amount of energy related to node activation and function assignment is given by:

$$E_{\text{act}} = \sum_{v \in \mathcal{V}} \sum_{f \in \mathcal{F}} F_{fv} y_{fv} + \sum_{v \in \mathcal{V}} F_v w_v. \tag{30}$$

The data processing and propagation energy consumption is weighted by the number of data samples processed by each virtual function and transmitted over physical links:

$$E_{\text{exec}} = \sum_{v \in \mathcal{V}} \sum_{f \in \mathcal{F}} \alpha_f E_{fv} n_f y_{fv} + \sum_{(u,v)\in\mathcal{E}} \sum_{(f,g)\in\mathcal{L}} \alpha_g E_{uv} n_{fg} z_{fg}^{uv}. \tag{31}$$

In addition, since the physical distance D_{uv} between two nodes is not considered in E_{exec}, we consider an additional term in the objective function to enforce having solutions that minimizes the distances in the physical network, and so to decrease the end-to-end latency between sources and the aggregator. To capture this, we introduce into the objective function a third term, E_{time}:

$$E_{\text{time}} = E_\tau \sum_{c \in C} \Big(t_c + \sum_{(u,v)\in\mathcal{E}} D_{uv} z_{ca}^{uv} \Big). \tag{32}$$

where E_τ is a scaling factor chosen such that E_{time} acts as a secondary objective.

Mathematical model. The resulting formulation of the DPS-AIF-CA problem is a binary quadratically constrained quadratic problem:

$$\begin{cases} \min \quad E_{\text{act}} + E_{\text{exec}} + E_{\text{time}} \\ \text{s.t. } (1) - (22), \ (25) - (29) \end{cases}$$

Remarks. Note that Constraints (12), (19) and (20) are redundant by optimality, as the objective function minimizes the assignment variables y_{fv} and data path variables z_{fg}^{uv}. However, keeping it in our formulation ensures that the solutions obtained by solving the problem within a restricted time limit are at least feasible for DPS-AIF-CA. Furthermore, the proposed formulation can be easily extended to a multi-aggregator setting.

4 Computational results

In this section we evaluate the computational performances of our formulation ($DPS - AIF - CA$) on three network topologies and three different virtual chaining settings.

4.1 Experimental setup and instance generation

Experimental environment. Our experiments were carried out on a laptop with an AMD Ryzen 5 5625U CPU (6 cores, 12 threads, 4.39 GHz) and 30 GB of RAM, under a Linux operating system. We use the solver `Gurobi 13.0.0` to solve the binary quadratic problem ($DPS-AIF-CA$). For all our experiments, we set a total time limit of 1800 seconds.

Network topologies. We consider three network topologies. The first one, *netOp* (67 nodes, 70 links), is derived from a realistic network operator dataset [8]. The two other topologies, *Geant* (22 nodes, 36 links) and *Brain* (161 nodes, 166 links), are taken from the SNDlib library [9]. *Geant* is a small network with long inter-node distances, while *Brain* is a larger network with smaller distances.

Parameter generation description. For *netOp*, the propagation delays D_{uv} are taken from [8], and for *Geant* and *Brain*, they are computed from node coordinates using the Haversine formula, assuming optical fiber links [5]. For all topologies, the set of nodes is partitioned in two sets: *Edge* nodes with limited capacities $A_v = (32\ GB, 6CPUs)$, and *Core* network nodes with larger capacities $A_v = (64\ GB, 28CPUs)$. *netOp* has 7 core and 60 edge nodes, *Geant* has 13 core and 9 edge nodes and *Brain* has 9 core and 152 edge nodes. In order to estimate per-function resource usage U_f, per-sample processing time T_f, and energy consumption F_{fv}, we ran an FL-based anomaly detection integrated with DPS [8]. The per-byte transmission energy consumption E_{uv} are taken from the literature [1]. All the parameter values are reported in Table 1.

We set for all experiments $B_{\min}$ to 50 and $B_{\max}$ to 200. The maximum end-to-end latency has been set such that both feasibility and data-freshness are ensured. Preliminary experiments allow finding relevant values that vary from 90 to 200. The sources exist only on edge nodes and we consider a maximum of one source per edge node. The number of data samples generated by each source follows a uniform distribution in [50, 100].

Table 1: Parameters of the *NetOp*, *Geant* and *Brain* topologies

	$f \in \mathcal{I}$	$f \in \mathcal{P}$	$f \in O$	$f \in C$	a
E_{fv}-Core (J)	$2.44 \cdot 10^{-5}$	$2.44 \cdot 10^{-5}$	$2.3 \cdot 10^{-5}$	$2.43 \cdot 10^{-3}$	$4.14 \cdot 10^{-5}$
E_{fv}-Edge (J)	$9.7 \cdot 10^{-6}$	$9.7 \cdot 10^{-6}$	$9.22 \cdot 10^{-6}$	$7.32 \cdot 10^{-4}$	$8.29 \cdot 10^{-6}$
F_{fv} (J)	4.75	4.27	4.7	4.73	5.06
$U_f(GB, CPUs)$	(0.5, 1)	(0.25, 1)	(0.25,1)	(2, 4)	(1,2)
T_f (ms)	0.18	0.21	0.2	1.5	–

Per-byte transmission energy: $E_{uv} \approx 9.65 \cdot 10^{-8}$
Scaling factor: $E_\tau = 0.01$

4.2 Experimental analysis

We first describe the virtual chaining settings before analyzing the impact of two structural parameters of the problem, end-to-end latency and number of sources.

Virtual chaining settings.

The DPS-AIF-CA formulation corresponds to our proposed fully flexible chaining configuration. An illustrative example of such a virtual function chain is shown in Figure 1(c), referred to as the *Flex* setting. To computationally evaluate DPS-AIF-CA with respect to the trade-off between total energy consumption and computational time, we investigate the impact of progressively restricting its flexibility. Specifically, we introduce two additional settings *noFlex* and *semiFlex* depicted in Figure 1(a) and Figure 1(b) respectively. These setting are obtained by incorporating additional constraints into the DPS-AIF-CA formulation (i.e. *Flex* setting) and and by using the index n to indicate from which point the virtual function chain is activated with respect to constraint (4).

- *noFlex*, where the virtual function chaining is fixed from the source, i.e. each virtual function chain is independent and there is exactly one chain from each source to the aggregator. This is enforced by adding the constraints

$$x_{s_n i_n} = 1, \quad x_{i_n j_n} \geq x_{s_n i_n}, \forall n \in [\![1, M_{\max}]\!], \tag{35}$$

  where $M_{\max} = |S|$.
- *semiFlex*, where the virtual function chaining is fixed from *ib*, i.e. merging chains is allowed at *ib*. This is enforced by adding the constraints

$$x_{i_n j_n} \geq x_{s i_n}, \forall s \in \mathcal{S}, \forall n \in [\![1, M_{\max}]\!], \tag{36}$$

  where $M_{\max} \leq |S|$.

Impact of the maximum end-to-end latency

Table 2: Results for *Geant* with varying $T_{\max}$

$T_{\max}$	noFlex		semiFlex		Flex	
	Opt	*Time (s)*	*Opt*	*Time (s)*	*Opt*	*Time (s)*
90	333.18	3.1	314.61	374.5	290.01	252.3
100	333.18	4.9	281.96	575.0	281.96	244.7
110	333.18	2.1	278.54	87.1	270.48	388.4
120	333.18	5.0	261.45	52.0	261.45	241.6
130	333.18	4.0	259.37	100.3	259.37	335.9
140	333.18	2.6	258.03	69.5	258.03	376.0
150	333.18	3.3	257.93	103.7	257.93	277.7

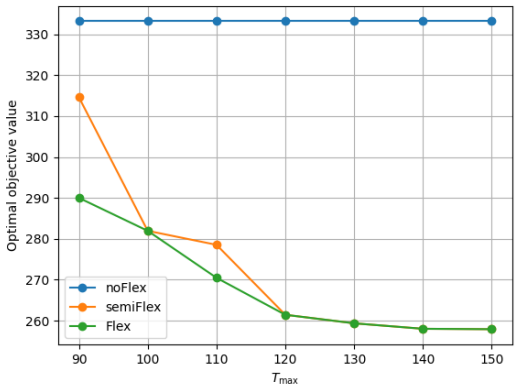

Figure 2: Impact of $T_{\max}$ on optimal energy consumption

We evaluate the impact of parameter T_{max} across the different settings with 6 sources on *Geant*. We present in Table 2 the results for the *Geant* network topology where each line corresponds to one instance. For each setting, we report the optimal value (*Opt*) of each instance, and the computation time to find the optimal

Table 3: Optimality gap results with varying number of sources on the three settings

Instance	#Sources	noFlex			semiFlex			Flex		
		Gap 10 min	Gap 20 min	Gap 30 min	Gap 10 min	Gap 20 min	Gap 30 min	Gap 10 min	Gap 20 min	Gap 30 min
Geant	4	0	0	0	0	0	0	0	0	0
	8	0	0	0	0.07 (0.21)	0.00 (0.00)	0.00 (0.00)	3.59 (5.30)	2.69 (4.72)	1.84 (3.85)
NetOp	4	0	0	0	0	0	0	0	0	0
	8	0	0	0	14.02 (27.98)	8.04 (13.13)	5.51 (9.14)	6.00 (9.54)	5.16 (8.48)	4.91 (8.09)
	16	1.02 (2.65)	0.49 (2.43)	0.30 (1.45)	44.05 (46.62)	27.48 (28.85)	26.13 (28.55)	48.84 (61.82)	43.78 (61.82)	31.11 (45.02)
Brain	4	0	0	0	0.29 (1.43)	0.24 (1.21)	0.19 (0.96)	2.60 (4.12)	1.02 (1.60)	0.60 (1.19)
	8	0	0	0	28.46 (32.20)	25.56 (30.17)	22.46 (28.99)	19.58 (31.79)	11.39 (17.80)	10.73 (15.67)
	16	3.29 (3.99)	2.35 (3.93)	1.77 (3.14)	100.00 (100.00)	100.00 (100.00)	82.32 (82.32)	100.00 (100.00)	70.54 (73.17)	68.09 (69.77)

solution (*Time*) in seconds. We observe that the average computation time is 3.6s, 194.6s and 302.4s for *noFlex*, *semiFlex* and *Flex* respectively. This result shows that a strong restriction on the virtual chaining can lead to fast computational time. Regarding the optimal energy consumption, allowing more flexibility in the virtual chaining induces a gain of 19, 4% in average. Figure 2 depicts the optimal values of each instance for each setting. We observe that parameter $T_{\max}$ has no impact on the optimal value in the *noFlex* setting, while for settings *semiFlex* and *Flex* energy consumption is reduced when $T_{\max}$ increases up to 130, beyond which the objective value stabilizes. From the objective value point of view, setting *Flex* outperforms the other settings. The choice of $T_{\max}$ reflects a trade-off between data freshness, and energy efficiency.

Impact of the number of sources. We report the results of experiments where the number of sources are set to 4, 8, and 16 for a fixed $T_{\max} = 150$ for *Geant*, and $T_{\max} = 200$ for *netOp* and *Brain*. Since *Geant* has only 9 edge nodes and since sources are placed exclusively on edge nodes, the 16-source case is not relevant for *Geant*. To ensure statistically reliable results, each experimental setting is executed 5 times with source nodes randomly selected among the edge nodes, sources are chosen randomly in the edge nodes and number of data samples N_s from a uniform law in $[50, 100]$.

The experimental results are reported in Table 3. The average gap (*Gap*) equals to $\frac{UB-LB}{LB} * 100$, where UB (resp. LB) is the best Upper Bound (resp. Lower bound) known after 10, 20, or 30 minutes of computation time. The values between parentheses represent the average maximum gaps computed for instances providing finite gaps.

For a small number of sources (4), all settings reach the optimal solution within 10 minutes. As the number of sources increases, the *noFlex* setting maintains low optimality gaps, whereas the *semiFlex* and Flex settings exhibit significantly larger gaps, particularly for large-scale network topologies such as*NetOp* and *Brain*. While increased flexibility enables higher potential energy savings, it also results in longer computation times and larger solution gaps.

5 Conclusion

We introduced DPS-AIF-CA, a model for the energy-efficient placement of DPS functions and AIF for in-network FL, ensuring that network data are routed to AIF clients within strict time bounds. The DPS-AIF-CA model enables chain merging at the *ib* and *p* functions to preserve data locality, which is a key constraint in FL settings. We proved that DPS-AIF-CA is NP-hard and formulated it as a binary quadratically constrained quadratic program.

Evaluation results on realistic network topologies show that the model finds an optimal solution with computation times below 10 minutes for small networks or a limited number of data sources, but scalability becomes challenging as the network size and the number of sources grow. Future work will therefore focus on developing scalable optimization algorithms and heuristics to address these limitations.

Acknowledgments

This work was funded by the TREES ANR project (ANR-24-TSIA-0004; trees.roc.cnam.fr).

References

[1] Vincenzo De Maio, Radu Prodan, Shajulin Benedict, et al. 2016. Modelling energy consumption of network transfers and virtual machine migration. *FGCS* (2016).
[2] Abdelfatteh Haidine, Fatima Zahra Salmam, et al. 2021. Artificial Intelligence and Machine Learning in 5G and beyond: A Survey and Perspectives. In *Moving Broadband Mobile Communications Forward - Intelligent Technologies for 5G and Beyond*. IntechOpen, London, Chapter 3.
[3] Siri Kim, Yunjung Han, and Sungyong Park. 2016. An Energy-Aware Service Function Chaining and Reconfiguration Algorithm in NFV. In *IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*.
[4] Ivana Ljubić, Ahlam Mouaci, Nancy Perrot, and Éric Gourdin. 2021. Benders decomposition for a node-capacitated virtual network function placement and routing problem. *Computers & Operations Research* (2021).
[5] E Maria, E Budiman, M Taruk, et al. 2020. Measure distance locating nearest public facilities using Haversine and Euclidean Methods. In *Journal of Physics: Conference Series*, Vol. 1450. IOP Publishing, 012080.
[6] Sevil Mehraghdam, Matthias Keller, and Holger Karl. 2014. Specifying and placing chains of virtual network functions. In *International conference on CloudNet*. IEEE.
[7] Ahlam Mouaci, Éric Gourdin, Ivana LjubiĆ, and Nancy Perrot. 2020. Virtual network functions placement and routing problem: Path formulation. In *2020 IFIP Networking*. IEEE, 55–63.
[8] Patient Ntumba, Nour-El-Houda Yellas, Salah Bin-Ruba, et al. 2024. Data Pipeline System Designs for In-network Learning. In *International Conference on Network and Service Management (CNSM)*. 1–9.
[9] Sebastian Orlowski, Michał Pióro, Artur Tomaszewski, et al. 2023. SNDlib Survivable Network Design Library. https://sndlib.put.poznan.pl/home.action
[10] Chuan Pham, Nguyen H. Tran, Shaolei Ren, et al. 2020. Traffic-Aware and Energy-Efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach. *IEEE Transactions on Services Computing* 13, 1 (2020), 172–185.
[11] Mohammad Ali Raayatpanah, Thomas Weise, Jocelyne Elias, et al. 2025. A Mixed-Integer Linear Programming Approach for Congestion-Aware Optimized NFV Deployment. In *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*.
[12] Sheng Wu, Ning Chen, Ailing Xiao, et al. 2024. Ai-empowered virtual network embedding: a comprehensive survey. *Communications Surveys & Tutorials* (2024).
[13] Xiaoning Zhang, Zhichao Xu, Lang Fan, Shui Yu, and Youyang Qu. 2022. Near-Optimal Energy-Efficient Algorithm for Virtual Network Function Placement. *IEEE Transactions on Cloud Computing* (2022).