Optimized Split Computing Framework for Edge and Core Devices

Andrea Tassi, Oluwatayo Yetunde Kolawole, Joan Pujol Roig

***Abstract*—With their ever-increasing number of users, beyond 5G mobile networks will face the challenge of ensuring a consistent and high-quality user experience in which Feed-Forward Neural Networks (FFNNs) models run on User Equipements (UEs) is a key element. This letter proposes an optimization framework for split computing applications where an FFNN model is partitioned into multiple sections executed by UEs, edge- and core-located nodes to resource the overall UEs' computational footprint while containing the inference time. An efficient heuristic strategy for solving the defined split computing problem is also provided. Numerical results show that the proposed framework efficiently offloads parts of the considered FFNN models from a UE to the serving base stations and a core-located node, which leads to a reduction of UEs' computation footprint of over 33%, in the considered settings.**

***Index Terms*—Split Computing, Feed-Forward Neural Network, 5G and 6G.**

I. Introduction

As the evolution of mobile networks progresses beyond 5G, there is mounting anticipation for the capability to support a large number of devices, ensure consistent and high-quality user experiences, and actualize instantaneous, zero-latency networks. Deep neural networks are being employed for the optimization of tasks, and the execution of complex inferences in challenging scenarios such as real-time navigation or user interactions in extended/augmented reality applications and gaming [1].

This letter focuses on the issue of offloading the computational and memory footprint associated with running inference tasks on pre-trained Feed-Forward Neural Networks (FFNNs) from low-end User Equipments (UEs) onto other network entities. However, the surge in predictive accuracy for real-time operations of modern FFNN models is accompanied by a spike in computational demands and energy consumption [2]. This makes deploying FFNN models in mobile applications particularly daunting, given smartphones' inherent power and computational limitations.

To overcome these limitations, a prevalent approach in the industry is offloading the entire computation and processing tasks to edge and/or cloud-based processing nodes [3]. Although alleviating the computation burden, these edge-/could-based computing methods do not intrinsically consider the quality of wireless links in the network. These connections are susceptible to erratic disruptions due to unpredictable interference and noise patterns. To this end, some works advocate creating bespoke, lightweight FFNN models tailored for resource-limited mobile devices [4]. Yet, besides this not always feasible, this practice often compromises the models' accuracy, resulting in a diminished quality of service. Hybrid strategies like Split Computing (SC) have emerged [5].

Originally, SC was a technique designed to partition neural network models into two sections: the head, which operates at the UE, and the tail, processed at an edge server. The head section handles initial data processing, while the tail section manages deeper analyses at the resource-rich edge server. A unique feature of SC is its dynamic splitting point, allowing the model to adjust in real time to both fluctuating wireless channel conditions and the specific constraints of the UE. This adaptability ensures efficient data transfer, optimized performance, and reduced energy consumption, making SC a pivotal solution for deploying neural network models in resource-constrained environments. Nevertheless, post proposals of SC introduce artificial bottlenecks to establish task-oriented compression and reduce transfer delays [6].

In contrast to the existing SC methods [4]–[6], this letter proposes a framework for determining an arbitrary number of splitting points while taking into account the link quality between the various devices running inference tasks and their computational capabilities. In the considered system model, UEs always initiate FFNN inference requests. Due to its computational and memory footprint, an FFNN inference task is shared among the UE that initiated it, its serving base station, and potentially a core network-located processing node. The proposed optimization model accounts for computational and memory resources available across all the network actors that may play a part in completing an FFNN inference task. At the same time, the proposed model minimizes the end-to-end communication delay associated with the wireless/wired network links interconnecting the actors. An efficient heuristic strategy for calculating feasible solutions to the problem above is also provided.

The rest of the paper is structured as follows. Section II presents the considered system model. The proposed split computing optimization problem and the procedure to calculate its heuristic solution are described in Sections III and IV, respectively. A performance evaluation of the proposed approach is given in Section V. Finally, in Section VI, we draw our conclusions.

II. System Model

We refer to a system model consisting of a set of devices $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{D}|}\}$ capable of running (partially or in full) an inference in a given FFNN model. The computational capacities of each device are listed in set $\mathcal{R} = \{(r_i^{(\text{CPU})}, r_i^{(\text{Mem})}), \forall i \in [|\mathcal{D}|]$ [1], where tuple $(r_i^{(\text{CPU})}, r_i^{(\text{Mem})})$ delineates the peak CPU capacity and the utmost memory allocation for device d_i, respectively.

[1]For positive integer K, $[K]$ denotes the set $\{1, 2, \ldots, K\}$

The authors are with the Advanced Network Research, Samsung R&D Institute UK (SRUK), UK (e-mail: {a.tassi, o.kolawole, j.pujolroig}@samsung.com). This work is a contribution by Project REASON, a UK Government funded project under the Future Open Networks Research Challengeov (FONRC) sponsored by the Department of Science Innovation and Technology (DSIT).

CPU capacities are articulated as an aggregate of individual capacities, with $r_i^{(\text{CPU})} \in [0,1]$, representing the allocation on each CPU core. This is then normalized concerning the highest CPU capacity across all devices. Similarly, memory capacities are normalized based on the maximum memory allocation of any device, thus $r_i^{(\text{Mem})} \in [0,1]$.

The connectivity blueprint of these devices, detailing both wired and wireless interconnections, is represented by a matrix $\mathbf{W}$ of dimensions $|\mathcal{D}| \times |\mathcal{D}|$. In this matrix, the element $w_{i,j}$ is assigned a value of 0 if there is no network link between device d_i and device d_j. If a link exists, the maximum data rate is assigned to this link (measured in bit/s).

We now formulate the following assumption:

Assumption 2.1: Device d_i can directly communicate with device d_j via direct wireless or wired link if and only if $j-i=1$. For any pair of devices d_t and d_u (where $u-t>1$, $t<|\mathcal{D}|$ and $u<|\mathcal{D}|$), d_t can communicate with device d_u only via the sequence of intermediate devices $[d_{t+1}, d_{t+2}, d_{t+3}, \ldots, d_{u-1}]$.

The aforementioned assumption introduces a hierarchical structure onto the devices in $\mathcal{D}$, which mirrors the scenarios described in Section I where the inference task always originated in the UE belonging to the lowest performance tier. Consequently, without loss of generality, we make the following assumption:

Assumption 2.2: Only device d_1 is the device capable of initializing an inference task.

We model a FFNN model as a tuple $(\mathcal{L}, \mathcal{C}, \mathbf{B})$. Set $\mathcal{L} = \{l_1, l_2 \ldots, l_{|\mathcal{L}|}\}$ consists of the FFNN model's layers, where l_i ($\forall i \in [|\mathcal{L}|]$) denotes the i-th layer in the model.

Execution costs of each layer are listed in set $\mathcal{C} = \{(c_i^{(\text{CPU})}, c_i^{(\text{Mem})}), \forall i \in [|\mathcal{L}|]\}$, where tuple $(c_i^{(\text{CPU})}, c_i^{(\text{Mem})})$ specifies the CPU cost and the memory cost for layer ℓ_i, respectively. Both CPU and memory costs are normalized using the previously mentioned factor. Data generated by each layer during a single FFNN model inference is detailed in the matrix $|\mathcal{L}| \times |\mathcal{L}|$ matrix $\mathbf{B}$. Here, the element $b_{i,j}$ represents the data volume (expressed in bit) traversing connections from layer l_i and terminating to layer l_j. If no connections exist from l_i to l_j, then $b_{i,j}$ is set equal to 0.

To align with contemporary FFNN models, we adopt the subsequent assumption:

Assumption 2.3: For any layer pair l_i and l_j (where $i, j \in [|\mathcal{L}|]$), if $i<j$, then $b_{i,j} \geq 0$. On the other hand, if $i \geq j$, then $b_{i,j}=0$.

III. Proposed Optimization Framework

We introduce integer optimization variables, denoted by $\mathbf{x} = [x_1, \ldots, x_\kappa]$, where κ ranges from 1 to $\hat{\kappa}$. Each variable represents a splitting point in an FFNN model. The maximum number of splitting points is given by $1 \leq \hat{\kappa} \leq |\mathcal{L}|$. The possible values for each optimization variable are defined as:

$$x_t = \begin{cases} \{1, \ldots, (|\mathcal{L}|-1)\} & \text{for } t = 1, \ldots, (\kappa-1) \\ \{|\mathcal{L}|\} & \text{for } t = \kappa. \end{cases} \tag{1}$$

The optimization variables partition the set $\mathcal{L}$ into non-overlapping subsets $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(\hat{\kappa})}$ as:

$$\mathcal{L}^{(t)} = \begin{cases} \{1, 2, \ldots, x_1\} & \text{for } t = 1 \\ \{x_{t-1}+1, x_{t-1}+2, \ldots, x_t\} & \text{for } t = 2, \ldots, \kappa. \end{cases} \tag{2}$$

It is evident that $\bigcup_{t=1}^{\hat{\kappa}} \mathcal{L}^{(t)} \doteq \mathcal{L}$. From relation (2), we deduce that $x_1 < x_2 < \ldots < x_{\hat{\kappa}}$.

Considering Assumption 2.2, for notation simplicity, we assume that FFNN model layers in set $\mathcal{L}^{(t)}$ map onto device d_t for $t = 1, \ldots, \hat{\kappa}$. Thus, $\hat{\kappa}$ equals $\min\{|\mathcal{D}|, |\mathcal{L}|\}$.

The constraints below ensure that the CPU cost of the FFNN model layers in $\mathcal{L}^{(t)}$ does not surpass the CPU capacity of device d_t:

$$\max_{i \in \mathcal{L}^{(t)}} \left\{ c_i^{(\text{CPU})} \right\} \leq r_t^{(\text{CPU})}, \quad \forall t \in [\hat{\kappa}]. \tag{3}$$

Each FFNN model layer's inference task can be divided into sub-tasks for each layer. These sub-tasks run sequentially. Hence, constraint (3) ensures no layer in $\mathcal{L}^{(t)}$ exceeds the CPU capacity of device t.

Similarly, we ensure that the total memory footprint of the FFNN model layers in $\mathcal{L}^{(t)}$ is within the memory capacity of device d_t:

$$\sum_{i \in \mathcal{L}^{(t)}} c_i^{(\text{Mem})} \leq r_t^{(\text{Mem})}, \quad \forall t \in [\hat{\kappa}]. \tag{4}$$

All FFNN model layers on a device are pre-loaded in its memory, irrespective of the inference sub-task status. Thus, unlike (3), device t must have sufficient memory to store layers in $\mathcal{L}^{(t)}$.

Considering a system with three devices ($|\mathcal{D}|=3$) and some FFNN model layers mapped to device d_1 connected to layers on device d_3, due to Assumption 2.1, the network link between device d_1 and device d_2 must accommodate data streams from the first to the second and third devices. Following this logic, we define the objective function:

$$\Psi(\mathbf{x}; \kappa) = \sum_{t=1}^{\kappa-1} \overbrace{\frac{1}{w_{t,t+1}} \sum_{\substack{i \in \bigcup_{h=0}^{t} \mathcal{L}^{(h)}, \\ j \in \bigcup_{u=t+1}^{\kappa} \mathcal{L}^{(u)}}} b_{i,j}}^{\psi^{(x_t)}}, \tag{5}$$

where term $\psi^{(x_t)}$ refers to the total data volume from device d_t to devices $d_{t+1}, \ldots, d_\kappa$, normalized by the network link bandwidth between devices d_t and d_{t+1}. Specifically, $\psi^{(x_t)}$ denotes the time required to transfer the entire data stream generated in a single FFNN inference from device d_t to device d_{t+1}. For convenience, we define $\psi^{(x_\kappa)}=0$. Notably, (5) only accounts for the data stream originating from layer partition d_t and directed to layers in another partition with an index $i \geq t$. This aligns with Assumption 2.3, which restricts an FFNN model ℓ from connecting to layers with indexes less than or equal to ℓ (for $\ell = 1, \ldots, |\mathcal{L}|$). We assume the time to transfer the output of an FFNN inference back to device d_1 is negligible, and thus, it is not considered in our model.

For a specified number of splitting points κ, the Split Computing Optimization (SCO-κ) problem is defined as:

$$\text{SCO-}\kappa: \quad \min_{\mathbf{x}} \ \Psi(\mathbf{x}; \kappa), \tag{6}$$

$$\text{subject to} \quad (3) \text{ and } (4). \tag{7}$$

Denote $\hat{\mathbf{x}}^{(\kappa)}$ as the optimal solution of the SCO-κ problem. Given its system-level implications, we prioritize split computing solutions with the fewest splitting points. Thus, the

Procedure 1 Split Optimization Procedure.

```
1:  procedure SO(κ̂)
2:      for κ = 1, ..., κ̂ do
3:          x_1 ← 0
4:          Initialize: ϵ ← 1, δ ← 0, ℓ ← 1, ν ← true, ψ ← [∞, ..., ∞]
5:          while ℓ ≤ |L| do
6:              if Conditions c_ℓ^(CPU) ≤ r_ϵ^(CPU) and δ + c_ℓ^(Mem) ≤ r_ϵ^(Mem) are met then
7:                  x_ϵ ← x_ϵ + 1
8:                  ψ[ℓ] ← ψ^(x_ϵ)
9:                  δ = δ + c_ℓ^(Mem)
10:                 ℓ ← ℓ + 1
11:             else
12:                 if ϵ + 1 > κ then
13:                     ν ← false
14:                     break
15:                 else
16:                     x_ϵ ← arg min{ψ}
17:                     ϵ ← ϵ + 1
18:                     x_ϵ ← x_{ϵ−1}, ℓ ← x_{ϵ−1}, ψ ← [∞, ..., ∞], δ ← 0
19:         If Solution is valid (ν == true) then return x
20:     return No valid solution found.
```

SCO-κ Solving Task

globally optimum split computing solution $\mathbf{x}^*$ for the SCO problem is:

$$\text{SCO} \quad \underset{\kappa=1,\dots,\hat{\kappa}}{\arg\min} \left\{ \Psi(\hat{\mathbf{x}}^{(\kappa)}; \kappa) \right\}. \tag{8}$$

IV. Proposed Split Optimization Procedure

We introduce the Split Optimization (SO) procedure to heuristically compute a feasible solution for the SCP at (8). The procedure's core steps are outlined in Procedure 1 where each iteration of the for-loop (lines 2-19) attempts to compute a feasible solution for an instance of the SCO-κ problem for $\kappa \in [\hat{\kappa}]$ (*SCO-κ Solving Task*). Upon finding a solution for one of the SCO-κ problems, Procedure 1 deems that solution valid for the SCO problem and returns (line 19).

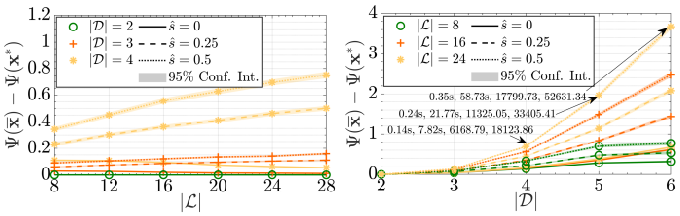
Procedure 1 attempts to solve each instance of the SCO-κ problem in the innermost while-loop (lines 5-18). In particular, this part of the procedure begins by setting the first splitting point in an FFNN model equal to the first layer ($x_1 = 1$). The value of x_1 is progressively incremented for as long as constraints (3) and (4) are met (lines 7-10). When any or both the aforementioned constraints are about to be violated, x_1 is set equal to the FFNN layer index minimizing the term $\psi^{(x_t)}$, for $t = 1, \dots, \kappa$ (line 16). Then, the procedure repeats by considering the next splitting point (x_2), which can only take values greater than x_1 (lines 18 and 7). The while-loop terminates as soon as all the FFNN model layers have been considered (namely, $\ell > |L|$) or the index ϵ if the currently considered splitting point exceeds κ. It is immediate to observe that the while-loop iterates for no more than $|L| + \kappa - 1$ times.

The following lemma demonstrates that Procedure 1 yields feasible solutions for the SCO-κ problem.

Lemma 4.1: Given $\kappa \in [\hat{\kappa}]$, if the while-loop in lines 5-18 of Procedure 1 concludes due to the violation of the loop condition at line 5, then the set $\{x_1, \dots, x_\kappa\}$ is a feasible solution for the corresponding SCO-κ problem.

Proof: The proof is derived from the fact that the while-loop in lines 5-18 of Procedure 1 minimizes the terms $\psi^{(x_t)}$ for $t = 1, \dots, \kappa - 1$. During this minimization, line 6 ensures constraints (3) and (4) are met. ∎

From Lemma 4.1, it is straightforward to prove the following theorem.



(a) Average cost difference vs. $|\mathcal{L}|$ (b) Average cost difference vs. $|\mathcal{D}|$

Fig. 1. Normalized average cost difference as a function of the total number of FFNN model layers and devices; 95% confidence intervals are also included. Each annotated point lists the average SCIP and SO procedure completion times (measured on a workstation equipped with a CPU AMD 5995WX with cores operated at 1.8 GHz), the average number of optimization variables and constraints of the linearized version of the SCO problem.

Theorem 4.1: Let $\overline{\mathbf{x}} = \{x_1, \dots, x_{\overline{\kappa}}\}$ be a solution returned by Procedure 1, where $\overline{\kappa} \leq \hat{\kappa}$. Then, $\overline{\mathbf{x}}$ is a feasible solution for the SCO problem.

V. Analytical Results

In this section, we refer to an equivalent linear formulation of the SCO-κ problem that can be obtained by adopting, after some manipulations, the transformations presented in [7, Eqs. (4)-(7), (11)-(14), (78)-(81)]. In doing so, the transformed SCO-κ problem is an integer linear problem equivalent to the formulation as per (6)-(7).

We optimally solved the linearized SCO problem by calculating the optimum solution of the SCO-κ problem, for $\kappa \in [\hat{\kappa}]$ by means of the Solving Constraint Integer Programs (SCIP) solver [8]. This makes it straightforward to establish $\mathbf{x}^*$ by directly solving (8). We remark that $\Psi(\mathbf{x}^*)$ and $\Psi(\overline{\mathbf{x}})$ are the values of the objective function (5) when the SCO problem is optimally solved and when the solution is obtained via the proposed SO procedure, respectively. Since $\Psi(\overline{\mathbf{x}}) \geq \Psi(\mathbf{x}^*)$, Fig. 1 shows how much a heuristic solution $\overline{\mathbf{x}}$ deviates from the corresponding optimal solution $\mathbf{x}^*$ by showing the *cost difference* $\Psi(\overline{\mathbf{x}}) - \Psi(\mathbf{x}^*)$ as a function of the number of an FFNN model's layers $|\mathcal{L}|$ and available devices $|\mathcal{D}|$.

To effectively investigate the performance of the proposed SO procedure, in Fig. 1, we considered FFNN models with a number of layers $|\mathcal{L}| \in [8, 28]$. Given an FFNN model, each layer ℓ_i is associated with a normalized CPU footprint $c_i^{(\text{CPU})}$ equal to 1, for $i \in [|\mathcal{L}|]$ – we regard $\overline{C}$ as the CPU normalization factor that is set equal to the largest CPU capacity among all the devices in $\mathcal{D}$. The normalized memory footprint of the aforementioned FFNN model's layer $c_i^{(\text{MEM})}$ is chosen uniformly distributed at random in $(0.01, 1.0]$ – we set the memory normalization factor $\overline{M}$ equal to the largest memory capacity among all the devices in $\mathcal{D}$. Matrix $\mathbf{B}$ is a direct function of the FFNN model's layer memory footprint. In particular, the number of bits $b_{i,j}$ to be transferred from layer ℓ_i to layer ℓ_j, for $1 \leq i < j \leq |\mathcal{L}|$, is set equal to $c_i^{(\text{MEM})}$. For $j \geq i+1$ and $j < |\mathcal{L}|$, we considered a probability $\hat{s} \in \{0, 0.25, 0.5\}$ of $b_{i,j}$ being non-zero. For a given number of devices and the FFNN model's layers, we considered $\overline{I} = 10^4$ Monte Carlo iterations. These simulation settings consist of a variable number of devices $|\mathcal{D}| \in [2, 6]$ each offering a normalized CPU capacity $r_i^{(\text{CPU})}$ equal to 1, and a normalized memory capacity $r_i^{(\text{MEM})} = (\sum_{i \in [\mathcal{L}]} c_i^{(\text{MEM})})/(|\mathcal{D}| - i + 1)$. For instance, if $|\mathcal{D}| = 2$ then d_2 is associated with $r_2^{(\text{MEM})}$ equal to the sum of the FFNN model's layer memory footprints,

TABLE I
MAIN SIMULATION PARAMETERS

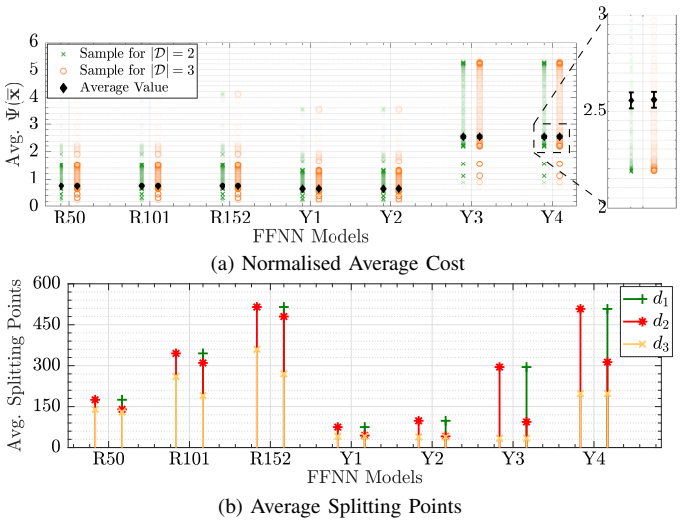
	Name	$\lvert\mathcal{L}\rvert$	Trainable Parameters $\times \cdot 10^{-7}$
FFNN Model	R50, R101, R152	177, 347, 517	2.56, 4.46, 6.03
	Y1, Y2, Y3, Y4	77, 100, 297, 510	5.99, 5.08, 6.15, 6.44

Communication Network	Deployment type	NR TDD, TR 38.901 channel model [9]
	UE to gNB distance	700 m with stationary UE
	UE and gNB antenna hight	1.5 m, 10 m
	Carrier Components ([Center Frequency, Bandwidth])	[28 GHz, 400 MHz], [29 GHz, 100 MHz]
	Network connecting the gNB to the computing node	10Gbps two-hop wired connection
	Transport protocol among FFNN model's layers	UDP

and d_1 is associated with $r_1^{(\text{MEM})}$ that is only half of $r_2^{(\text{MEM})}$. The normalized bandwidth $w_{t,u}$ of the wireless/wired link interconnecting devices d_t and d_u, for $1 \leq t < u \leq |\mathcal{D}|$, is set equal to $1/(|\mathcal{D}|-1)$ – the bandwidth normalization factor $\overline{B}$ is set equal to largest bandwidth value among all the links. These modeling assumptions capture edge devices' reduced memory capacities and communication bandwidths.

Fig. 1a shows the average cost difference (averaged across all the randomly generated instances of $(\mathcal{L}, \mathcal{C}, \mathbf{B})$) as a function of the number of FFNN models' layers. For a number of devices $|\mathcal{D}| = 2$, we observe that the average cost difference is equal to zero, regardless of the considered number of FFNN model's layers and the value of $\hat{s}$ – thus, signifying the proposed SO procedure returns a solution associated with the same cost of the solution returned by SCIP while solving the linearized version of the SCO problem. As the number of devices increases to $|\mathcal{D}| = 3$, the average cost difference tends to remain stable for a given value of $\hat{s}$. Overall, as the value of $\hat{s}$ increases, the average cost difference also tends to increase. For instance, this can be observed for $\hat{s} = 0.5$, when $|\mathcal{L}|$ increases from 8 to 28, the average cost difference increases from 0.089 to 0.158. At the same time, for $|\mathcal{L}| = 28$, as the value of $\hat{s}$ increases from 0 to 0.5, the average cost difference increases from 0.011 to 0.158. Overall, if $|\mathcal{D}| = 3$, the value of $\hat{s}$ impacts the average cost difference more than the total number of FFNN model layers. This is not surprising as, for a relatively small number of devices in the system model, the structure of matrix $\mathbf{B}$ determines the complexity of the SCO problem. However, as soon as the number of devices increases to 4, both the number of FFNN model layers and the value of $\hat{s}$ have an impact on the overall complexity of the SCO problem, which results in average cost differences that increases as the values of $|\mathcal{L}|$ and/or $\hat{s}$ increases. If we consider the case where $\hat{s}$ is set to its largest value (0.5), the average cost difference moves from 0.343 (for $|\mathcal{L}| = 8$) to 0.751 (for $|\mathcal{L}| = 28$).

Fig. 1b shows the average cost difference as a function of the number of devices $|\mathcal{D}|$, in settings that are equivalent to the ones considered in Fig. 1a. As an extension of the considerations made above, we observe that if the number of the FFNN model's layers is more than twice the number of available devices, the average cost difference increases with both $\mathcal{D}$ and $\hat{s}$. On the other hand, when the value of $|\mathcal{D}|$ approaches $|\mathcal{L}|$, the average cost difference plateaus. Once more, this was expected as, for a given value of $\hat{s}$, the more the number of devices approaches the number of FFNN



(a) Normalised Average Cost

(b) Average Splitting Points

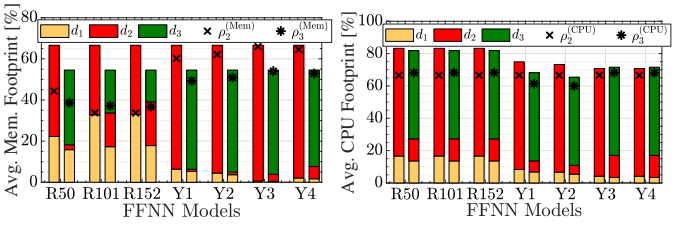
Fig. 2. Normalized average cost and splitting points for three ResNet architectures (R50, R101, and R152) and four YOLO architectures (Y1, Y2, Y3, and Y4), for $|\mathcal{D}| = \{2, 3\}$. Fig. 2a includes the 95% confidence intervals that are superimposed to the average values and scatter plot of the cost difference data points calculated at the end of each Monte Carlo iteration.

model's layers, the fewer the number of ways the model's layers can be mapped onto the available devices. In turn, this simplifies the SCO problem. We remark that Fig. 1 shows average performance gaps as a measure of time normalized by the term $\overline{B}/\overline{M}$. If we converted the aforementioned average performance gaps to seconds, we would need to multiply each data point by $\overline{M}/\overline{B}$, which is likely significantly smaller than one for practical scenarios.

For $|\mathcal{L}| = 24$ and $|\mathcal{D}| = \{4, 5, 6\}$, Fig. 1b also highlights the average time the proposed SO procedure and the SCIP solver need to solve the (linearized) version of the SCO problem, along with the average number of optimization variables and constraints forming the instances of the linearized version of the SCO problem. When $|\mathcal{D}|$ increases from 4 to 6, we observe that the average number of optimization variables (constraints) increases by a factor of 2.88 (2.9). Yet, the average time the SCIP solver (the proposed SO procedure) needs to solve the problem increases from 7.82 s to 58.73 s (from 0.14 s to 0.35 s) – thus, reinforcing the need of formulating a heuristic strategy.

In the remainder of this section, we will benchmark the performance of the proposed SO procedure across a selection of widely known FFNN models summarized in Tab. I. In particular, we considered three Residual Neural Network (ResNet) architectures (hereafter referred to as R50, R101, and R152) [10] and versions 1-4 of You Only Look Once (YOLO) (hereafter referred to as Y1, Y2, Y3, and Y4) [10]. To establish a consistent cost model across the considered FFNN models, we set each model's layer CPU and memory footprint equal to the number of trainable parameters of each layer. CPU and memory normalization factors $\overline{C}$ and $\overline{M}$ have been calculated as in the case of Fig. 1.

Figs. 2 and 3 consider a system model consisting of a stationary UE that is served by a gNodeB (gNB) by means of a 5G NR RAN. The gNB is connected to a 4G Evolved Packet Core (EPC) core network. In this setting, the FFNN model inference task is initiated by the UE, and it may be executed in the gNB and in a dedicated computing node that is ideally part of the EPC. As such, for the purpose of our performance model, the devices that can run (partially or in



(a) Average Memory Footprint (b) Average CPU Footprint

Fig. 3. Average memory and CPU footprints and UE's footprint reductions associated with the numerical results shown in Fig. 2.

full) an inference in a given FFNN model can either be the UE and gNB ($|\mathcal{D}| = 2$), or the UE, gNB and a computing node part of the EPC ($|\mathcal{D}| = 3$). To account for the variability of the propagation conditions between the UE and gNB, we set $\overline{I} = 10^4$. Key simulation details are summarized in Tab. I.

Fig. 2a compares the (normalized) average cost of the considered ResNet and YOLO FFNN models. In the case of the ResNet FFNN models, the proposed SO procedure ensures a similar average cost among the models, regardless of the considered number of devices and in spite of the considerable difference in terms of the number of FFNN model's layers – R50 consists of 175 layers while R152 is associated with 515 layers (see Tab. I). This signifies that the SO procedure can efficiently establish splitting points aiming at minimizing the overall cost (subject to meeting the capacity constraints). In the case of the YOLO FFNN models, we observe that Y1 and Y2 are relatively similar in their FFNN model structure, which leads to similar average costs. On the other hand, the number of the model's layers and the density of interconnections among layers associated with Y3 and Y4 are higher than in the case of Y1 and Y2 (see Tab. I). This results in larger average costs for the Y3 and Y3 models. Yet, despite Y4 consisting of nearly twice as many layers as Y3, the proposed SO procedure returns splitting solutions with similar cost.

Fig. 2b shows the splitting points solutions $\overline{\mathbf{x}}$ calculated with the proposed SO procedure. In particular, for each of the considered FFNN models, the figure shows the average splitting points solution defined as $(\sum_{i=1}^{\overline{I}} \overline{\mathbf{x}}_{[i]})/\overline{I}$, where $\overline{\mathbf{x}}_{[i]}$ signifies a solution returned by the SO procedure at the end of the i-th Monte Carlo iteration. In the case of the considered ResNet models, the proposed SO procedure returns splitting solutions where most of the FFNN model's layers are mapped onto the UE (case $|\mathcal{D}| = 2$) or the UE and the gNB (case $|\mathcal{D}| = 3$). However, as the complexity of the FFNN model increases, the SO procedure establishes solutions where, on average, most of the model's layers are mapped onto the gNB and/or a dedicated computing node in the EPC. For instance, this is the case of Y4 where only 38.9% of the layers are mapped onto the UE and the remaining onto the gNB and the dedicated computing node in the EPC, for $|\mathcal{D}| = 2$ and 3.

Fig. 3 compares the average memory and CPU footprints associated with results shown in Fig. 2. In particular, with regard to the assumption made at the beginning of this section, if $|\mathcal{D}|$ is equal to 2 (3), the UE is associated with 33.3% (18.16%) of the overall memory and CPU capacity across the devices. In contrast, the gNB is associated with 66.6% (27.27%), while the dedicated computing node in the EPC is associated with 54.55% (when $|\mathcal{D}| = 3$). With regards to Fig. 3a, in the case of the ResNet models, regardless of the value of $|\mathcal{D}|$, the SO procedure tends to favor solutions associated with a larger footprint on the UE. However, as the complexity of the FFNN model increases, the SO procedure tends to return splitting solutions associated with a reduced memory footprint on the UE and/or the gNB. This is particularly evident in the case of the YOLO FFNN models – in the case of Y4 and $|\mathcal{D}| = 3$ only 1.7% and 5.9% of the average memory footprint of the returned solution is associated with the UE and gNB, respectively. Similar considerations apply to the average CPU footprint shown in Fig. 3b. Fig. 3 also shows the average UE's memory and CPU footprint reduction ($\rho^{(\text{Mem})}_{|\mathcal{D}|}$ and $\rho^{(\text{CPU})}_{|\mathcal{D}|}$, respectively) defined as the average reduction of the memory/CPU footprint experienced by the UE in the case the UE had enough memory and CPU capacity to run FFNN model inferences on its own. In particular, we observe that the SO procedure ensures an average UE's memory (CPU) footprint reduction of at least 33.6% or 36.6% (66.6% or 60%) for $|\mathcal{D}|$ equal to 2 or 3, respectively.

VI. Conclusions

We developed a novel optimization framework for SC applications that divides the FFNN model into sub-sets and executes them across a variable number of devices, starting from a UE and progressively considering devices further from the edge while minimizing end-to-end communication delay from wireless/wired network links. We also proposed an efficient heuristic approach to solve the formulated optimization problem. The deviation between solutions calculated with the proposed heuristic strategy and a branch-and-cut approach is marginal in the considered settings. Our numerical results considered 5G cellular network set-ups with a number of devices with heterogeneous computational capacities. We also validated the proposed heuristic procedure on multiple state-of-the-art FFNN models with a heterogeneous number of layers, trainable parameters, and model's layer interconnections. Our numerical results show that the proposed heuristic procedure establishes valid splitting points while fulfilling device-specific computational capacity constraints.

References

[1] Hexa-X, "Final 6G Architectural Enablers and Technological Solutions," H2020-ICT-2020-2, Deliverable D5.3, Apr. 2023.
[2] M. Tan and CVPR, "Efficientdet: Scalable and Efficient Object Detection," in *Proc. of IEEE/CVF CVPR*, 2020.
[3] X. Wang *et al.*, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2020.
[4] T. Liang *et al.*, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.
[5] A. Bakhtiarnia *et al.*, "Dynamic Split Computing for Efficient Deep Edge Intelligence," in *Proc. of IEEE ICASSP*, 2023.
[6] Y. Matsubara *et al.*, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.
[7] M. Asghari *et al.*, "Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey," *Mathematics*, vol. 10-2, 2022.
[8] K. Bestuzheva *et al.*, "The SCIP Optimization Suite 8.0," Zuse Institute Berlin, ZIB-Report 21-41, December 2021. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:0297-zib-85309
[9] 3GPP, "Study on Channel Model for Frequencies from 0.5 to 100 GHz," 3rd Generation Partnership Project (3GPP), TR 38.901, Mar. 2022.
[10] T. Diwan *et al.*, "Object Detection Using YOLO: Challenges, Architectural Successors, Datasets and Applications," *Multimed. Tools Appl.*, p. 9243–9275, 2022.