# Learning-Enabled CPS for Edge-Cloud Computing

Luigi Capogrosso*, Shengjie Xu†, Enrico Fraccaroli*†, Marco Cristani*, Franco Fummi*, Samarjit Chakraborty†
*University of Verona, Verona, Italy †University of North Carolina at Chapel Hill, USA
Email:*{luigi.capogrosso, enrico.fraccaroli, marco.cristani, franco.fummi}@univr.it, †{sxunique, samarjit}@cs.unc.edu

***Abstract*—Many Cyber-Physical System (CPS), such as autonomous vehicles and robots, rely on compute intensive Machine Learning (ML) algorithms, especially for perception processing. A growing trend is to implement such ML algorithms in the cloud. However, the data transfer overhead and the delay introduced in the process necessitate some form of *edge-cloud* solution. Here, a part of the processing is done locally and the rest on the cloud, and how to do this partitioning is being explored in the body of work referred to as Split Computing (SC). In this position paper, we explore different SC architectures and discuss their implications on controller design for CPS. In particular, we discuss the delay and state estimation accuracy of these different SC architectures and how they would impact the design of the feedback controllers using them.**

***Index Terms*—Split Computing, Early Exit, Deep Neural Networks, Cyber-Physical Systems, Edge Devices.**

## I. Introduction

Any Cyber-Physical System (CPS), by definition, involves a tight interaction between physical systems and software running on an embedded computing platform [1]. The software in such systems typically involves a feedback controller that (i) senses the state of the system, (ii) estimates the value of the system state using the sensed information, (iii) computes the control input using the estimated value, and finally (iv) actuates the system using the control input. The steps (i) – (iv) are carried out in an infinite loop to impose a desired behavior on the overall system (the physical system and the controller). In other words, the goal is to ensure that the system's state evolves in a particular manner, *e.g.*, it follows a desired trajectory in the state space, or avoids some states/region of the state space, or stays within a certain region of the state space [2], [3].

As a concrete example of such a CPS, let us consider an autonomous vehicle that should follow a specified trajectory and not collide with vehicles in front of it. Using sensors such as cameras or lidar, the vehicle senses its environment. The output from such sensors – such as images from the camera or a point cloud from the lidar – is fed into a neural network that, *e.g.*, estimates the distance between the vehicle and the one in its front. The control algorithm uses this estimated distance to adjust the vehicle's acceleration to maintain a safe distance between the two vehicles. Similarly, the output from the neural network could be an estimate of the distance between the center of the vehicle and two-lane boundaries (left and right), with the goal of the controller being to keep the vehicle in the center of the lane. While the neural network for such applications can reside on the computational platform on the vehicle, for more complex

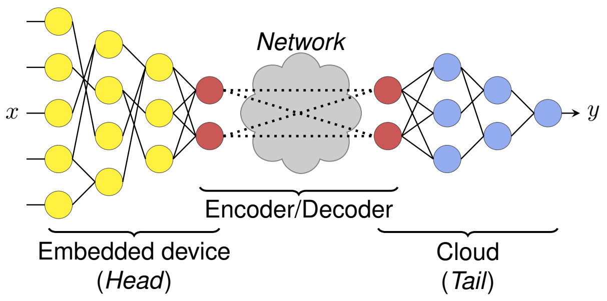


Figure 1: SC architecture – Head & Tail components.

autonomous features, the computational bandwidth available on the vehicle might not be sufficient. In such cases, a Deep Neural Network (DNN) is implemented on a cloud computing platform. While this enables implementing powerful DNNs, they are also associated with significant delays that impact control performance, *i.e.*, the dynamics or behavior of the system. A middle ground is to use Split Computing (SC), as shown in Figure 1. Here, a DNN is partitioned into "Head" and "Tail" components, with the former being implemented locally on an embedded computing platform and the latter on the cloud. While such DNN [4] architectures have been studied in the past, how they should be designed when used in conjunction with feedback controllers in a CPS has not been investigated.

It is evident that the partitioning between the head and the tail of a DNN in SC will determine the delay as well as the classification or estimation accuracy of the DNN. However, it remains unclear how these partitions should be "connected" to the performance of a feedback controller. Additionally, the impact of different types of connections on the resulting dynamics of the closed-loop system is not well understood. In this position paper, we present a few of these connections between SC architectures and feedback controllers and discuss their implications on the behavior of the closed-loop dynamics of the system. We believe that this discussion will trigger useful research and open up a potentially new field – "*Split Computing CPS*" – that has not been explored until now.

## II. Overview of split computing and early exit

We start by introducing different distributed deep learning application architectures. We focus on architectures operating through a DNN model $\mathcal{M}(\cdot)$, whose task is to produce the inference output $y$ from an input $x$. We can identify four major types of architectures used for distributed deep learning applications in the literature: Local-only Computing (LoC), Remote-only Computing (RoC), SC, and Early Exit (EE). Their structures are shown in Figure 2.

### A. Local-only computing

Under this policy, the entire computation is performed on the edge device. As shown in Figure 2a, the edge device entirely executes the inference function $\mathcal{M}(x)$. Its advantage lies in offering low latency due to the proximity of the computing element to the sensor/controller in our setup [5]. However, it may not be compatible with DNN-based architectures that demand robust hardware capabilities. Usually, simpler DNN models $\bar{\mathcal{M}}(x)$ that use specific architectures (*e.g.*, depth-wise separable convolutions) are used to build lightweight networks, such as MobileNetV3 [6]. Besides designing lightweight neural models, in the last few years, significant progress has been made in the area of DNN compression. Compression techniques, such as network pruning and quantization [7], or knowledge distillation [8] achieve a more efficient representation of one or more layers of the neural network, but with a possible quality degradation.

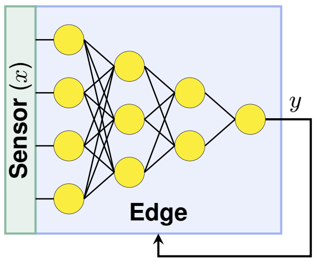
### B. Remote-only computing

As shown in Figure 2b, here the input $x$ is transferred from the edge device through a communication network and then is processed remotely through the function $\mathcal{M}(x)$. This architecture preserves full accuracy considering the higher power budget of the remote system, but it leads to high latency and bandwidth consumption due to the input transfer.

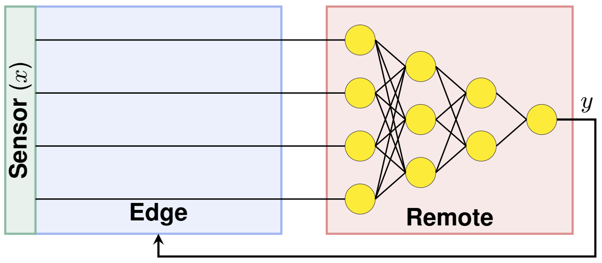
### C. Split computing

A typical SC scenario is discussed in [9], where it is shown that neither LoC nor RoC approaches are optimal, and a split configuration performs better. The general structure of SC is shown in Figure 2c, which shows how the SC paradigm divides the DNN model into a head, executed by the edge device, and a tail that is executed by the remote system. It combines the advantages of both LoC and RoC, thanks to the lower latency and, more importantly, reduced transmission bandwidth requirements. Such reduction may also be obtained by compressing an input $x$ to be sent, through the use of an autoencoder [10]. We define the encoder and decoder models as $z_l = \mathcal{F}(x)$ and $\bar{x} = \mathcal{G}(z_l)$, which are executed at the edge, and remotely, respectively. The distance $d(x, \bar{x})$ defines the performance of the encoding-decoding process. One of the earliest works on SC [11] show that the initial layers of a DNN are the most suitable candidates for partitioning, as they optimize both latency and energy consumption. Additionally, latency reduction is usually achieved through quantization, as explored in [12], and by using lossy compression techniques before data transmission, as investigated in [13]. In addition to lossy compression techniques, lossless techniques to encode intermediate results without modifying the machine learning model have also been studied [14]. Finally, the concept of employing autoencoders to compress the data further to be transferred is discussed in [15].

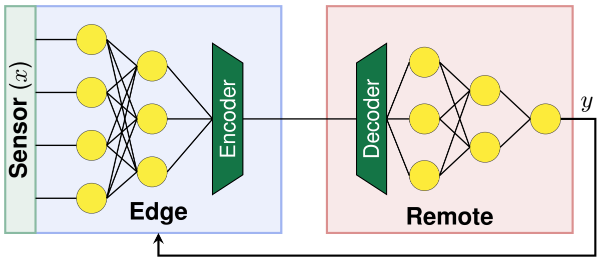
The prevalent methods for identifying potential splitting points have evolved from architecture-based techniques to more refined neuron-based methods. Within the domain of architecture-based approaches, candidate split locations can



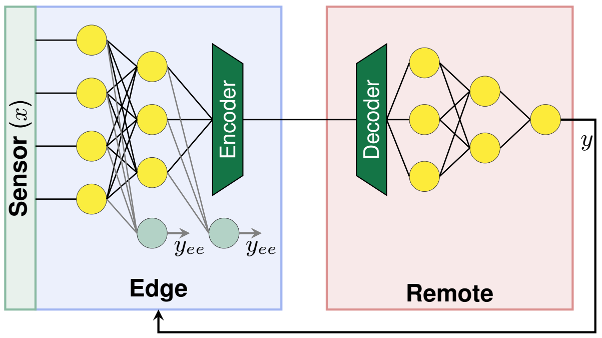
(a) With LoC, the edge device entirely executes the model inference.



(b) With RoC, the input is transferred to the remote device, where it is processed, and the result is then sent back to the edge device.



(c) With SC, the computation is split between the edge and the remote devices. The transmission bandwidth is reduced by compressing it using encoder and decoder models.



(d) When combining SC and EE, the computation is split between the edge and the remote devices; however, we have intermediate classification branches, producing an estimate $y_{ee}$ of the desired output $y$.

Figure 2: Comparing the four major types of architectures used for distributed deep learning applications, *i.e.*, LoC, RoC SC, and EE.

be where the size of the DNN layers decreases [16]: the rationale is that compressing information by autoencoders, where compression would still occur due to the shrinking of the architecture, seems reasonable. Instead, I-SPLIT, demonstrates that the architecture of the layers and the saliency of individual layers is a crucial factor [17]. Specifically, a neuron's saliency is determined by its gradient in relation to the accurate decision. Along similar lines, [18] introduces Split-Et-Impera, a fast and user-friendly framework that eases the design of a distributed architecture executing one or more DNNs. Split-Et-Impera not only accurately mimics diverse communication protocols and application requirements, but also offers a unique capability. It suggests the proper configuration to match the application's Quality of Service (QoS) requirements, ensuring optimal performance in terms of accuracy and latency. Since manipulating diverse SC configurations may require days of computation, Split-Et-Impera allows the elimination of several configurations through communication-aware simulations.

At the same time, current state-of-the-art approaches in different deep learning applications rely on advanced learning procedures, such as Multi-Task Learning (MTL). In particular, MTL is a paradigm in which multiple related tasks are jointly learned to improve the general applicability of a model by using shared knowledge across different aspects of the input. In [19], a method to partition multi-tasking DNNs for deployment within an SC framework is discussed. The proposed MTL-Split design handles multiple tasks concurrently, shifting the focus from Single-Task Learning (STL) in SC, and through MTL, increases task performance, overcoming the challenge of preserving only the performance of the main task. Moreover, in [20], the effect of predefined sparsity within the SC paradigm is presented. This approach, demonstrably practical for the first time in an SC scenario, significantly reduces computational, storage, and energy demands during training and inference, regardless of the hardware platform.

### D. Split computing and early exit

This scenario adds an EE branch to a standard SC architecture, as depicted in Figure 2d. Formally, we can define $B_i,\ i = 1 \ldots N$ (with $N = L$, and $L$ is the number of layers of the DNN) as the branch model that takes as input $z_l$ and produces an estimate of the desired output $y$. In practice, the EE architecture modifies an existing neural network by adding one or more classification branches, where the confidence of the intermediate result is checked before the computation of all network layers. If the confidence is sufficient, the intermediate result is considered the final output [21].

The EE architecture can be leveraged in distributed deep learning applications, where the intermediate result can either be directly transmitted, as in local computing, or further refined on the remote side, as in SC. In this scenario, the level of transmission traffic depends on the input, thus varying stochastically. Therefore, the interdependencies between computation and communication cannot be analytically modeled, and real experiments are needed to validate a given implementation.

## III. Basics of Controller design

When neural networks are used for sensor data processing before the data is fed into a controller in a CPS, the overall system performance is impacted not only by the latency and accuracy of the neural network, but also the underlying dynamics of the physical system. This section introduces the basics of feedback control systems used in the rest of the paper. One common representation of control systems is the state-space model, where the state of the system is represented by a state vector $x(t) \in \mathbf{R}^p$ and the input to the system by $u(t) \in \mathbf{R}^q$. For simplicity, we discuss Linear Time-Invariant (LTI) control systems here, but the principles discussed in this work apply to any type of control system with learning-enabled components.

The state-space model of a continuous LTI system is:

$$\dot{x}(t) = A_c x(t) + B_c u(t)\ , \tag{1}$$

where $A_c \in \mathbf{R}^{p\times p}$, and $B_c \in \mathbf{R}^{p\times q}$ are matrices encoding the system's dynamics. Equation (1) shows that the rate of change of the system state $\dot{x}(t)$ depends both on the current state $x(t)$ and the control input $u(t)$. To enable feedback control, the control input $u(t)$ is computed by a periodic real-time task running on a processor. Computing $u(t)$ requires discretizing the continuous state-space model with a constant sampling period $h$. Assuming periodic sampling, *i.e.*, $t_{k+1} - t_k = h$, matrices $A$ and $B$ can be derived from Equation (1) such that:

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k)\ .$$

For simplicity, we denote $x(t_k)$ as $x[k]$ and $u(t_k)$ as $u[k]$ to obtain the discrete state-space model:

$$x[k+1] = Ax[k] + Bu[k]\ . \tag{2}$$

In the simplest case, the control input $u[k]$ is computed by:

$$u[k] = Kx[k]\ , \tag{3}$$

where $K \in \mathbf{R}^{q\times p}$ is the feedback gain. Many methods exist to design the feedback gain $K$ with various stability, energy, and complexity considerations.

## IV. Cyber-physical systems safety

Multiple issues mentioned in Section III impact the safety of a CPS. For example, the choice of the sampling period $h$ affects the response time of the control system to external disturbances. A sampling period that is too large leads to unstable systems, while a period that is too small induces unnecessary load on the processor and may cause other control tasks utilizing the same processor to become unsafe. Furthermore, the system state $x[k]$ in Equation (3) is usually unknown and requires some form of sensing and estimation, *i.e.*, the control input $u[k]$ is computed by:

$$u[k] = K\hat{x}[k]\ , \tag{4}$$

where $\hat{x}[k]$ is an estimation of the ground truth state $x[k]$. While some sensors used for state estimation can be fairly accurate, *e.g.*, speed and temperature sensors, other sensing methods, especially ones involving neural networks, can have non-negligible uncertainties in their estimation. Changes in the sampling period and errors in state estimation can both lead to deviation from the intended behavior of a system, potentially violating its safety requirements.

Here, we introduce two quantitative measures of CPS safety over a finite time horizon $H$. First, assuming a nominal (ideal) behavior of the system $x_{nom}$ (*e.g.* defined by the trajectory of the system in its state space), the safety of the system can be expressed in terms of the maximum deviation D from that ideal behavior:

$$\mathtt{D} = \max_{k \in [0,H]} d(x[k],\ x_{nom}[k]) \ , \tag{5}$$

for any defined distance metric $d$ (such as the Euclidean distance) between two points in the state space. A smaller deviation $D$ from the nominal behavior implies a safer system. Second, if the nominal behavior is not known or not applicable, we measure the safety of the system in terms of the maximum diameter of reachable states over the time horizon $H$. Starting from an initial set of states $X[0]$, the reachable sets can be calculated by extending Equations (2) and (4):

$$X[k+1] = \bigcup_{\chi \in X[k]} A\chi \oplus BK\hat{\chi}, \tag{6}$$

where $\hat{\chi}$ denotes the set of possible estimations of $\chi$, and $\oplus$ denotes the Minkowski sum of two sets. The maximum diameter Diam can then be calculated as:

$$\mathtt{Diam} = \max_{k \in [0,H]} \left( \max_{x,y \in X[k]} d(x,y) \right) \ . \tag{7}$$

Similar to the safety measure with maximum deviation, a smaller maximum diameter of reachable sets implies a safer system. Given different SC architectures, our aim is to use such safety measures to evaluate their suitability in a CPS. In addition, we propose to use the above safety metrics to drive suitable splitting decisions in DNNs between edge and cloud computing resources.

## V. Splitting decisions and illustrative results

This section shows how the split point is usually determined and which factors impact these decisions, providing experimental results based on the current state-of-the-art methods. As an example, we focus on the image classification task, and as DNN, we use the PyTorch implementation of the VGG16 [22]. We train our model on the CIFAR-10 dataset [23] up to 20 epochs with a learning rate of $5 \times 10^{-3}$, using Adam [24] as an optimizer. CIFAR-10 has to be considered as a placeholder for more extensive datasets (*e.g.*, ImageNet [25]); nonetheless, the focus here is to discuss how different SC options will influence control performance in a CPS and not beat the state-of-the-art in a specific computer vision challenge.

All experiments have realized the split point by placing an autoencoder with a 50% compression rate. For the training of the encoder/decoder, we run up 50 epochs with a learning rate of $5 \times 10^{-4}$, always using Adam as the optimizer. In the experiments in which we evaluated the communication network aspects, we always simulated a 1 GB/s Full-Duplex network channel.

### *A. Model-based split point selection*

Figure 3 shows the results following both, the architecture-based [16] and the neuron-based split point search. In this figure, the so called Cumulative Saliency CS [18] is a function of the layer compared with the accuracy of the DNN split in that layer [17]. The peaks in the CS curve correspond to the points where accuracy is preserved despite split injection. As such, the layers in which CS has a local maximum are the best candidates for splitting. Layers marked with an asterisk (*) represent VGG16 max-pooling layers, *i.e.*, down-sampling layers, which reduce the spatial dimensions of the input data.

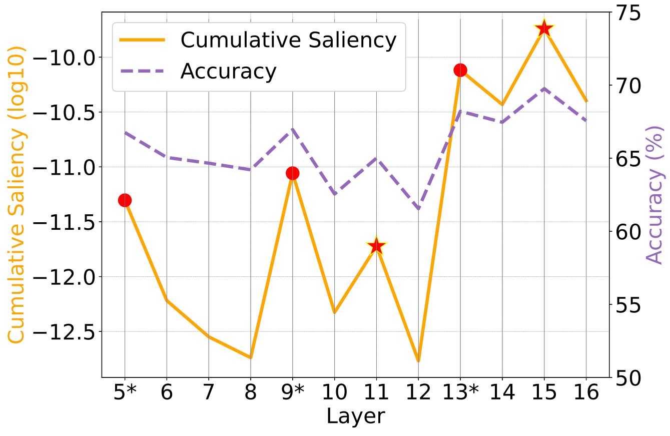


Figure 3: Cumulative Saliency (CS) as a function of the layer compared with the accuracy of the DNN split in that layer [18].

In Figure 3, the architecture-based approach identifies candidate split points (red dots) layers 5, 9, and 13 (dense data), corresponding to block2_pool, block3_pool and block4_pool. Instead, the neuron-based approach also identifies two additional points (red stars) at layers 11 and 15 (informative data), corresponding to block4_conv2 and block5_conv2, respectively. It is worth noting how layers with the same dimensionality, *i.e.*, convolutional layers belonging to the same VGG16 block, do not express the same importance, as shown by the CS curve. Given the model-based split point selection results, due to the lack of space, in the next section, we present the communication network-based results only after splitting the DNN at layers 11 and 15.

### *B. Communication network-based split point selection*

Figure 4 shows the results on the impact of the communication network on the split point selection, using the simulation framework for SC and EE in [18]. In this experiment, we assumed that we have a real-time application with a constraint on the maximum frame latency of 0.05 seconds (*i.e.*, runs at 20 FPS). Figure 4 highlights how the latency increases with the packet loss rate due to TCP re-transmission in case of packet loss. However, this preserves the maximum accuracy of the application. Specifically, the dashed curve shows that splitting at layer 15, the application requirements are always satisfied independently of the packet loss rate. The dotted curve shows that with the split at layers 11, the 20 FPS constraint cannot be satisfied when the packet loss rate is more than 3%. This behavior meets expectation, *i.e.*, by splitting the network at layer 11, the amount of transmitted data is more significant than the one obtained by splitting the network at layer 15, and because of the retransmissions, the latency increases, up to a point that violates the application constraints represented by the dashed red line.

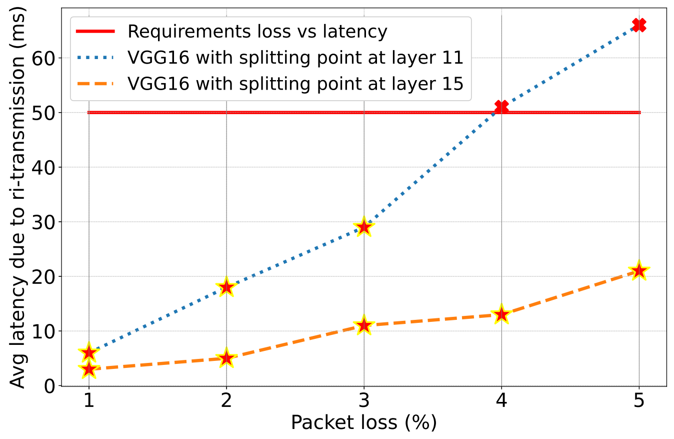
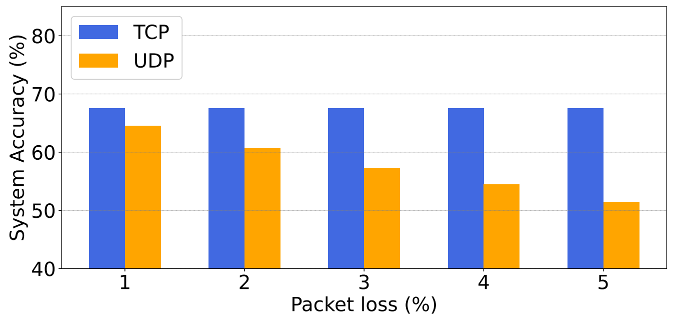
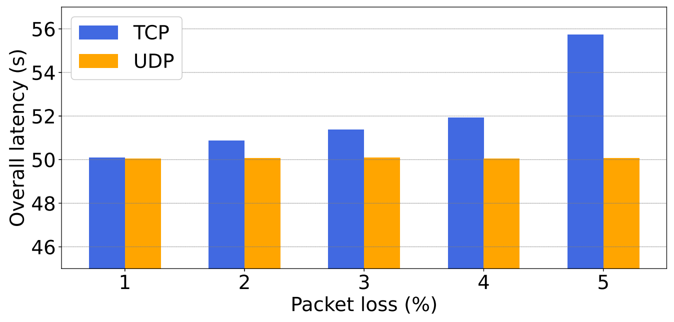


Figure 4: Evaluation of the impact of the communication network on split point selection [18].



(a) Accuracy comparison between TCP and UDP.



(b) Latency comparison between TCP and UDP.

Figure 5: Comparison between the TCP and the UDP protocols [18].

## C. *Latency-based communication network protocol selection*

Based on the experiments in [18], in Figure 5, we show a comparison between system accuracy and the overall latency using the TCP and UDP protocols. Figure 5a shows that application accuracy does not depend on the packet loss rate when using TCP. Figure 5b shows that this, however, feature comes at a price: with TCP, the overall latency is much greater, so it is required to ensure that this is compatible with the application requirements. UDP protocol shows a dual behavior: the latency is minimized and kept independent of the packet loss rate, but the accuracy decreases in case of loss since no error checking and recovery services are provided. Finally, Figure 6 shows the bandwidth utilization comparing the SC (blue curve) and EE scenarios (gray curve), both using the TCP protocol. The curves show that SC consumes a constant bandwidth while EE consumes less bandwidth when inference is stopped early.

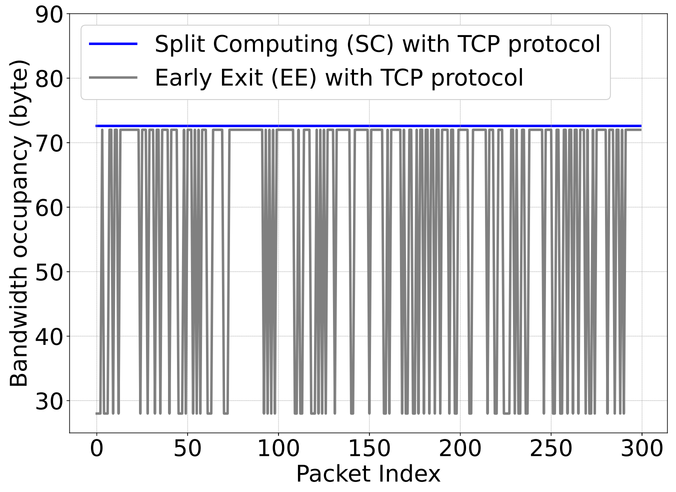


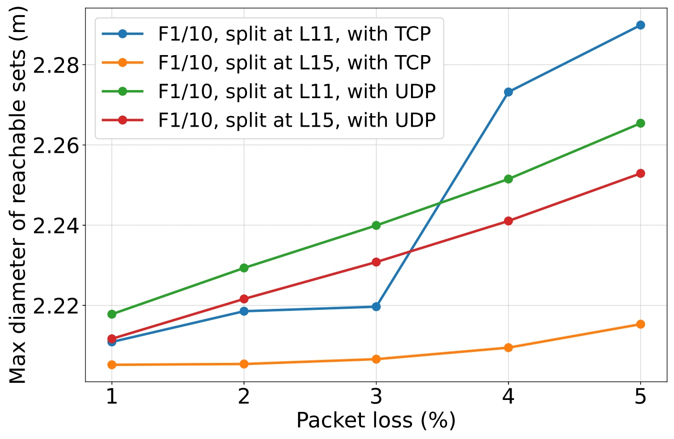
Figure 6: Network bandwidth utilization over time, comparing the SC (blue) and EE scenario (gray), both using the TCP protocol.

## D. *Split computing and cyber-physical systems safety*

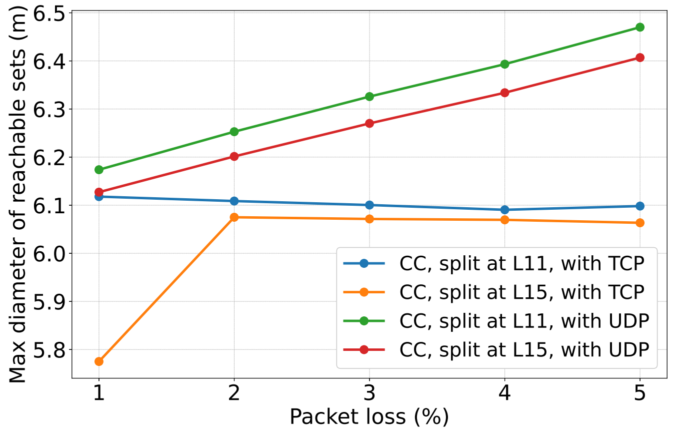
We have seen how different SC configurations yield varying trade-offs between accuracy, latency, and resource requirement. Their effects on the safety of CPSs are many and must be investigated considering the properties of the specific CPS at hand. As a case study, we selected two control systems – a simplified F1/10 car model derived from [26] and a cruise control model derived from [27] – and measured the maximum diameter of reachable sets as introduced in Section IV. For each control system, we evaluated four SC configurations: (1) splitting at layer 11 using TCP, (2) splitting at layer 15 using TCP, (3) splitting at layer 11 using UDP, and (4) splitting at layer 15 using UDP. We used the latency of each configuration as the sampling period $h$ for the control systems. We assumed that the neural network accuracy $a \in [0, 1]$, translates to a state estimation error of $(1 - a)^2$.

Figure 7 shows the maximum diameters of reachable sets for the above two control systems using these four SC configurations, with packet loss rates ranging from $1\%$ to $5\%$. Figure 7a reports the maximum diameters of reachable sets for the F1/10 car model, while Figure 7b reports the maximum diameters of reachable sets for the cruise control model. A key observation is that for the F1/10 car model, the maximum diameter of the reachable set using the configuration `L11 TCP` (*i.e.*, splitting at layer 11 and using TCP) increased significantly between packet loss rates $3\%$ and $4\%$, making it the worst configuration at a loss rate of $4\%$ or higher. In contrast, for the cruise control model, the maximum diameter using the same configuration (`L11 TCP`) does not change much between packet loss rates of $3\%$ and $4\%$. In fact, for the cruise control model, `L11 TCP` yields lower maximum diameters than both UDP configurations at all packet loss rates.

As shown in Figure 4, increasing the packet loss rate from $3\%$ to $4\%$ causes a latency increase of more than



(a) Max. diameters of reachable sets for F1/10 (lower is better).



(b) Max. diameters of reachable sets for cruise control (lower is better).

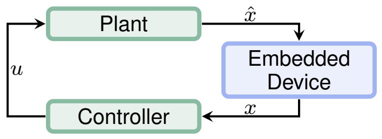
Figure 7: Maximum diameters of reachable sets for two control systems with neural network components, with latency and accuracy values from Figure 5.

$20ms$ for the `L11 TCP` configuration. This difference in behavior reveals that the F1/10 car model is more sensitive to control period changes than the cruise control model and highlights the necessity of considering the properties of the CPS when assessing different split-computing configurations, as the latency and accuracy of the neural networks alone can not tell a complete story about the overall system safety.

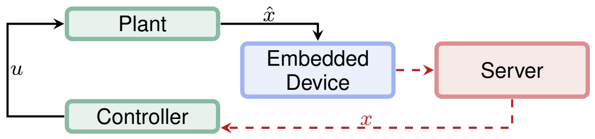
## VI. Controller design for split computing

We now discuss multiple SC-augmented feedback controller architectures. These are shown in Figure 8. From their descriptions, it will become clear that these are not exhaustive and variations of these architectures are possible. As mentioned earlier, our goal in this position paper is not to conduct an exhaustive study of this topic but to initiate the first discussion.

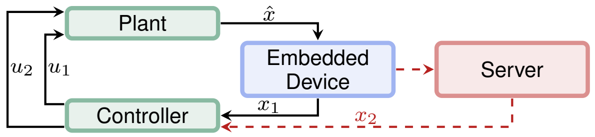
Figure 8a shows the most basic scenario, where the entire DNN is implemented locally on an embedded system. Any embedded platform's relatively low computational bandwidth will restrict the DNN's size, compromising its classification or estimation accuracy. As shown in Section V, a higher state estimation error results in a more extensive reachable set, thereby compromising system safety.



(a) Basic Edge Computing only scenario.



(b) Edge-Cloud (or Split) Computing scenario.



(c) SC and EE scenario.

Figure 8: SC architectures for a feedback controller.

Figure 8b shows a typical SC architecture. A part of the inference is done on a neural network running locally on an embedded platform, and the subsequent inference is done on a server in the cloud. By using the cloud – and thereby supporting a bigger DNN – the inference accuracy will be higher, and therefore, the state estimation error will be lower than in the case shown in Figure 8a. However, a higher delay will be involved in communicating to and from the cloud, resulting in a higher sensor-to-actuator delay. Depending on the communication protocol used, as shown earlier, there might also be data loss, which will also impact the size of the reachable set and, therefore, the system's safety. The research question is to determine an appropriate split point to minimize the size of the system's reachable set and maximize safety. What is important to note here is that because of the sensor-to-actuator delay associated with each splitting point, the DNN architecture with SC that maximizes inference accuracy will not necessarily be the optimal SC architecture for maximizing system safety, motivating a study of SC for CPS.

Finally, Figure 8c shows a scenario, where if sufficient inference accuracy is reached using the local DNN then early exit may be used, where $x_1$ is used to compute a control input $u_1$. This results in a lower sensor-to-actuator delay and avoids any communication with the cloud. But if the inference accuracy using a local-only DNN is not deemed sufficient, then the additional computation is conducted in the cloud, and $x_2$ is instead used by the controller to compute $u_2$.

However, there could be several additional possibilities here. First, irrespective of the inference accuracy achieved locally using the edge DNN, a "preliminary" state estimate, henceforth referred to as $x_1$, can be used to compute an earlier control input $u_1$. A more accurate state estimate $x_2$ using the cloud DNN may be used later to apply a second control input $u_2$. Such a controller addresses the importance of lower delays and the need for more accurate state estimates. The controller needs to be appropriately designed, since it might be that the control input $u_2$ is applied during a later sampling period than the one in which $u_1$ is applied. Either the entire $\hat{x}$ may be sent to the cloud, or as shown in Figure 8c, the data sent to the cloud might be first processed by the edge DNN.

As yet another possibility, some components of the sensed state $\hat{x}$ may be processed by a local DNN on edge to compute $x_1$, and the remaining components of $\hat{x}$ may be computed by the DNN on the cloud to compute $x_2$. The set of $\hat{x}$ components sent to the edge and the cloud need not be disjointed. How to partition state components between the edge and the cloud, and how to design the resulting controller to maximize system safety, are again open questions that need to be studied.

## VII. Concluding remarks and Outlook

This paper introduced "*Split Computing CPS*", a topic that we believe has not been studied before. Although there exists a rich literature on implementing DNNs on edge devices, and more recently on split computing, the focus has almost exclusively been on maximizing inference accuracy of the DNN. However, we have argued in this paper that an optimal DNN implementation architecture when evaluated in isolation might no longer be optimal when used as a part of a larger system, *e.g.*, a CPS. We used autonomous systems [28] that increasingly use DNNs for perception processing as examples to illustrate this. Here, we have used the size of the reachable set of a feedback controller as a measure of system safety and showed that different SC architectures result in different degrees of system safety. Finally, we introduced different CPS-SC architectures and showed a large design space that needs to be studied by the CPS and SC research communities.

The research direction discussed in this paper is related to the problem of control/architecture co-design [29], [30] that has lately attracted considerable attention. Such co-design is motivated by control strategies being designed with simplistic assumptions on the implementation platform that are not valid in reality. As a result, verification or certification [31], [32] results at the model level fail to carry over to an implementation. While techniques such as progressively refining a controller by introducing more implementation details and simulating after each such refinement step [33], [34] are common, they come with significant overheads. Alternatively, co-design approaches start with a partial specification of a set of controllers and their implementation options, and attempt to explore different implementation choices and the corresponding optimal parametrizations of the controllers [35]–[37]. Here, the implementation choices could involve task mapping and task and communication scheduling [38]–[42]. Each of these implementation choices is associated with different delays experienced by the control tasks, for which optimal control parameters may be synthesized [43]–[45]. Taking such co-design a step further, recent work has considered different notions of system-level safety (such as those discussed earlier in this paper) and studied scheduling and controller design techniques that allow non-ideal timing behaviors such as deadline misses [46]–[49]. As CPS implementation platforms and their communication architectures – such as in the automotive domain – become more complex, distributed, and increasingly wireless [50], [51], there is also a need for new timing analysis techniques [52], [53] to support such co-design.

This paper extends such co-design for CPS to consider machine learning components [54], [55]. With increasing design complexity, in the future, co-design techniques need to be extended to consider the design space of not only control strategies and the implementation of control tasks and signals [56], but also the design space of machine learning components, and security mechanisms [57], [58]. Such holistic co-design approaches can focus on *system-level* safety [2] and performance metrics instead of following current design methods that attempt to optimize individual components (such as machine learning, real-time schedulers, or security) and assemble multiple individually-optimized components. Focusing on such system-level metrics will allow more design flexibility and help design better resource provisioned and more cost-effective CPS and autonomous systems.

## VIII. Acknowledgments

This study was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043), and by the European Union's Horizon Europe research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 101109243. This manuscript reflects only the Authors' views and opinions. Neither the European Union nor the European Commission can be considered responsible for them. This work was also partially supported by the US NSF grant 2038960.

## References

[1] D. Goswami *et al.*, "Challenges in automotive cyber-physical systems design," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2012.

[2] C. Hobbs *et al.*, "Quantitative safety-driven co-synthesis of cyber-physical system implementations," in *15th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2024.

[3] ——, "Safety analysis of embedded controllers under implementation platform timing uncertainties," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4016–4027, 2022.

[4] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, 2022.

[5] L. Capogrosso *et al.*, "A machine learning-oriented survey on tiny machine learning," *IEEE Access*, 2024.

[6] A. Howard *et al.*, "Searching for MobileNetv3," in *IEEE/CVF International Conference on Computer Vision*, 2019.

[7] T. Liang *et al.*, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, 2021.

[8] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, 2021.

[9] A. E. Eshratifar *et al.*, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, 2019.

[10] Y. Matsubara *et al.*, "Distilled split deep neural networks for edge-assisted real-time systems," in *Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2019.

[11] Y. Kang *et al.*, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” *ACM SIGARCH Computer Architecture News*, 2017.

[12] G. Li *et al.*, “Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge,” in *27th Intl. Conf. on Artificial Neural Networks (ICANN)*. Springer, 2018.

[13] H. Choi and I. V. Bajić, “Deep feature compression for collaborative object detection,” in *25th IEEE International Conference on Image Processing (ICIP)*, 2018.

[14] D. Carra and G. Neglia, “DNN split computing: Quantization and run-length coding are enough,” in *IEEE Global Communications Conference (GLOBECOM)*, 2023.

[15] A. E. Eshratifar *et al.*, “Bottlenet: A deep learning architecture for intelligent mobile cloud computing services,” in *IEEE/ACM Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2019.

[16] M. Sbai *et al.*, “Cut, distil and encode (CDE): Split cloud-edge deep inference,” in *18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2021.

[17] F. Cunico *et al.*, “I-split: Deep network interpretability for split computing,” in *26th Intl. Conference on Pattern Recognition (ICPR)*, 2022.

[18] L. Capogrosso *et al.*, “Split-Et-Impera: A framework for the design of distributed deep learning applications,” in *26th Intl. Symp. on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023.

[19] L. Capogrosso, E. Fraccaroli, S. Chakraborty, F. Fummi, and M. Cristani, “Mtl-split: Multi-task learning for edge devices using split computing,” *arXiv preprint arXiv:2407.05982*, 2024.

[20] L. Capogrosso, E. Fraccaroli, G. Petrozziello, F. Setti, S. Chakraborty, F. Fummi, and M. Cristani, “Enhancing split computing and early exit applications through predefined sparsity,” *arXiv preprint arXiv:2407.11763*, 2024.

[21] C. Lo *et al.*, “A dynamic deep neural network design for efficient workload allocation in edge computing,” in *IEEE International Conference on Computer Design (ICCD)*, 2017.

[22] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[23] A. Krizhevsky *et al.*, “Learning multiple layers of features from tiny images,” 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[24] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[25] J. Deng *et al.*, “Imagenet: A large-scale hierarchical image database,” in *IEEE conference on computer vision and pattern recognition*, 2009.

[26] M. O’Kelly *et al.*, “F1tenth: An open-source evaluation environment for continuous control and reinforcement learning,” in *NeurIPS 2019 Competition and Demonstration Track*. PMLR, 2020.

[27] K. Osman, M. F. Rahmat, and M. A. Ahmad, “Modelling and controller design for a cruise control system,” in *5th International Colloquium on Signal Processing & Its Applications*, 2009.

[28] U. D. Bordoloi *et al.*, “Autonomy-driven emerging directions in software-defined vehicles,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[29] D. Roy *et al.*, “Multi-objective co-optimization of FlexRay-based distributed control systems,” in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[30] D. Goswami, R. Schneider, and S. Chakraborty, “Co-design of cyber-physical systems via controllers with flexible delay constraints,” in *16th Asia South Pacific Design Automation Conference (ASP-DAC)*, 2011.

[31] P. Kumar *et al.*, “A hybrid approach to cyber-physical systems verification,” in *49th Annual Design Automation Conference (DAC)*, 2012.

[32] G. Georgakos *et al.*, “Reliability challenges for electric vehicles: from devices to architecture and systems software,” in *50th Annual Design Automation Conference (DAC)*, 2013.

[33] G. Tibba *et al.*, “Testing automotive embedded systems under X-in-the-loop setups,” in *35th International Conference on Computer-Aided Design (ICCAD)*, 2016.

[34] J. Oetjens *et al.*, “Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges,” in *The 51st Annual Design Automation Conference (DAC)*. ACM, 2014, pp. 113:1–113:6.

[35] L. Zhang *et al.*, “Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems,” in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.

[36] D. Roy *et al.*, “Tool integration for automated synthesis of distributed embedded controllers,” *ACM Trans. Cyber Phys. Syst.*, vol. 6, no. 1, pp. 3:1–3:31, 2022.

[37] R. Schneider *et al.*, “Constraint-driven synthesis and tool-support for Flexray-based automotive control systems,” in *9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[38] F. Sagstetter *et al.*, “Schedule integration framework for time-triggered automotive architectures,” in *51st Annual Design Automation Conference (DAC)*, 2014.

[39] D. Roy, S. Ghosh, Q. Zhu, M. Caccamo, and S. Chakraborty, “Good-spread: Criticality-aware static scheduling of CPS with multi-qos resources,” in *41st IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[40] S. Chakraborty and L. Thiele, “A new task model for streaming applications and its schedulability analysis,” in *Design, Automation and Test in Europe Conference and Exposition (DATE)*, 2005.

[41] M. Lukasiewycz *et al.*, “Modular scheduling of distributed heterogeneous time-triggered automotive systems,” in *17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012.

[42] H. Voit *et al.*, “Optimizing hierarchical schedules for improved control performance,” in *IEEE Fifth International Symposium on Industrial Embedded Systems (SIES)*, 2010.

[43] D. Goswami, R. Schneider, and S. Chakraborty, “Re-engineering cyber-physical control applications for hybrid communication protocols,” in *Design, Automation and Test in Europe (DATE)*, 2011.

[44] W. Chang, D. Goswami, S. Chakraborty, and A. Hamann, “Os-aware automotive controller design using non-uniform sampling,” *ACM Trans. Cyber Phys. Syst.*, vol. 2, no. 4, pp. 26:1–26:22, 2018.

[45] D. Roy *et al.*, “Tighter dimensioning of heterogeneous multi-resource autonomous CPS with control performance guarantees,” in *56th Annual Design Automation Conference (DAC)*, 2019.

[46] S. Xu *et al.*, “Safety-aware flexible schedule synthesis for cyber-physical systems using weakly-hard constraints,” in *28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023.

[47] B. Ghosh *et al.*, “Statistical verification of autonomous system controllers under timing uncertainties,” *Real Time Syst.*, vol. 60, no. 1, pp. 108–149, 2024.

[48] S. Xu *et al.*, “Safety-aware implementation of control tasks via scheduling with period boosting and compressing,” in *29th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2023.

[49] A. Yeolekar, R. Metta, C. Hobbs, and S. Chakraborty, “Checking scheduling-induced violations of control safety properties,” in *20th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, ser. Lecture Notes in Computer Science, vol. 13505. Springer, 2022.

[50] P. H. Kindt *et al.*, “Energy modeling for the bluetooth low energy protocol,” *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 2, pp. 13:1–13:32, 2020.

[51] ——, “Neighbor discovery latency in ble-like protocols,” *IEEE Trans. Mob. Comput.*, vol. 17, no. 3, pp. 617–631, 2018.

[52] E. Fraccaroli *et al.*, “Timing predictability for SOME/IP-based service-oriented automotive in-vehicle networks,” in *Design, Automation & Test in Europe Conference (DATE)*, 2023.

[53] D. Roy *et al.*, “Timing debugging for cyber-physical systems,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[54] Z. Wang, C. Huang, Y. Wang, C. Hobbs, S. Chakraborty, and Q. Zhu, “Bounding perception neural network uncertainty for safe control of autonomous systems,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[55] S. Xu *et al.*, “Neural architecture sizing for autonomous systems,” in *15th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2024.

[56] D. Roy *et al.*, “Semantics-preserving cosynthesis of cyber-physical systems,” *Proc. IEEE*, vol. 106, no. 1, pp. 171–200, 2018.

[57] P. Mundhenk *et al.*, “Security in automotive networks: Lightweight authentication and authorization,” *ACM Trans. Design Autom. Electr. Syst.*, vol. 22, no. 2, pp. 25:1–25:27, 2017.

[58] H. Liang, Z. Wang, D. Roy, S. Dey, S. Chakraborty, and Q. Zhu, “Security-driven codesign with weakly-hard constraints for real-time embedded systems,” in *37th IEEE International Conference on Computer Design (ICCD)*, 2019.