SUSTAIN Deliverable

D2.1 System modelling of the distributed and node intelligence: initial common strategy

Grant Agreement number	101071179
Action Acronym	SUSTAIN
Action Title	Smart Building Sensitive to Daily Sentiment
Type of action:	HORIZON EIC Grants
Version date of the Annex I against which the assessment will be made	28th March 2022
Start date of the project	1st October 2022
Due date of the deliverable	M3
Actual date of submission	31st January 2023
Lead beneficiary for the deliverable	AALTO
Dissemination level of the deliverable	Public

Action coordinator's scientific representative

Prof. Stephan Sigg
AALTO –KORKEAKOULUSÄÄTIÖ,
Aalto University School of Electrical Engineering, Department of Communications and Networking
stephan.sigg@aalto.fi

Funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or [name of the granting authority]. Neither the European Union nor the granting authority can be held responsible for them.

Authors in alphabetical order		
Name	**Beneficiary**	**e-mail**
Martin Andraud	Aalto University	martin.andraud@aalto.fi
Giovanni Iacca	University of Trento	giovanni.iacca@unitn.it
Lingyun Yao	Aalto University	Lingyun.yao@aalto.fi
Kasim Sinan Yildirim	University of Trento	kasimsinan.yildirim@unitn.it

Abstract

This document is the first deliverable related to work package (WP) 2 of the SUSTAIN project. The objective of this document is double: (1) to give a first report about the common strategy used to model the distributed intelligence mechanisms which will be designed and implemented in the project, and (2) to lay out an initial strategy to integrate the distributed learning mechanisms in SUSTAIN. It is meant to be used as reference for the consortium to know the intended capabilities of the distributed intelligence framework and use the proposed framework to integrate all parts of the SUSTAIN node. It links to forthcoming activities and deliverables, especially in WP2 and WP3 and, to a lesser extent, to those in WP4 to WP6.

Note on version 2 (this version)

In the initial project proposal, the first task defined two working points, namely the implementation and the integration of the distributed intelligence.

"Both AAL and TRE teams will first prepare their own wishes regarding the final system, while discussing together the main tools that will be used for modelling and implementation. AAL and TRE teams will then physically meet for a workshop where the implementation and the integration of the distributed learning strategy will be discussed."

In the first version of the deliverable, we had put more emphasis on the integration, i.e. how the distributed intelligence will be modelled and integrated at a system level, reflecting the discussions we had with regards to the task. The initial task was planned with a 3 month-duration, hence providing only an initial strategy for the project. Yet, we did not put enough weight on the implementation part, which we intend to correct here.

Contents

1. Introduction and purpose of this deliverable

This report is the first deliverable of Work Package (WP) 2: “Node-level intelligence with embedded probabilistic learning”. The overall goal of WP2 is to design two main hardware blocks embedded in the final node:

- An AI accelerator, enabling **node-level intelligence** mechanisms based on probabilistic models. This node-level intelligence will interact with the **distributed meta-level intelligence** developed in WP3.
- A **processor** able to control this intelligence, as well as any other functionalities of the node, if needed (sensing, data gathering, communication).

The objective is to integrate both in a single circuit for the final system demonstration in WP7.

In the project, WP2 has very strong links with WP3 “Distributed intelligence”. These two WPs both deal with the intelligence part of the system, whether it is at node level (WP2) or at a meta-level (WP3). Hence, the objective was to start discussions early between WP2 and WP3, to exchange on the possible strategy to have implement the distributed intelligence mechanisms. This strategy includes both the technical solutions and the integration of those in a single simulation framework.

WP2 also has links with WP4, WP5 and WP6, as it is planned to integrate the various blocks developed in the project with the processor and propose an adaptation methodology for the node using probabilistic models.

Purpose of the deliverable

The purpose of this deliverable is to report the first discussions in between various project members (from Aalto University and University of Trento), regarding the implementation and integration of the intelligence mechanisms in SUSTAIN. *It focuses on two aspects: (1) the simulation framework and the chosen methodology used to integrate the different blocks developed for the project, and (2) an initial implementation for the distributed intelligence*, as described in the project proposal. It should be noted that we consider this deliverable as the first version of a living document, which shall be updated as the project evolves.

The structure of the deliverable is as follows. First, we will give an overview of the envisaged distributed intelligence in SUSTAIN, elaborating on the content initially described in the project proposal. Then, we will list challenges in modeling large-scale systems with hardware/software integration. Finally, we will further detail our proposed strategy for modeling and developing the different hardware and software blocks of the project.

2. Overview of the envisaged intelligence in SUSTAIN

a. Distributed intelligence

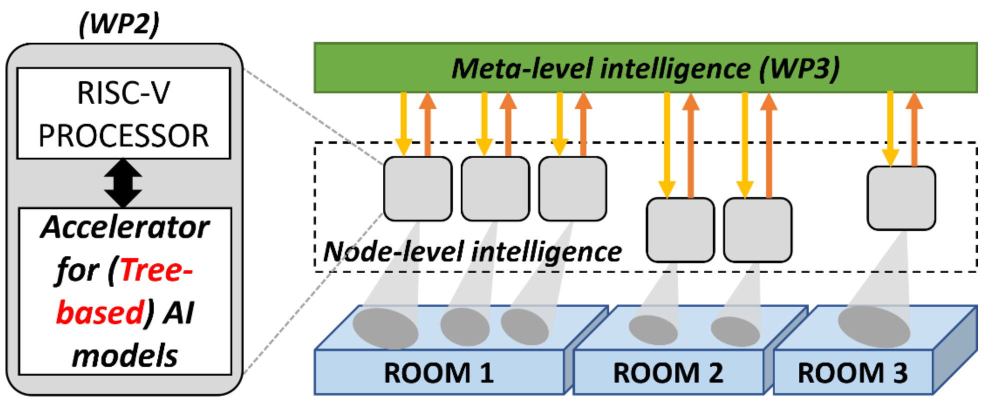

(WP2)
RISC-V PROCESSOR
Accelerator for (Tree-based) AI models
Meta-level intelligence (WP3)
Node-level intelligence
ROOM 1
ROOM 2
ROOM 3

Figure 1 - Overview of the distributed intelligence mechanisms developed in SUSTAIN

As detailed in the initial project proposal and illustrated in Figure 1, the overall intelligence strategy will rely on distributed intelligence, both at node level and at meta level. More specifically, it will be articulated around these two levels:

- **Node-level intelligence:** each node will embed a dedicated intelligence block, in the form of a hardware accelerator controlled by a customized RISC-V processor. In the intelligence architecture, a tree-based model (see details in subsection b) allows to gather *local* information about the current conditions and/or configure the other parameters of the sensor node (i.e., sensor sampling, communication, data processing, etc.) to optimize its performance and power consumption. It should be noted that there are various hardware-efficient models in the form of trees, such as decision trees, behaviour trees [2,3] or Probabilistic Circuits [1]. As a secondary objective, we will investigate how we could retrain these models online to increase their ability over time. This issue has *not yet been tackled in the current literature*. The inference process will be controlled by a customized RISC-V processor specifically developed in Aalto University. This processor can also be used to embed the different functionalities of the sensor node, developed in WP4, WP5 and WP6 (RF sensing, energy harvesting, etc.).
- **Meta-level intelligence**: at this level, we will integrate the node-level intelligence into a coherent meta-level framework. More specifically, starting from the node-lebel models described above, we will expand the node capabilities by also including various kinds of black-box models (e.g., neural networks) as well as "transparent" (also called glass-box or white-box) models, such as decision or behavior trees, and combinations thereof. These models will be optimally designed by using Neural Architecture Search (e.g., Evolutionary Algorithms), considering computational constraints (limited memory/CPU) and energy consumption. The trade-offs between training time, amount of training data, and model performance will be assessed. We will then consider the addition of a meta-level framework on top of the individual node-level intelligence. This framework will allow, for instance, that each node selects (or switches to) a different available node-local model from a predefined set

of models, depending on the node working regime, the specific energy/hardware constraints, and its context. Such predefined set models can be either made available in a centralized manner, or in a distributed manner, by allowing models to migrate from one node to another [5]. We will also explore different distributed learning approaches, based on consensus and other forms of aggregation (e.g., majority, Fisher's method, descriptive statistics on confidence levels) of the outputs of the node-local models, to achieve a global goal, such as an accurate estimate of the human presence in the monitored environment. Federated and split learning approaches will be considered to ensure a flexible architecture capable of guaranteeing a proper separation of concerns (different nodes model different parts of the monitored environments) and data protection/privacy. The meta-learning framework will then automatically decide which kind of distributed learning approaches are more appropriate to the specific task at hand, which form of aggregation, and which kind of split over the network. In essence, the meta-learning framework will "learn how to learn" over the network. Please note that, to enable meta-learning, we consider a scheme for probabilistic communication & computation (that will be detailed in T3.3).

b. From PCs to Tree-based models (note for this deliverable version)

As a starting point, the envisioned models to be embedded in the node to be probabilistic circuits (PCs), which is a relatively recent and increasingly popular class of probabilistic models [1]. The main advantage of PCs in terms of implementation on resource-constrained devices is that they offer tractable inference, i.e., exact inference with limited computations. During our discussions, we agreed that we could open the discussion on the exact model used in the node intelligence to evaluate and compare several options. Towards this direction, the emergence of neural-network models based on trees and recent decision tree approaches, for instance, fast feedforward neural networks (FFFs) [10,11,13] made us update the initial strategy to include them as possible candidates for the node intelligence.

Hence, we agreed that at the later stage of the project in task T2.2, we will develop a dedicated accelerator for tree-based models and not only PCs. This is possible because probabilistic circuits are also tree-based models, for instance relying on similar computations that FFFs. Enlarging the scope of the accelerator to include several models will allow for a broader impact of the designed circuit, and for a comparison between various models in SUSTAIN.

In terms of work task division, WP2 will still focus on developing hardware-efficient PC implementations, along the hardware accelerator and processor for the SUSTAIN node. In parallel, WP3 will work on emerging decision trees and FFF-based models, while developing federated learning strategies for the distributed intelligence. All results will be integrated in a single simulation framework enabling a co-simulation between hardware and software parts of the system. This allows for having multiple path towards the distributed intelligence mechanisms in SUSTAIN, maximizing the chances of success.

2. Challenges in hardware/software modeling

Before entering into details about the proposed simulation methodology and the intended implementation of the distributed intelligence, this section details several challenges exist in implementing distributed intelligence in resource-constrained environments [7-11]. These challenges are:

- **Missing deployment support.** Several frameworks, such as Tensorflow Lite and EdgeImpulse, allow building machine learning (ML) models for resource-constrained edge devices. These frameworks provide several features to train and compress neural networks (via weight pruning, sharing, and quantization) to decrease the memory footprint of the ML models and optimize energy consumption. However, these common frameworks target only a limited number of microcontroller-based platforms, making them rigid and almost impossible to customize for specific hardware platforms. Therefore, optimizing and mapping trained models to the proposed processor and accelerator is difficult as of now.

- **Missing distributed/on-device learning support.** Mentioned ML frameworks enable the deployment of trained models for on-device inference, but they do not support on-device training which is challenging. For instance, in federated learning, edge devices share common observations to build a model collaboratively. However, exchanging model parameters under energy constraints and aggregating model parameters under memory constraints is not trivial.

- **Compatibility with custom-designed hardware blocks.** Custom-designed hardware is typically simulated and tested on computer-aided design (CAD) frameworks. This simulation process takes time and resources and becomes quickly impractical for large-scale testing as we wish to achieve in this project. The most common solution to this issue is to use behavioural models in various intermediate levels, whether suited for system hardware modelling (system Verilog, VHDL AMS, etc.) or more generic languages (C, C++, Python). The challenge is to include all the main functionalities of the node while reducing simulation time to a minimum.

To start an exploration of distributed intelligence in resource-constrained environments, a simulation environment has been initiated (under WP3). As of now, this environment can simulate a specific federated learning scenario on a PC. Briefly, the simulator environment emulates the distributed training algorithm and the communication between edge devices and the central entity. With this setup, the strategies to optimize energy consumption and memory footprint of the distributed learning approaches can be evaluated. Moreover, the initial strategies that can be used to map the trained models to the probabilistic circuits or other tree-based models can be developed and elaborated.

3. Simulation framework developed for the project (main focus in WP2)

Considering the challenges discussed in the previous section, we aim to maximize the chances to build a successful final demonstrator at the end of the project (in WP7). For that, the proposed strategy is to start building a *simple but fully functional system first*, and to *increasingly improve* it with the components developed in the project. For that, we have created a common repository where all members of the project can access and contribute to (https://gitlab.com/m.andraud/sustain/). Each project member can then create their own blocks and model to be further integrated in the system.

a. Hardware system simulations through the SyDeKick

The general strategy is to base common developments on a dedicated simulation platform called the SyDeKick [4]. In this platform, the different blocks constituting the system can be modelled using various levels of abstraction, from high-level python behavioural modelling to synthesizable hardware.

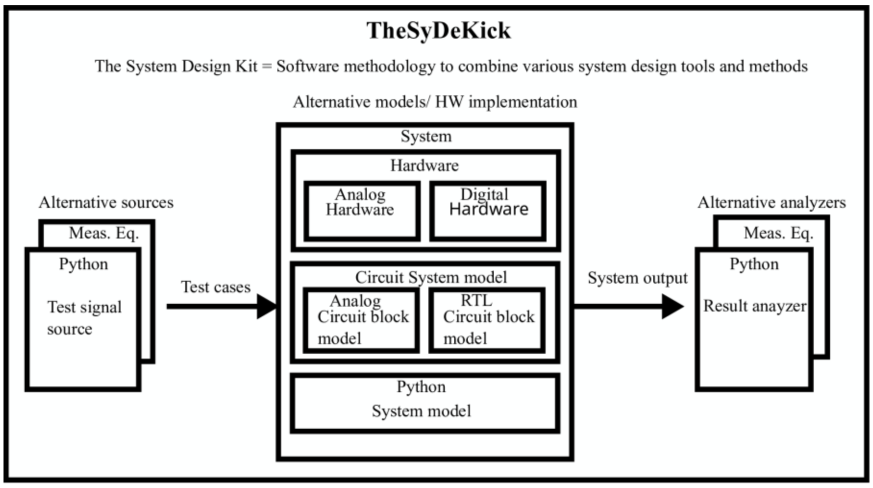


Figure 2 - Overview of the System Development Kit (SyDeKick) framework

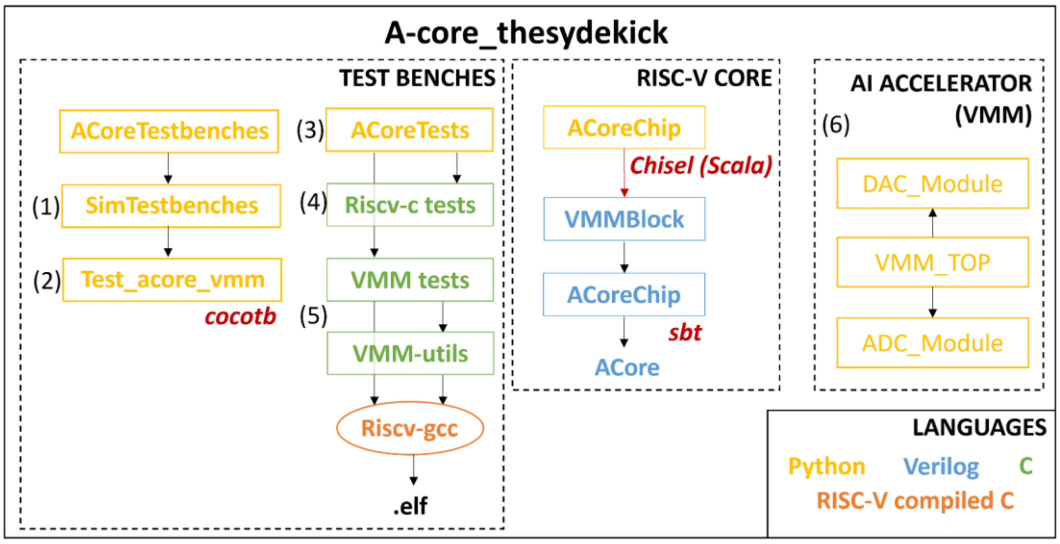
For instance, the complete model can be built at a behavioural level using python, including software routines if needed. Each hardware block can subsequently be replaced by their hardware equivalent, for instance as a Register Transfer Level (RTL) block for digital implementation, or a netlist for analog implementation. All other blocks can be written in Python, such as test benches, sensor implementations or data processing algorithms. The SyDeKick also supports a co-simulation between RTL and python using the cocotb tool (https://docs.cocotb.org/en/stable/quickstart.html).

As an initial target, we will model the hardware system with the SyDeKick, and make the system as a standalone unit that can transmit data at certain time intervals to the meta-level intelligence (as a sensor node would do in a real scenario). The model can then be extended to receiving information about configuration when it will include the embedded PC model as well. In that way, we can have the hardware as a standalone block, which can be used as a data-generation node for data generation, training and validation of the meta-level intelligence.

Indeed, the meta-level intelligence framework will require a large amount of training data to develop the intended adaptive mechanisms. As the final hardware implementation of the node and its embedded intelligence will be finalized only towards the end of the project, we wish to start collecting data with commercial tools and off-the-shelf components. Collecting data from real sensor nodes is a common strategy employed in the field of distributed intelligence as it does not require creating a model with synthetic data. In addition, there exist various sensor node platforms, based on microcontrollers (e.g., Arduino) that can provide a subset of the functionalities that we intend to develop. We will start by using such platforms and integrate step-by-step the innovations brought by the project.

b. Baseline implementation with hardware/software integration

Before adding the various blocks corresponding to new elements of the SUSTAIN node, we created a baseline implementation showing the possibilities offered by the SydeKick. This baseline contains an open-source RISC-V processor called A-core, and includes a model of an AI accelerator developed in the context of another project. This serves as an example system to show the integration possibilities of the SyDeKick. This baseline is totally open source and available here: https://gitlab.com/a-core.



(1): gathers all sources and calls the cocotb testbench for the RTL-Python co-simulation

(2): contains the RTL test bench. It stores the .elf file generated by ACoreTests and runs the co-simulation RTL-Python

(3): compiles the AcoreChip to Verilog with sbt, compiles the testbench file .elf with the gcc compiler and runs the simulation testbench.

(4): standard tests for the A core chip.

(5): tests and utility functions for the VMM

(6): Python model of the VMM, could be replaced by the hardware implementation

The complete system is embedded into The SyDeKick framework. It comprises three main parts: the RISC-V processor core A-core, with a RTL design written in Verilog, an AI accelerator in the form of a Vector-Matrix Multiplier (VMM) developed in another project, and various testbenches for the system. The interface between the processor and the accelerator is realized through an AXI-4 bus, implemented in the "VMMBlock" file together with the processor in Verilog. As the VMM is implemented in Python, it is necessary to have a co-simulation between RTL (processor and AXI-4 interface) and python (VMM). For this purpose, the file "ACoreTestbenches" sets up a simulation framework where for each clock cycle of the simulation at the RTL level, it halts the simulation and runs the "VMM_top" file to simulate the VMM. It means that data can be transferred to the VMM python file in both directions through "VMMBlock". As such, there is no need for a long simulation time of the VMM, while enabling it to have the exact same test-bench (compiled from C code by gcc) than if the physical chip were available. That code has also been tested on the taped-out circuit containing Acore and the VMM, currently under publication.

Details on the Aalto A-core RISC-V processor

At the heart of the sensor node usually lies a microprocessor. Its role is to schedule the node's operations. This includes setting up the node's sensor frequencies, collecting data, and controlling the communication with the meta-level intelligence. In this project we will employ the A-core processor developed in Aalto University. A first version of A-core has been taped-out in December 2022, with the following characteristics:

- RISC-V 32 IMFC
- 7 stage pipeline with multi-cycle execution
- 64kB on-chip Program Memory, 64kB on-chip RAM
- 20 MHz operating frequency
- UART and GPIO peripherals
- JTAG programming interface

This platform will be used to test the ability of the processor to handle the scheduling of the node. In addition, the embedded AI accelerator could be used to try several adaptation algorithms in-situ.

c. Software simulation for the distributed intelligence and learning

In addition to hardware simulator, Trento University has initiated a basic simulator development that will enable high-level evaluations of several distributed learning scenarios. This custom simulation environment will be used to evaluate training, model development, and optimization procedures. It can be further integrated with the hardware system simulation as both run with Python.

In addition, Trento University has software tools that can automatically generate code that will execute ML models on resource-constrained microcontrollers. To support on-device training, these tools need to be extended. We will develop customized and highly optimized libraries that will realize learning procedures on our target devices. For energy and computational efficiency, these libraries will benefit from the specific hardware features proposed by our target platforms.

d. Other open-source frameworks that can be used in the project

Aalto AutoPC hardware generator. The node-level intelligence block will be realized with a dedicated and embedded machine-learning algorithm. Initially, we plan to use Probabilistic Circuits (PCs) for that purpose. Several PC implementations have been implemented over the last years. They all have their specificities, and the best performing PC implementation depends on the application. To obtain a fast hardware performance comparison and prototyping of PC, Aalto is developing ***AutoPC***, a framework for the automatic compilation and hardware generation of PCs, on FPGAs. The framework is currently being published; hence details will be available in the next deliverables of WP2. This platform is completely open source and freely available here: https://gitlab.com/a-core.

4. Proposed strategy for node and distributed intelligence (main focus in WP3)

As known, the Machine Learning field has seen a remarkable growth in the last decades, which enabled exciting applications in a wide variety of fields, such as autonomous driving, healthcare, human-computer interaction, and others. However, the field still suffers from two major issues: the **need of energy-hungry, special-purpose hardware**; and the need for strongly **centralized architectures**, which need all the entities of an organization to share their data, potentially exposing them to data leaking events. These two aspects are of paramount importance for the SUSTAIN project. In the following, we briefly summarize how we intend to handle them, reflecting the two layers of intelligence described above (respectively, node-level and distributed meta-level).

a. Node-level intelligence: tiny Machine Learning (tinyML) with tree-based models

Regarding the first issue, we aim to develop fast, energy-efficient methodologies for machine learning that can be executed on tiny devices, which have strong constraints on the amount of memory and on energy consumption. To this end, we will investigate the use of **novel architectures**, e.g., based on shallow neural networks (for instance FFFs), orthogonal and oblique decision trees, differentiable decision trees or probabilistic circuits. In parallel, WP2 will develop an accelerator for such models (figure 2). This change is possible because all these models have all similar characteristics for hardware acceleration: they are tree-based and sparse. Hence, we plan to build an accelerator handling sparsity and a tree-based connection between nodes, which will be suitable for FFFs, decision trees of PCs. More implementation details will follow in the course of task T2.2.

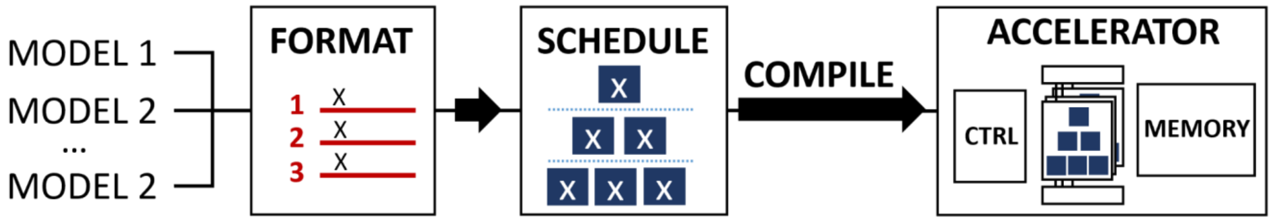


Figure 2 - Envisaged compilation and execution pipeline for the node-level intelligence (in task T2.2)

Yet, the memory footprint of those architectures may still be a drawback, since the number of parameters of such models will increase significantly while the depth increases. Recently, **weight virtualization** has been used as a weight-sharing technique to reduce the memory footprint and physical hardware requirements of an embedded system, especially for different tasks and deep neural networks [12]. To overcome the memory footprint problem, we aim to deploy weight virtualization on the novel architectures we will use. Additionally, we plan to implement **virtualized memory** of such techniques to reduce the physical requirements of our targeted embedded platform. It should be noted that weight virtualization requires some specific care also on the hardware side. Therefore, a matching table that maps virtualized weights and weights of a network will be implemented.

b. Distributed meta-level intelligence: Federated Learning (FL)

Federated learning (FL) is a decentralized approach to machine learning that enables multiple clients to collaborate and train on shared models while ensuring that raw data remains locally on the devices.

Traditional centralized machine learning requires aggregating all data into a single server, which poses significant privacy and security risks. FL mitigates these risks by ensuring data remains localized, enhancing user privacy and complying with regulations such as GDPR. Despite these advancements, FL faces several significant challenges that hinder its widespread adoption and effectiveness:

- **Communication Overhead:** The frequent exchange of model updates between the central server and edge devices can be bandwidth-intensive, leading to high communication costs and latency, especially in resource-constrained environments.
- **Data Heterogeneity:** The non-independent and identically distributed (non-IID) nature of data across different clients can result in biased models and degraded performance. Each client's local data may differ significantly in terms of distribution and quantity.
- **System Heterogeneity:** The diverse computational capabilities and network conditions of participating devices can cause stragglers, where slower devices delay the overall training process.
- **Privacy and Security:** While FL enhances data privacy, it is not immune to attacks. Techniques such as model inversion and poisoning attacks can still threaten the integrity and confidentiality of the training process.

To fully understand the implications of Federated Learning, it is essential to evaluate its advantages and disadvantages:

- **Advantages:**
  - *Enhanced Privacy:* By keeping data on local devices, FL minimizes the risk of data breaches and misuse, enhancing user trust and compliance with privacy regulations.
  - *Reduced Latency:* Localized training can lead to faster model updates as data does not need to be transferred to a central server for processing.
  - *Scalability:* FL can efficiently scale across numerous devices, leveraging their computational power and diverse data for comprehensive model training.
- **Disadvantages:**
  - *Communication cost:* Continuous synchronization and model update exchanges incur high communication overhead, particularly in bandwidth-constrained environments.
  - *Handling Heterogeneous Data:* Variations in data distributions across clients complicate model convergence and can degrade performance.
  - *Resource Constraints:* The varying computational capabilities of edge devices may limit the complexity of models that can be trained effectively.

In our project we will leverage the strengths of FL while addressing its challenges to develop a robust framework for distributed intelligence. One particular aspect we will focus on is the fact that, in most cases, FL requires a central authority to *define* the network architecture that will be used by all the clients. While this may seem obvious, it is important to note that, due to the fact that the IID assumption does not hold in real-world FL scenarios, different architectures entail different inductive biases that can be optimal for some of the clients and suboptimal for others. For this reason, we will investigate the applicability of algorithms capable of **automatically tailoring the network architecture to all of the clients simultaneously**. This aspect will realize one of the most important properties of the envisioned meta-level distributed intelligence system. More specifically, we will investigate the use of techniques such as **Neural Architecture Search and**

Neuroevolution, which can learn and adapt architectures based on the specific tasks and data characteristics of each client.

Finally, we aim to merge the outcomes from these two phases to have both **efficient and secure machine learning on tiny, even batteryless devices**, and carry out learning in a federated way. To do so, we will adapt the FL strategies proposed in the corresponding phase to the architectures developed for energy-efficient machine learning. For instance, hypothesizing that differentiable decision trees turn out to be the best architecture for tiny ML in our scenario, we will adapt the proposed NAS and Neuroevolutionary approaches to produce only this type of models.

5. Summary of the detailed steps in the proposed strategy

To conclude, the proposed phases of our strategy are:

Phase 1: build the hardware and software simulation frameworks and start collecting data using commercial sensor nodes. In this step, the two simulators presented in section 3 will be developed. As their integration is at a later stage of the project, and to minimize the risks, a first version of the node-level intelligence will be realized with commercial hardware platforms in parallel to these simulator developments. These sensor nodes will collect data for the meta-intelligence level. We will choose commercial platforms with a RISC-V processor to be able to port the code in the custom design when available. The collected data can be used to train several models and compare their performance on the node. We will consider the initial sensors developed by our partners in WP4, 5 and 6 to integrate them with the framework. Regarding the node-level intelligence, WP3 will investigate novel architecture for tinyML based on FFFs and decision trees, while WP2 will focus on PCs. In parallel, WP3 will develop the distributed (federated) learning.

Phase 2: customize and improve the framework with "SUSTAIN" intelligence mechanisms. In this phase, in WP2 we will integrate step by step the different hardware blocks developed in the project in the proposed SyDeKick framework. These include the iteration of the A-core processor, as well as successive versions of the accelerator. As an intermediate step, a FPGA implementing a chosen PC implementation can be tested on hardware through AutoPC. Then, the tinyML models developed in WP2 and WP3 will be ported to the new processor and accelerator.

In parallel to the two aforementioned phases, we will develop finer models for the hardware, to be able to generate accurate and meaningful synthetic data, even though the final versions of the sensors and intelligence mechanisms in the node will be under development. This requires extensive simulation and integration work. It is expected that each partner will provide behavioural models of their hardware parts, so that we can integrate all these parts in the system model.

Phase 3: build the final framework with final nodes and test the system in-situ. In this phase we will eventually integrate all the components of the framework and test the proposed system on a demonstrator.

6. References

[1] A. Vergari, Y. Choi, R. Peharz, and G. Van der Broeck. ”Probabilistic circuits: Representations, inference, learning, applications (tutorial)”. [Online]. Available: http://web.cs.ucla.edu/~guyvdb/talks/ECAI20-tutorial/

[2] Martin Andraud, Ahmed Hallawa, Jaro De Roose, Eugenio Cantatore, Gerd Ascheid, and Marian Verhelst. 2018. Evolving hardware instinctive behaviors in resource-scarce agent swarms exploring hard-to-reach environments. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '18).

[3] J. De Roose, H. Xin, A. Hallawa, G. Ascheid, P. J. A. Harpe and M. Verhelst, "Flexible, Self-Adaptive Sense-and-Compress SoC for Sub-microWatt Always-On Sensory Recording," in *IEEE Solid-State Circuits Letters*, vol. 3, pp. 362-365, 2020.

[4] The system development kit (SyDeKick), [online] https://github.com/TheSystemDevelopmentKit/

[5] Yuan, B., Wolfe, C. R., Dun, C., Tang, Y., Kyrillidis, A. and Jermaine, C. (2022) Distributed Learning of Fully Connected Neural Networks using Independent Subnet Training. Proceedings of the VLDB Endowment, 15(8), pp. 1581-1590. Available from: https://doi.org/10.14778/3529337.3529343

[6] Filho, Carlos Poncinelli, et al. "A systematic literature review on distributed machine learning in edge computing." Sensors 22.7 (2022): 2665.

[7] Disabato, Simone, and Manuel Roveri. "Incremental on-device tiny machine learning." Proceedings of the 2nd International workshop on challenges in artificial intelligence and machine learning for internet of things. 2020.

[8] Ren, Haoyu, Darko Anicic, and Thomas Runkler. "How to Manage Tiny Machine Learning at Scale: An Industrial Perspective." arXiv preprint arXiv:2202.09113 (2022).

[9] Predd, Joel B., Sanjeev B. Kulkarni, and H. Vincent Poor. "Distributed learning in wireless sensor networks." IEEE signal processing magazine 23.4 (2006): 56-69.

[10] Chen, Mingzhe, et al. "Distributed learning in wireless networks: Recent progress and future challenges." IEEE Journal on Selected Areas in Communications 39.12 (2021): 3579-3605.

[11] Bonawitz, Keith, et al. "Towards federated learning at scale: System design." Proceedings of machine learning and systems 1 (2019): 374-388.

[12] Seulki Lee and Shahriar Nirjon. “Fast and scalable in-memory deep multitask learning via neural weight virtualization.” In Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services, MobiSys ’20, pages 175– 190, New York, NY, USA, 2020. Association for Computing Machinery.

[13] Peter Belcak and Roger Wattenhofer “Fast Feedforward Networks”, arXiv, 2023.