

## Deliverable

### D3.6 Software node and services architecture

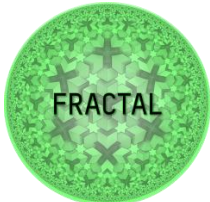
Deliverable Id:	<b>D3.6</b>
Deliverable Name:	<b>Software node and services architecture</b>
Status:	<b>Approved</b>
Dissemination Level:	<b>Public</b>
Due date of deliverable:	<b>30.9.2022</b>
Actual submission date:	<b>20.10.2022</b>
Work Package:	<b>WP3</b>
Organization name of lead contractor for this deliverable:	<b>Offcode Oy</b>
Author(s):	<b>Antti Takaluoma</b>
Partner(s) contributing:	<b>Alexander Flick, PCL2 Jérôme Quévremont, Thales Kévin Eyssartier, Thales Jaume Abella, BSC Ramon Canal, BSC Edurne Palacio, Ikerlan Lauri Loven, UOULU Leticia Pascual, Solver (SML)</b>
<p><b>Abstract: This deliverable (D3.6) is the third of a series of deliverables that describe the software work for the FRACTAL project software nodes. This is the third of three deliverables on software node, and it will be updated throughout the project with D3.2, and D3.4.</b></p>	




This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056



Co-funded by the Horizon 2020 Programme of the European Union under grant agreement No 877056.

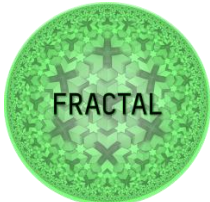
	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

History.....	4
1 Summary.....	5
List of WP3 components.....	7
1.1 Introduction.....	8
1.2 Application owner view.....	8
1.2.1 Support for business logic.....	8
1.2.2 Support for development and testing.....	9
1.2.3 Support for commissioning, deployment and provisioning.....	9
1.3 End-user view.....	9
2 WP3 and related WPs.....	10
2.1 WP3 and WP4.....	10
2.1.1 Supporting FRACTAL developments on safety.....	10
2.1.2 Supporting FRACTAL developments on security.....	10
2.1.3 Supporting FRACTAL developments on low power.....	10
2.2 WP3 and WP5.....	11
2.2.1 Supporting FRACTAL developments on AI.....	11
2.2.2 LEDEL to develop and execute AI-based models in a FRACTAL node.....	17
2.2.3 Drivers for the software diverse redundancy library (WP3T34-02).....	25
2.2.4 Driver for the Edge-Oriented Monitoring Unit (WP3T34-01).....	27
2.2.5 Driver for SIEFRACC accelerator (WP3T35-01).....	30
2.3 WP3 and WP5.....	31
2.3.1 Supporting FRACTAL developments on cognitive awareness.....	31
2.3.2 Idiom Recognition.....	34
2.4 WP3 and WP6.....	36
2.4.1 Supporting FRACTAL framework consistency.....	36
2.4.2 Supporting FRACTAL application consistency.....	36
3 Role of different platforms in FRACTAL Use Cases.....	37
3.1 Customizable node (RISC-V based PULP).....	38
3.1.1 Pulp onboard resources.....	40
3.1.2 Safety considerations on Nuttx Pulp – WP4.....	41
3.1.3 AI processing on Nuttx Pulp -- WP5.....	41
3.1.4 Application orchestration on Nuttx Pulp – WP6.....	42
3.2 Versal node.....	43

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

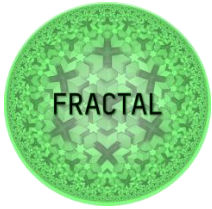
3.2.1	Versal onboard resources.....	46
3.2.2	Safety considerations on Versal – WP4 .....	46
3.2.3	AI processing on Versal -- WP5.....	47
3.2.4	Application orchestration on Versal – WP6.....	47
3.3	Other nodes .....	47
3.3.1	NOEL-V .....	48
3.3.2	ARIANE/CVA6.....	49
3.3.3	Yet additional platforms .....	50
4	Interaction of UCs with FRACTAL nodes .....	51
5	Conclusions.....	52
6	Next steps.....	53
6.1	Risks and Mitigation plans .....	53
7	Deviations from workplan.....	55
8	Bibliography .....	56
9	References .....	57
10	List of figures .....	58
11	List of tables.....	59
12	List of Abbreviations .....	60

## Contents

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## History

Version	Date	Modification reason	Modified by
0.17	11.10.2022	Fixed names and added references to figures. Fixed grammar mistakes	Leticia Pascual (SML)
0.16	10.10.2022	Formatting (removed textboxes, added captions to figures / tables, added bibliography, fixed header table, fixed footer, removed white spaces before paragraph breaks) TODO: fix references in text	Levente Tamas (OFFC)
0.15	3.10.2022	Document format refactored	Levente Tamas (OFFC)
0.14	23.7.2022	BSC contributions included	Jaume Abella (BSC)
0.13	21.6.2022	Copied to be a base for D3.6	Antti Takaluoma
0.12	30.3.2022	Final formatting changes	Antti Takaluoma
0.11	30.3.2022	Updates based on internal review and various optimization to document.	Antti Takaluoma
0.10	11.3.2022	Moving chapters	Antti Takaluoma
0.9	1.3.2022	Ready for review by Matti Vakkuri and Igor Bisio	Wp3 meeting
0.8	28.2.2022	Proposal to be approved	Antti Takaluoma
0.7	28.2.2022	Reviews and updates	Alexander Flick
0.6	25.2.2022	Added "big pictures"	Jaume Abella (BSC)
0.5	23.2.2022	Reviews and updates	Jérôme Quevremont
0.4	17.2.2022	Proposal, prepared to project internal review	Antti Takaluoma (OFFC)
0.3	31.12.2021	Draft, filled D3.2 data to new structure, yellow text to be revisited	Antti Takaluoma (OFFC)
0.2	30.12.2021	Draft, adding content	Antti Takaluoma (OFFC)
0.1	29.12.2021	Draft, basic structure revisited	Antti Takaluoma (OFFC)

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 1 Summary

The main objective of the FRACTAL project is to “create a cognitive edge node enabling a fractal Edge that can be qualified to work under different safety-related domains”. Furthermore, it is stated in the DoA that “This computing node will be the basic building block of intelligent, scalable and non-ergodic IoT”. As such the hardware node is a central part of the FRACTAL project around which 28 partners collaborate, investigate and industrial partners develop their use cases.

This deliverable (D3.6) is the third of a series of deliverables that describe the software work for the FRACTAL project hardware nodes. These documents will be delivered throughout the project with D3.2 (M12), D3.4 (M18), and D3.6 (M20). These three deliverables are also paired with the “hardware node and services” deliverables D3.1, D3.3 and D3.5.

The FRACTAL project brings together many partners (28) both from industry and academia, working on varied and challenging topics as well as eight industrial use cases. It was already a challenging task to provide a set of solutions for the hardware node in this context and combined with restrictions around COVID and worldwide supply disruptions for electronic components, partners in WP3 had to face additional challenges.

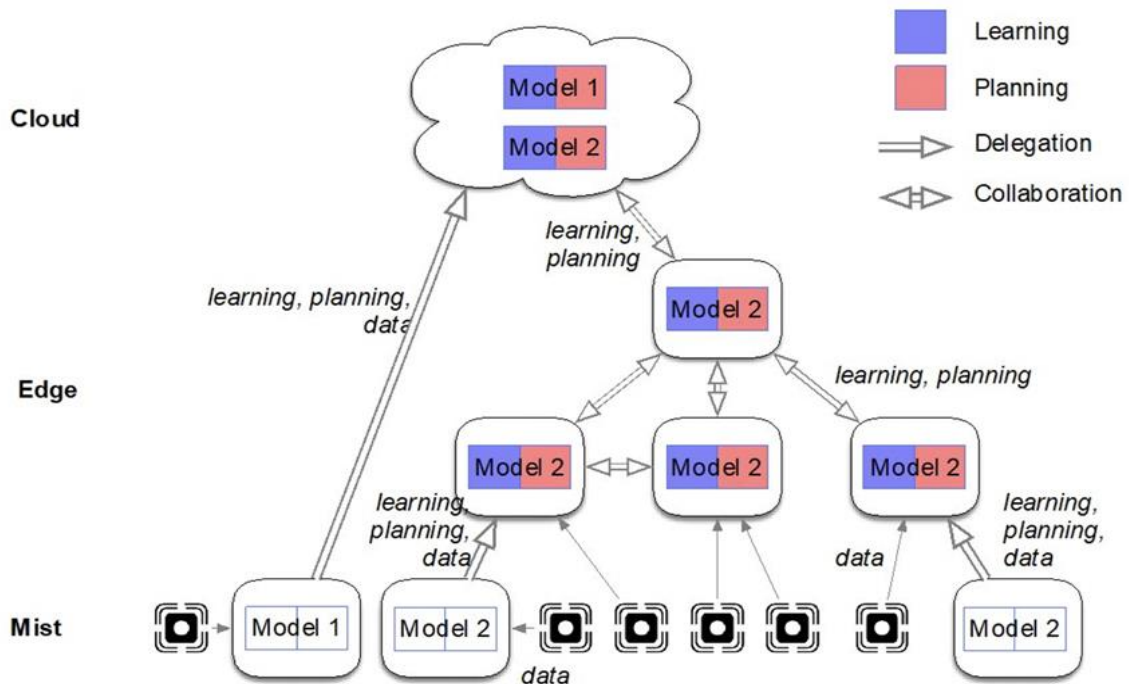



Figure 1 A schematic drawing of a possible FRACTAL system deployment using three different tiers of FRACTAL hardware nodes with different capabilities (drawing from WP5 technical meetings).

In the project the following options for the hardware nodes were used:

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

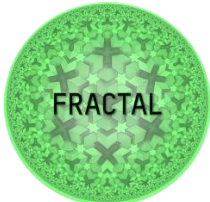
1. **Edge node** based around the Xilinx VERSAL ACAP (Adaptable Compute Acceleration Platform)
2. **Low-end node** (also as mist node) -- based around the open-source RISC-V based PULP platform

Additionally, some specific platforms (e.g., Ariane/CVA-6, NOEL-V) are used to demonstrate some specific technology concepts and use cases.

**Note**, in D3.2 also terms **customizable node** and **commercial node** were used. For now, on the Fractal domains are described by terms: **cloud, edge node** (Versal) and **low-end node** (Pulp). In case of cases where other platforms were used, their context will be clarified on text.

The organization of the deliverable is as follows.


Chapter 2 provides a general introduction to the Fractal framework and the stake holders using it. Chapter 3 summarizes WP3 technical relations to other WPs. Chapter 4 looks WP3 requirements for the Pulp platform point of view. Chapter 5 looks WP3 requirements for the Versal platform point of view. Chapter 6 introduces the special WP3 cases that were demonstrated by other platforms. Chapter 7 summaries and refers to the Fractal use cases.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## List of WP3 components

Component ID	(Sub)Component / Development Name	Partner	Deliv.	Section
<b>WP3-AI</b>				
<b>AI accelerator (hardware and software support)</b>				
WP3T32-01	HW accelerator (SIEFRACC)	SIEM	D3.5	
WP3T32-05	ML inference demo PULPissimo	ETHZ	D7.3	
WP3T32-07	Age and Gender identifier at the edge	UNIVAQ	D3.5	
WP3T32-10	VERSAL accelerator building-blocks	IKER	D3.5	
WP3T34-03	Versal Model deployment layer	PLC2	D3.5	
WP3T35-01	SW driver for HW accelerator	SIEM	D3.6	2.2.6
WP3T35-02	Accelerator Adaptation to AI library	UPV	D3.5	
WP3T35-03	LEDEL (Low Energy EDDL)	SML	D3.6	2.2.2
WP3T35-04	Deep learning based automatic iris diagnosis	MODIS	D3.6	2.2.2.1
<b>WP3-CPU/OS</b>				
<b>CPU and OS support</b>				
WP3T32-02	PULPissimo platform for IoT applications	ETHZ	D3.5	
WP3T32-02b	Ariane for Linux capable RISC-V platform	ETHZ	D3.5	
WP3T32-03	PULP training	ETHZ	D3.5	
WP3T32-04	FreeRTOS port to PULP	ETHZ	D3.6	3.1.2
WP3T32-08	Real-time aware caches	ACP	D3.5	
WP3T32-11	Smart Interrupt distribution system	ACP	D3.5	
WP3T32-12	Security services - TL2AXI adapter	ACP	D3.5	
WP3T33-03	CVA6 (former Ariane) RISC-V core	THA	D3.5	
WP3T36-01	Linux for CVA6 (former Ariane)	THA	D3.6	3.3.2
WP3T36-02	Load Balancing Module	MODIS	D3.6	2.3.1.1
WP3T36-03	Nuttx on PULP	OFFC	D3.6	3.1
<b>WP3-Safety</b>				
<b>Safety and security features for CPU</b>				
WP3T31-01	Edge-oriented monitoring unit	BSC	D3.5	
WP3T31-02	Interconnect to support Accelerators integration	UPV	D3.5	
WP3T31-03	Safety and security hardware support	UPV	D3.5	
WP3T32-06	Redundant Acceleration Scheme	UPV	D3.5	
WP3T32-09	Runtime Bandwidth Regulator	UNIMORE UNIVAQ	D3.5	
WP3T34-01	Driver for the edge-oriented monitoring unit	BSC	D3.6	2.2.4
WP3T34-02	Drivers for the SW diverse redundancy library	BSC	D3.6	2.2.3

Table 1 The FRACTAL components (according to D2.3) related to WP3. Some components will be described in D3.5 (and one in D7.3), for others the section in this deliverable is given

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 1.1 Introduction

FRACTAL is a system that offers a framework for developing modern distributed applications. Distributed applications can be executed in embedded nodes -- near the individual processes, centrally at the cloud, or as distributed into both of these domains. The supporting Framework should offer seamless connectivity, application integrity and the required security and safety. As addition the FRACTAL framework offers also an integrated AI tools also to the domains. Additionally, FRACTAL offers application specific hardware accelerations to the embedded nodes.

In the project proposal, we identified four strategic objectives of FRACTAL to reach this goal:

- **Objective 1:** Design and Implement an Open-Safe-Reliable Platform to Build Cognitive Edge Nodes of Variable Complexity. This part is mainly being addressed as part of WP3.
- **Objective 2:** Guarantee extra-functional properties (dependability, security, timeliness and energy-efficiency) of FRACTAL nodes and systems built using FRACTAL nodes (i.e., FRACTAL systems), which has determined the tasks of WP4
- **Objective 3:** Evaluate and validate the analytics approach by means of AI to help the identification of the largest set of working conditions still preserving safe and secure operational behaviors, which is the topic of WP5
- **Objective 4:** To integrate fractal communication and remote management features into FRACTAL nodes, which will be covered by WP6.

Looking outside, FRACTAL can be seen from various points of view. Most important are the end-user view and the application owner views.

## 1.2 Application owner view

Applications are developed according to the developer's business logic. Initially the business owner will assume that on the market the application will offer an added value to the end customer(s). To benefit from this added value, business owners need strategies to enter the market and secondly keep and improve this position.


### 1.2.1 Support for business logic

Framework should offer freedom to implement various application scenarios according to their business opportunity. While the business case (market) develops the application developments should be easy to deploy.

Main purpose of FRACTAL framework is to minimize this work, without limiting too much the application developer freedom.

**Fractal Use Cases described at D3.3, chapter 5.**



	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

### 1.2.2 Support for development and testing

Framework should offer tools for actual development, testing and verification. Mainly these tools are “standard” development tools, but the Framework itself should have specific tools to identify (and prevent) unwanted behavior of application specific components.

While in some extreme cases embedded electronics are custom developed, in most cases the framework should offer seamless hardware acceleration.

Yet another important aspect of distributed applications is application integrity. All parts of the application must be consistent with each other. Framework should offer tools for safe application deployment and ensure that parts of the distributed application are genuine.

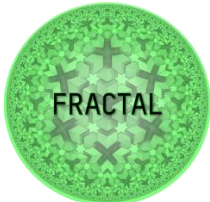
### 1.2.3 Support for commissioning, deployment and provisioning

When the end-users are attracted by the business logic, the application is ramped to a specific end-user. This may require physical installations on site and/or configurations to the cloud. Application logic – by support of the framework – should distribute the configurations to end-user specific nodes.

Some cases there may be needs to collect end-user specific information – e.g., billing of further marketing needs – this information transfer must be secure.

## 1.3 End-user view

For the end-user (or the customers of the end-user) the distributed application integrates directly to their processes (**Fractal Use Cases described at D3.3, chapter 5**). Depending on the application, availability, operational safety, and information security are typically important aspects.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 2 WP3 and related WPs

---

WP3 is focusing on the development of nodes – both the physical HWs and the necessary firmware. There are VERSAL based high-end nodes and low-end PULP based nodes. Both platforms have toolsets for application software and HW acceleration.

### 2.1 WP3 and WP4

The WP4 looks at the FRACTAL framework from point of safety -- how to manage physical defects on mechanics and electronics and the exceptional cases on software. Thus, these are highly related both to node hardware and node firmware, WP4 is deeply related to both WP3 nodes. Some exceptional cases are not necessarily possible to demonstrate on these platforms, so special platforms may be used.

#### 2.1.1 Supporting FRACTAL developments on safety

Today safety is mainly based on process and system assessments. As such, it is not a plain software feature, but more like process and documentation issue.

#### 2.1.2 Supporting FRACTAL developments on security

Two approaches are differentiated in the developments covering security related features.

##### 2.1.2.1 Linux based systems

Linux offers good tools for security. With HW support those can be strengthened to meet the requirements derived from specific use cases.

Additionally, for the VERSAL node (Linux based system as well), those use case applications that require device-level security could implement boot image encryption and authentication, functionalities that are natively supported by VERSAL.

##### 2.1.2.2 RTOS based systems

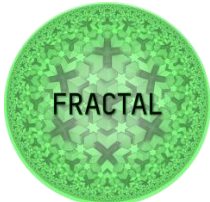
By nature, RTOS based systems have little native security features. With dedicated HW support those can be strengthened to meet the requirements, but in most cases security features are application specific implementations.

#### 2.1.3 Supporting FRACTAL developments on low power

Two approaches are differentiated in the developments related to low consumption features.

##### 2.1.3.1 Linux based systems

Linux offers good tools for low power operations. For further needs the RTOS can be utilized. RTOS runs in additional processor or preferably one of the system cores. When low power requires Linux to switch itself off, it yields the responsibility to the RTOS. RTOS keeps processing events and, when defined conditions are met, it wakes up the Linux.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

Additionally, for the VERSAL node (Linux based system as well), as shown in **iError! No se encuentra el origen de la referencia.** a centralized Platform Management Controller (PMC) that handles device management control functions is available. A flexible management control could be done through this PMC. This platform management handles several scenarios and allows the user to execute power management decisions through its framework (equivalent to what it is done in Linux, which provides basic power management capabilities like CPU frequency scaling).

However, some limitations apply. Because of the heterogeneous multi-core architecture of VERSAL, individual processors can't make autonomous decisions about power states of individual components or subsystems. Instead, a collaborative approach is taken, where a power management API delegates all power management control to the platform management controller. This PMC is the key component in coordinating the power management requests received from the other processing units, and the coordination and execution from other processing units through the power management API. This framework manages resources such as power domains, power islands, clocks, resets, pins and their relationship to CPU cores, memory, and peripheral devices.

Therefore, the natively provided power management API would be used for VERSAL node, since this platform management framework abstracts the complexity associated to administrate the power-management of a multiprocessor heterogeneous system.

### **2.1.3.2 RTOS based systems**

By nature, RTOS based systems offer good tools low-power operations. The RTOS level low-power features are typically extended with specific support by the underlying HW (processor).

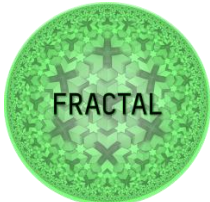
Another additional layer of low power is typically obtained by application architecture. Event-based application structure is by nature easier for low-power than applications that are coded based on infinity loops, however this is not in scope of framework.

## **2.2 WP3 and WP5**

WP5 focuses on integrating AI to the Fractal framework. While AI is in the scope of whole project, the WP3 has some special concerns related to implementation in the low resource environment.

### **2.2.1 Supporting FRACTAL developments on AI**

Several alternative methods are studied for deploying AI/ML models on FRACTAL nodes. First, a model may be pre-built, that is, trained by a third-party actor, downloaded from a public repository, and uploaded to the node. Second, a model may be learned from data available to a node, possibly augmented with annotations which indicate the expected model output for each data point. Third, a few nodes may co-operate to train a model, e.g., with a federated learning approach.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

In each case, the model may need updating due to model drift, that is, the accuracy of the model output slowly degrading. In such cases, new training data must be collected (and possibly annotated), and the model updated to reflect the data. Further, the model update cycle must be managed such that model quality is monitored and update launched when necessary. Tools and methods conducting model lifecycle management are commonly referred to as MLOps.

Inference-time, when the model is turning input data into model output, federated approaches may improve the quality of the outputs in some use cases. For example, if a number of nodes each employ an independently trained (i.e., with different data) but otherwise identical models, the models may be used as an ensemble, with the same input data fed to all of them, and the results combined into one.

WP5 is studying all above approaches in close co-operation with WP3, focusing on theoretical study of distributed learning and inference, the FRACTAL cloud platform, the architecture and orchestration of the FRACTAL network, as well as the AI methods required to fulfill the requirements of the use cases.

#### **2.2.1.1 Raw imaging preprocessing requirements for IRIS disease detection (WP3T35-04)**

To properly execute the classification algorithms for the monitoring and detection of diabetic retinopathy, three preprocessing techniques that proved to be the most effective in this field, had been chosen:

1. *Cropping*: a technique for bringing images to a square shape.
2. *Resize*: for bringing images with the same resolution.
3. *CLAHE*: advanced histogram equalization technique to improve the contrast of raw images.


Even though these techniques are well-known ones in literature, lately they are well implemented separately in a cloud environment within which, in a broad sense, there are no resources constraints. Our *new* approach is to use them all together in sequence in order to deploy the pre-processing *service* in a *commercial environment*, with resources limitations, to address specific research topics and to support the Fractal ecosystem.

These techniques are computationally expensive; therefore, it is necessary to study the limits, e.g., the minimum requirements, within which it is possible to run the preprocessing service avoiding the use of an oversized or undersized microprocessor board.

An evaluation of the average execution times was carried out to be able to evaluate the implemented Iris Recognition algorithm and be able to define the minimum hardware requirements to be able to run it correctly. This analysis was carried out on **a Zynq UltraScale + MPSoC ZCU102 board** using the entire IDRiD dataset (516 images).

The Zynq UltraScale + MPSoC ZCU102 board has the following features:

- Quad-core Arm Cortex-A53 processor

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

- CPU frequency up to 1.5GHz
- 4GB RAM Memory

To perform the evaluation, we made use of the *IDRiD dataset*. The IDRiD dataset (Indian Diabetic Retinopathy Image Dataset) contains fundus images that were captured by a retinal specialist containing 516 images. The size of each image is about 800 KB.

The preprocessing phase of an image consists, as mentioned before, of the operation of Crop, Resize, and Clahe.

We found that the average execution times, evaluated on 4 Cores of the Cortex A53 processor, for the Crop, Resize, and Clahe operations are shown in the following table:

Preprocessing Phase	Average Execution time (s)	Number of Core
Crop	0,550808249	4
Resize	0,429059386	4
Clahe	104,5466895	4
<b>Total Time (s)</b>	<b>105,5271729</b>	

Figure 2 Average execution time of Crop, Resize and Clahe algorithms on Zynq with 4-core running

The Crop operation required 0,550808249 s, the Resize operation required 0,429059386 s, and finally, the Clahe operation required 104,5466895 s. Therefore, the total average time for preprocessing phase is 105,5271729 s (1m 45s).

In detail, it was created a boxplot to display the summary of the set of data values having properties like a lower whisker, lower quartile, median, upper quartile, and upper whisker.

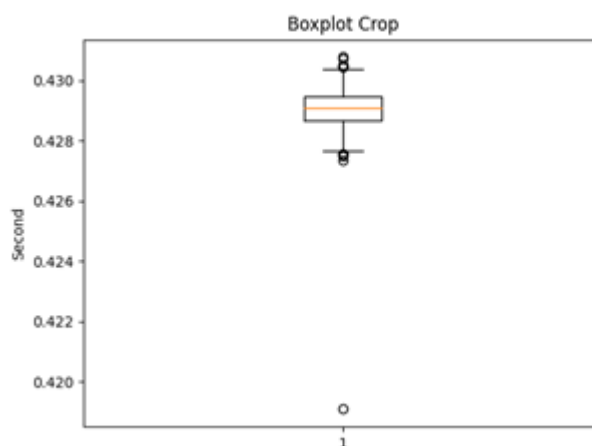
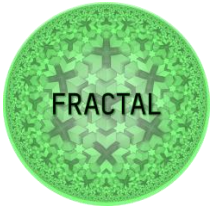


Figure 3 Boxplot for Crop Algorithm with 4-core running

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

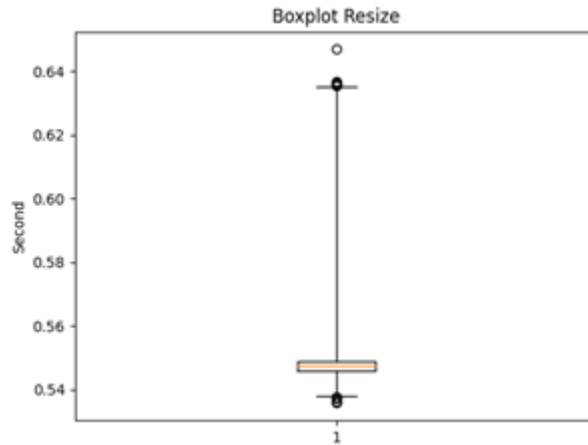


Figure 4 Boxplot for Resize Algorithm with 4-core running

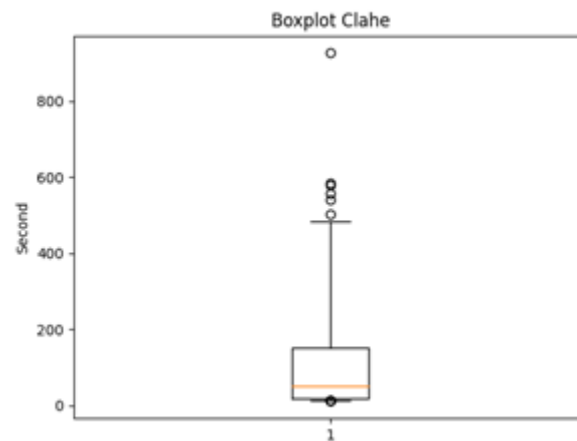
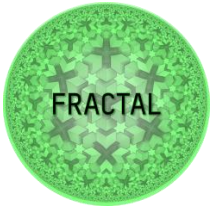


Figure 5 Boxplot for CLAHE Algorithm with 4-core running

<b>LABEL</b>	<b>LOWER WHISKER [s]</b>	<b>LOWER QUARTILE [s]</b>	<b>MEDIAN [s]</b>	<b>UPPER QUARTILE [s]</b>	<b>UPPER WHISKER [s]</b>
<b>Box Crop</b>	0.427664	0.428654	0.429077	0.429022	0.430383
<b>Box Resize</b>	0.537745	0.545758	0.547378	0.547167	0.635081
<b>Box CLAHE</b>	12.438491	16.811439	50.663041	41.275369	483.239338

Table 2 Statistical insights for algorithms running with 4-core.

The subsequent study has been conducted calculating the average execution times, with only 1 Core of the Cortex A53 running, for the Crop, Resize, and Clahe.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

Preprocessing Phase	Average Execution time (s)	Number of Core
Crop	0,55360711	1
Resize	0,429484704	1
Clah	108,0786817	1
<b>Total Time (s)</b>	<b>109,0623796</b>	

Figure 6 Average execution time of Crop, Resize and Clah algorithms on Zynq with 1-core running

The Crop operation required 0,55360711 s, the Resize operation required 0,429484704 s, and finally, the Clah operation required 108,0786817 s. Therefore, the total average time for preprocessing phase is 109,0623796 s (1m 49s).

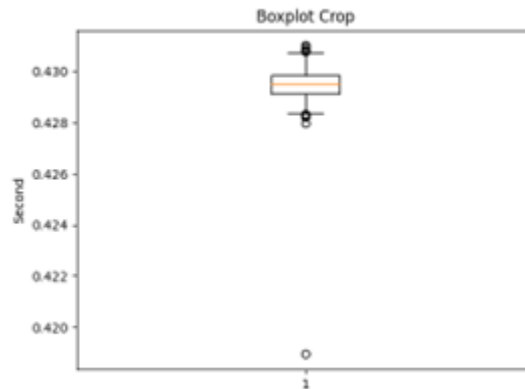


Figure 7 Boxplot for Crop Algorithm with 1-core running

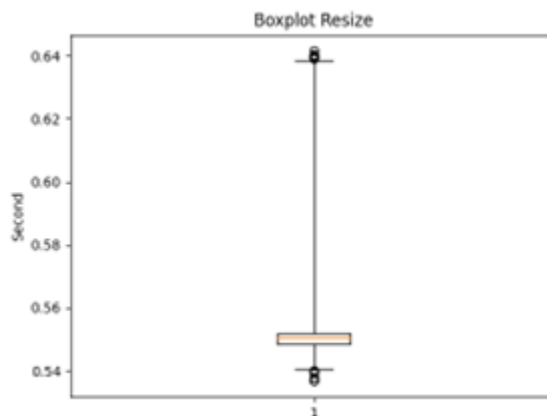
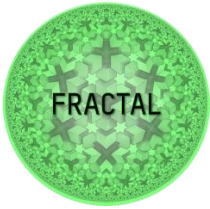


Figure 8 Boxplot for Resize algorithm with 1-core running

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

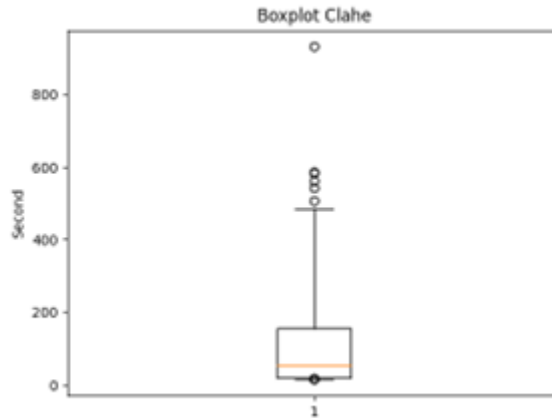


Figure 9 Boxplot for CLAHE algorithm with 1-core running

<b>LABEL</b>	<b>LOWER WHISKER [s]</b>	<b>LOWER QUARTILE [s]</b>	<b>MEDIAN [s]</b>	<b>UPPER QUARTILE [s]</b>	<b>UPPER WHISKER [s]</b>
<b>Box Crop</b>	0.428366	0.429135	0.429511	0.429462	0.430748
<b>Box Resize</b>	0.540493	0.548339	0.550389	0.550140	0.638475
<b>Box CLAHE</b>	16.173421	20.232431	54.147033	44.750463	486.486875

Table 3 Statistical insights for algorithms running on 1-core processor.

Another step studying the minimum hardware required was on the variation of the RAM memory and the number of the Cores of the Cortex A53 processor made available for the image preprocessing operation. The following table shows the results of the evaluation with the variation of RAM and with 4 Cores of the Cortex A-53 processor.

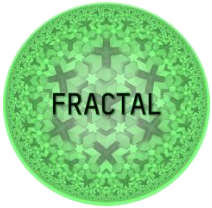
<b>RAM Memory Used with 4 CPU core</b>	<b>Result</b>	<b>Number of Image pre-processed sequentially</b>
300 Mbyte	Insufficient RAM memory	1
400 Mbyte	Insufficient RAM memory	1
500 Mbyte	Insufficient RAM memory	1
600 Mbyte	Crop error, overload resolution failed, insufficient RAM memory	1
700 Mbyte	Pre-Processing executed	1
700 Mbyte	Insufficient RAM memory	2
800 Mbyte	Insufficient RAM memory	2
900 Mbyte	Insufficient RAM memory	2
1 GByte	Pre-Processing executed	2
1 Gbyte	Pre-Processing executed	2+

Table 4 Minimum RAM memory required if 4-core running

It is noteworthy how we obtained different results increasing RAM memory and number of images to be processed. In particular:

- For only one image to be processed 700MB of RAM is enough.



	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

- For two or more images to be processed sequentially it needs 1GB RAM.

The following table shows the results of the evaluation with the variation of RAM and with 4 Cores of the Cortex A-53 processor.

RAM Memory Used with 1 CPU core	Result	Number of Image pre-processed sequentially
300 Mbyte	Insufficient RAM memory	1
400 Mbyte	Insufficient RAM memory	1
500 Mbyte	Pre-Processing executed	1
500 Mbyte	Insufficient RAM memory	2
600 Mbyte	Pre-Processing executed	2
600 Mbyte	Pre-Processing executed	2+

Table 5 Minimum RAM memory required if 1-core running

It is possible to notice how we obtained different results as it increases RAM memory and the number of images to be processed. In particular:

- For only one image to be processed 500MB of RAM is enough.
- For two or more images to be processed sequentially it needs 600MB RAM.

From this evaluation, it comes out that the minimum requirements for preprocessing a large image dataset with Crop, Resize and CLAHE algorithms are listed in the following table:

Number of cores required	RAM memory
1 Core Cortex A-53 processor (or similar)	1GB

In order to provide full operability and access to these features to any application, we arranged the algorithms in a docker service. It can be executed from any application with the following command:

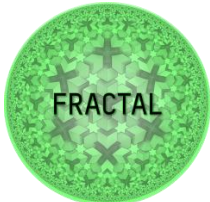
- `docker run modis_irisrecognition_app`

The logic underneath is pretty simple, taking an image as input, processing sequentially this raw image with Crop, Resize and Clahe algorithm and produce a preprocessed image as output.

### 2.2.2 LEDEL to develop and execute AI-based models in a FRACTAL node

The aim of this work is to port EDDL to be used in a RISC-V architecture, so machine learning algorithms can be executed in such hardware.

There has been attempts to extend the RISC-V Instruction Set Architecture with the goal of accelerating the inference process of CNN. In this attempts a new processor, extending an already existing processor with the new set of instructions, was built into an FPGA,

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

where its performance was tested showing a x2 performance improvement using TensorFlow Lite models [<https://ieeexplore.ieee.org/abstract/document/9071197>].

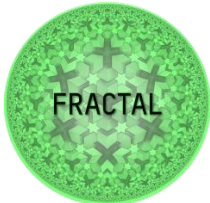
Another work extending the instruction set of RISC-V architecture to allow TensorFlow Lite cross-compilation and execution [[https://www.researchgate.net/publication/339298701\\_Towards\\_Deep\\_Learning\\_using\\_TensorFlow\\_Lite\\_on\\_RISC-V](https://www.researchgate.net/publication/339298701_Towards_Deep_Learning_using_TensorFlow_Lite_on_RISC-V)].

During the RISC-V Summit of 2019, LG presented the LG Neural Engine (LNE), a processor with customized ISA extensions of RISC-V to support neural network functions. The LNE software framework can support the inference of trained models in Caffe, TensorFlow and PyTorch [<https://riscv.org/wp-content/uploads/2019/12/12.10-15.50a-Scalable-Configurable-Neural-Network-Accelerator-Based-on-RISC-V-Core.pdf>].

Another interesting work a Neural Network framework base on the RISC-V architecture is develop, achieving the execution of lightweight models with accuracy results like the same models run using Keras [[https://link.springer.com/chapter/10.1007/978-3-030-49556-5\\_8](https://link.springer.com/chapter/10.1007/978-3-030-49556-5_8)].

### ***2.2.2.1 LEDEL in the FRACTAL project***

In Figure 10 LEDEL development in FRACTAL we can observe the scope of the task in WP3 in the context of LEDEL in the FRACTAL project, which is the adaptation of the EDDL to become LEDEL. Such adaptation consists of compiling the EDDL in a RISC-V platform, reassuring all the libraries and dependencies that it needs are also available and fully functional in the reduced instruction set architecture. The platform chosen for this aim is NOEL-V. This platform is scheduled to be available before the end of the year.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

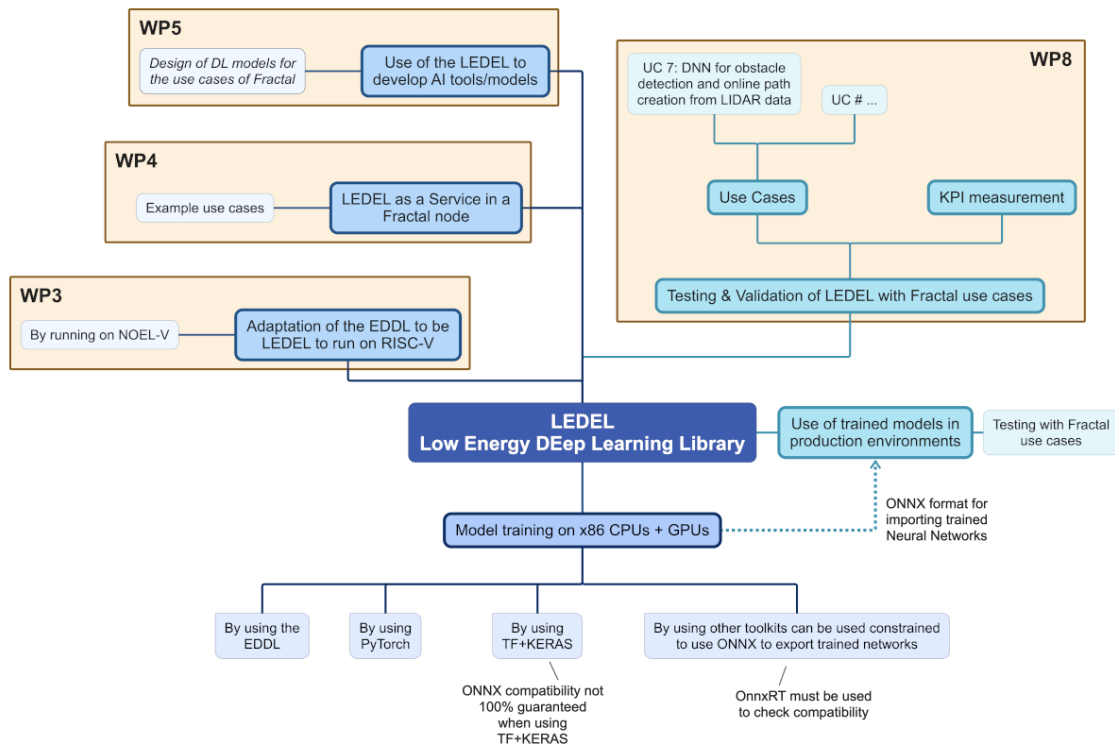


Figure 10 LEDEL development in FRACTAL

Thus, in order to check that the LEDEL could be ported to this hardware, we have used an emulated environment for the RISC-V architecture based on QEMU software. For this purpose, we have used an already created and compiled Linux Debian image named "Artifacts"

[https://gitlab.com/api/v4/projects/qiomasce%2Fdqib/jobs/artifacts/master/download?job=convert\\_riscv64-virt](https://gitlab.com/api/v4/projects/qiomasce%2Fdqib/jobs/artifacts/master/download?job=convert_riscv64-virt)

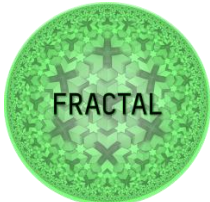
from the project repository

<https://gitlab.com/qiomasce/dqib#debian-quick-image-baker-dqib>

Once the image has been installed and running, EDDL has been compiled in this RISC-V virtualized environment. All the dependencies work completely fine. And a few simple tests have been executed checking their proper behavior.

One can train a model using the EDDL on a computer without limitation of resources and export it using the ONNX format. Afterwards, the model can be imported by the LEDEL in a FRACTAL node, and then used to infer from data received in the node.

As an example, it has been possible to train a simple model for the MNIST digit dataset, and then use it for inference. Obviously, as all infrastructure is being emulated, this execution process has been quite slow. Also, it has been possible (i) to train this model

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

using an “outside” computer, (ii) to save it in ONNX format, (iii) to import it in the emulated machine and (iv) to infer.

The use of this pre-baked image of Linux running on RISC-V emulated platform has allowed us to check if the portability of the EDDL to this architecture was possible. Furthermore, it has given us the advantage of moving forward with T4.1 (LEDEL as a service in a FRACTAL node), and now we are able to test the deep learning model for the UC7.

This intermediate solution has been documented and packed using a docker and is available in the repository of the FRACTAL project: [<https://github.com/project-fractal/WP3/tree/main/Components/WP3T35-03%20LEDEL>].

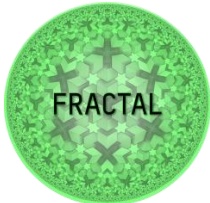
### ***2.2.2.2 Presentation and preparation of the emulated NOEL-V platform***

We have used the software named isar-riscv provided by our partner Siemens in <https://github.com/siemens/isar-riscv/blob/main/README.md>.

The requirements of such software need to be installed and deployed in a Debian Linux System. To this aim, a docker container with the Debian GNU/Linux 11 distribution has been prepared. In it the isar-riscv solution has been deployed, following the instructions from the official repository.

We have also prepared a manual in a more detailed manner to prepare all the infrastructure needed to use the LEDEL, as well as to instruct how one can install and use the library to create ML algorithms in the FRACTAL node [<https://github.com/project-fractal/WP3/tree/main/Components/WP3T35-03%20LEDEL>]. We basically have followed the official documentation with the addition of a couple of extra steps for a better and friendlier performance. It should be noted that we can custom upgrade the docker file to our taste or necessities.

Since the EDDL installation is already explained in the official documentation and, also, in the user manual, in this document we will focus on how the LEDEL works in a RISC-V based architecture machine, emulated in this case. Due to the fact that we have reduced space, the simplest and safest way to achieve this goal is to download the original code of the

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

library and compile it. As said before, all steps are well documented in FRACTAL project Github repository.

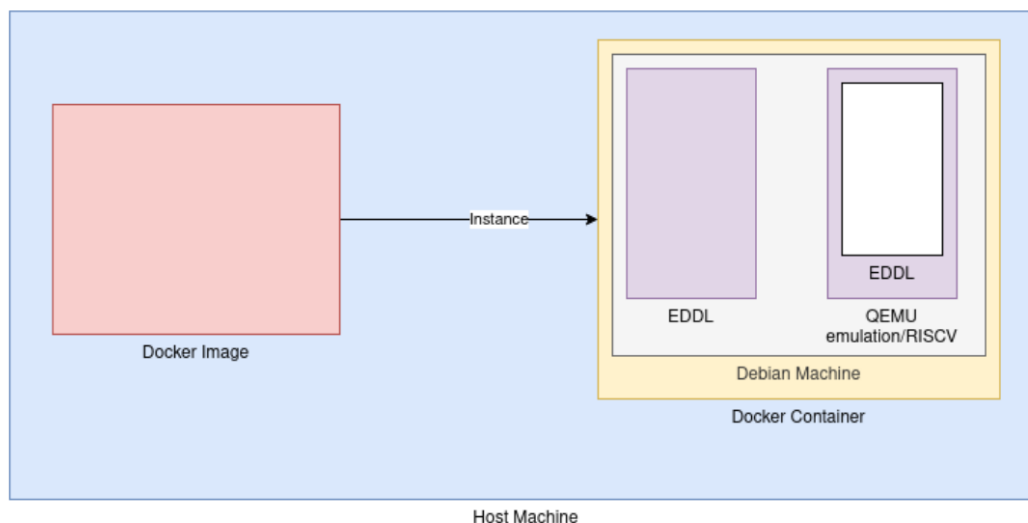


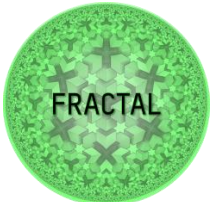
Figure 11 Abstraction of the docker file presented

Finally, Figure 11 Abstraction of the docker file presented, shows the content of the docker container created to mount the LEDEL infrastructure. We have installed the EDDL library to write and compile algorithms using it. Later, they can be exported using ONNX and transferred to the FRACTAL node. In this case, the QEMU emulation of RISC-V acts as the FRACTAL node. After creating the ONNX model, the file created is imported inside the RISC-V emulation to be tested and executed correctly using a network loader implemented with what is already LEDEL. We will refer to the emulation process as the FRACTAL node since it is the exact same process to follow once the FRACTAL node hardware is available.

The steps followed to achieve this are related in more detail below. Basically, we have trained two simple models using two different technologies. The first one is implemented using EDDL library. It consists of a simple neural network implementation that trains using the MNIST dataset. Code used to illustrate this example can be accessed following the link: <https://deephealthproject.github.io/eddl/usage/intermediate.html>

After the training, the ONNX file is created with the weights in a .bin file. We can observe in Figure 12 Example of training process using LEDEL in RISC-V, the results of the training process.



	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

```

#include <cstdio>
#include <cstdlib>
#include <iostream>

#include "eddl/apis/eddl.h"
#include "eddl/serialization/onnx/eddl_onnx.h"

using namespace eddl;

int main(int argc, char **argv) {

    // Load dataset
    Tensor* x_test = Tensor::load("mnist_tsX.bin");
    Tensor* y_test = Tensor::load("mnist_tsY.bin");

    // Preprocessing
    x_test->div_(255.0f);

    // Load net
    Net* net = import_net_from_onnx_file("mlp_mnist.onnx");

    // Build model
    build(net,
        rmsprop(0.01), // Optimizer
        {"soft_cross_entropy"}, // Losses
        {"categorical_accuracy"}, // Metrics
        //CS_GPU({1}) // one GPU
        //CS_GPU({1,1},100) // two GPU with weight sync every 100 batches
        CS_CPU(),
        //CS_FPGA({1})
        false //Disable model initialization, since we want to use the onnx
        //weights
    );

    // View model
    summary(net);

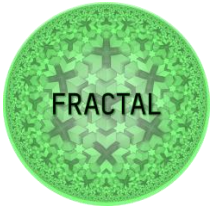
    // Evaluate
    evaluate(net, {x_test}, {y_test});
}

```

Figure 13 Code used to load the ONNX file using the LEDEL





	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

To sum up, we have created three docker containers to illustrate and prove that we can build and use the EDDL library to become the LEDEL, being compiled, installed, and used in two different emulated environments, artifacts first and isar-riscv later. All the code and details carried out as well as a manual for its different uses are available in the FRACTAL project dedicated to this component in WP3.

Once we had all these containers created, they allowed us to perform the validation of the LEDEL library, which is documented in D4.2 from WP4, and prepare the tests for UC7.

Unfortunately, we could not try the LEDEL in real hardware platform NOELV, since the chipset was not available to us.

### 2.2.3 Drivers for the software diverse redundancy library (WP3T34-02)

BSC's software-only diverse redundancy support builds upon a monitor process creating redundant instances of the application to be run with diverse redundancy (see Figure). In particular, the monitor process spawns the redundant execution of the application in two cores, one thread the head one, and the other the trail one. The monitor guarantees that the head thread is at least a given number of instructions ahead of the trail thread, where such number is platform dependent and must be large enough so that the trail thread cannot catch up with the head one between two consecutive checks of the monitor process. The monitor checks periodically the progress of the head and trail threads, and if, eventually, the trail thread is too few instructions behind the head process, the monitor stalls the trail process until the next monitoring check. When, eventually, the staggering (in terms of instructions) between the head and the trail is large enough, or if the head trail finishes its execution, the monitor allows the trail thread to resume execution. This guarantees that the state of the cores where redundant processes run differs at any time, and hence, a fault affecting both cores similarly will produce different errors that will be detected upon comparison of the outcomes of the head and trail threads

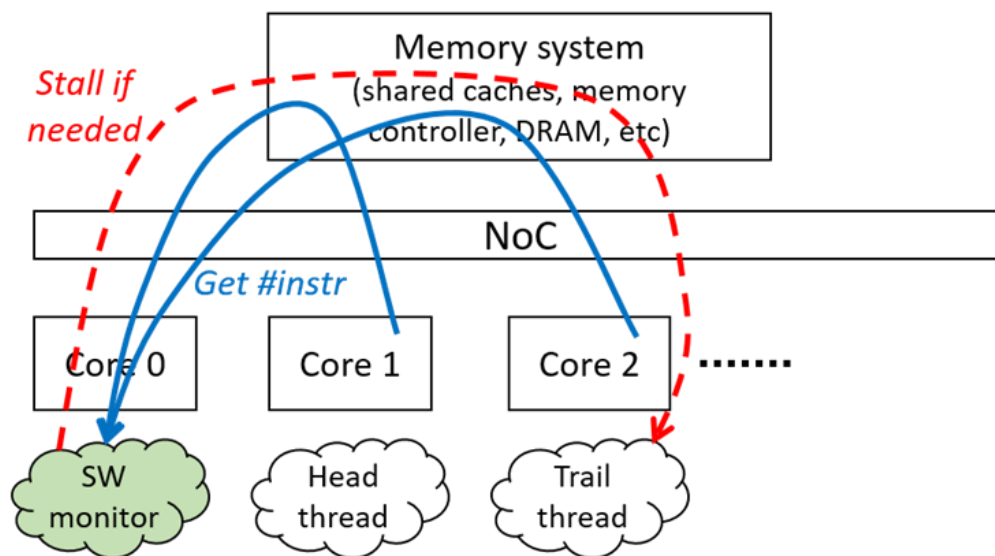



Figure 16 Figure: A schematic of the software-only support for diverse redundancy.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

BSC software-only diverse redundancy support is deployed on top of the NOEL-V based platform and is intended to run as a Linux library. This type of service has been prototyped in the past for Arm-based platforms with Ubuntu Linux distributions<sup>1</sup>. Hence, the challenge in FRACTAL is threefold:

1. Porting this service from Arm to RISC-V using a different infrastructure (i.e., a FPGA board interfaced through a host instead of a directly accessible ASIC-based platform), and a different Linux distribution (*Buildroot* instead of Ubuntu).
2. Generating a standalone library easing the integration in use cases, rather than resorting to handcrafted prototyping in ad-hoc experiments as done in previous work.
3. Validate the implementation against a few relevant test cases prior to its integration in any of the FRACTAL use cases.

Those steps span across WP3 and WP4. In particular, the work in WP3 relates to the porting of the basic functionalities on which to build the service, whereas work in WP4 is restricted to the use of those basic functionalities to deliver the service itself.

The first step, namely the porting of this feature from Arm to the particular target RISC-V platform consists of porting the following functionalities: (a) a call to spawn a new thread in a remote core, which will be invoked by the monitor process to create the head and trail threads; (b) a call to reset the instruction count of a remote core, where either the head or trail thread runs; (c) a call to retrieve the number of instructions executed in a remote core, i.e., the cores where the head and trail threads run; and (d) calls to stop and resume the execution of the trail thread, which runs in a remote core. Progress so far has led to the successful porting of those calls, which show to work properly. Building the service on top of those calls is part of WP4.


The second step is mostly within the scope of WP4 and, in the context of WP3, only requires validating that the calls in the first step can be properly encapsulated as part of a library, which we have already validated.

The third step, namely the validation of the overall service, falls within the scope of WP4. However, in the scope of WP3 we have the validation of the individual calls, as well as the tailoring of the staggering (in terms of instructions) between the head and trail threads to guarantee that the trail thread cannot catch up with the head thread. The latter, tailoring, has already been performed successfully. The former, namely the validation of the calls, has been successfully completed, but further tests are being conducted as part of the integration of this component with UC7.

**Related work:** ASIL D compliant ST Microelectronics SPC56XL70 and Infineon AURIX processor family implement Dual-Core LockStep (DCLS), whereas some Arm Cortex-R5

---

<sup>1</sup> S. Alcaide et al. Software-only based diverse redundancy for ASIL-D automotive applications on embedded HPC platforms. In DFT, 2020

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

designs implement Triple-Core Lockstep. However, half of the cores are not visible for the user, who cannot use them to run independent (non-critical) applications.

Some recent work attempts to provide the advantages of DCLS, but allowing cores to be used independently whenever needed<sup>2</sup>. Such work, referred to as SafeDE, while highly efficient, imposes hardware changes. A software-only counterpart has been presented recently<sup>3</sup>. However, it has only been hand-crafted for specific programs, thus not being usable for other applications, and its target was Arm cores.

1. Our work provides a library implementing the latter without hardware support, with a simple software interface, and able to run on Linux. In particular, component WP3T34-02 provides the drivers needed for such implementation to access instruction counts remotely and stall a core whenever needed.

#### 2.2.4 Driver for the Edge-Oriented Monitoring Unit (WP3T34-01)

As it will be shown in D4.4, the edge-oriented monitoring unit (WP3T31-01) is used to provide timing monitoring capabilities. The service in D4.4 (WP4T43-01) builds on the following main functions that must be provided by the driver:

- `pmu_counters_disable()`
- `pmu_register_events()`
- `pmu_counters_reset()`
- `pmu_counters_enable()`

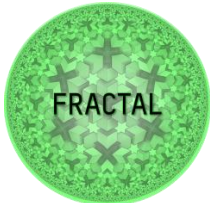
The driver, which can be found in the following public repository ([https://gitlab.bsc.es/caos\\_hw/riscv/linux\\_driver\\_safesu/-/tree/main](https://gitlab.bsc.es/caos_hw/riscv/linux_driver_safesu/-/tree/main)), implements the aforementioned functions, as well as other functions needed for the use and configuration of the WP3T31-01 component.

```
void pmu_counters_enable(void) {
    if(!driver_fd) open_pmu_driver();
    write_pmu_reg(PMUCFG0, read_pmu_reg(PMUCFG0)|0x00000001);
#ifdef __PMU_LIB_DEBUG__
    printf("Enable counters\n");
    printf("CFG0 = 0x%08x\n", read_pmu_reg(PMUCFG0));
#endif
}
```

Figure 17 Source code of the `pmu_counters_enable()` function.

<sup>2</sup> F. Bas et al. SafeDE: a flexible diversity enforcement hardware module for light-lockstepping. In IOLTS, 2021.

<sup>3</sup> S. Alcaide et al. Software-only based diverse redundancy for ASIL-D automotive applications on embedded HPC platforms. In DFT, 2020

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

It is not the goal of this section providing the full implementation in the driver. In fact, for component WP3T34-02 in previous section, we do not even provide the code explicitly. However, in the case of the driver for WP3T31-01, since it is a new hardware component instead of a driver building on existing platform features, we provide lower level details. For illustration purposes, Figure 17 shows the code of the `pmu_counters_enable()` function, part of the driver. As shown, it mostly builds upon the functions `write_pmu_reg()` and `read_pmu_reg()`, which receive as parameter the register to be written/read, and in the case of the former, also the value to be written. Those two functions are the basis for most of the functions in the driver, and they provide a simple interface to access physical registers of the WP3T31-01 component.

```

inline static unsigned int read_pmu_reg(unsigned int offs) {
    unsigned int value;
    lseek(driver_fd, offs, SEEK_SET);
    read(driver_fd, &value, 4);
    return value;
}

inline static void write_pmu_reg(unsigned int offs, unsigned int value) {
    lseek(driver_fd, offs, SEEK_SET);
    write(driver_fd, &value, 4);
}


```

Figure 18 Source code of the `read_pmu_reg()` and `write_pmu_reg()` functions.

The source code of those functions is depicted in Figure 18. As shown, building on the open file pointer `driver_fd`, which points to the memory range where the corresponding registers are mapped, those functions shift the pointer to the corresponding register and perform the read or write operation requested.

Overall, as shown, the driver has been developed using simple functions that, ultimately, reduce the risk of experiencing bugs and ease validation, as needed in safety-related systems.

**Related work:** The relevant related work for this component is already given in D3.5 for component WP3T31-01.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

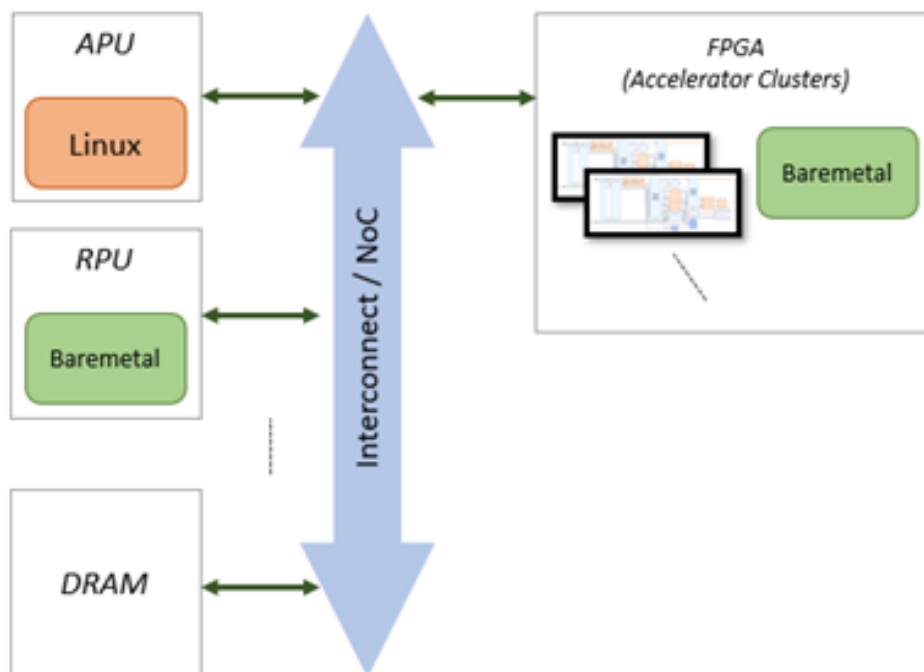


Figure 19 Software architecture for memory interference study


### 2.2.5 Support for the Runtime Bandwidth Regulator (WP3T32-09)

In order to simplify our analysis process on APU, we compiled a Linux kernel image based on *PetaLinux*. We decided to combine the *PetaLinux* system with a custom root file system based on the Ubuntu 20.04.2 distribution to take advantage of its rich ecosystem of software packages. To quantify the interference on host cores, we implemented two micro-benchmarks, which can carry out sequential and random memory traffic patterns towards the DRAM.

Both benchmarks are tuned to maximize the number of cache misses, to ensure the issued requests are in fact serviced from the DRAM (and not intercepted by the cache hierarchy). For the sequential access pattern, the memory reads are performed with stride equal to the L2 Cache Line Size. For the random-access pattern, the stride is randomic, but always a multiple of the cache line size. Typically, this pattern exhibits a higher average miss latency, as the prefetching mechanisms in the DRAM itself (e.g., row buffers) are bypassed.

These two memory access patterns represent the worst case for realistic patterns that can occur in a real-life scenario. Our micro-benchmarks are modeled after the *lmbench* test suite <http://lmbench.sourceforge.net/>.

As for the RPU interference study, we used the same benchmarks used for APUs, but recompiled for bare metal. Finally, for the SmartDMA component, used in this case as a traffic generator, we implemented a simple standalone application, which allows the softcore to control the DMA.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

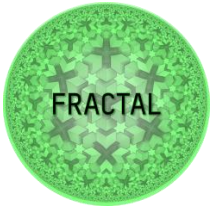
### 2.2.6 Driver for SIEFRACC accelerator (WP3T35-01)

As part of the Fractal Edge Node a hardware accelerator for execution of AI's convolutional layers is available. The algorithm of the driver that manages the hardware accelerator is shown in Figure 20.

The first step of the algorithm is to call ***init\_RAM()*** and ***init\_DMA()*** functions sequentially. The DMA has reserved memory locations inside the RAM that are non-cacheable. These memory locations are used by the application for locating the data that are transferred by the DMA from one memory location to another. The first function ***init\_RAM()*** has the responsibility to define new pointers for each of these locations in RAM and to assign the virtual address of the pointers to the physical address space. This step makes the physical memory location reserved for DMA to be accessible also by the application. The second function ***init\_DMA()*** resets the DMA control registers and puts it into a state to be ready for new data transfers.

The AI application can take images of different size as input and stores them in different memory locations. Therefore, the ***locate\_image()*** function is called to define a pointer that points to the memory location of the image and defines another variable for the size of the image. These two variables are used as input arguments for the following ***image\_transfer()*** function. The job of this function is to relocate the image from cacheable memory location in RAM to the previously defined non-cacheable memory location reserved for DMA and the predetermined weights used for convolution operation from application memory space to the DMA reserved location.

Once the above functions are executed, the driver enters a loop that transfers partially the image and the weights to the hardware accelerator, triggers and controls the hardware accelerator and as the last step it returns the results from the hardware accelerator to the system memory. The loop consist of ***DMA\_write()***, ***activate\_HWACC()***, and ***DMA\_read()*** functions. The first function configures the DMA to transfer part of the image from system memory to the local memory of the hardware accelerator. Next, the function performs the same operation to transfer the weights from system memory to the local memory of the accelerator. When both transfers are finished, the ***activate\_HWACC()*** is called to start the accelerator to perform the convolution operation on the part of the image that was previously transferred. While the accelerator performs the convolution, the ***activate\_HWACC()*** function enters an infinite loop and periodically checks the status register of the accelerator. When the convolution operation is done, the accelerator writes the results into its local memory and changes the state of its status register. This triggers a condition in a loop that enables the loop to exit iterations and step to the next function. The last function ***DMA\_read()*** transfers the results from local memory of the hardware accelerator to the system memory. The loop iterates continuously until the whole picture has gone through the convolution process. Within the loop a set of variables is defined to keep track of the actual size of the image that is left to be transferred. In addition, a pointer to the location of the rest of the image that needs to transfer is continuously updated, as well as pointers to the weights and the location of the resulting image in the system memory.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

**Related work:** This component is strongly related to component WP3T32-01 - HW accelerator (SIEFRACC). Both can only be used in combination.

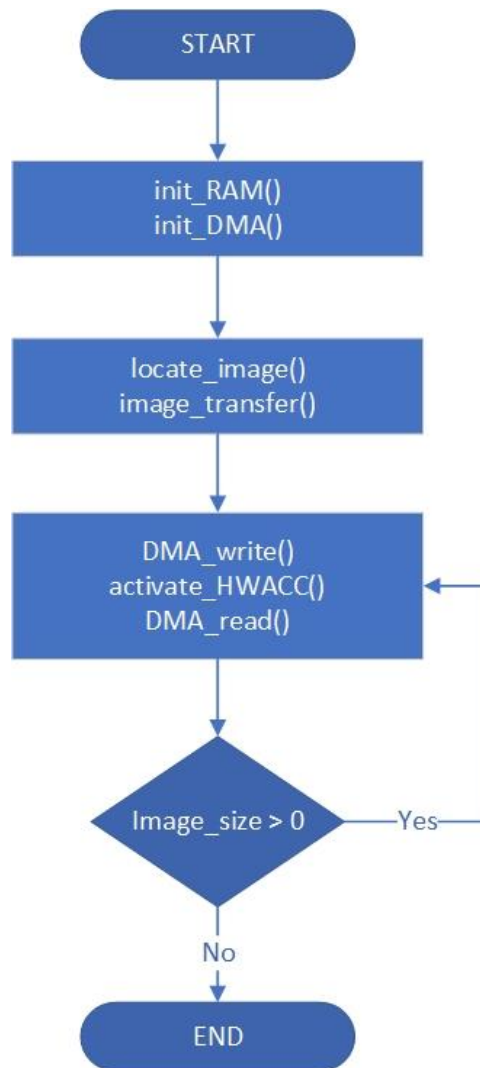



Figure 20 Algorithm for SIEFRACC hardware accelerator driver

## 2.3 WP3 and WP5

### 2.3.1 Supporting FRACTAL developments on cognitive awareness

There may be some software services for providing cognitive awareness to FRACTAL nodes. Those components may include libraries, drivers or software blocks to interface the hardware accelerators implemented in the nodes, which may be connected over different interfaces to the main processing unit (e.g., AXI/APB, shared memory).

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

With reference to the accelerator for age and gender recognition under development to be part of FRACTAL nodes, the application will be composed by the model and a Flask python server to provide REST API to external services and machines. All the software services will be packed inside a single docker image ready to acquire images and return predicted values. The only requirement to run the services will consist in the availability of the docker daemon in the operating system, together with the required hardware resources to load the model in main memory and to perform the inference.

### **2.3.1.1 The Load Balancer service as support for cognitive awareness (WP3T36-02)**

The Load Balancer (LB) literature typically copes with techniques to divert internet traffic to not-overloaded nodes to avoid traffic congestions in distributed ecosystems. This concept applies an OSI level 3, but this is not suitable if we would address specific application-level topics.

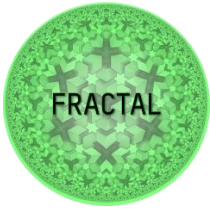
In these cases, the more usable service of load balancing is the simplest, and then we are going to use the *notion* of Load Balancer to provide (or better "to get") *the identity of node less computationally busy (loaded)* within a specific network. Within this scope, the load balancing will be general enough to be used as a dockerized service, in order to expose it to any high-level applications. Its use will be very simple: making a simple request and receiving a node identifier to which the node itself may send computational tasks. In this way, the load balancer becomes a utility service for all applications, simply changing the specific "formula" on which the calculus is made.

The LB then become a software service packed in a docker container able to identify the less busy node in the network able to perform a specific and predetermined task: for this implementation it is specifically used for the a&g recognition task. This ability falls into the cognitive-awareness capabilities in order to gain quick access to the a&g results when a specific node is over-loaded, and for collaborative purposes.

Having said that, the LB component represents therefore a new and innovative way to think and implement the concept of "load balancing".

The load balancing component works as described in the following diagram:



	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

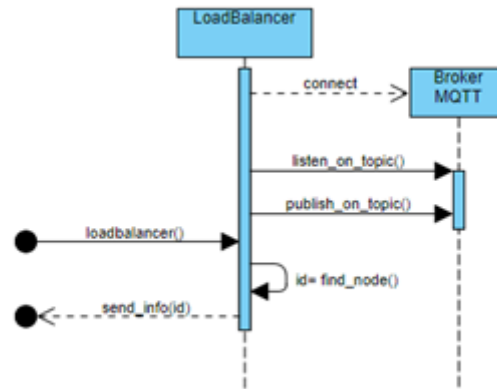


Figure 21 Sequence diagram for using LB service

The control flow can be summarized as follows:

1. Configuration of the node (*ID, IP, PORT*)
2. Configuration of the REST API (*IP, PORT*)
3. Configuration of the MQTT Client (publish/subscribe on Topic)
4. Evaluate the CPU and Memory parameters
5. Send the useful information, through REST API, to perform the load balancing

After initializing and configuring an MQTT client-server and API REST, client workload evaluation metrics have been defined and stored in "node resource table". That metrics will be used to define whether a client is not very busy, busy, or very busy. They were formatted in a JSON dictionary and later published on the MQTT Topic.

The computational load is evaluated using the average weight of the values that were collected before:

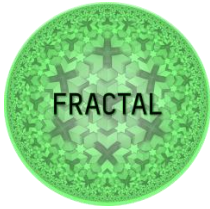
Where:

- CPU: CPU usage, defines a percentage of CPU usage
- MEM: Memory usage, defines the use of memory, compared to the total memory
- APR: Active process, defines the number of active processes.
- $p\{CPU, MEM, APR\} = \{CPU, MEM, APR\}$  parameter
- $w\{CPU, MEM, APR\} = \text{weight of } \{CPU, MEM, APR\}$  parameter

In load balancing procedure the node with the lowers computational load will be chosen, therefore with the lowest "average load" parameter.

The Load Balancer component interfaces with the other applications using the following interfaces:

Interface	Type	Description
-----------	------	-------------

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

LB/id_node	REST API	entry point to find the node that will perform the computation.
LB/reading	MQTT topic	It publishes the node resources and receive the resources of the other nodes.

### 2.3.2 Idiom Recognition

The Idiom Recognition (IR) module is used to perform automatic language recognition based on speech processing. The module outputs the language of the current speaker by capturing and analyzing an input audio stream.

The main objective of the IR module is to enable the Intelligent Totem with language recognition in order to automatically provide customized content to users in their spoken language. Currently supported hot-words and corresponding languages are:

- *italiano* | *benvenuto* [ ITALIAN ]
- *english* | *welcome* [ ENGLISH ]
- *deutsch* | *willkommen* [ GERMAN ]
- *français* | *bienvenue* [ FRENCH ]
- *español* | *bienvenidos* [ SPANISH ]

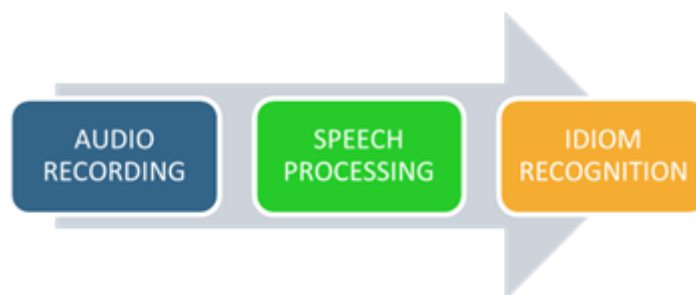
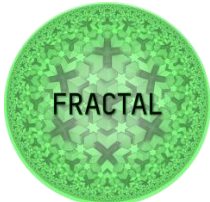


Figure 22 The IR module process: from input to output

The IR module can either perform real-time recording or take as input an already stored audio speech file to process. Machine learning based Speech-to-Text (STT) solutions are then used to obtain the transcript of the audio file and a keyword search algorithm is employed to detect specific hot-words inside the speech sentence. Database search and matching is finally employed to identify the language corresponding to the recognized hot-word. Information about the detected language is used by the Intelligent Totem to provide contents to the user in his/her native language. The specific working stages are reported in Figure Y.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

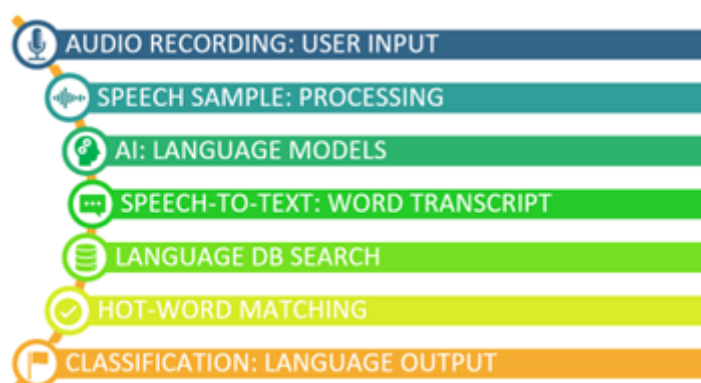


Figure 23 IR module working stages

The IR module needs a speech sample to process and output the corresponding language. The specific approach leverages existing speech processing and STT libraries to obtain the transcript out of the recorded audio speech file. In particular, the Vosk speech recognition API is used to identify words inside the audio sample and ad-hoc language models are employed to obtain the speech transcript. Vosk compatible language models are lightweight and pre-trained with thousands of hours of speech data. Once the transcript has been extracted, the recognized words are processed by a keyword detection algorithm to identify potential hot-words related to the supported languages. Pre-defined hot-words are stored in a database, which is hence queried for hot-word matching. The IR module output (i.e., the recognized language) is then sent to the Runtime Manager for interaction with other Intelligent Totem modules. In case of congestion, a recording of the audio speech sample is sent to the Runtime Manager for load balancing purposes.

A diagram showing data flow is reported in Figure Z.

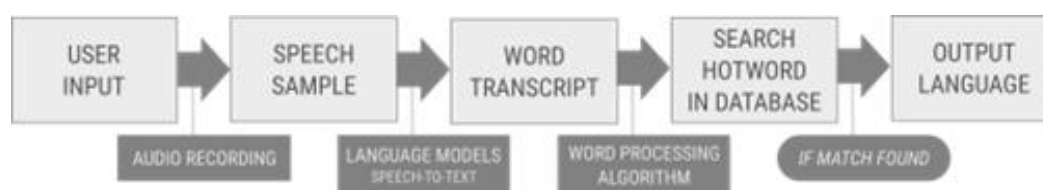
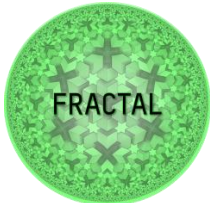


Figure 24 IR module data flow diagram

The IR module exploits transparent interactions with the user, employing data collected from user actions. In particular, the user is invited to speak by the Intelligent Totem, and the provided speech is recorded in order to be processed by the IR module. The outcome of data pre-processing, such as the audio transcript, are available to be presented to the user. The final output of the IR module is the recognized user spoken language, which is then used by the Intelligent Totem to provide customized contents. Hence, the AI results are available to the user in terms of language output and, in case the system is not able to recognize the idiom, the user is informed and invited to speak again.

In the IR module the interaction with the user is minimal. As described above, the output of the building block is the recognized user language. Therefore, the result is easily understandable by the

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

user, who is able to visualize the transcript of its speech to assess if the IR module provided a correct processing and decision.

## 2.4 WP3 and WP6

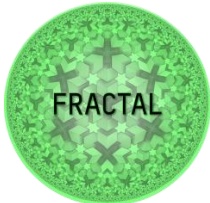
WP6 focuses on safe and secure orchestration of the distributed application domain -- how to manage the deployment and management of the distributed application and the data. Like WP5, also here the WP3 nodes need to offer required interfaces and resources while there are limitations on platforms.

### 2.4.1 Supporting FRACTAL framework consistency

WP6 will introduce methods for ensuring the framework consistency. How to ensure that all components of the framework are always consistent between each other. And how the framework raises and processes exceptions if any inconsistency happens.

### 2.4.2 Supporting FRACTAL application consistency

WP6 will introduce methods for ensuring the applications in the framework are consistent. While keeping the complex framework consistent, the evolving application brings another layer of potential problems in play.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

### 3 Role of different platforms in FRACTAL Use Cases

The comprehensive discussions with all FRACTAL partners during the preparation of D2.1 “Platform specification (a)” showed a number of issues with our initial approach regarding how FRACTAL software nodes will be demonstrated as part of the use cases.

Work on Pulp SW nodes focuses on developing a software system for the low-end FRACTAL platforms. These will be limited in memory and storage but will benefit from price and energy consumption. Readers must understand that these FRACTAL low-end platforms enable a huge market segment of devices that cost a few euros and/or run years with a set of AAA-batteries.

As stated, the **Commercial Node** software components are already provided by Xilinx. Therefore, this document gathers the information related to those software components that may need some customization or integration effort to be adapted to FRACTAL node requirements.


It is a fact that the **Customizable Node** (PULP) will have much less resources available than the **Commercial Node**, e.g., onboard processor performance and volatile/non-volatile memory will be multiple decades smaller. Due to these facts, the high-end programming tools, such as Java/Python, are not necessarily available. However, the software node will offer POSIX standard APIs and C/C++ standard development tools for application development.

Despite the limitations above, the **Customizable Node**, if carefully designed, will meet application specific performance easily Figure 1. presents the three tiers of the system architecture. If a node exists in mist tier, it is a good candidate to be a Pulp-based low-end node.

Another limitation on **Customizable Nodes** is caused by the high level of optimization of the node's hardware. Having a complete node software framework for all the platforms is out of scope of this project. Some of the use case features may need to be demonstrated at multiple (different) HW platforms.

As the use case is a concrete demonstration for the use case provider, it should not be surprising that the main goal of the use case provider is to make sure that the use case can run without issues within the FRACTAL project. As a result, some project partners expressed rather extensive requirements for their own use cases in order not to be limited by the hardware capabilities in the future, and some others expressed interest in using systems that they are more familiar with. In practice, this has led to several use case providers stating the need for a symmetric multi-core system running a standard Linux distribution.

In all cases, these requirements are perfectly understandable and most of them could be implemented using the commercial node of FRACTAL, the Xilinx VERSAL platform. As outlined in chapter 4.2, the basic customizable node has been targeted towards simpler IoT applications and lacks the power to fulfil several of these requirements. At first sight

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

this creates an apparent imbalance of utilization between the commercial and the customizable node.

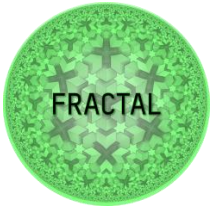
FRACTAL partners have discussed various approaches to provide a solution and have decided on a number of measures to make sure that the ideas developed as part of FRACTAL are validated on common platforms that are available to all project partners. From those discussions come the following recommendations.

- There are several use cases (see Section 4) that are content to use the FRACTAL hardware nodes as provided.
- FRACTAL has identified three tiers of FRACTAL hardware nodes: low (Mist), medium (Edge), high (Cloud). Node versions that all share similar interfaces and interact with each other Figure 1. shows an illustration of such an organization where simpler nodes are acquiring data and delegating more complex tasks to nodes with higher complexity. The figure is meant as an example, and different allocations of tasks are currently under discussion within WP5/6. In this model, the industrial node covers the higher-end version, while the customizable node is seen as the lower-end version. As described chapter 4.3, partners have suggested several alternatives for the medium-end nodes.
- Some partners are relying on their prior work and experience to implement some of their contributions. Most of these are based on hardware systems that are similar and/or compatible with FRACTAL nodes but have some differences. These include implementations in earlier models of Xilinx MPSoC platforms than the VERSAL as well as other openly available RISC-V systems. Out of practical considerations, FRACTAL partners have added these as additional platforms to the initially identified hardware nodes.

It was also recognized that official FRACTAL nodes could be instrumental for research aspects involving developments in WP4/5/6 and the experience from these explorative works could then be used to evaluate the potential of these developments in use cases that consider more traditional solutions. As a concrete example, novel safety solutions with hardware support could be explored on a small scale in the customizable node as part of WP4. The results of this exploration could then be used to directly estimate the gains achievable by this approach in a use case that employed an alternative hardware node. The work done throughout the first part of the project allowed partners to realize different possibilities and showed that the key point was that all developments from FRACTAL technical work packages should be accessible for all FRACTAL partners. While the initially identified Hardware Nodes cover a large range of the specification spectrum, partners could also make use of additional hardware nodes as long as this work could be used/verified/evaluated by all partners.

### 3.1 Customizable node (RISC-V based PULP)

The customizable node in FRACTAL is a RISC-V based PULP platform (particularly the PULPissimo microcontroller) which is further described in D3.5. For the sake of completeness, Figure 25 depicts PULPissimo as part of the FRACTAL big picture.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

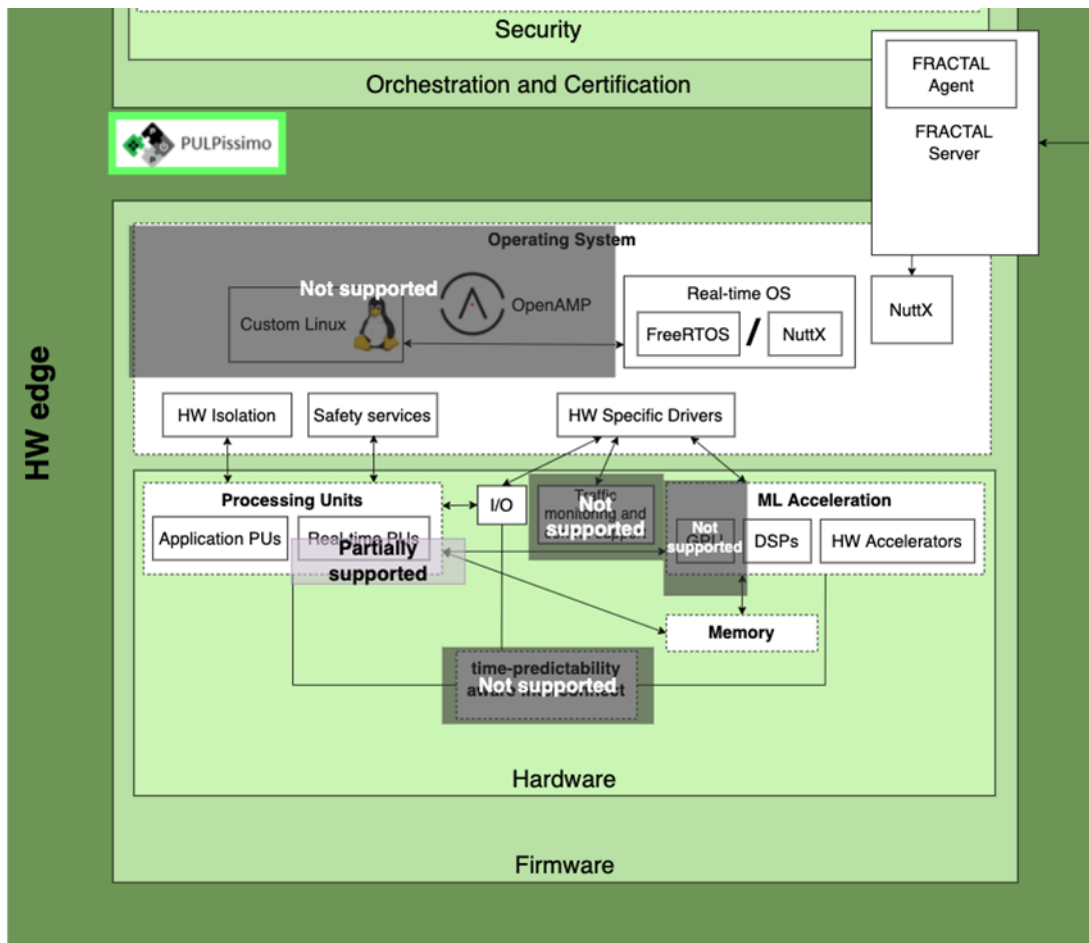



Figure 25 PULPissimo as part of the FRACTAL big picture

PULPissimo is a RISC-V based system, with resources adopted to IoT workloads – RAM (volatile) up to a few megabytes and flash (non-volatile) some hundreds of megabytes. System clock speeds are typically below 1GHz and often the processor may lack the memory protection modules.

RISC-V is based on proven RISC principles from the 80's. Recently it has had additional benefits, as an open ISA. This enables development of both open and commercial applications. This is interesting both for Industry and Academia as it lowers the barriers to share developments on the ISA between various partners and benefits the cumulative research results. The RISC-V licensing terms make it possible to develop open-source hardware.

As a platform optimized for the IoT applications, the PULP platform has a much lower power budget in the milliwatts or even microwatts range. This is especially interesting for applications where battery powered nodes are expected to operate for years.

PULPissimo systems have FPGA based implementations, where all digital parts (including the processor) exist as FPGA code, allowing the peripheral and HW-accelerators to

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

optimize purely to the application. Potentially parts of the application logic may be implemented in hardware and even altered in runtime. An interesting option is the implementation of AI engine primitives by HW.

By converting the FPGA design to the ASIC, the cost of PULP based systems (in high volumes) can go down to the cents.

For software point of view, the advanced operating systems such like Linux are out of scope due to the limitations of the underlying platform. On the other hand, plain bare-metal applications will be too complex to be managed by the Fractal framework. There exists numerous RTOS's, but for FRACTAL Pulp nodes the open source Nuttx RTOS has been chosen. Main benefit of Nuttx is the POSIX compatibility. This RTOS offers primitives such as threads, devices and sockets. Due this the libraries and the applications may develop to be fully Linux compatible – some existing Linux libraries and application can be just compiled into the Nuttx. Physical communication devices can be integrated into the sockets. By utilizing the IP-stack, the local and wide area networks can be seamlessly hidden below IP-networking. This again eases the application development and isolates the network configuration. Existing authentication and encryption methods can be utilized. As result, the application development does not require any RTOS specific code.

For the development the standard GNU C/C++ development environment is available for the application developer. Also, tools such as GDB/JTAG offer test/debug features on platform.

Due to HW limitations – mainly RAM/flash -- tools such as java and python are typically not available (some limited versions do exist).

Due the limited memory resources -- the containers and hypervisors does not make sense on Nuttx. By increasing the processor complexity and sizes of memories, these problems can be solved, but when moving in that direction, the better solution is to switch to the Linux.

### **3.1.1 Pulp onboard resources**

#### ***3.1.1.1 Key management***


Key security element of safe software deployment and authentication is secure key management. To do this in a safe way an additional hardware module is required.

#### ***3.1.1.2 Hardware acceleration***

There are two typical approaches for HW acceleration in Pulp platform.

RISC-V offers a mechanism to introduce new instructions to the processor. Utilizing this mechanism makes the acceleration totally seamless to the application software. Drawback is the requirement of a customized compiler. Less general the acceleration is, less practical this approach is.



	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

Another typical approach is to build a custom peripheral that offers a memory mapped register interface to the software.

One interesting case is the HW acceleration of AI-primitives.

If the Pulp is running on FPGA platform, software may upload different functions to FPGA, thus it can dynamically change the behavior of the HW acceleration. By utilizing this mechanism, the parts of application software can be executed on HW – however this is outside scope of this document.

In task WP3T36-03 the Nuttx OS were port to RISC-V processor and to the Pulp platform. There was a deprecated RISC-V support on official Nuttx repositories, but it wasn't covering the newer releases. As a result of this task the RISC-V is now on Nuttx mainline and maintenance responsibility has been taken to OFFC.

### 3.1.2 Safety considerations on Nuttx Pulp – WP4

Refencing the following publication:

#### **RISC-V for Real-time MCUs - Software Optimization and Microarchitectural Gap Analysis**

<https://ieeexplore.ieee.org/document/9474114>

Abstract:

Processors using the RISC-VISA are finding increasing real use in IoT and embedded systems in the MCU segment. However, many real-life use cases in this segment have realtime constraints. In this paper we analyze the current state of real-time support for RISC-V with respect to the ISA, available hardware and software stack focusing on the RV32IMC subset of the ISA. As a reference point, we use the CV32E40P, an open-source industrially supported RV32IMFC core and FreeRTOS, a popular open-source real-time operating system, to do a baseline characterization. We perform a series of software optimizations on the vanilla RISC-V FreeRTOS port where we also explore and make use of ISA and micro-architectural features, improving the context switch time by 25% and the interrupt latency by 33% in the average and 20% in the worst-case run on a CV32E40P when evaluated on a power control unit firmware and synthetic benchmarks. This improved version serves then in a comparison against the ARM Cortex-M series, which in turn allows us to highlight gaps and challenges to be tackled in the RISC-VISA as well as in the hardware/software ecosystem to achieve competitive maturity.

### 3.1.3 AI processing on Nuttx Pulp -- WP5

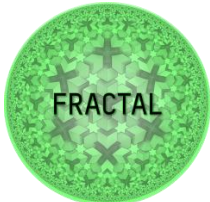
On this project the AI scenarios on Pulp node are based preloading models to the Node. The learning of model is done elsewhere (i.e Fractal cloud) and machine learned model is uploaded to node(s). This is demonstrated by UC3.

#### **3.1.3.1 Software approach**

In the case of low-end nodes, the lack of high-level tools such as python and Java and the limited RAM memory make the plain SW approach challenging. Infesting cases would be a hybrid solution where software loads pre trained modules to be executed in HW.

#### **3.1.3.2 Hardware approach**

A very interesting case would be the HW acceleration of AI-primitives. This would result in a case where models can be loaded to software, but the actual processing happens in HW.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

Referencing following (Fractal) publication:

**Ternarized TCN for  $\mu$ J/Inference GestureRecognition from DVS Event Frames:**

<https://ieeexplore.ieee.org/document/9406333>

Abstract:

Heavily quantized fixed-point arithmetic is becoming a common approach to deploy Convolutional Neural Networks (CNNs) on limited-memory low-power IoT end-nodes. However, this trend is narrowed by the lack of support for low-bitwidth in the arithmetic units of state-of-the-art embedded Microcontrollers (MCUs). This work proposes a multi-precision arithmetic unit fully integrated into a RISC-V processor at the micro-architectural and ISA level to boost the efficiency of heavily Quantized Neural Network (QNN) inference on microcontroller-class cores. By extending the ISA with nibble (4-bit) and crumb (2-bit) SIMD instructions, we show near-linear speedup with respect to higher precision integer computation on the key kernels for QNN computation. Also, we propose a custom execution paradigm for SIMD sum-of-dot-product operations, which consists of fusing a dot product with a load operation, with an up to  $1.64 \times$  peak MAC/cycle improvement compared to a standard execution scenario. To further push the efficiency, we integrate the RISC-V extended core in a parallel cluster of 8 processors, with near-linear improvement with respect to a single core architecture. To evaluate the proposed extensions, we fully implement the cluster of processors in GF22FDX technology. QNN convolution kernels on a parallel cluster implementing the proposed extension run  $6 \times$  and  $8 \times$  faster when considering 4- and 2-bit data operands, respectively, compared to a baseline processing cluster only supporting 8-bit SIMD instructions. With a peak of 2.22 TOPs/s/W, the proposed solution achieves efficiency levels comparable with dedicated DNN inference accelerators and up to three orders of magnitude better than state-of-the-art ARM Cortex-M based microcontroller systems such as the low-end STM32L4 MCU and the high-end STM32H7 MCU.

For additional information see:

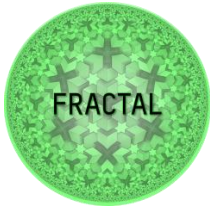
[https://pulp-platform.org/pulp\\_sw.html](https://pulp-platform.org/pulp_sw.html)

<https://pulp-platform.github.io/pulp-dsp/tutorial-index/>

### 3.1.4 Application orchestration on Nuttx Pulp – WP6

Fractal application deployment is based on containers. While Nuttx lack resources to run containers as such, the problem is solved by having special containers on cloud (or high-end edge nodes), that connects to specific nodes and upload specific binaries to Pulp nodes. While most of this work is done in scope of WP6 here is just described the services that node offers.

This further studied on D6.2.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

### 3.2 Versal node

The Xilinx VERSAL ACAP is expected to be deployed as part of the VCK190 Evaluation Kit board, which provides support for several I/O interfaces and memory devices. For completeness, Figure 26 shows the VERSAL platform in the FRACTAL big picture.

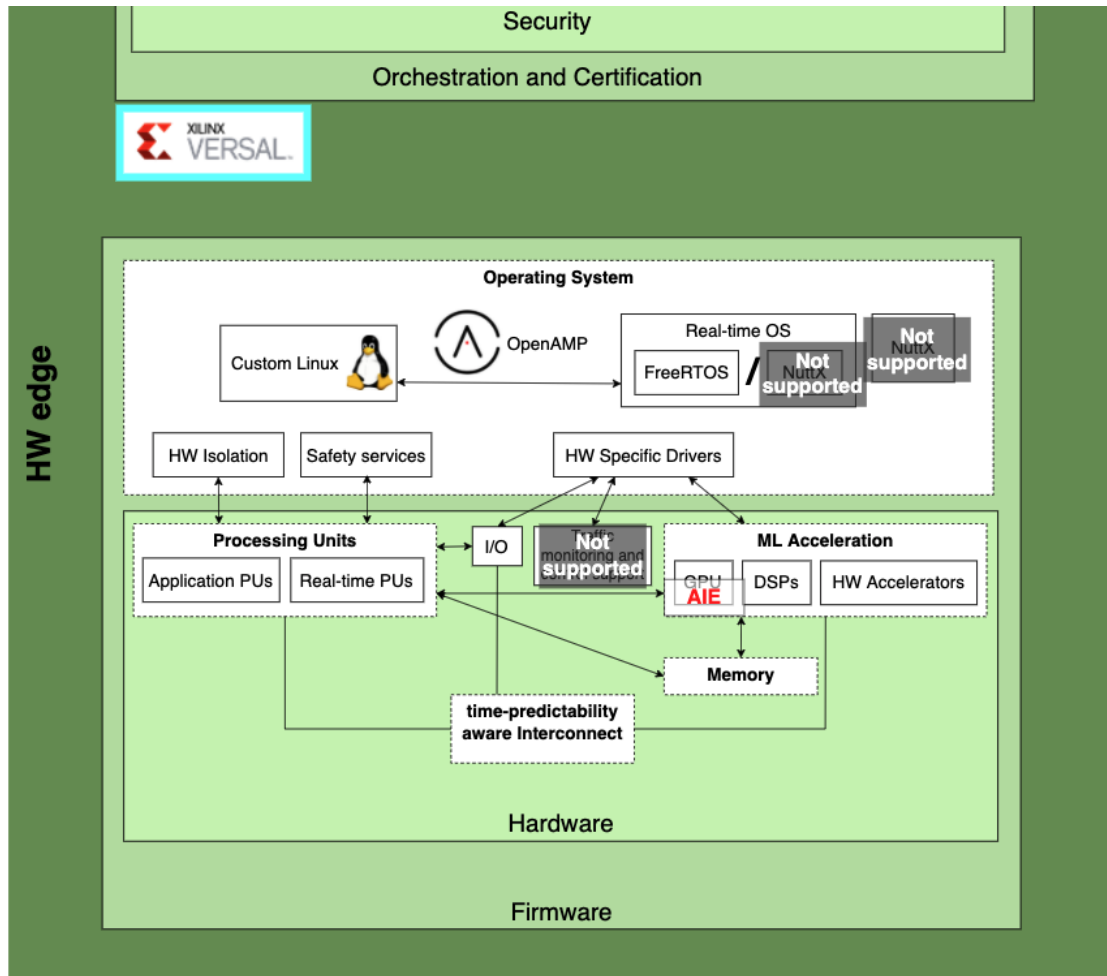
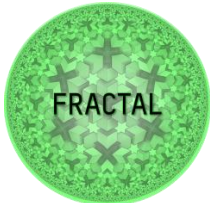


Figure 26 The VERSAL platform in the FRACTAL big picture

The VERSAL architecture (Figure 27) combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC).

Like the earlier Xilinx Zynq MPSoC products, VERSAL ACAP devices still offer the two main components:

- Processing System (PS)** consists of a dual high-performance ARM Cortex A72 cores that can run Linux or other operating systems. This system is augmented by a dual-core ASIL-C certified real-time processing subsystem based on Arm Cortex R5F cores. Together these systems address the needs of most modern computing needs using a traditional programming interface.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

- **Programmable Logic (PL)** allows this system to be augmented by hardware accelerators customized to a particular compute function.

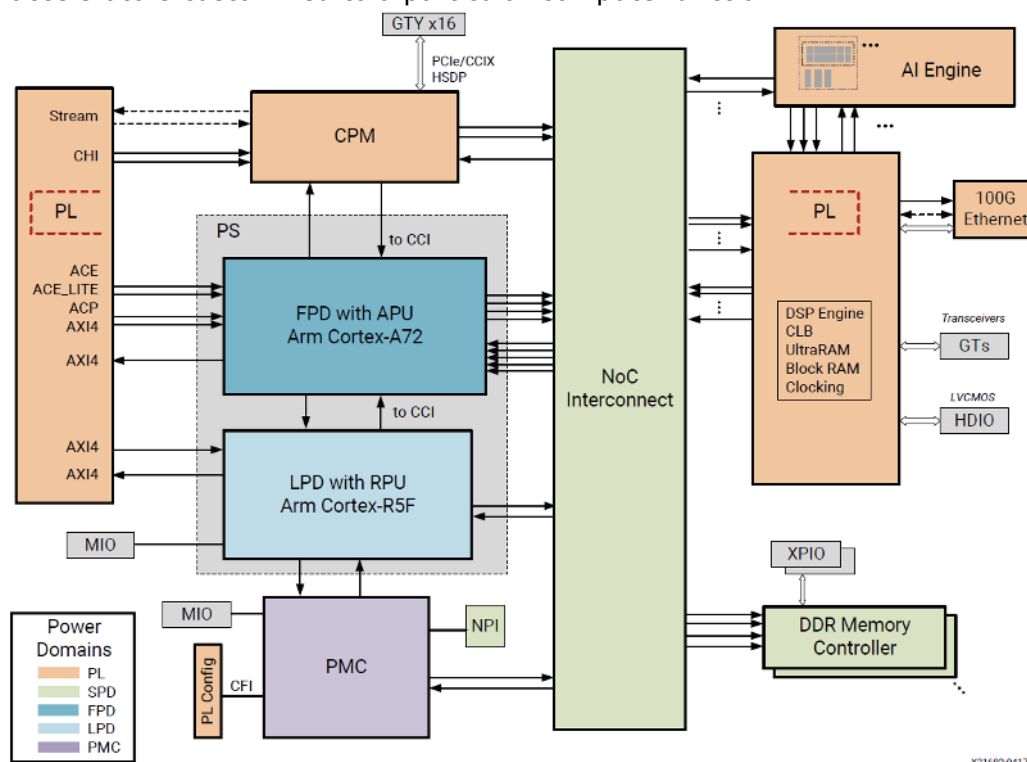
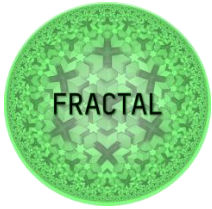


Figure 27 Top level schematic of Xilinx VERSAL ACAP

On one hand, VERSAL designs are enabled by the Vitis™ tools, libraries, and IP. The Vitis IDE lets the developer program, run, and debug the different elements of VERSAL AI Engine application, which can include AI Engine kernels and graphs, PL, high-level synthesis (HLS) IP, RTL IP, and PS applications. Vitis offers two development approaches:

- **Accelerated Flow.** It allows to build a software application using the OpenCL or the open-source Xilinx Runtime (XRT) native API to run the hardware kernels on accelerator cards, or on a Linux-embedded processor platform. The Vitis tool includes the v++ compiler for the hardware kernel on all platforms, the g++ compiler for compiling the application to run on an x86 host, and Arm® compiler for cross compiling the application to run on the embedded processor of a Xilinx device.
- **Embedded Flow.** It provides a complete environment for creating software applications targeted for the embedded processors. It includes a GNU-based compiler toolchain, C/C++ development toolkit (CDT), JTAG debugger, flash programmer, middleware libraries, bare-metal BSPs, and drivers for all the Xilinx IPs. It also includes a robust IDE for C/C++ bare metal and Linux application development and debugging.

On the other hand, the Peta Linux tools (built on top of the Yocto Project) offer everything necessary to customize, build, and deploy embedded Linux solutions on Xilinx processing systems. Tailored to accelerate design productivity for SoC devices, the solution works

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

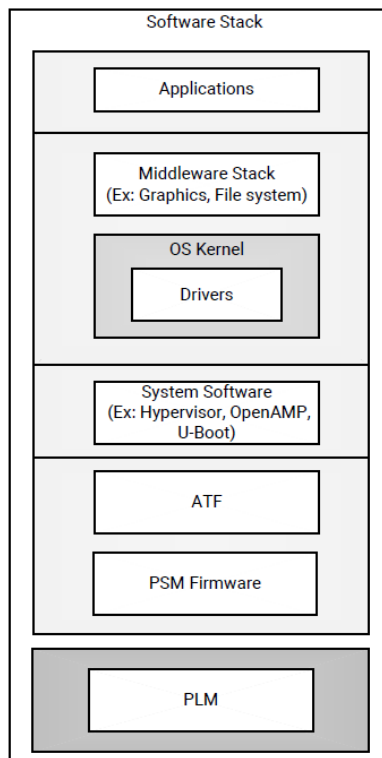
with the Xilinx hardware design tools to facilitate the development of open-source Linux systems for VERSAL devices.

FRACTAL nodes will use the tools provided by Xilinx to build the system as stated in the different use cases. This deployment will include OS or system software customization (e.g., hypervisor, bootloader, kernel) to match the requirements of FRACTAL nodes, and file system creation, including all the software packages and configuration stated in the platform description requirements (e.g., Python, JAVA, net-tools, others). It will also consider the development of use case specific high-level applications or low-level drivers required by custom building-blocks (e.g., security related modules).

It is important to mention that even though the developments will be very use case specific, they will share a common development stack, based on the options and considerations established by Xilinx (

Figure 28 VERSAL Linux development stack

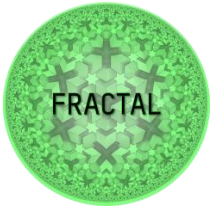
).



X23439-071020

Figure 28 VERSAL Linux development stack

As VERSAL provides many different implementation possibilities, in the beginning of the project the main approach was not only to analyze, understand and determine the requirements coming from every use case, but also the proposed roadmap in order to

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

achieve use case objectives. Reference software architecture of a cognitive edge computing node with FRACTAL properties will be defined and a common repository of generic qualified components will be set up. Particular attention will be paid on providing flexible computing nodes, that are reusable by others and that efficiently support the software on providing acceleration for the learning part.

The reference software architecture design for the VERSAL based FRACTAL node will be described in the following deliverable (D3.6), so that it reflects the customization and integration made on top of the FRACTAL nodes based on the software components provided by Xilinx.

### 3.2.1 Versal onboard resources

Native Linux applications that include the Xilinx runtimes for different hardware targets can still be built as native applications and run against the support of the runtime of the Versal Linux host. This may be beneficial in nodes where the FRACTAL orchestration and system interface level is only used to provide data or model and otherwise has low interaction with the application itself. Such applications can make use of all features on the board as the device provides the complete underlying hardware description.

The FRACTAL Edge Software stack uses microservices to build up an application. As applications in the node need to be exchanged or scheduled from the system level context aware scheduling, it is desirable to fetch these from the image store and thus run these applications in a containerized version.


To achieve this, a docker container is devised that is based on an Ubuntu Linux and holds the full addition of libraries to satisfy the respective runtime, as an example XRT or VART. In such a docker environment the application can run and reach out to the devices through the kernel. The scaling of this approach will be further investigated, but it is required to control the scope and reach of the application.

### 3.2.2 Safety considerations on Versal – WP4

The Versal reference platform is being set up in two base designs. While one of these is not imposing restrictions on actual computation cores, the second version is intended to build the certifiable platform along D2.3.

This safety focused setup is guided by the separation capabilities of the Versal ACAP architecture. As already pointed out in Chapter 3 Versal devices offer a hardware abstraction of core-related power functionality and allow access to these through the PMC. While the PMC exposed features can be used rather freely, this is not reasonable for a proper safety approach.

To satisfy these safety concerns, the separation of the elements is considered from ground up and the services on the FRACTAL node level need to settle to a particular central provider that actually then as a proxy call into the PMC. In the safety focused platform all control requests for power and scaling the actual low-level accesses are collected on one of the RPUs. This RPU will carry out all PMC related transactions on behalf of the node. This platform setup also incorporates a secure boot mechanism to ensure the guaranteed boot into the safety

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

environment. The exposure of the service interface to the RPU is carried out within the respective WP4 tasks.

The support for the time triggered scheduling and node-level and system-wide features as from WP4, are attributed to the RPU in the designs where these real-time capabilities are required. In this setup the Adaptive Time-Triggered Network Interface is controlled by this core to provide the scheduling services on top. This RPU can unify the node control transactions with this service or push transactions to the respective other RPU if it is available in the design.

The communication between the heterogenous cores, from the Fractal Edge Software Linux to the RPU is carried out using OpenAMP. To cater for such a setup, the RPU projects are deploying FreeRTOS.

### 3.2.3 AI processing on Versal -- WP5

The Versal based platform, specifically with the VCK190 and the VCKC1902 device offers specific ML acceleration means by deploying the AI Engines. These are supported by the DPU IP hardware structure in the reference platform design.

In the Versal platform BSP for Linux the supporting device tree elements for all particular features of the device are made available and the respective Vitis AI runtime (VART) libraries are installed. The effective pre-trained model deployment is carried out through a proprietary Xilinx toolchain, Vitis AI, that is capable of processing ML models from a variety of toolchains, like Tensorflow and PyTorch, but also reading from ONNX models.

The analysis and translation of such models yields a combination of hardware configuration information and embedded code artifacts for multiple computation engines. The translated models themselves are exchangeable and may be retrieved from the model repository on demand. The translation itself can also be carried out as a service in the FRACTAL cloud.

A translated model is deployed through the VART runtime in a Linux application. Such an application is typically triggered through the orchestration level of the Fractal Edge software and can be added to a specific application container.

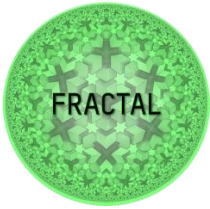
### 3.2.4 Application orchestration on Versal – WP6

The FRACTAL reference design based on Versal ACAP is set up to include the basic application support along the WP6 proposal. Current basic Petalinux setup has been operating Docker and Kubernetes (MicroK8s) and Mosquitto. The full set is planned to use Prometheus and Juju. All additional orchestration features are deployed in respective containers.

## 3.3 Other nodes

On Fractal the Versal and the Pulp platforms were selected to use as node HW-platforms. However, to demonstrate some specific developments partners have chosen to use some other platforms.

During this study performed in the first part of the project, several use cases stated the need for more traditional RISC-V based systems (capable of running single-core or SMP

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

Linux) which resulted in some additional hardware nodes being added. The software capabilities of these additional RISC-V nodes will be covered in this section.

### 3.3.1 NOEL-V

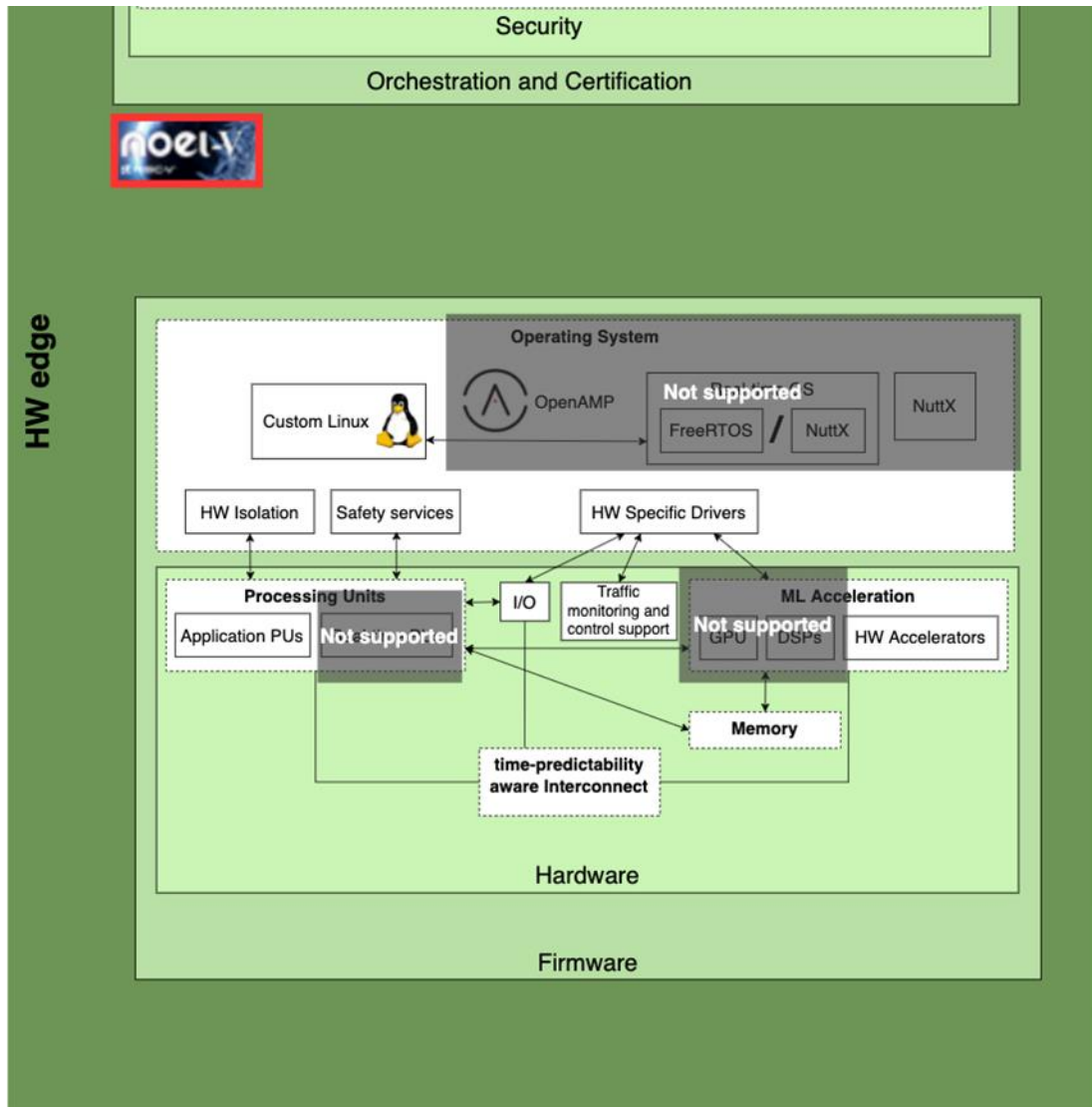



Figure 29 NOEL-V as part of the FRACTAL big picture

The NOEL-V based SoC builds upon the GPL platform provided by the H2020 SELENE project (<https://www.selene-project.eu/>). The SELENE SoC has been synthesized in a Xilinx Virtex UltraScale VCU118 FPGA and the original NOEL-V SoC is also available for the KCU115, although it can be ported to other boards. For completeness, Figure 9 shows the NOEL-V based SoC as part of the FRACTAL big picture.

The NOEL-V SoC supports memory management units, and implements Translation Lookaside Buffers (TLBs), both for data and instructions, locally in each core. The SoC also



	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

provides support for cache coherence. Those features allow booting SMP Linux and RTEMS operating systems among others and allow sharing data across cores.

The Linux image, on which current developments are performed, has been built with buildroot (2021.02LTS), and the required sources are provided by Cobham Gaisler at <https://www.gaisler.com/index.php/downloads/sw-noelv-downloads>.

The platform implements the RV64I RISC-V ISA along with the G, C and H extensions.

Standard software tools for compiling, debugging, and the like are supported since the platform adheres to the RISC-V standard.

### 3.3.2 ARIANE/CVA6

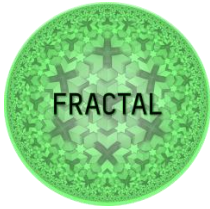
Ariane/CVA6 supports Linux, both 32-bit on CV32A6 and 64-bit on CV64A6. In a first step of the FRACTAL project, Linux has been ported to CV32A6 with recent versions of the various components (BBL, Buildroot 2021.5.rc1, Linux kernel 5.10.7).

CV64A6 has supported 64-bit Linux for a longer time and work has been performed to update to the same versions as CV32A6. Compilation is supported by GCC 9.3, and software simulation is supported by Spike. As CVA6 is aligned with RISC-V standard extensions, we can expect software support by other generic tools, such as Clang/LLVM, without further port.

Since D3.4 release, the support of popular up-to-date components have been added: UBoot as an alternative to BBL (Berkely Boot Loader), as well as the OpenSBI firmware. The support of Yocto embedded Linux image builder, as an alternative to Buildroot, is the latest addition. (component WP3T36-01).

To the best of our knowledge, this is the first time UBoot, OpenSBI and Yocto are supported on CVA6.

On top of the Linux operating system, the Ariane/CVA6-based node will support the FRACTAL software components depicted in the figure below to deliver the services needed in UC4. On the AI side, only inference will be supported.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

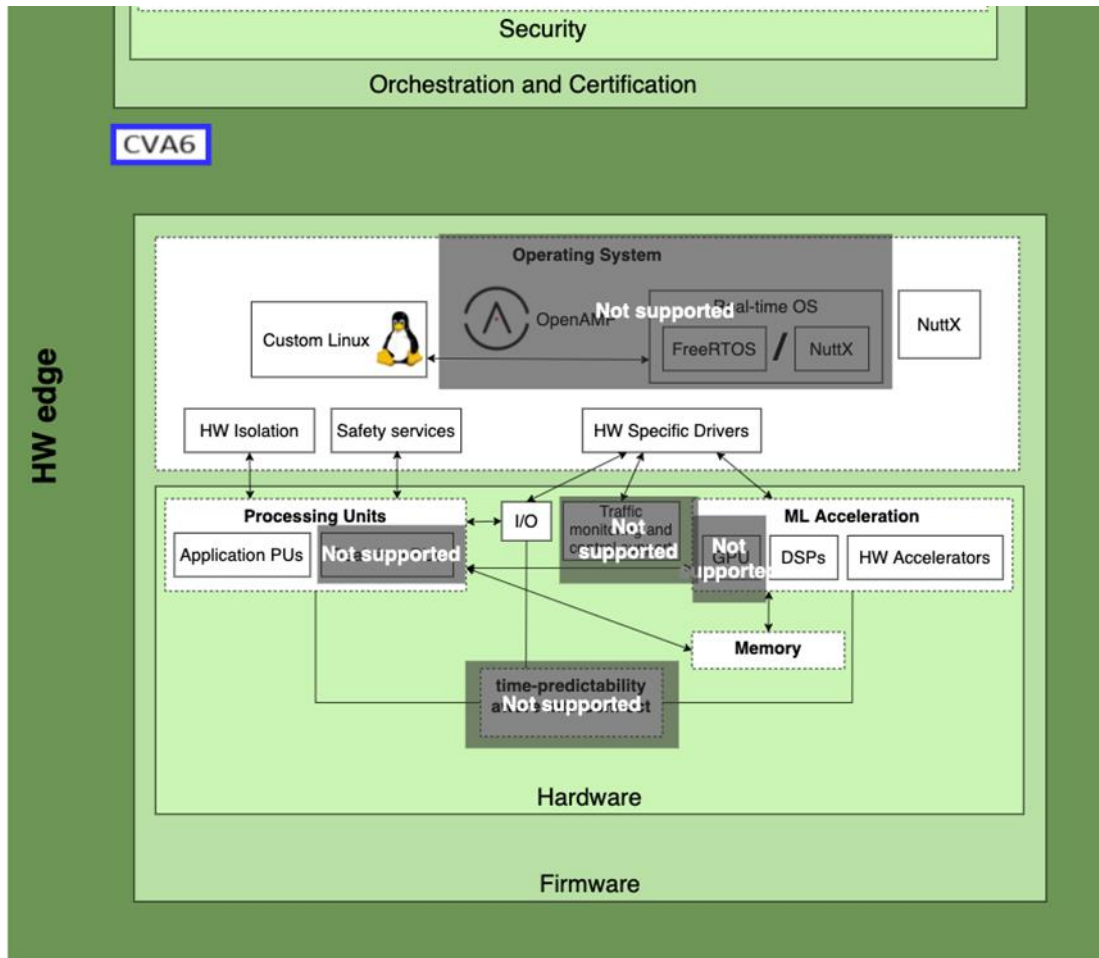



Figure 30 CVA6 as part of the FRACTAL big picture

### 3.3.3 Yet additional platforms

Some special (technical) cases may be implemented and/or demonstrated by yet other platforms (i.e. Raspberry PI). Those cases will be presented here.

#### 3.3.3.1 Asymmetric multiprocessing AMP

Yet another interesting option is to utilize both Linux and *NuttX*. Cost efficient way to do this is to use Asymmetric Multi Processing (AMP) in a multicore processor. One for the cores is running NuttX, while others run Linux. Main benefits are real-time processing. As NuttX is RTOS, it can process the real-time deadlines better than Linux. Another benefit is the low-energy operation. Linux can put to deep sleep, while NuttX handles basic operations and when required wake up the Linux. This is not possible to demonstrate on a PULPissimo platform.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 4 Interaction of UCs with FRACTAL nodes

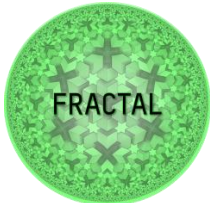
---

The FRACTAL cloud SW development has begun in top-down approach. The main AI-applications are under development and aim to demonstrate with simulated nodes.

The FRACTAL Pulp node SW development progresses in by bottom-up approach, where the basic features are ramped up. This includes the board OS-wake-up, simple LED-Blink demonstrations and setting up SW-repositories and tools. While development progresses the service layers AI, Connectivity, Security will be integrated. The final phase of integration is the application – use case integration, where node SW and cloud SW are integrated as complete solution.

On FRACTAL Versal the use case development can begin after the FRACTAL adaptation interfaces are agreed.


The detailed interaction between FRACTAL SW nodes and use cases will be described in subsequent versions of this deliverable (D3.4, D3.6). In document **D3.1\_Preliminary\_HW\_node** (chap 5) presents the resource-based allocation of use cases on FRACTAL nodes.

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 5 Conclusions

---

The two main hardware nodes (commercial and customizable) are being made available to all FRACTAL partners. Following discussions regarding the design requirements, it was seen that partners would benefit from additional nodes that fall in between the two default options. WP3 partners have discussed providing such solutions in agreement with other partners from technical WPs 4/5/6 as well as the UCs

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 6 Next steps

The development and adaptation of HW nodes continues as part of WP5/6/7/8 activities that are involved with actual implementations of UCs.

Before the start of the project, the following hardware platform related risks had already been identified:

FPGA based PULP platform implementations will not have the necessary performance (speed/cost/power) profile needed for use cases (Medium likelihood/Low Impact).

For some use cases, this has been indeed the case. However, the issue was not the FPGA implementation of the platform, but the desire to have systems that are more like traditional computing systems complete with running large software ecosystems running on full-fledged operating systems (i.e. Pytorch) which is beyond the practical capabilities of the system. In part this has been remedied by providing additional platforms (NOEL-V and Ariane). As the projects mature and partners become more experienced with different FRACTAL platforms, it is highly probable that more partners will make use of the experimental and customizable aspects of the provided systems.

### 6.1 Risks and Mitigation plans

As described in D3.5:

***"Diversity/maturity of tools/development environments for RISC-V systems is lower than expected (Low likelihood/Medium impact).***


***As mentioned in the project proposal, the development rate of the RISC-V ecosystem is quite high, and so far this has not been a major issue.***

***Unable to properly and timely integrate multiple services developed. Due to incompatibilities or services not provided by selected hardware platforms (Low likelihood/impact).***

***The integration of services are still at an early stage, so it is too early to tell about the possible impact. However, the effort in WP2 has allowed partners to anticipate some of the issues, and there is good momentum in the project that leads us to believe these issues could be mastered.***

***Difficulties to integrate hardware developments from different partners (Medium likelihood/impact).***

***This is one of the main challenges that faces the developments around the HW platform at the moment. However, all partners are aware and are looking for solutions. In some cases, one solution will be to demonstrate technical solutions running in parts, and not in concert for a given platform, use case combination.***

	Project	FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node
	Title	Software node and services architecture
	Del. Code	D3.6

***I.e. a security service maybe demonstrated on a smaller scale, allowing the use case partner to be able to judge and evaluate its impact, but the overall use case could still use a more traditional approach.***

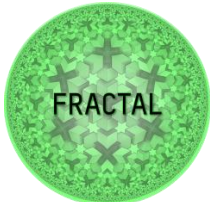
***Part of the mitigation efforts also led partners to add two additional hardware nodes, as not to spend initial efforts on porting previous work from architectures they were familiar with to the two nodes presently available.***

***In addition, the following issues have been detected, and efforts have been put in place to mitigate the effects:***

***Delivery difficulties with the VERSAL board. There is a global shortage on electronic components, and unfortunately the delivery of the development boards have been hit with longer delays than anticipated. For some of the projects that do not rely on exclusive VERSAL properties, suggestions were made to use more previous generation Xilinx MPSoC boards until the VERSAL shipments can be organized.***

***Node computation demands are too high. Some partners on technical workpackages are working on solutions that require significant resources from the hardware nodes. While the VERSAL board can satisfy these requirements, both its price and its power envelope is higher than what could be expected for IoT applications.***

***The addition of a lower-end (mist) node will help address this issue. The FRACTAL system will have nodes with less capabilities that defer to more capable nodes for higher complexity operations. This will allow hardware nodes within mW power envelope to be part of the FRACTAL system.”***

	Project	FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node
	Title	Software node and services architecture
	Del. Code	D3.6

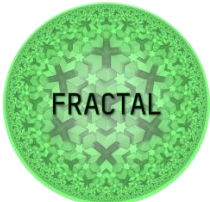
## 7 Deviations from workplan

---

As described in D3.5:

***"Following the feedback from the first project review, it was decided to align the deliverables and reports to a consolidated big picture of the FRACTAL project. This big picture was discussed and agreed on as part of the 2nd Technical Workshop that took place in early February. As a result, D3.3 as well as this follow-up D3.5 were delayed slightly allowing the deliverables to be updated.***


***It must be noted that lack of face-to-face meetings due to COVID restrictions have impacted such discussions, and while the FRACTAL consortium has put in a very good effort to maintain virtual meetings and discussions, the level of interaction of such meetings cannot replace hands on workshops and technical discussions to bring agreement between a large group of participants. We are happy to report that the last technical meeting was held face to face and has brought back much needed personal interaction. Unfortunately, the efforts of WP3 have been completed, but partners are actively working on supporting UCs in WP7/8 until the end of the project."***

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 8 Bibliography

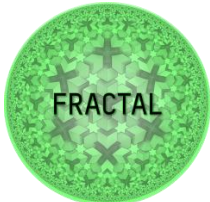
---



	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

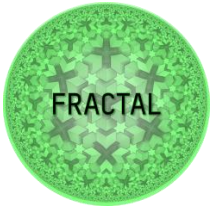
## 9 References

- Amazon Web Service. (2019). *AWS documentation*. Retrieved from Message Broker for AWS IoT: <https://docs.aws.amazon.com/iot/latest/developerguide/iot-message-broker.html>
- Amazon Web Service. (2019). *AWS Documentation*. Retrieved from AWS IoT SDK: <https://docs.aws.amazon.com/iot/latest/developerguide/iot-sdks.html>
- Amazon Web Service. (2019). *AWS Documentation*. Retrieved from Security and Identity: <https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html>
- Amazon Web Service. (2019). *AWS IoT Core*. Retrieved from Features: <https://aws.amazon.com/iot-core/features/>
- Bosch. (2019). *Gateway software*. Retrieved from <https://www.bosch-si.com/iot-platform/iot-platform/gateway/software.html?ref=ot-2-inst-de-2017h1-sales-contact-forms-iot-platform>
- Comer. (2000). Datagram Size, Network MTU, and Fragmentation. In *Sect. 7.7.4* (p. p. 104).
- Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. Irvine: University of California.
- Lampkin, V., Tat Leong, W., Olivera, L., Rawat, S., Subrahmanyam, N., Xiang, R., . . . Locke, D. (2012). *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. IMB Redbooks.
- Saint-Andre, P., Smith, K., & Tronçon, R. (2009). *XMPP: The Definitive Guide. Building Real-Time Applications with Jabber*. O'Reilly Media, Inc.
- Shelby, Z. a. (2011). *6LoWPAN: The wireless embedded Internet* (Vol. 43). John Wiley & Sons.
- Siemens. (2018). <https://www.plm.automation.siemens.com>. Retrieved from [https://www.plm.automation.siemens.com/media/global/en/Siemens-MindSphere-Whitepaper-69993\\_tcm27-29087.pdf](https://www.plm.automation.siemens.com/media/global/en/Siemens-MindSphere-Whitepaper-69993_tcm27-29087.pdf)
- UbuntuPit. (2018). <https://www.ubuntupit.com/>. Retrieved from <https://www.ubuntupit.com/choose-the-right-iot-platform-top-20-iot-cloud-platforms-reviewed/>

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 10 List of figures

Figure 1 A schematic drawing of a possible FRACTAL system deployment using three different tiers of FRACTAL hardware nodes with different capabilities (drawing from WP5 technical meetings). .....	5
Figure 2 Average execution time of Crop, Resize and Clahe algorithms on Zynq with 4-core running .....	13
Figure 3 Boxplot for Crop Algorithm with 4-core running .....	13
Figure 4 Boxplot for Resize Algorithm with 4-core running .....	14
Figure 5 Boxplot for CLAHE Algorithm with 4-core running .....	14
Figure 6 Average execution time of Crop, Resize and Clahe algorithms on Zynq with 1-core running .....	15
Figure 7 Boxplot for Crop Algorithm with 1-core running .....	15
Figure 8 Boxplot for Resize algorithm with 1-core running .....	15
Figure 9 Boxplot for CLAHE algorithm with 1-core running .....	16
Figure 10 LEDEL development in FRACTAL .....	19
Figure 11 Abstraction of the docker file presented .....	21
Figure 12 Example of training process using LEDEL in RISC-V .....	22
Figure 13 Code used to load the ONNX file using the LEDEL .....	23
Figure 14 Inference from ONNX file .....	24
Figure 15 PyTorch example execution .....	24
Figure 16 Figure: A schematic of the software-only support for diverse redundancy.....	25
Figure 17 Source code of the pmu_counters_enable() function. ....	27
Figure 18 Source code of the read_pmu_reg() and write_pmu_reg() functions. ....	28
Figure 19 Software architecture for memory interference study .....	29
Figure 20 Algorithm for SIEFRACC hardware accelerator driver .....	31
Figure 21 Sequence diagram for using LB service .....	33
Figure 22 The IR module process: from input to output.....	34
Figure 23 IR module working stages .....	35
Figure 24 IR module data flow diagram .....	35
Figure 25 PULPissimo as part of the FRACTAL big picture .....	39
Figure 26 The VERSAL platform in the FRACTAL big picture .....	43
Figure 27 Top level schematic of Xilinx VERSAL ACAP .....	44
Figure 28 VERSAL Linux development stack.....	45
Figure 29 NOEL-V as part of the FRACTAL big picture .....	48
Figure 30 CVA6 as part of the FRACTAL big picture.....	50

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 11 List of tables

---

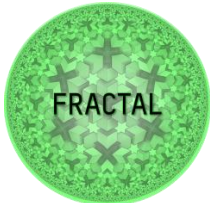
Table 1 The FRACTAL components (according to D2.3) related to WP3. Some components will be described in D3.5 (and one in D7.3), for others the section in this deliverable is given ..... 7

Table 2 Statistical insights for algorithms running with 4-core..... 14

Table 3 Statistical insights for algorithms running on 1-core processor..... 16

Table 4 Minimum RAM memory required if 4-core running..... 16


Table 5 Minimum RAM memory required if 1-core running..... 17

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

## 12 List of Abbreviations

---

ACAP	Adaptive Compute Acceleration Platform (relates to VERSAL)
AI	Artificial intelligence
APB	Advanced Peripheral Bus
API	Application Programming Interface
APU	Application Processing Unit
ASIC	Application-specific integrated circuit
AXI	Advanced eXtensible Interface
BBL	Berkeley Boot Loader
BSP	Board Support Package
CDT	C/C++ Development Toolkit
CLang	C Language
CPU	Central processing unit
DMA	Direct Memory Acces
DoA	Description of Action
DRAM	Dynamic/Distributed random-access memory
EDDL	European Distributed Deep Learning Library
FPGA	Field-Programmable Gate Array
GCC	GNU Compiler Collection
GDB	GNU Debugger
GPL	General Public License
GNU	GNU is Not Unix
HLS	High-Level Synthesis
HW	Hardware
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Intellectual Property
ISA	Instruction Set Architecture
JTAG	Joint Test Action Group
LB	Load Balancer
LED	Light-Emitting Diode
LEDEL	Low Energy DEep Learning Library
LLVM	Former initialism of Low Level Virtual Machine. Concept currently expanded.
MLOps	Compound of “machine learning” and the continuous development practice
MPSoC	Multiprocessor System-on-Chip
NoC	Network-on-Chip
ONNX	Open Neural Network eXchange
OpenCL	Open Computing Language
OS	Operating System
PL	Programmable Logic
PMC	Platform Management Controller
POSIX	Portable Operating System Interface

	Project	<b>FRACTAL: Cognitive Fractal and Secure Edge Based on Unique Open-Safe-Reliable-Low Power Hardware Platform Node</b>
	Title	<b>Software node and services architecture</b>
	Del. Code	<b>D3.6</b>

PS Programmable System  
 PULP Parallel Ultra Low Power  
 QEMU Quick EMUlator  
 REST Representational state transfer  
 RISC-V Reduced Instruction Set Computer  
 RPU Real time Processing Unit  
 RTEMS Real-Time Executive for Multiprocessor Systems  
 RTOS Real Time Operating System  
 SMP Symmetric Multi-Processing  
 SoC System on a Chip  
 SW Software  
 TLB Lookaside Buffer  
 UC Use Case  
 WP Work Package  
 XRT Xilinx Runtime