



D5.2 “First report on development and optimization of use-cases”

Version 1.0

Document Information

Contract Number	780681
Project Website	https://legato-project.eu/
Contractual Deadline	31. July 2019
Dissemination Level	Public
Nature	Report
Author	Micha vor dem Berge (CHR)
Contributors	Nils Kucza (UBI), Michaela Körber (CHR), Raúl de la Cruz (BSC), Xavier Martorell (BSC), Sigrun May (HZI), Amani Al-Mekhlafi (HZI), Frank Klawonn (HZI), Hans Salomonsson (MIS)
Reviewers	Gunnar Billung-Meyer (CHR), Tobias Becker (MAX)

The LEGaTO project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement No 780681

Changelog

Version	Date	Description of Change
vo.1	02.07.2019	Initial ToC
vo.2	09.07.2019	Initial input to sections 3-6 by application partners included
vo.3	15.07.2019	First internal review of sections 3-6
vo.4	23.07.2019	Input to section 7
vo.5	26.07.2019	Update revised input of sections 3-6 Writing of chapters 1, 2 and 8
vo.6	26.07.2019	Second internal review of the whole deliverable
vo.7	30.07.2019	Merging partner's input after second internal review
v1.0	31.07.2019	Final version

Index

1	Executive Summary	5
2	Introduction	6
3	Smart Home.....	8
3.1	Smart Mirror Prototype Hardware Setups	9
3.1.1	The Second Prototype.....	10
3.1.2	Next Planned Prototype.....	10
3.2	Smart Mirror Software Optimization Progress	11
3.2.1	Module Reconstruction and Optimizations with C++ and Acceleration with CUDA.....	11
3.2.2	Evaluating Darknet for OmpSs Optimizations	12
3.3	Baseline Benchmark and Performance	12
3.4	Development of additional Features	13
3.4.1	Implementation of Tracking using Kalman Filters	13
3.4.2	Dataset of Hand Gestures for easy control	14
3.4.3	Behaviour Prediction of the User Interaction	14
3.5	Next Optimization Possibilities	14
4	Smart City	15
4.1	Metrics & Optimization Goals.....	16
4.2	Baseline Benchmark.....	17
4.3	Development Status & Optimization Path	20
5	Infection Research.....	21
5.1	The background of biomarker discovery.....	21
5.2	Description of the application	22
5.3	Metrics & Optimization Goals.....	23
5.4	Baseline Benchmark.....	24
5.5	Development Status & Optimization Path	25
5.6	Further Plans.....	26
6	Machine Learning.....	28
6.1	Metrics & Optimization Goals.....	28
6.2	Baseline Benchmark.....	28
6.3	Development Status & Optimization Path	29
7	Secure IoT Gateway	31
7.1	Use case description and motivation	31

7.2	Development Status	32
7.3	Baseline Benchmark.....	38
8	Conclusion	40
9	References	41

1 Executive Summary

The main focus of Workpackage 5 is the development and optimisation of different real use cases with the help of the LEGaTO workflow. The actual development and optimisation status is reported within the deliverable. There are five different use cases

1. Smart Home
2. Smart City
3. Infection Research
4. Machine Learning
5. Secure IoT Gateway

All of them except the latter have been already able to be optimised using one of the toolflows that the LEGaTO project provides.

The Smart Home use case concentrates on a Smart Mirror demonstrator of which two versions have been developed within the project so far. A first demonstrator was built, after that a second enhanced demonstrator was built with many manual improvements that have been carried out and also optimised standard hardware was used. By doing this, the energy efficiency was already increased by factor 3. Still, the hardware uses around 430 Watt when running at 25 FPS. The goal is to use around 50 Watt when targeting the newly developed LEGaTO edge server platform.

The Smart City use case simulates the air quality in urban areas. Existing code has been ported from Fortran to C, now running on x86 as well as ARM64 architectures. A first implementation on an FPGA has been developed, showing an energy efficiency increase by a factor of 9. A full-size simulation run takes ~1.3 kWh of energy, but as it runs multiple times a day with new input data, there is a lot of optimisation potential.

The Infection Research use case uses statistical methods to research on the effectiveness of drugs, vaccination strategies and harmfulness of pathogens. The major problem of this use case are the extremely long runtimes of simulations when running with a full-size dataset of several years. To reduce this, the MaxJ toolflow was used to optimise one algorithm, resulting in a speedup of factor 822. A full run with a real dataset would take more than 1.5 years and consume around 187 kWh, so there is a huge potential in both, speed and energy efficiency improvements.

The Machine Learning use case on the one side develops neural networks with a focus on automotive usage, but also optimises existing neural networks with a new kind of deep learning optimisation tool called EmbeDL. With the help of EmbeDL, well known neural networks could be optimised to reach a speedup of up to a factor of 15, while also energy efficiency could be improved by up to a factor of 15.

The Secure IoT Gateway makes it easy to secure many types of network connections to and from IoT devices. It bases on well established technologies like OpenVPN, OPNsense and openWrt and adds an easy to use configuration Web GUI on top of it. As analysis of the used components did not reveal any optimisation possibilities for computation speed or energy efficiency, this use case will be used to enhance the security for the Smart Home and potentially also the Smart City use case.

2 Introduction

This document gives an overview about the actual state of the implementation, optimisation and measurements of baselines of the LEGaTO use cases, represented by the work carried out in Work Package 5 as shown in Figure 1.

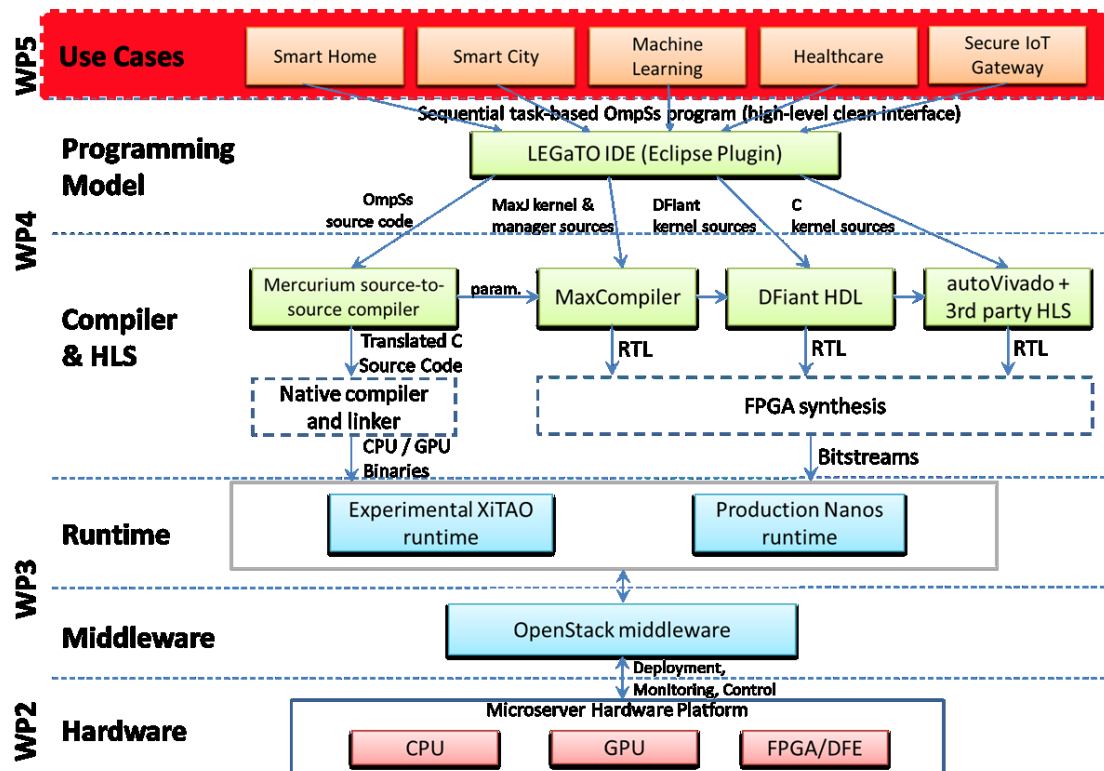


Figure 1: WP5 and the LEGaTO stack

In general, the four use cases of Smart Home, Smart City, Infection Research and Machine Learning have the goal to improve in terms of diverse metrics, such as energy efficiency, as mentioned below. To do so, these use cases have been deeply analysed for their requirements and computational characteristics as presented in the previously published D2.1 [1]. As described in this document, in some cases these use cases have been firstly optimised by hand, improving the algorithms or other by other manual optimisations. Doing this, already good results could be obtained, e.g. in the Smart Mirror demonstrator of the Smart Home use case. As a second step, the already hand optimised algorithms are under improvement by using one of LEGaTO's programming models, as described in detail in D4.2:

- 1) OmpSs & XiTAo
- 2) MaxJ
- 3) DFiant

This has already been achieved for some use cases, revealing great improvements, e.g. for the Infection Research use case using the MaxJ programming model.

The fifth use case, the Secure IoT Gateway, is a development project, adding extra security to the Smart Home use case and potentially the Smart City use case.

The optimisation targets of LEGaTO that we are targeting for are the following, as already depicted in chapter 3 of D2.1:

1. Improve Energy Efficiency
2. Increase MTBF (by 5x)
3. Increase code base security (by 10x)
4. Increase designer productivity (by 5x)

In addition, we also consider the following objectives. These are not explicitly stated as objective in the DoA but could also be relevant:

5. Reduce TCO
6. Reduce Latency
7. Reduce Cost

Still, as all use cases have different core metrics to be optimised for, these will be individually named in the following chapters. See the Conclusion for a wrap-up, which use cases optimised for which metric.

The structure of this document is quite simple, as every use case has its own chapter. In each chapter, a short introduction of the specific use case is given, followed by the actual implementation and optimisation state. For most use cases, there have been some non-optimised baseline measurements done, presented and compared to early optimised benchmarks.

In case of the Secure IoT Gateway, the state of the actual development is described.

The document finalises with a conclusion.

3 Smart Home

As already introduced in D2.1, chapter 3.3: "Current smart living environments are based on the simple automation of subsystems consisting of sensors, information processing, and actuators. The development of assisted living can be seen as a move from isolated applications realized as simple embedded systems, towards cyber-physical systems gathering and processing large amounts of data from a high number of distributed smart devices. Additionally, the smart home has to process interaction of different users simultaneously; conflicting actions have to be recognized (e.g., one user opening a window and another one closing it again) and compromises can be suggested. Different interaction schemes can be combined adaptively, e.g., switching from touch to speech interaction while cooking or using text-based output while phoning. Providing this functionality is highly computationally intensive."

In smart home environments the smart mirror is a more and more frequently used interface for interaction. It is based on a display with a semi-transparent foil applied on it. This device shows personalized information and enables controlling of other smart components and services, e.g., operating the automated wardrobe, turning on/off the lights or opening/closing the entrance door. For this use case a demonstrator based on the open source project MagicMirror is developed and extended with the most needed features of smart homes. Therefore, object and gesture detection are implemented with a neural network as a single shot detector called YOLO [2]. This can recognise 80 different object types and 32 different hand gestures at the moment. A face recognition module is implemented using TensorFlow, widerFace, and FaceNet. For Speech recognition Mozillas DeepSpeech is utilized. With these functions the smart mirror can understand the user and his wishes and offer help for the everyday life. Hardware evaluations and performance analysis with YOLO and Darknet.

As most frameworks base on YOLO, it can be used as an indication for possible performance and hardware requirements. This is especially helpful in selecting the appropriate hardware components for the next prototypes. Therefore, the performance of a single YOLO instance is evaluated to get the feel of the performance of multiple instances.

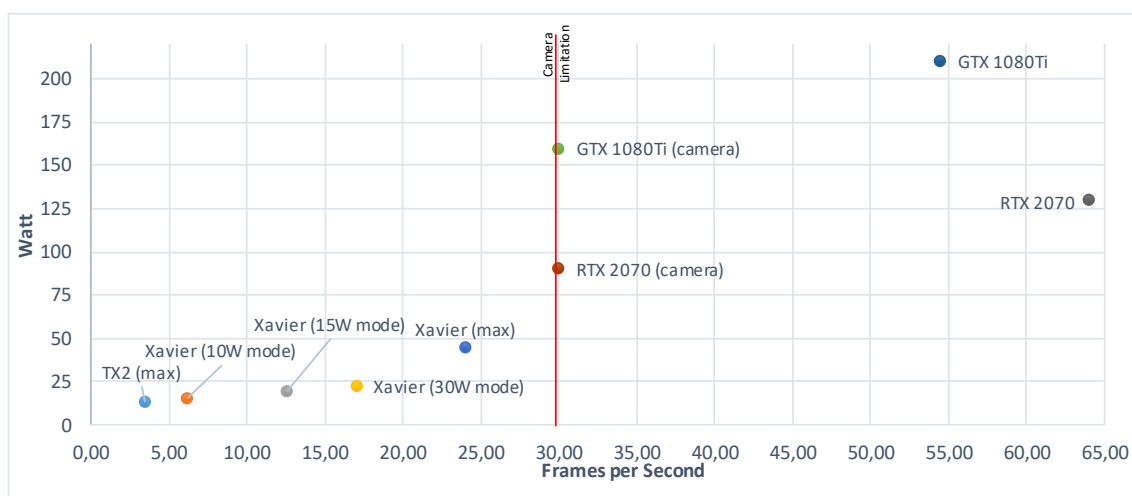


Figure 2 FPS and Watt of YOLO running on TX2, Xavier, GTX 1080Ti and RTX 2070. The application is limited by the maximum output framerate (30 fps) of the Intel RealSense D435i (red line).

In this chapter, the performance on different hardware architectures is evaluated with regards to FPS and power consumption of a single instance. This is visualized in the following graph for NVidia TX2, Xavier embedded modules and GTX 1080Ti or RTX 2070 GPUs. The in the prototype used Intel RealSense supports a maximum framerate of 30 FPS and is therefore marked by the red line. To measure the max performance on the GPUs, a video was used. The GTX 1080Ti runs one YOLO instance with 30 FPS and 160 Watt power consumption with the RealSense being the bottleneck. With a video, around 54 FPS are reached but the power consumption is increased to 210 Watt. This is way too much performance and power consumption for an embedded hardware solution. The introduction of Tensor Cores [3] on the RTX 2070 shows a possible solution for this. While bound by the RealSense 30 FPS, about 90 Watt are reached. The maximum performance of one YOLO instance is 64 FPS and a power consumption of 128 Watt. If the Tensor Cores are not utilized, the maximum performance is reduced by 10 FPS and the power consumption is increased by around 30 Watt. This demonstrates the potential of specialized hardware accelerators for neural networks. For possible embedded hardware solutions NVidia TX2 modules [4] are evaluated, but with 13 Watt and 4 FPS, the performance is not sufficient for the smart mirror use case. The NVidia Xavier module [5] on the other hand shows a decent performance of 24 FPS at a power consumption of around 40 Watt in max performance mode. The integrated Tensor Cores of these modules leverage a possible embedded hardware solution. For the next iteration of the smart mirror two of those Xavier modules will be connected via PCIe to evaluate their full potential.

3.1 Smart Mirror Prototype Hardware Setups

The smart mirror prototype is composed of a display with an applied semi-transparent mirror foil, a camera and a workstation. While the first tests were conducted with a small monitor and normal webcam, the later version of the demonstrator is utilizing an Intel RealSense and a big TV as a display. The RealSense was chosen for the featured depth images. At both recent smart mirror prototypes, the overall application was distributed to two accessible GPUs, due to memory limitations on the GPU and to enhance the hardware utilization and user experience. The smart mirror prototypes were also shown at several fairs to promote the LEGaTO project.

The performance of the image pipeline with face, object and gesture recognition running simultaneously on the first prototype forms the baseline for this use case. Based on this, all optimizations and enhancements are evaluated. The First Prototype

The workstation of the first prototype was composed of two GeForce GTX 1080Ti and an Intel i7-7700K. On this setup, the first version of the demonstration was running with around 12 FPS in all detections simultaneously (face, gestures and objects) and a power consumption of 650 Watt. After porting the limiting python scripts to C and C++ with CUDA optimizations, especially for the Intel RealSense camera handler, the performance was enhanced to 16 FPS for the three detections. This can be traced back to a high CPU computation, for example due to image scaling in python. After the evaluation of the RTX 2070 the focus was shifted towards the second prototype.



Figure 3: The first prototype setup of the Smart Mirror

3.1.1 The Second Prototype

The second prototype is composed of two GeForce RTX 2070 and an Intel i9-9900K. The introduction of Tensor Cores increased the performance to around 20 FPS for all three detections (face, gestures and object). At the same time, the power consumption was decreased to around 430 Watt for the workstation. After additional optimization of the communication structure between the modules within the smart mirror application (number of image streams, partitioning), the performance was increased to 25 FPS for all detections. The latest features like tracking are added on this setup with no performance reductions.

3.1.2 Next Planned Prototype

The newly released feature for the Nvidia Xavier modules allows using them as a PCIe endpoint. The combination of two modules should be enough to run a version of the smart mirror with reduced but sufficient performance. As shown in the previous section, the high FPS rate of the second prototype will not be achieved, but the resource efficiency should be increased, and it should be closer to the power goal of around 50 Watt of the smart mirror. Nevertheless, further optimizations of the software are indispensable to be able to use all available resources. Especially the 16 cores of the combined Xavier modules have to be mentioned here, since making them usable across systems is a particular challenge.

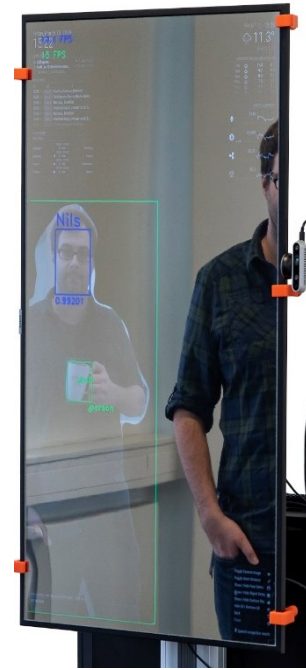
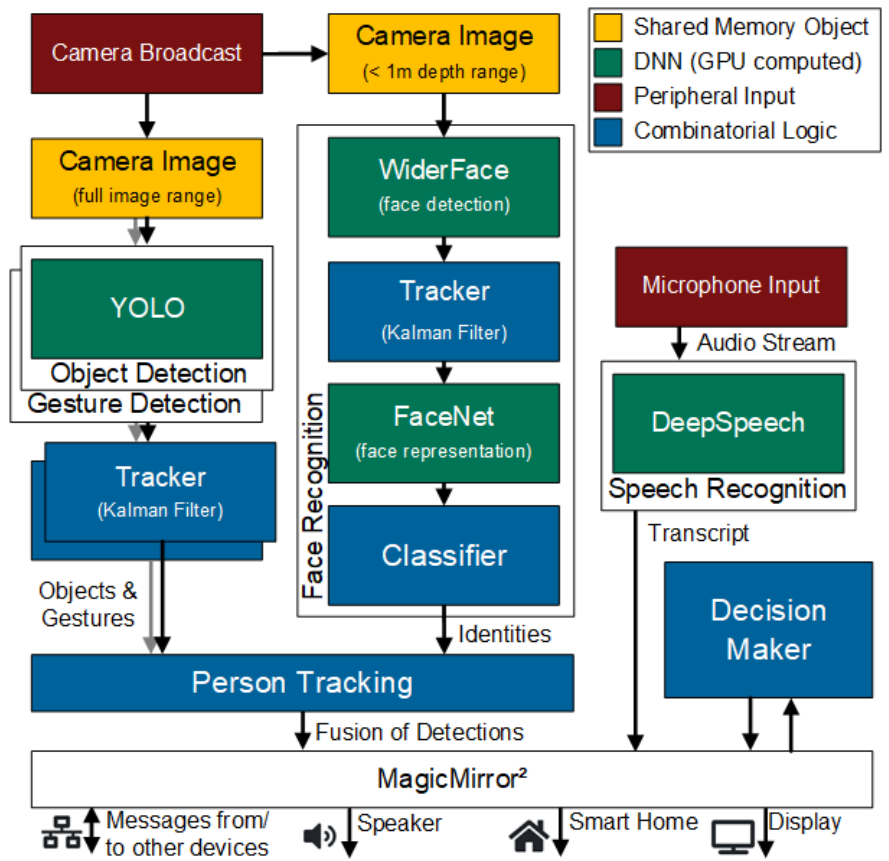


Figure 4: The second prototype setup of the Smart Mirror

3.2 Smart Mirror Software Optimization Progress

To improve the user experience and increase performance and energy efficiency, the data flow between the modules was changed, modules were reconstructed, and Kalman filters were introduced. These changes have increased the performance and reduced the power consumptions of the smart mirror demonstrator. Starting from about 12 FPS in each detection at a power consumption of 650 watts, the last iteration reaches 25 FPS at a power consumption of 430 watts while all detections run simultaneously. Additional features have been implemented that increase usability and developer friendliness, such as a decision maker who calls applications and displays content based on all information. The following chapter will provide further details about all changes made and newly added features.



3.2.1 Module Reconstruction and Optimizations with C++ and Acceleration with CUDA

Several modules were first implemented in python, because it is the common language for neural networks and therefore a fast development could be done. An efficient image processing was neglected at first. After the initial evaluation, the following improvements were made.

- The main camera script, which is preparing the RGB and depth image, was ported to C++ and accelerations with CUDA were made. The CPU load has been reduced by 50% at the cost of a slightly increased graphics card load, which is designed for such image processing. This script provides the image to all other scripts via gstreamer appsink.
- Every neural network which is currently running on the smart mirror, needs a unique image resolution. In first the python version this downscaling was done by the CPU and thereby slow and inefficient and thus was rewritten with GPU acceleration. The Darknet framework is also written in C and uses its own image format. This flow of scaling and converting is next to be combined and optimized.
- The first implementation of face recognition tried to identify each face in each image. This can cost a lot of computation due to a large number of possible faces. This could already be reduced by introducing image depth. By implementing tacking, faces do not have to be

identified in every frame. In the current version, the identity is only checked once every second. This reduced the resource requirement enormously.

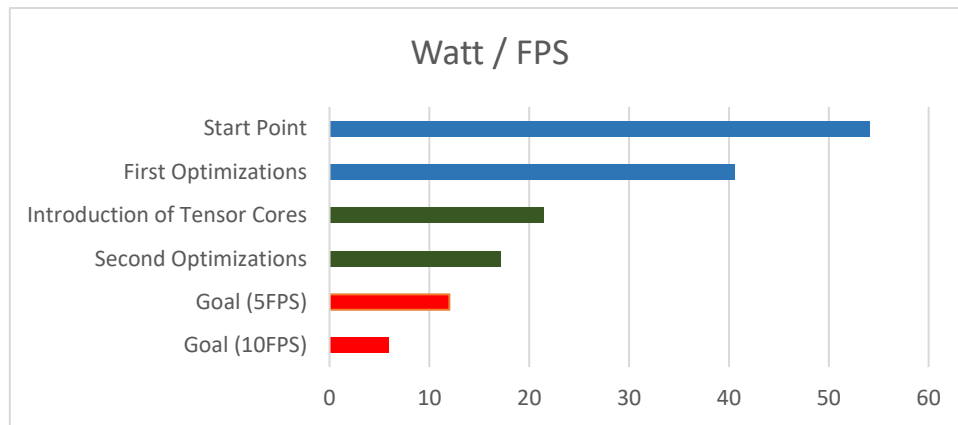
These changes together with the introduction of Tensor Cores have increased the performance from around 12 FPS to 25 FPS with a reduced power consumption to 430 Watt from 650 Watt in the first version. The improvements made show further opportunities for improvement with the LEGaTO toolset. As an example, image conversions and scaling are currently performed with CUDA, but could be performed more efficiently on FPGAs. The Kalman implementation is also not efficient and requires massive CPU computation time. Outsourcing of parallel calculations to graphics cards and FPGAs using the LEGaTO toolset is conceivable.

3.2.2 Evaluating Darknet for OmpSs Optimizations

In every recent prototype setup, a fixed allocation of the neural networks is done. Because of this, the developer must map every instance of neural networks to the accelerators (e.g. GPU or FPGA). In order to solve this problem, possibilities were sought to use the functionalities of OmpSs to give the overall system more flexibility and efficiency. Thus, a GPU shall not be the bottleneck for multiple graphs, while a second GPU is free and idling. The commonly used framework Darknet for the YOLO graph is used as a starting point. It has been compiled using the Mercurium compiler for a threaded CPU version. In this version a defined amount of threads is used to compute the neural network. In the next step, currently in progress, CUDA kernels will be used to exploit the available number of GPUs. Due to the complexity and the data dependencies in a neural graph, this is a big challenge. This is because before the inference can be carried out on the images, the entire graph is loaded into the memory of the graphics card and only the image is transferred during runtime. Due to this procedure and the dependence between the layers of the neural network, the benefit that can be achieved with OmpSs is unclear.

3.3 Baseline Benchmark and Performance

As the baseline for the entire Smart Mirror, the total power consumption and the achieved frames per second in all detections simultaneously are taken into account. As shown in the deliverable D2.1 the initial prototype achieved a performance of 12 frames with a power consumption of 650 watts with less features than the later version. This results in a value of around 54 W/FPS. After the first optimizations and translation from python to C++ with CUDA, 16 FPS with a power consumption of 650 Watt were achieved. This results in a value of around 40 W/FPS. With the second hardware prototype and the introduction of Tensor Cores, the performance was increased to 20 FPS with a reduced power consumption of 430 Watt, which corresponds to a value of around 22 W/FPS. After modification of the communication infrastructure and a more fine granular subdivision, the performance was increased to 25 FPS with the same power consumption. Thereby around 17 W/FPS is achieved. The efficiency was thus increased by factor 3 due to the optimizations carried out so far.



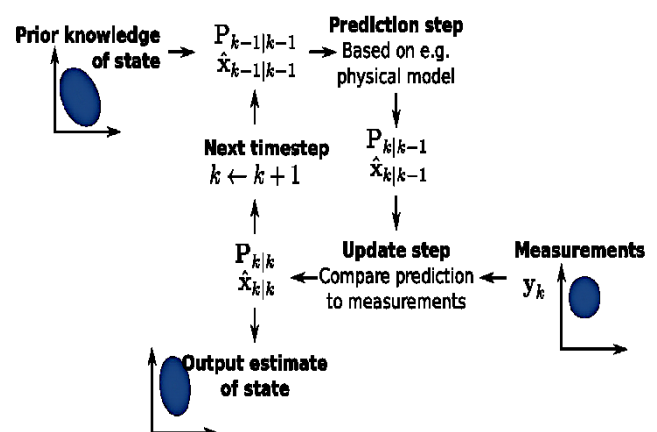
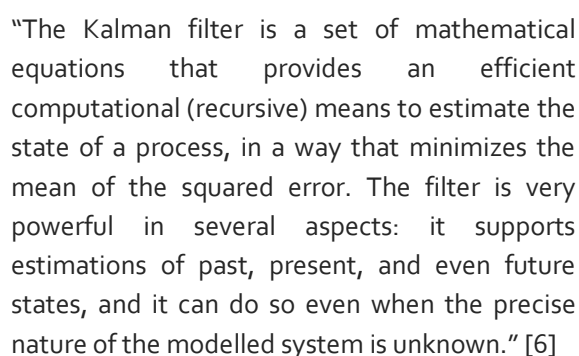
The goal until the end of the project is a targeted power consumption of around 50 Watt with an adequate performance. Based on discussions with potential users, 50 Watt of power consumption seem to be an acceptable quantity for a Smart Mirror. A Xavier module that executes a single instance of YOLO reaches about 2 W/FPS, which should fit the approximate target.

3.4 Development of additional Features

Besides the optimizations with the Legato toolset, further algorithms are developed in this use case, which should contribute to the improvement of user-friendliness. These include hand gesture recognition to control the mirror, Kalman filter for tracking detected objects, and a behaviour prediction. These features are not intended as a direct optimization step, but can lead to an improvement in performance.

3.4.1 Implementation of Tracking using Kalman Filters

A recently added feature is tracking the recognition of faces, objects, and gestures. This allows many simplifications and increases usability. In the past, all decisions were based on findings only, but now the findings tracked are used to make decisions. These can now also be combined and users can be tracked, even if they turn their face away, for example. Kalman Filters were introduced after each detection.



In our case the input of these filters are the bounding boxes of each detection in each frame. After a small number of frames an ID can be assigned for each detection. It predicts the following value and can track objects even if they are not visible for a short time. Due to the distribution and implementation of the detections, each object type needs a separate Kalman filter instance. Unfortunately, the implementation of this mathematical algorithm is very CPU intensive and has to be done after each frame. An optimized version of this filter is thereby desirable.

3.4.2 Dataset of Hand Gestures for easy control

In addition to the commonly used object detection, YOLO is also used for gesture detection in this use case. In scope of this easy to use gesture control, a dataset consisting of 32 hand gestures had to be gathered. No freely available data record exists for this scope. Currently, the gathered dataset for this purpose contains images of around 20 persons with at least 2000 images of each gesture and will be further extended. Evaluations with different YOLO configurations trained with this dataset are currently conducted and a promising first result is used to control the mirror. Due to the nature of this images anonymization would be necessary if the data set should be made publicly available. Solutions for this are currently evaluated.

3.4.3 Behaviour Prediction of the User Interaction

For a really intelligent mirror a form of behaviour prediction is mandatory. Therefore, a first simple RNN structure is currently evaluated. Currently it takes the sequence of visible applications of the mirror as input and predicts the next accessed application. This simple approach shows an accuracy of around 72%. In the next step, additional information like the output of the detections or any other application status will be provided as additional input. The efficient data structure for this additional information are currently evaluated as well as the first the implementation into the smart mirror. It would be desirable for the mirror to recognise the current situation and offer the user possible next actions. As soon as the behaviour of the user can be predicted, strong deviations can be detected, and conclusions can be drawn on possible anomalies which would be the second goal for an intelligent mirror. With this feature unexpected behaviour is recognisable. For example, if the user is having a medical condition like a stroke, he will behave contrary to his other natural behaviour. The developed prediction RNN is designed with focus on generality to allow easy adoption in other areas.

3.5 Next Optimization Possibilities

For the next performance increase, the following system components will be evaluated. The utilization of the available hardware resources is still in the foreground:

- Highly computationally intensive algorithms in the current implementation are above all the Kalman filter for tracking. These require a lot of CPU computation and are limiting the FPS number. Each object type requires its own filter, which ends in a total number of currently 113.
- A second widely used algorithm is image scaling. Before any neural network, the RGB image must be scaled down. Currently the image is scaled 5 times to a certain unique size, because every neural network needs a different input. FPGAs in particular promise an efficient solution.

4 Smart City

In many urban areas air quality and associated impacts on public health are matters of growing concern. The emission and dispersion of critical pollutants (PM_{10} , NO_2 and ground-level O_3) correlate with cancer, asthma, cardiorespiratory problems, brain development in children and reduction of life expectancy in general [7]. As a consequence, air quality monitoring networks and modelling forecasting systems are critical to increase awareness and, ultimately, to assist decision-makers on the adoption of measures to protect public health.

The Smart City use case is composed of four different components that, individually, meet a specific project objective and, in combination, allows the development of an operational air quality modelling system at street-level resolution based on CFD. Figure 6 shows the key components of the air quality modelling system. Most of the individual components have already been developed within other research projects. In this operational workflow, different inputs are required by the CFD model to simulate urban-scale winds:

- **Meteo data:** CFD models require initial boundary conditions to confine the physical problem into a finite computational domain (e.g., a city mesh). These boundary conditions set the inputs of our CFD simulation, defining how the fluid, or wind in our case, enters (inlet) or leaves (outlet) the domain. In other words, boundary conditions connect the region of interest (our urban area) with its surroundings.
- **Sensor assimilation:** In order to model and forecast urban-scale pollutant dispersion, it is not only necessary to dispose of high-resolution near-surface wind fields, but also to characterize the sources of pollutants at street-level (mainly derived from vehicle combustion) through sensors or emission inventories
- **CFD-based wind and pollutant dispersal modelling:** Within the Smart City use case, the CFD-based simulator is the main core of the whole application. To this end, an urban-scale wind forecasting system was developed, based on coupling the meso-scale WRF model (a numerical weather prediction system) [8], with BSC's Alya modelling system [9], a CFD-based simulator. In order to simulate the air flow through the urban-scale morphologies (buildings), the CFD model solves the incompressible Navier-Stokes equations on a computational mesh that represents the city geometry. However, due to its physical complexity, the use of CFD techniques requires massive computing resources.
- **Data gathering and streaming:** After the simulation, data is collected directly from the CFD model, and the environmental network. Pollutant data is post-processed and published using an Open Data format in a BSC repository. The post-processing may require large amount of memory resources to be interpolated and stored in databases depending upon the mesh size and refinement. Finally, the data provided might be used by visualization and analysis tools for their study.

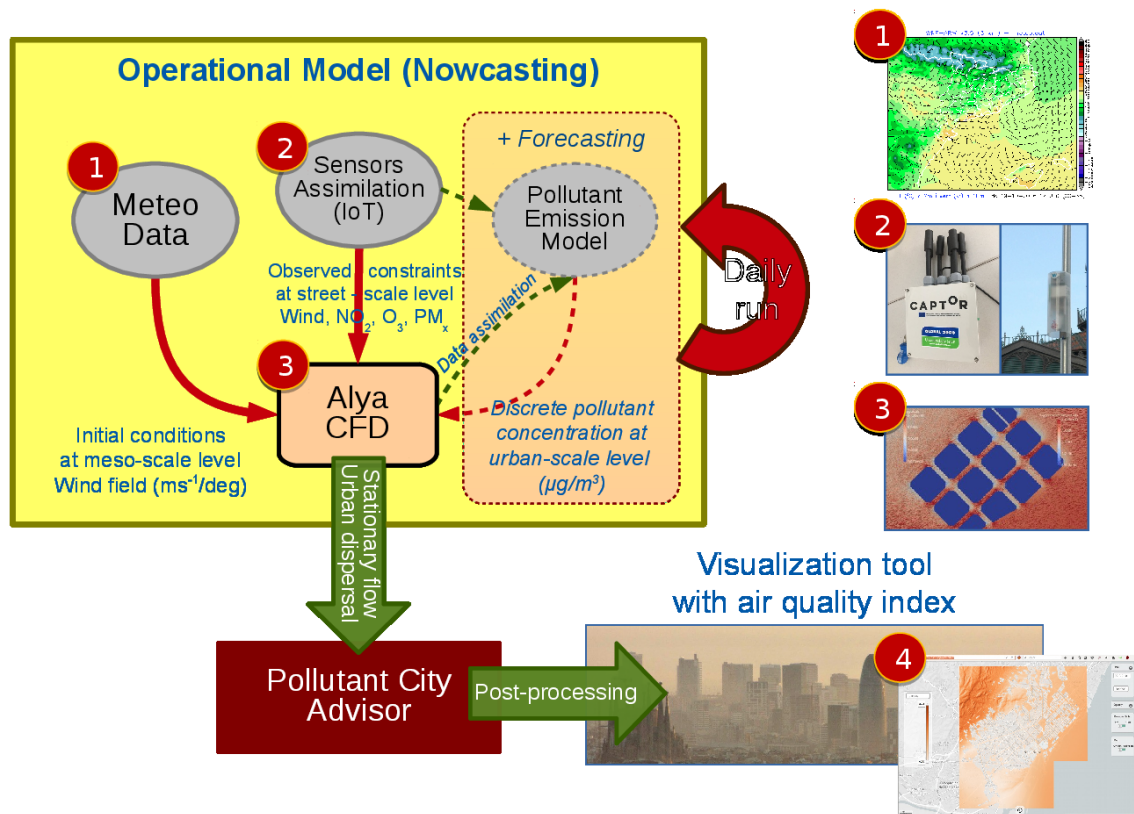


Figure 6. Conceptual sketch of the urban-scale air quality forecast system

On top of this model, the most critical components have been chosen to take advantage of the LEGaTO platform. Actually, within this workflow, the Alya CFD-based simulator is the main compute-intensive component and the one that takes longer to run, and therefore some of its kernels are the main target to be ported to the LEGaTO stack.

This use case aims at demonstrating that monitoring of urban air quality through CFD simulations is feasible for short term forecasts (also known as nowcasting) in an operational workflow built on top the LEGaTO stack.

4.1 Metrics & Optimization Goals

Due to the operational context of the Smart City use case, the air quality model must be constantly executed every update period with new input data from pollution sensors and the meteorological agency. The update period can be so small, between 30 minutes and 1 hour, that it can be critical for running the whole simulation. In this scenario, the LEGaTO stack is pivotal, both in terms of leveraging the processing capabilities to shorten simulation times and improving the energy-efficiency of a highly-demanding urban-scale air quality modelling system. In order to evaluate the LEGaTO implementation, two main metrics must be gathered:

- Use case performance: elapsed time per CFD simulation (*metric 1*).
- Energy-efficiency: Joules per CFD simulation and FLOPs/watt (*metric 2*).

Using the previous two metrics, the following LEGaTO optimization goals will be evaluated for this use case:

- Allow complex urban area simulations (*metric 1*): leverage total execution time objective (10x faster).
- Increase update frequencies in the operational context (*metric 2*): one order of magnitude in energy-efficient objective (10x energy savings).
- Improve FPGA designer productivity: ease the porting of compute intensive kernels to FPGAs devices using OmpSs model.
- Increase the MTBF factor: support fault tolerance by means of agnostic FPGA task checkpointing allowing task replication and reduction of failures during simulations (5x increase).

4.2 Baseline Benchmark

Once that the functionality of the LEGaTO version is checked, a baseline test case must be used to benchmark the Smart City use case. This test set will be used to evaluate the optimization goals of the LEGaTO implementation with respect to the original code using the agreed metrics. The test set was run on MareNostrum IV supercomputer, and the most important metrics were obtained as a reference for the baseline. These metrics will be used in future analysis to evaluate the improvement of the LEGaTO implementation.

The baseline evaluation is performed using a pure-MPI Alya CFD version running the Smart City application. Then, we will evaluate the application by applying the LEGaTO optimizations mentioned above. These will include OmpSs taskification to express efficient parallel execution of the application, using single or mixed-precision floating point with minimal loss of quality, and vectorization for efficient execution on accelerators.

The test set used for the baseline comparison is composed of two tests, a minimal mesh and a full-size mesh:

- *Test 1*: proof of concept mesh with a single square building block of 22 meters height and 150 meters long on each side (see Figure 7). The whole urban-area mesh is composed of almost 1 million tetrahedral elements, with a higher element refinement on the boundary layers of the building and on its leeward wall (downwind side of the building).
- *Test 2*: full-sized mesh of Barcelona city geometry of 500 meters high (see Figure 8). The whole urban-area mesh is composed of 80 million tetrahedral elements, each element with a minimum resolution (length of the element faces) of 15 meters at surface level (streets and building walls).

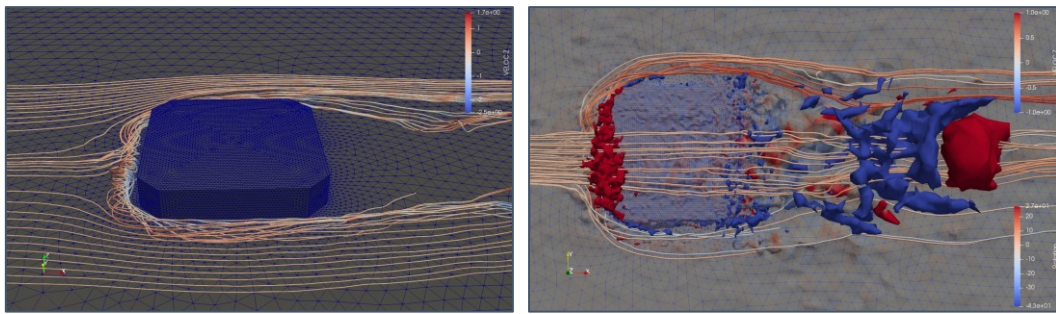


Figure 7. Mesh with a single building block

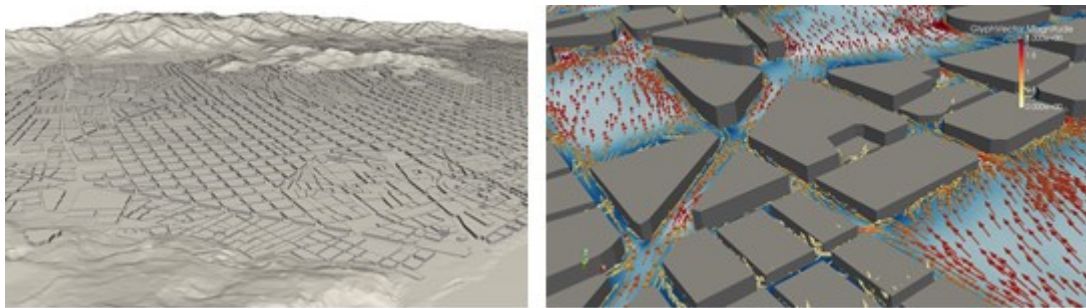


Figure 8. Mesh with a full-sized urban mesh from Barcelona city

As initial condition for the simulations, a west-component (180 degrees) logarithmic wind field profile (10 m/s^{-1} on the top boundary) is used.

Each MareNostrum IV node in our testbed platform is composed of 2 sockets with Intel Xeon 8160 processors with 24 cores each. They run at a frequency of 2.1 GHz and have 96GB of main memory. One single node was used for *test 1* (48 MPI tasks) and 10 nodes were used for *test 2* (480 MPI tasks). Using this configuration, the Alya's CFD model was run in a pure-MPI parallel way over 100 time-steps for both tests. Different metrics were taken from the job execution for future comparisons with the FPGA versions.

On the other hand, we also run some initial tests on the AXIOM board platform. This is an ARM64 bit architecture with a Xilinx FPGA accelerator [10]. However given the memory constraints on this platform, only the ARM64 processor was feasible to be used with the single building block test (*test 1*). The results obtained on both platforms for metrics 1 and 2 were:

Table 1: Metrics for test 1 on MN₄ using 48 MPI tasks

Baseline metrics	Explicit momentum	SpMV	Whole simulation
Elapsed time	12.63s	2.08s	28.75s
Energy consumption	-	-	9741 Joules (2.7Wh → 0.3Wh in the FPGA)
Performance per Watt	-	-	273 MFlops/watt 92.5 GFLOPs (per node)

Table 2: Metrics for test 1 on AXIOM board using 4 OpenMP threads

Baseline metrics	Explicit momentum	SpMV	Whole simulation
Elapsed time	-	-	29800s
Energy consumption	-	-	195190 Joules (54.2Wh)
Performance per Watt	-	-	12.96 MFlops/watt 84.92 MFLOPs (per node)

Table 3: Metrics for test 2 on MN₄ using 480 MPI tasks

Base line metrics	Explicit momentum	SpMV	Whole simulation
Elapsed time	654.9s	268.7s	1505.45s
Energy consumption	-	-	4733167 Joules (1.3KWh → 0.1KWh in the FPGA)
Performance per Watt	-	-	175 Mflops/watt 55.2 GFLOPs (per node)

4.3 Development Status & Optimization Path

Within the Smart City use case, the CFD-based simulator is the main core of the whole application. To this end, an urban-scale wind forecasting system was developed, based on coupling the meso-scale WRF model with BSC's Alya modelling system. On top of this model, the most compute-intensive kernels are being ported to take advantage of the LEGaTO platform. Two main hotspots have been identified as targets:

- Explicit momentum solver: long numerical code where the submatrix for each element in the mesh is computed and assembled in the global system. This is the main optimization target due to their computational cost (~60% of the global time). (*kernel 1*)
- Sparse Matrix Vector operation (SpMV): Sparse L2 BLAS operation used in both explicit momentum (just once per time-step) and implicit pressure solver (iteratively until convergence criteria is reached). Due to its relative computational cost (~10-15% of the execution time) this operation has been also considered to be ported. (*kernel 2*)

The porting of those two kernels to the LEGaTO stack requires at least two operations: the full rewrite of the Fortran kernel codes to C language, and the kernel taskification with OmpSs programming model.

So far, we have fully ported a basic implementation of the explicit momentum solver (*kernel 1*) to C and has been taskified with OmpSs. The kernel has successfully been compiled and run on Intel and ARM64 platforms (Intel Xeon Platinum 8160, Cavium ThunderX2 64-bit ARMv8 [11] and AXIOM board [10]), where the baseline test-case for the SmartCity use case was executed to validate the correctness. This kernel has also been annotated with OmpSs@FPGA directives in order to run on the Xilinx Zynq Ultrascale+ FPGA on the AXIOM board. However, the existing memory on that device was not sufficient to execute the test-case. Its 4GB of memory were not enough to load the mesh, allocate Alya physical variables, use the OmpSs backend (libnanos), and allocate the required buffers to transfer data back and forth to the FPGA processor. As a workaround and for future testing, the development has been moved to a PCIe Alaphadata FPGA board [12] which incorporates more memory.

On the other hand, the SpMV operation (*kernel 2*) has been ported to C, and it is being taskified to use OmpSs programming model. Some preliminary tests are being conducted and tested on Intel platform. In the following months, this kernel will be also tested and evaluated on the PCIe Alaphadata FPGA board.

Our next steps include:

- Check correctness of *kernel 1* & *2* and its FPGA implementation on the PCIe Alaphadata board. [M24]
- Optimize *kernel 1* & *2* by improving memory access, adding tiling and specific HLS pragmas [M30]
- Benchmark Smart City use case on LEGaTO testbed [M24-M32]
- Evaluate fixed-point and reduced floating-point implementation for numerical instructions (right now are FP32-bit) [M32].

5 Infection Research

Recently, Biomarkers are used widely in medical researches to detect the disease early, for diagnosis, in making prognosis assessments or risk assessment, which nowadays plays an important role in modern clinical and preventive medicine. By the new technologies like microarray, next generation sequencing, and mass spectrometry, the researchers can get many biomarkers that may exceed ten thousands [13] [14] [15]. With this development come the courage to find biomarkers to detect the risk to have a disease or diagnose it with high confidence.

To avoid wasting time and money, researchers do pilot studies with a small number of observations as necessary first step for biomarker discovery. Because of the small number of cases and the large number of biomarker candidates, any result might be caused by random effects and statistical significance cannot be proven. For that, the researchers try to reduce the number of biomarkers and extract the most informative ones from these pilot studies to then increase the sample size and achieve an adequate statistical power.

5.1 The background of biomarker discovery

Definition: "Biomarkers are objective indicators of certain, often abnormal, biological states, including pathogenic processes, or pharmacologic responses to a therapeutic intervention. Biomarkers can serve many unique purposes, including screening for early signs of the disease [...], confirmation of the diagnoses, monitoring effects of the treatments, or the progression of the disease and prediction of clinical outcomes." [16]

Commonly known biomarkers are for example the pulse and blood pressure for possible diseases of the heart or the blood sugar level for diabetes mellitus. A biomarker must be measurable and reproducible. It indicates a biological state or condition. Biomarkers can not only be used to identify a disease, a state of a disease or to give a prognosis. But also they are utilized in research to examine organ function, to develop new drugs or to design an individual therapy for personalized medicine.

Biomarkers can be found through quantification of various molecules in biological fluids and tissues. Today high throughput technologies enable to examine those molecules. High throughput screening allows to quickly carry out millions of chemical, genetic, or pharmacological tests generating huge amounts of data. Even though profiling became cheaper in the last years gaining a single sample is still expensive.

In consequence of the rapid development of next-generation sequencing devices, standardized procedures and data formats as well as comprehensive quality management considerations are rare. [17] Having different experiment designs and different data formats the achieved results of the different next-generation sequencing devices are hardly comparable. Even using different equipment of the same type can lead to varying results. In addition also variability in data normalization and other steps of the data processing can affect the consistency between separately generated data sets. [16]

High throughput data from biological experiments are gathered into vectors whose dimension correspond to the number of simultaneous measurements. Biological datasets from various sources are hardly comparable because of reasons described above and even analysing a single sample is expensive. Therefore we normally have a small sample size n of comparable data and a big space of dimension p (a high number of biomarker candidates). [16]

The number of samples (n) must be larger than the number of biomarker candidates (p) for good classification performance to get a reliable diagnosis.

To avoid wasting time and money, researchers do pilot studies with a small number of observations as necessary first step for biomarker discovery. Because of the small number of cases and the large number of biomarker candidates, any result might be caused by random effects and statistical significance cannot be proven. For that, the researchers try to reduce the number of biomarkers and extract the most informative ones from these pilot studies to then increase the sample size and achieve an adequate statistical power.

5.2 Description of the application

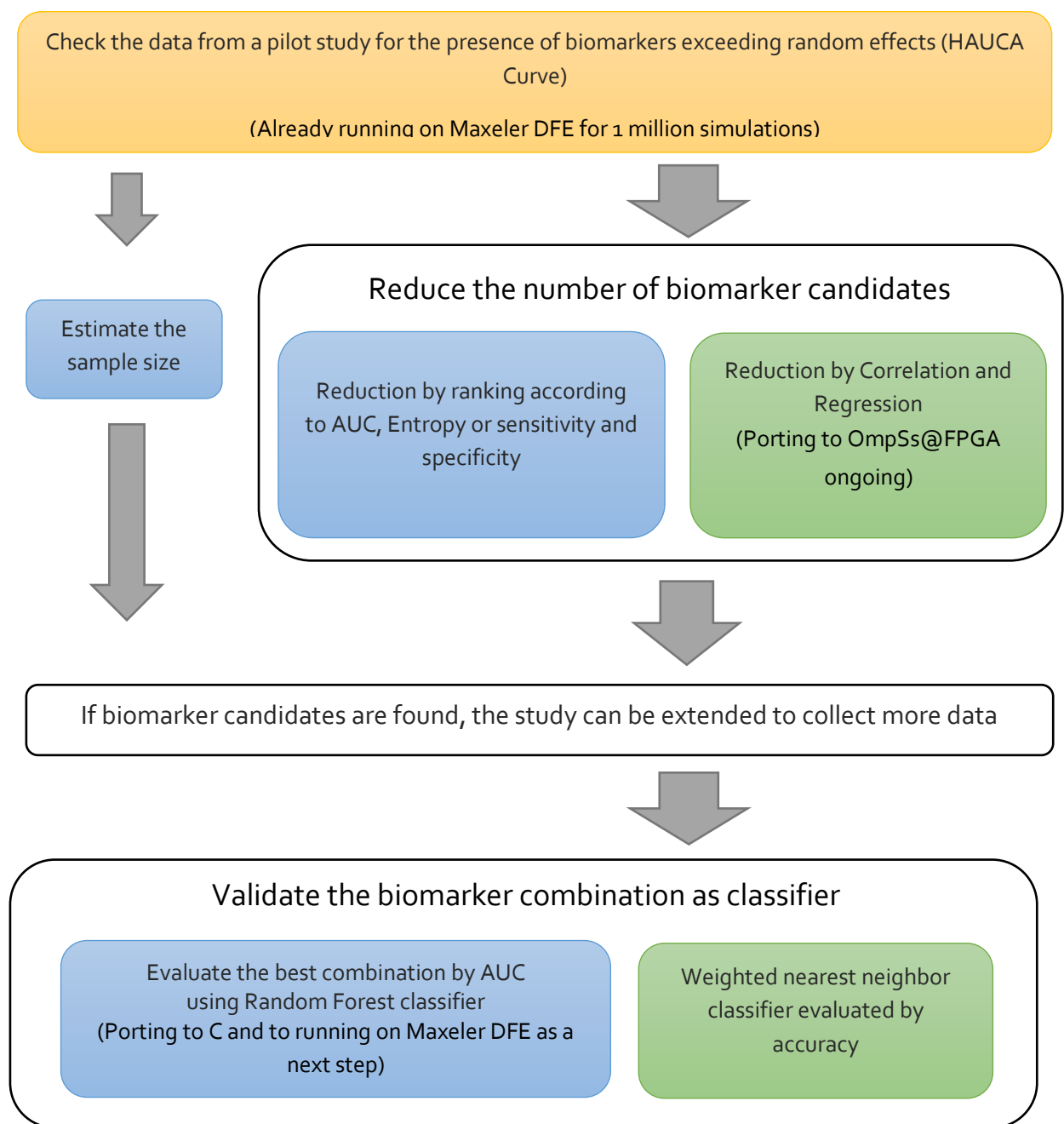


Figure 9: Workflow of the Infection Research Use Case

In a first step (marked in yellow) we check the data for the presence of biomarkers that have an association, with an outcome exceeding the pure random effect. The HAUCA curve method shows how many biomarker candidates in the real dataset exceed a specific value of AUC or entropy (two of the performance measurements) compared to a random dataset and a confidence interval. To measure the probability of how much biomarkers exceed each value of AUC can be calculated. However, for entropy values simulations have to be done.

To reduce the number of biomarkers in the first algorithm (marked in blue) we extract the most informative ones by ranking them according to AUC, entropy, sensitivity or specificity. We estimate the sample size for extended studies in order to achieve an adequate statistical power. After collecting enough data we repeat the biomarker selection process and evaluate the best combination of biomarkers by AUC using Random Forest.

In a second approach (marked in green) we also want to discover biomarker combinations by regression and classification. In this algorithm we do not directly predict the class from the training data. Instead we predict every biomarker candidate by regression. From the importance of the label for the prediction and the regression error we calculate a weight for this specific biomarker candidate. We assume that the higher the weight, the more relevant the biomarker candidate is. As regression model we use lightGBM as a gradient boosted tree implementation. This approach is computationally very expensive. To be able to apply it to real biological data with up to 50,000 biomarker candidates extreme powerful hardware and further optimization of the algorithm is needed.

5.3 Metrics & Optimization Goals

Our goal is to decrease the calculation time and the energy consumption by optimizing parts of the workflow. We also want to handle larger data sets (more biomarker candidates) and multiclass problems. The number of biomarkers and the size of the data sets we will be able to handle is crucial for our future projects. We are part of the project consortium i.Vacc (Paving the way towards individualized vaccination). The project will start in 2020 funded with one million Euro within the framework “Big data in the future life sciences” of the Lower Saxony Ministry for Science and Culture in Germany. In contrast to our actual projects where we usually handle data from one so-called omics fields (metabolomics, proteomics, transcriptomics, genomics), we will look jointly at different types of omics, increasing the number of biomarker candidates drastically. At the moment, we can barely carry out the required computations for data of one omics type. Analysing data from three or four different omics fields jointly does not mean that we have to apply our algorithms three or four times but the size of the biomarker candidate data set increases by factor of three or four. Without a significant speed-up, we will not be able to cope with such projects as i.Vacc in the future. Our goal is therefore to work with data sets at least three times as large as the ones we can handle at the moment. This means we want to extend the number of biomarker candidates to >50,000 and the sample size to >100.

To evaluate our progress we measure the runtime and the energy consumption of our algorithm. Furthermore we count the number of additional biomarker candidates we can include in our calculations.

5.4 Baseline Benchmark

To check the data from a pilot study for the presence of biomarkers exceeding random effects with HAUCA Curves we determined the most time and therefore also energy consuming part of the code.

For the benchmark below we considered the cut index function. We generated an artificial dataset with 50 observations, 4 classes and 1 million simulated biomarker candidates. It should be noted that due to technical restrictions, the measurements were taken on a comparatively weak standard hardware. But since the number of computation steps tends to increase exponentially with the number of biomarker candidates for the combinations, a more powerful standard hardware would not bring too much advantage.

Number of biomarker candidates	Number of simulations	Calculating time (standard hardware/ original R-Code)	Energy consumption (standard hardware/ original R-Code)
10 000	1 000 000	2,5 hours	126000 Joule

We try to find all possible combinations of the features that we have already selected before to find the best combination of biomarkers. After finding all possible combinations we measure the performance of each model (each combination). For the benchmark above we selected 18 from 112 biomarker candidates, due to the impossibility of finding a best model from a large number of biomarkers (>20) which can take years. For instance, to find a combination from 25 biomarkers we estimate a duration around 1 year and 4 months while to find a combination from 30 biomarkers we estimate already around 41 years. The run time increases dramatically when we increase the number of biomarkers, due to the increase of possible combinations ($2^k - 1$ combinations where k is the number of selected biomarkers).

Number of observations	Number of biomarkers	Calculating time (standard hardware/ original R-Code)	Energy consumption (standard hardware/ original R-Code)
66	18	~ 4 days (345600 seconds)	4147200 Joule
66	20	~ 2 weeks (estimated)	n/a
66	25	~ 1.3 years (estimated)	n/a
66	30	~ 41 years (estimated)	n/a

The benchmarks were run on the following hardware, which are development machines. It is planned to later run parts of our code on the cluster of HZI to enhance the run time and the energy consumption.

The hardware and R-Version used for the HAUCA Curve and the biomarker combination is:

- Intel Core i7-6600U (2 Cores, 2.60 GHz, 3.40 GHz Turbo, 4 MB cache, 15 Watt TDP)
- Installed RAM: 16 GB
- R version: 3.5.2 (2018-12-20)

For the biomarker candidate (feature) selection by regression the components of the test system are:

- Intel Xeon E-2186M (6 Cores, 2.90 GHz, 4.80 GHz Turbo, 12 MB cache, 45 Watt TDP)
- DDR4-RAM, 16 GB, 2666 MHz, ECC
- The application runs currently with Python 3.7.3

The test data includes 50,000 biomarker candidates and 16 samples. The calculation time of one biomarker candidate takes about 17 Minutes. The estimated total runtime of 1.7 years does not take into account that considering correlations within the data will prolong the calculation. So the real calculation time will be even longer.

Biomarker Candidate(s)	Calculation time	Energy consumption
1	1037 seconds (~ 17 Minutes)	13 481 Joule
50 000	~ 1.7 years (estimated)	~ 674 050 000 Joule or 187,2 kWh (estimated)

5.5 Development Status & Optimization Path

	Number of biomarker candidates	Number of simulations	Calculating time (standard hardware/ original R-Code)	Energy consumption (standard hardware/ original R-Code)	Calculating time DFE Cut index	Energy Consumption DFE Cut index
done	10 000	1 000 000	2,5 hours	126000 Joule	5.49 seconds	4524

As presented in the table above, so far, benchmarks with up to 10 000 biomarker candidates have been carried out with 1 million simulations, the next step will be to do simulations with 50 000 biomarker candidates and 5 million simulations.

	Original R code	C++ Cut Index	DFE Cut Index
Time(s)	4514	74.8	5.49
Speedup	1(baseline)	60	822

In cooperation with Maxeler and their MaxCompiler, the computing time has been reduced dramatically by accelerating key parts of the computation with Maxeler DFEs. As a first step the application was profiled for its most computationally intensive components. These ones naturally represent the best opportunity for application speed-up. Due to the close link between efficient processing in terms of speed and energy on DFEs, successful application speed up by offloading compute intensive parts to DFEs also typically leads to significant energy reductions. In this case, the cut index function which is part of the R package "discretization" was chosen for DFE offloading and acceleration. This process meant the cut index functionality was re-implemented in Maxeler's MaxJ language and compiled to a state-of-the-art MAX5 DFE cards with MaxCompiler. In order to

facilitate the integration of the DFE accelerator functionality into the original R application, a C++ interface layer was created in order to call the Maxeler SLIC API. The application was then run and evaluated on the testbed system located at the Jülich Supercomputing Center (JSC). This testbed system provides the following system configuration:

- A high-end CPU server with
 - 2x AMD EPYC 7601 CPUs
 - 1 TB RAM
 - 9.6 TB SSD storage
- A Maxeler MPC-X 3000 dataflow node with
 - 8 x MAX5 DFEs

The above system is described in more detail in Deliverable D2.2. On this system, the accelerated version of the biomarker candidate application with cut index computations carried out on the DFEs, 1 million simulations could be computed in a time of 5.49 seconds while in the normal computer it took 2.5 hours.

5.6 Further Plans

We are about to look at more than 50,000 biomarkers so we need to estimate a probability smaller than $1/50000$, which means we need to run up to 5 million simulations (to have a 95% chance that the (relative) error is less than $\pm 20\%$). Because of an even longer calculation time it can only be done with Maxelers DFEs.

To find biomarker combinations from bigger sets of biomarkers than 20 (2 weeks) we plan to port the most time and therefore also energy consuming part of this code from R to C. We plan to accelerate the application by adapting and running it on Maxelers DFEs.

The next steps are running 5 million simulations for larger datasets, calculate possible combinations for more than 20 biomarkers and involve multiclass data.

The baseline to calculate one biomarker candidate by regression is about 17 Minutes per Biomarker Candidate.

If we would calculate the whole dataset with 50,000 biomarker candidates, it would take more than one year to finish. Hence the calculation time and analogous the energy consumption of the complete calculation must be reduced.

To achieve the reduction of the calculation time, we plan to further enhance the algorithm itself and adapt it using OmpSs and the hardware from UNIBI. The core of the regression algorithm (Microsoft/lightGBM with MIT licence), a well-known machine learning tool, is now implemented with OpenMP. The conversion from OpenMP to OmpSs@FPGA has started in order to run the algorithm on the hardware provided and supported by the LEGaTO project. To achieve this goal, we installed and tested successfully a ZedBoard Zynq-7000 ARM/FPGA SoC Development Board as test environment. A first compilation of OmpSs is done. We started to analyse the Code for the most promising and useful part to be ported to OmpSs@FPGA.

The next steps include the further optimization of the algorithm and to run first examples in our test environment. This will be followed by adapting, compiling and running parts of lightGBM with OmpSs@FPGA in collaboration with UNIBI and BSC. The goal is to run 5 million simulations for larger datasets, calculate possible combinations for more than 20 biomarkers and involve multiclass data.

6 Machine Learning

Although there are many success stories of deep learning (DL), which has spurred a wave of public and corporate interest in artificial intelligence (AI) and machine learning (ML), there are still many unsolved issues. One of these is how to make the DL models more efficient; both from an energy perspective as well as cost perspective. This is especially true for embedded applications, e.g. autonomous driving, IoT and robotics. To solve this problem, we are developing a new kind of DL optimisation tool (EmbeDL) that will make DL models use significantly less energy and require less computations in order to be deployed to cheaper hardware. We will demonstrate the technology developed on vision applications relevant for autonomous driving: image classification, object detection and pixelwise semantic segmentation.

6.1 Metrics & Optimization Goals

Multiple of the metrics in the LEGaTO Project are highly interesting for the ML application. As mentioned, the energy consumption of today's state-of-the-art perception system are too high and needs to be decreased. Thus, energy is the most important metric for the ML use case. The second most important metric is to make the models more computationally efficient. By doing so, the same original model can be deployed to cheaper hardware. Latency is also very important, since lower latency can be the difference between an accident or not. Increased MTBF and code base security is of course also important for having operational and secure autonomous vehicles. Thus, in order of importance, the metrics and goals are:

- 10x energy efficiency [frames/second/watt] (Metric #1)
- 10x faster execution [frames/second] (Metrics #5 & #7)
- 5x lower latency [milliseconds] (Metric #6)

In D2.1 we specifically mentioned one DL model and target goals for that. However, we would like to demonstrate the model agnostic capabilities of the optimisation engine by evaluating on several models. We then compare with TensorFlow, by Google, as the baseline for the model.

6.2 Baseline Benchmark

The models we have used for our benchmark are VGG16, GoogleNet and ResNet. These models are commonly used as the main component in basically all image/video related applications.

VGG16 (Very Deep Convolutional Networks for Large-Scale Image Recognition) is a very influential paper on how to design deep learning models for image analysis. It has over 15 000 citations and is regularly used as a reference model. [18]

GoogLeNet (aka Inception v1) was developed by Google and achieved a new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC2014). [19]

ResNet34 (Deep Residual Learning for Image Recognition) was developed by Microsoft and is the foundation in models used to win competitions like ILSVRC & COCO 2015, ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. [20]

We did benchmarks of these three common vision models on Nvidia's Jetson TX2 platform locally on a development board. As a baseline, we compared with the most used DL framework, TensorFlow, developed by Google. Comparing with TensorFlow is the baseline we will use in general when possible. However, some hardware, e.g. FPGA, does not have support for TensorFlow directly. In these cases we will compare against what is considered the de facto standard of running DL on that platform.

The results of these initial benchmarks on an Nvidia TX2 development board can be seen in the tables below.

TensorFlow	<i>VGG16</i>	<i>GoogleNet</i>	<i>ResNet34</i>
<i>Speed (FPS)</i>	22	34	69
<i>Energy (FPS/W)</i>	4.0	6.3	20.4

EmbeDL	<i>VGG16</i>	<i>GoogleNet</i>	<i>ResNet34</i>
<i>Speed (FPS)</i>	332	108	214
<i>Energy (FPS/W)</i>	61.5	25.1	59.4

Relative Improvement	<i>VGG16</i>	<i>GoogleNet</i>	<i>ResNet34</i>
<i>Speed (FPS)</i>	15x	3.2x	3.1x
<i>Energy (FPS/W)</i>	15x	4.0x	2.5x

VGG16 is the model we have analysed more thoroughly when large parts of the optimiser have been developed, which might explain the difference in optimisation results between VGG16 and the other models. We are quite confident that there are some low hanging fruits for improvements on the GoogleNet and ResNet34 models.

6.3 Development Status & Optimization Path

We have made great progress with our optimisation engine, EmbeDL. We have also implemented three baseline models, described above. Regarding the integration of OmpSs, a first implementation of an OmpSs taskified DL model has been done for CPU and this implementation has also been ported to OmpSs@FPGA. The integration is in early phase and improvements will be done.

EmbeDL relies on multiple techniques and we have more techniques on the roadmap throughout the rest of the project. A core piece of our technology developed in LEGaTO is the hardware aware hyperparameter optimiser that combines different optimisation techniques and parameters, e.g. on a per layer basis set the pruning level. This is vital when the number of techniques and corresponding

parameters grow in size and makes it infeasible to combine methods by hand. This technology is useful now, but will be of paramount importance going forward adding more techniques and methods for optimisation. The DL optimiser shows some very promising results. However, we have several more techniques in the pipeline that we would like to integrate as well as making the tool completely automated.

To accurately measure energy usage and to develop and verify extensive hardware support, we will in the future do the evaluation and benchmarking on the LEGaTO testbed. The benchmarking done up to this point has been for the internal development of the DL optimisation engine. To improve and demonstrate the model agnostic capabilities of the optimiser, benchmarks on several models needs to be done. We are currently working on support for object detection and pixel-wise semantic segmentation to complement our current image classification models.

The next steps include:

- Complete implementation of PSPNet and YOLO [M24]
- Complete OmpSs integration for CPU and FPGA [M30]
- Benchmark on LEGaTO testbed [M30]
- Develop a demo [M30]
- Continue development of DL optimisation engine [M36]

7 Secure IoT Gateway

The Internet of Things (IoT) is on everyone's lips. There are completed products or instructions for do-it-yourself projects for nearly all interest groups. Proceedings in that are already having effects on our lives as more and more of our devices are communicating among each other, no matter if we are a private or commercial user. The security of IoT is often unattended for different reasons: it would be unnecessary or it would be way too complex, to name just two often heard explanations.

To provide a solution to reduce the complexity of IoT security, the Secure IoT Gateway is being developed.

7.1 Use case description and motivation

Talking about Industry 4.0 and Internet of things, the security of communication should be a main focus. Many devices are communicating among each other or to a server. This could be in a local area network like a company network as well as a wide area network like the internet. No matter whether an IoT device is located in a local or wide area network, the communication through the network needs to be secure for any attacks. As any part of the communication can be attacked, an effective security concept is needed, covering the whole path between sender and receiver. The security of IoT and/or remote network devices is really complex, so the Secure IoT Gateway will support the Smart Home and possibly Smart City use cases by simplifying this complexity. So the Secure IoT Gateway is a special use case in the LEGaTO project as its main goal is not to optimize the energy efficiency as the other ones but to reduce the complexity of IoT security.

The Secure IoT Gateway is an application composed of four components working together to create a secure communication.

- 1) The **IoT Bridge** is a small local device running a Linux system with OpenWrt. The IoT Bridge is placed at the IoT device that should be secured. The connection will be encrypted on-the-fly and can be established with a Gateway Cluster or a Network Gateway. As a component in a local area network, the IoT Bridge is not multi-customer capable.
- 2) The **Network Gateway** is a local network component and therefore not multi-customer capable. The Network Gateway is a server with several LAN ports and runs an OPNSense instance. It allows secure connections to IoT Bridges. All the communications between the components can be regulated by rules. The Network Gateway secures the local area network.
- 3) The **Gateway Cluster** is a wide area network component and will be hosted by a systems house. It is a Linux system running several virtualized OPNSense instances, one per customer. By doing this, the Gateway Cluster is the central component for providing gateways. Inside a virtualized OPNSense instance, a Master Gateway provides a secure network connection to the datacenter/server for the customer's Gateway and allocates the public IP addresses. Each customer is automatically supplied with an autonomous gateway, so it's separated from the other customers gateways. To handle several customers, the Gateway Cluster is multi-customer capable. The Gateway Cluster is the only optional component of the Secure IoT Gateway as it would not be needed if the customer's IoT data stays on-premise.
- 4) The **Network Cockpit** is a GUI the user (normally the admin) will use to configure and monitor the IoT Bridges, Network Gateways and Gateway Clusters. Normally, the user just

Single Page Application with its interactivity so the user experience isn't suffering caused by long loading times.

As a web application, the Network Cockpit comes with access authorization. So just authorized users are allowed to use the Network Cockpit. The access to the web application will be expired after an hour. The authorized users will be furthermore differentiated in several permission roles:

- 1) A **User** just has the permission to read the configuration of the devices. With the exception of the own settings like username, password and application language, no configuration can be done. A user could be an employee or technician and is bounded to a customer.
- 2) An **Admin** has the permissions to read and set configuration of the devices. In addition to configure the existing devices, the admin is allowed to create new accounts, new devices and Secure IoT Gateway components like IoT Bridges. An admin is normally the internal admin of the customer and is bounded to this.
- 3) A **Super Admin** has all permissions like the Admin: creation of users, devices and Secure IoT Gateway components and setting the configuration of devices and components plus the Super Admin maintains the Gateway Network and is able to see all customers with their secured connection environments which are hosted by the systems house. As a consequence, the Super Admin is an employee of the systems house.

As application language, you can choose between English and German by now. The internationalization is given by the nuxt-i18n plugin.

All data is stored in a MongoDB, a NoSQL database, with Strapi used as content management system in order to create, read, update and delete data. Although Strapi comes with a lot of useful features, the email plugin was customized so personalized emails can be sent from the web application. By now, emails will be just sent if a new account was created. To keep the data in a consistent state, Vuex is part of the application as a State Management System. This means that every change in a data set is global so there won't be different states in different parts of the application.

After login to the Network Cockpit, the user gets to the dashboard, see Figure 10. The user can use the dashboard or the navigation bar in the header to navigate to the different parts of the application.

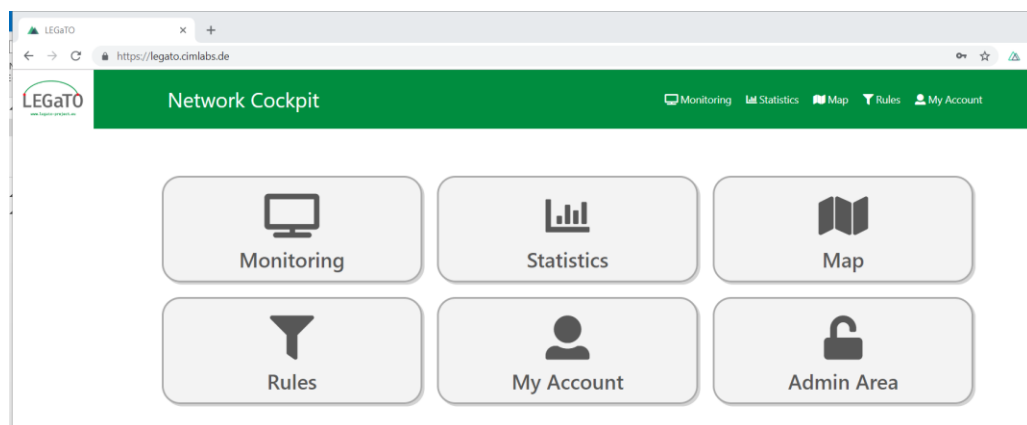


Figure 11: Dashboard of the Secure IoT Gateway

The **Monitoring** page is an overview of the hosted IoT bridges, see Figure 12. It shows several information like the assigned customer, status, IP address, uptime and the traffic. As Super Admin you see all IoT Bridges, as an Admin bounded to a customer you just see the IoT Bridges which belongs to your company.

Network Cockpit

Monitoring

Statistics

Map

Rules

My Account

Bezeichnung	Gateway	Client	Status	Location	IP address	Uptime	VPN Traffic	MacAddress	Firmware	Devices
Bridge R1234	Customer's Gateway	Customer	ok	Room1234	172.20.13.11	47d 9h 42m	603 MB		OpenWrt 19.5	0
Bridge	J.Doe Gateway	John Doe	ok		172.45.10.50	32d 18h 33m	426 MB		OpenWrt 19.5	2
Bridge Server Room	Customer's Gateway	Customer	ok	Server Room	172.20.113.10	46d 9h 29m	821 MB		V3.0	3

Figure 12: Monitoring page of the Secure IoT Gateway

A click on an entry in the table will open a detailed view as overlay, see Figure 13. In this overview more information about the IP configuration, the IoT Network, WLAN configuration and hardware like used RAM, CPU usage and bandwidth are shown. Also the connected IoT devices are listed.

Bridge Server Room

MAC:

Description: Bridge Server Room

Location: Server Room

Location: V3.0

IP Configuration:

IP Address: 172.20.113.10

Subnetmask: 255.255.255.0

Connected to Gateway: Customer's Gateway

Gateway IP:

VLAN ID:

IoT-Network DHCP: off

DHCP Range:

IoT Network: 172.20.113.20

WLAN Configuration:

SSID:

Preshared Key:

WLAN Network:

VPN Summary

Tunnel Network:

Incoming Port:

RAM (MB):

used RAM (MB):

CPU Usage (%):

Bandwidth (bps):

Last error:

Connected device(s):

Weatherstation

FibonacciClock

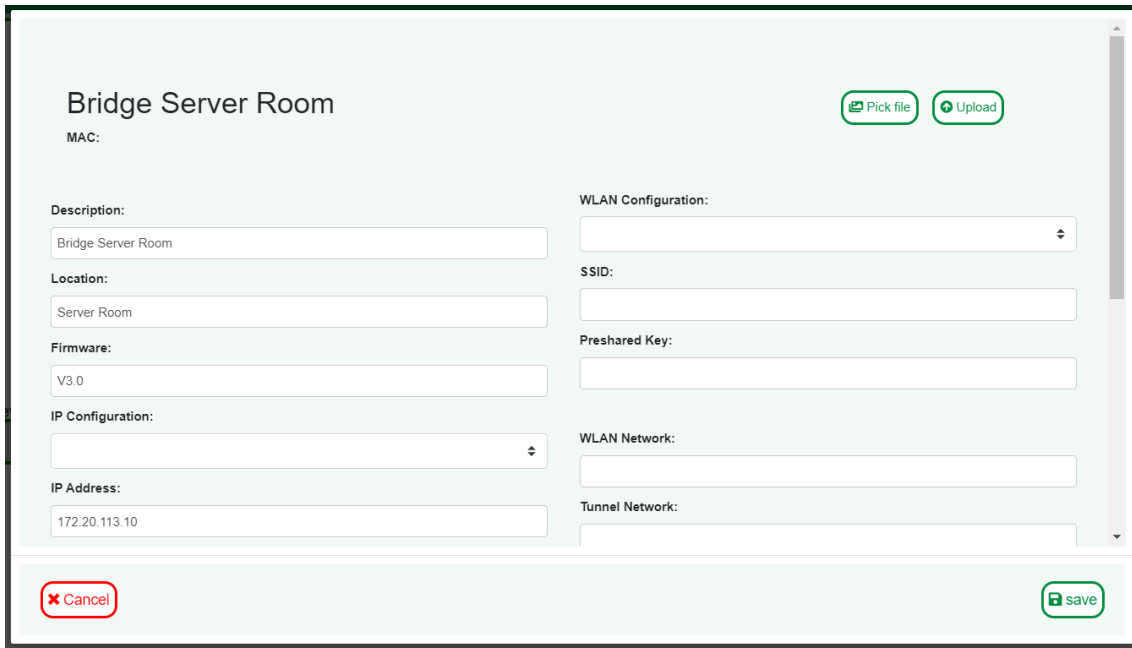
Smart Mirror

Edit

OK

Figure 13: Information dialog of a Bridge device

As Admin or Super Admin you are allowed to edit the data, so you can choose Edit in the right corner of the overlay to get to an editable overview, see Figure 14. Also an image can be picked and saved as file so it will be shown on detailed view.



The image shows a web-based configuration interface for a 'Bridge Server Room'. At the top, the title 'Bridge Server Room' is displayed. To its right are two buttons: 'Pick file' and 'Upload'. Below the title, there is a 'MAC:' label followed by an empty input field. The main configuration area is divided into two columns. The left column contains fields for 'Description:' (with 'Bridge Server Room' entered), 'Location:' (with 'Server Room' entered), 'Firmware:' (with 'V3.0' entered), 'IP Configuration:' (a dropdown menu), and 'IP Address:' (with '172.20.113.10' entered). The right column contains fields for 'WLAN Configuration:' (a dropdown menu), 'SSID:' (empty), 'Preshared Key:' (empty), 'WLAN Network:' (empty), and 'Tunnel Network:' (empty). At the bottom of the dialog, there is a 'Cancel' button on the left and a 'save' button on the right.

Figure 14: Edit dialog of a Bridge device

To see more information about the connected IoT devices, one can click on the link in detailed view and another overlay is opened with detail information of the choosen IoT device, see Figure 15. Some information of the device itself can be seen like firmware and hardware specifications as well as its IP address, the connected IoT Bridge, the used protocol, the healthcare status, bytes sent and received and when the last error was thrown.

As mentioned before, it is switchable to an editable view by clicking on the Edit button in the right corner.

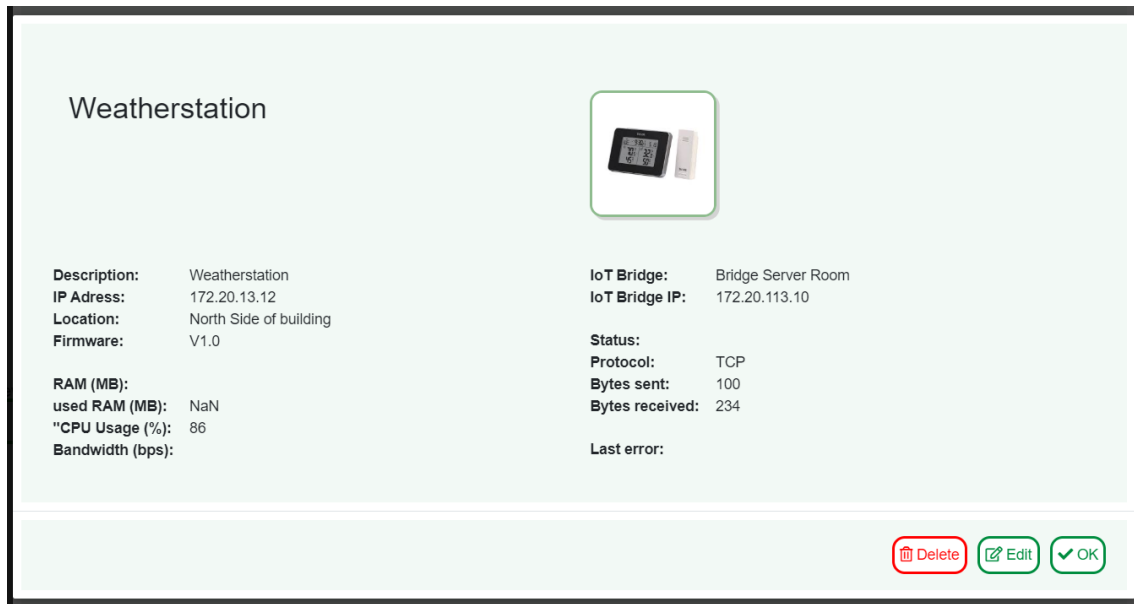


Figure 15: Information dialog of a connected Device

The **Statistics** page, see Figure 16, shows measurable values like the data speed for incoming or outgoing communication of a component (IoT device, IoT Bridge or Network Gateway).

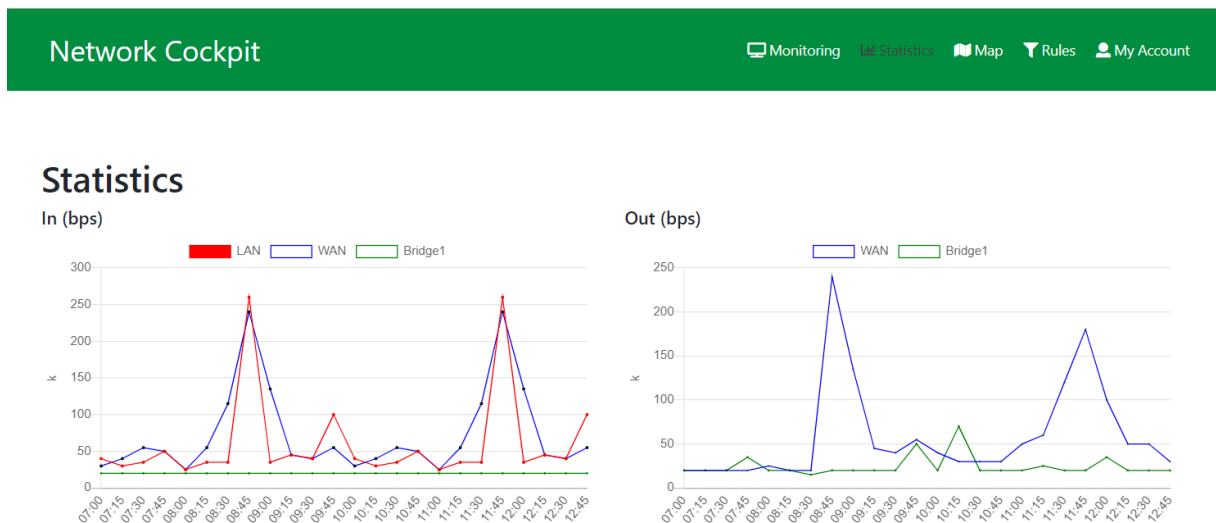


Figure 16: Statistics view of the Secure IoT Device

Later there will be more data shown and filters will be implemented for the timescale in order to be able to do time travelling to an error.

With **Map**, an overview in style of a network plan is given. The Super Admin sees all the Network Gateways and their connected IoT Bridges and IoT devices while the Admin just sees the own IoT environment, see Figure 17.

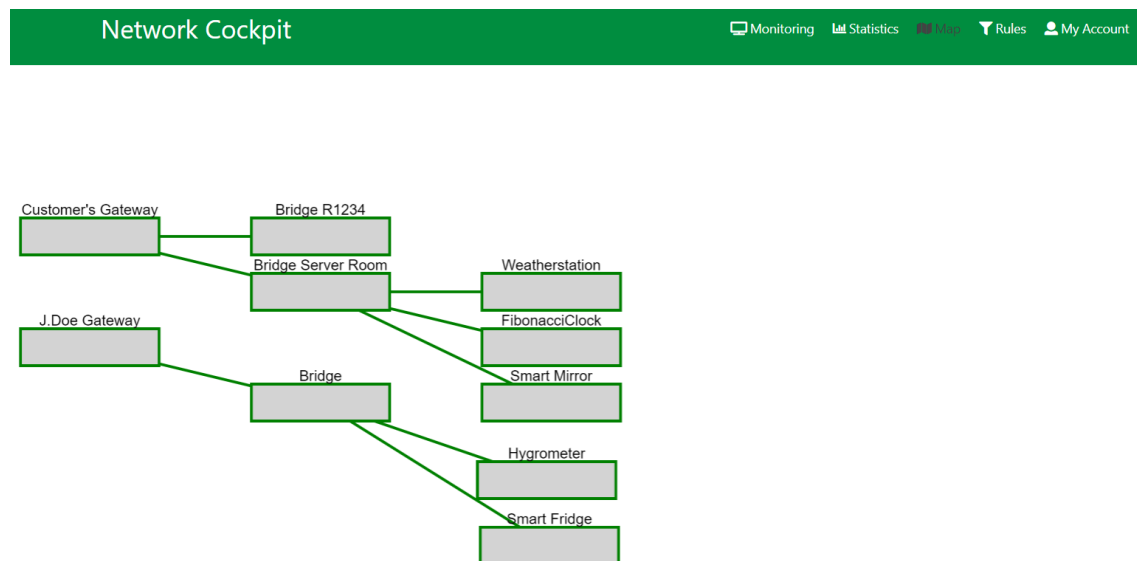


Figure 17: Map view of the Secure IoT Gateway

Every part of the Map can be clicked to get further information: an edge symbols here the connection between two components, see Figure 18. In case of a hardware component like the Network Gateway, the detailed view of respective component will be opened. The Map view is realized using the cytoscape plugin for Vue.js. This needs to be customized to reach more possibilities in styling and functionality.

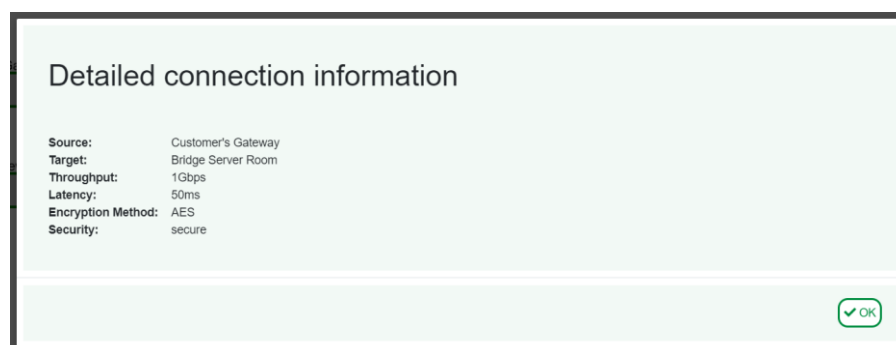


Figure 18: Details about an established connection

4) With **Rules** you can regulate the communication between the Network Gateway and the connected IoT Bridges. For example in a foster home with Smart Home living rooms it should not be allowed that the Smart Home environment of one resident can communicate with the environment of another one. This settings can be done by the Admin for it's own IoT environment. In addition to this, the Super Admin could undertake this for the in the systems house hosted IoT environments of the customers.

To change the own account details like username, password or application language, you would choose **My Account**.

In the **Admin Area**, Admins and Super Admins can create new accounts, IoT devices, IoT Bridges and Network Gateways in their environment responsibilities.

The Network Cockpit will be connected to the Gateway Cluster or the Network Gateway using an customized OPNSense plugin. With this plugin, the web application will be able to read the settings of customers IoT environment like the connected Network Gateway (when hosted in a cloud by a systems house) with its connected IoT Bridges and their connected IoT devices. For the specific component, the IP address, the status and the hardware workload could be read. Also connection details like encryption method, latency and throughput could be read from the OPNSense instance. The plugin will also allow to write several settings like the encryption method, DHCP ranges of an IoT Bridge, incoming ports, preshared keys.

One main goal of the Network Cockpit will be a provisioning service in order to ease the configuration and installation. New local IoT Bridges and Network Gateways can auto configure themselves with the help of a preregistering service. The configuration will be customized in advance and the device authenticates itself at the provisioning service with its serial ID and gets the corresponding configuration data.

7.3 Baseline Benchmark

To test the VPN connections among the Gateway Cluster, Network Gateway and IoT Bridges, a testbed containing adequate hardware and operating systems was prepared. The VPN connection was successfully configured using pre-shared keys. The possibility of using certificates to establish a secure communication among several IoT devices via the Network Gateway will be elaborated in further steps.

In order to rate the choice of technology to implement a secure and effective VPN, a benchmark comparison between WireGuard and OpenVPN was made.

WireGuard aims to provide a secure VPN being pretty simple to configure and highly effective using Curve25519 for key exchange, ChaCha20 and Poly1305 for data authentication and BLAKE2s for hashing - this combination is in NaCL (part of the Noise protocol framework) an alternative to AES which is used by OpenVPN.

In this benchmark test, WireGuard and OpenVPN were both tested using server reference (2x Intel Xeon Gold 5120 @ 2.2GHz (28C [56T] and 2x Intel Xeon Gold 5120 @ 2.2GHz (28C [56T])) and IoT like hardware (Intel Atom C2558 @ 2.4 GHz (4C [1T]) and Intel Xeon D-1518 @ 2.2GHz (4C [2T])).

The installation of WireGuard is in fact very simple with the "Quick Start Guide" while the installation of OpenVPN is more complex. Out of the box, WireGuard is faster than OpenVPN. But when the 2p2 modus, AES-256-CBC as cipher and UDP instead of TCP/IP is choosen and also the MTU of the TUN device is set to 64k, fast-io is added and the internal fragment is deactivated, OpenVPN is similar to WireGuard when using 1GbE (both have a bandwidth 0,9 Gbits/s or so with four parallel tunnel). But after upgrading the network to 10GbE, OpenVPN has a bandwidth of 6,8 Gbits/s using four parallel tunnels while WireGuard just reaches 2,1 Gbits/s. For detailed benchmarking results, see Figure 19.

Some CPUs support hardware-accelerated AES encryption being typically faster than encryption performed by software implementation on the same processors. In Order to compare the encryption with equal conditions, the hardware acceleration for AES encryption was not used in this testbed.

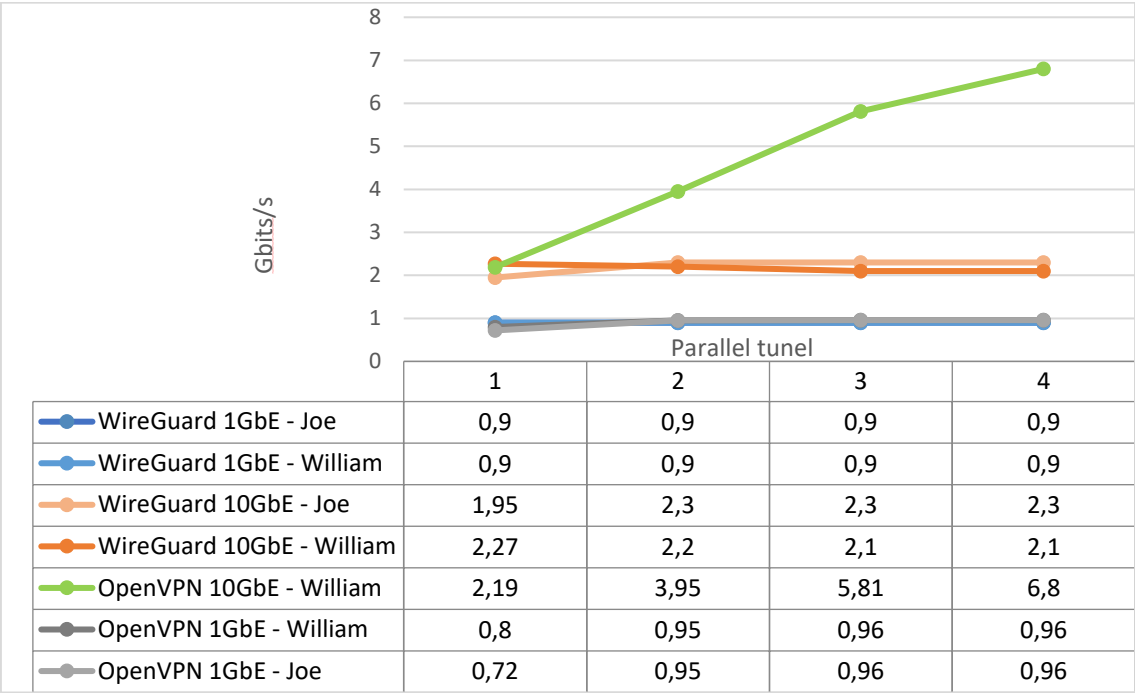


Figure 19: Benchmarking results of WireGuard vs OpenVPN for several configurations

8 Conclusion

This document gives an overview of the actual state of development and optimisation of the five use cases within LEGaTO.

Figure 20 gives an overview of the optimisation objectives for each use case, as already presented in D2.1. But as it was noted in the individual application sections, not all programming models are equally relevant to all applications and we described which programming models and optimization strategy appears to be most promising.

Programming Model	Smart Home	Smart City	Machine Learning	Infection Research
OmpSs, XiTao	1,4,7	1,2,4	1,2,3,5,6,7	1,3
MaxJ	1,4,7	1,4		1,3
DFiant	1,4,7		1,4	

Figure 20: Optimisation objectives for each use case

All developments have been progressed very well as not only baseline benchmarks have been presented, but also impressive improvements in terms of different metrics:

- Total energy consumption [Metric 1]: Smart Home, Smart City
- Energy efficiency [Metric 1]: Smart City, Machine Learning
- Computation time: Infection Research, Machine Learning
- Throughput: Secure IoT Gateway

Besides the described improvements, this deliverable also gives an outlook of actions to be taken within the second half of the LEGaTO project: Most use cases need to work on, or finalise, the usage of the LEGaTO programming models, and maximise the usage of those. This includes to verify the correctness of the optimised code, benchmarking and a final TCO analysis. For the Smart Home use case, a porting towards the edge server is planned. For the Machine Learning use case, examples of using the EmbeDL optimiser in an automotive area will be developed. The Secure IoT Gateway needs to be further developed and bundled for rollout, first within the LEGaTO project, but also later towards early adopters.

A follow-up report on these topics will be given at the end of the project in D5.3 and D5.4.

9 References

- [1] H2020 LEGaTO Project, "Deliverable 2.1: Architecture Definition and Evaluation Plan for Legato's Hardware, Toolbox and Applications," 2018.
- [2] J. Redmon, "YOLO: Real-Time Object Detection," [Online]. Available: <https://pjreddie.com/darknet/yolo/>.
- [3] NVIDIA, "NVidia Tensor Cores product page," [Online]. Available: <https://www.nvidia.com/en-us/data-center/tensorcore/>. [Accessed 31 07 2019].
- [4] NVIDIA, "NVidia TX2 product page," [Online]. Available: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/>. [Accessed 31 07 2019].
- [5] NVIDIA, "NVidia Xavier product page," [Online]. Available: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/>. [Accessed 31 07 2019].
- [6] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, 2006.
- [7] European Environment Agency, "Air quality in Europe, 2018 Report, EEA Report No. 12/2018," European Union, 2018.
- [8] National Center for Atmospheric Research (NCAR), "The weather research and forecasting model," [Online]. Available: <https://www.mmm.ucar.edu/weather-research-and-forecasting-model>. [Accessed 17 7 2019].
- [9] CASE Department – BSC, "Alya – High Performance Computational Mechanic Simulator," [Online]. Available: <https://www.bsc.es/research-development/research-areas/engineering-simulations/alya-high-performance-computational>. [Accessed 19 7 2019].
- [10] AXIOM Project, "The AXIOM Board has arrived!," [Online]. Available: <http://www.axiom-project.eu/2017/02/the-axiom-board-has-arrived>. [Accessed 2019 7 17].
- [11] Cavium, "Mont-Blanc project selects Cavium's ThunderX2™ processor for its new ARM-based HPC platform," [Online]. Available: <https://www.cavium.com/News-Release-Mont-Blanc-project-selects-Caviums-ThunderX2-processor-for-its-new-ARM-based-HPC-platform.html>. [Accessed 19 7 2019].
- [12] Alpha Data, "ADM-PCIE-7V3 – High Performance Computing," [Online]. Available: <https://www.alpha-data.com/dcp/products.php?product=adm-pcie-7v3>. [Accessed 19 7 2019].
- [13] R. Pepperkok and J. Ellenberg, "High-throughput fluorescence microscopy for systems biology," in *Nature Reviews Molecular Cell Biology* 7, 690-696, 2006.

- [14] A.-J. WAN, K. Wang, H.-C. ZHANG, H.-L. LI and D.-N. WANG, "Modercarbohydrate microarray biochip technologies," in *Chinese Journal of Analytical Chemistry* 40(11): 1780–1788, 2012.
- [15] W. W. Soon, M. Hariharan and M. P. Snyder, "High-throughput sequencing for biology and medicine," in *Molecular systems biology* 9, page 640, 2013.
- [16] V. B. Patel and V. R. Preedy, "General Methods in Biomarker Research and their Applications," 2015.
- [17] C. Endrullat, J. Glökler, P. Franke and M. Frohme, "Standardization and quality management in next-generation sequencing," *Applied and Translational Genomics*, 2016.
- [18] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR 2015*, San Diego, CA, USA, 2015.
- [19] C. Szegedy, W. Liu, J. Yangqing, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [20] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *CoRR*, 2015.