CEI UPM | Centro de Electrónica Industrial

POLITÉCNICA

Architectures and Algorithms for Evolvable Self-* Systems. The A-IQ Ready Proposal

SEPTEMBER 18-22 2023 – ALGHERO, SARDINIA, ITALY

CYBER PHYSICAL SYSTEMS SUMMER SCHOOL 2023

Andrés Otero, Alfonso Rodriguez

joseandres.otero@upm.es, alfonso.rodriguezm@upm.es

UNIVERSIDAD POLITÉCNICA DE MADRID

cei@upm.es

POLITÉCNICA

The Speakers

Andrés Otero is a Telecommunications Engineer from the University of Vigo, where he graduated with Honors in 2007. He completed a Master's and Doctorate in Industrial Electronics from the **Universidad Politécnica de Madrid (UPM)** in 2009 and 2014, respectively. Currently, he serves as an Associate Professor at UPM. His areas of interest are embedded and/or reconfigurable systems, evolutionary computation, and machine learning at the *edge*. Contact: joseandres.otero@upm.es

Alfonso Rodríguez has a MSc on Industrial Electronics (2014) and a PhD on Electrical and Electronics Engineering (2020), both by Universidad Politécnica de Madrid (UPM). Currently, he works as an Associate Professor at **UPM** and carries out its research activity as a member of the Center of Industrial Electronics (CEI-UPM). His main research interests are high-performance embedded systems, reconfigurable computing, open computer architectures (e.g., RISC-V), and edge AI/ML.
Contact: alfonso.rodriguezm@upm.es

Many Other Contributors

Eduardo de la Torre

Javier Mora

Rubén Salvador

MSc students: Alberto García, Guillermo San Llorente, Javier Laserna…

External Collaborations with Prof. Lukáš Sekanina (Brno Univ. Technology) and Prof. José Luis Nuñez-Yañez (Linköping University)

Centro de Electrónica Industrial (CEI)

CEI is a Research Center that belongs to the **Universidad Politécnica de Madrid.** We are located at the **E.T.S.I. Industriales**, in the city center of Madrid.

INDUSTRIALES
ETSII | UPM

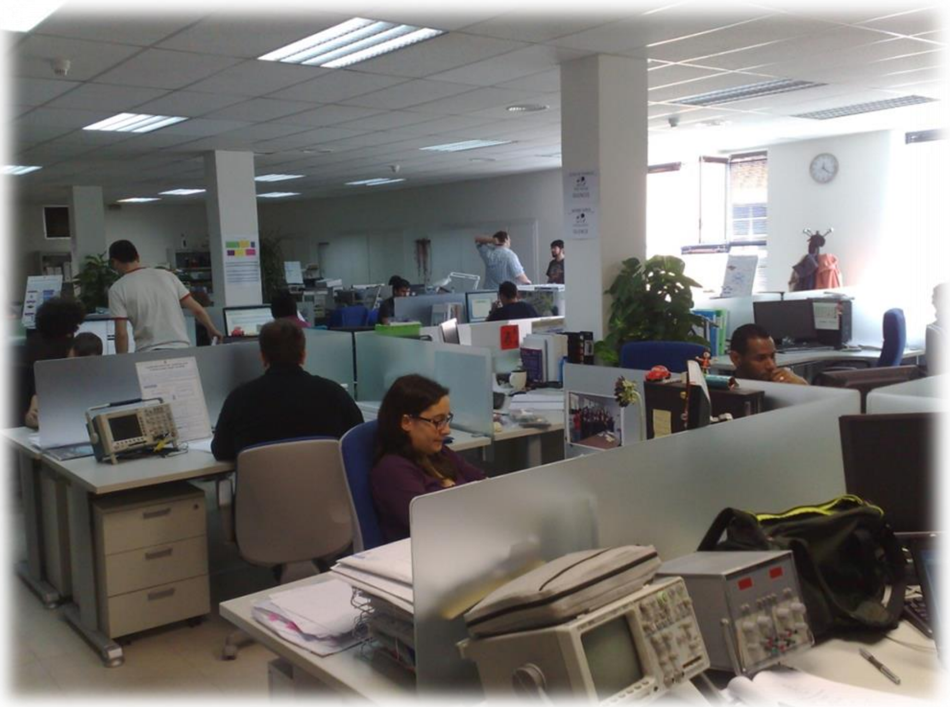
CEIUPM | Centro de Electrónica Industrial

CEIUPM

POLITÉCNICA

Centro de Electrónica Industrial (CEI)

CEI UPM



Digital Embedded Systems

- Reconfigurable Embedded Systems
- Sensor Networks & Internet of Things
- Embedded Intelligence

Power Electronics

- Power Converters
- Power Supply Architectures
- Modeling & Simulation

Some Numbers

- 48 Full-time researchers
  - 18 Doctors (Faculty, Contracted researchers)
  - 30 Full-time Students (18 Doctorate, 12 Master)
- 19 Part time, sponsored students
- 3 Administration & Technicians

Digital Embedded Systems @CEI – Who we are?

- 7 Professors
  - Teresa Riesgo*, Eduardo de la Torre, Yago Torroja , Jorge Portilla, Andrés Otero, Gabriel Mujica, and Alfonso Rodriguez.
- 8 Full-time PhD Students:
  - Jaime Señor, Juan Encinas, Daniel Vázquez, Juan Gallego, Junjiao Sun, Rogelio Hernández, Fernando Pérez, Luis Waucquez, Alejandro Redondo.

*: On temporary leave

INDUSTRIALES
ETSII | UPM

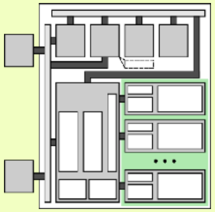
CEI UPM

Digital Embedded Systems – Research Lines

Internet of Things

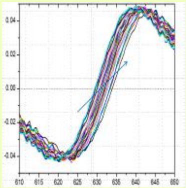
Networked embedded systems to face challenges related to the era of smart and sustainable cities, comprising the integration of heterogeneous hardware and software technologies.

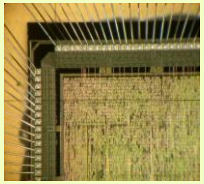
Reconfigurable Hardware



Developing embedded parallel computing platforms based on HW acceleration and design tools, to obtain energy-efficient, scalable, and run-time adaptive solutions.

Machine Learning at the Edge





Embedded circuits for near-sensor decision making, based on HW/SW embedded signal processing and machine learning techniques

Index of the Presentation

1. Introduction to Self-* Computing Systems

2. Principles of Evolutionary Computation and Evolvable Hardware

3. Classic Evolvable Hardware: Application to Image Processing

4. Neuroevolution: Combining Evolutionary Computation and ANNs

5. Deep Neuroevolution: Evolutionary Computation + Deep Learning

6. From Neuroevolution to Neuromorphic: Evolutionary Spiking Neural Networks

7. The A-IQ Ready View: hybrid computing platforms

INTRODUCTION TO SELF-* COMPUTING SYSTEMS

Index of the Presentation

1. **Introduction to Self-* Computing Systems**

2. Principles of Evolutionary Computation and Evolvable Hardware

3. Classic Evolvable Hardware: Application to Image Processing

4. Neuroevolution: Combining Evolutionary Computation and ANNs

5. Deep Neuroevolution: Evolutionary Computation + Deep Learning

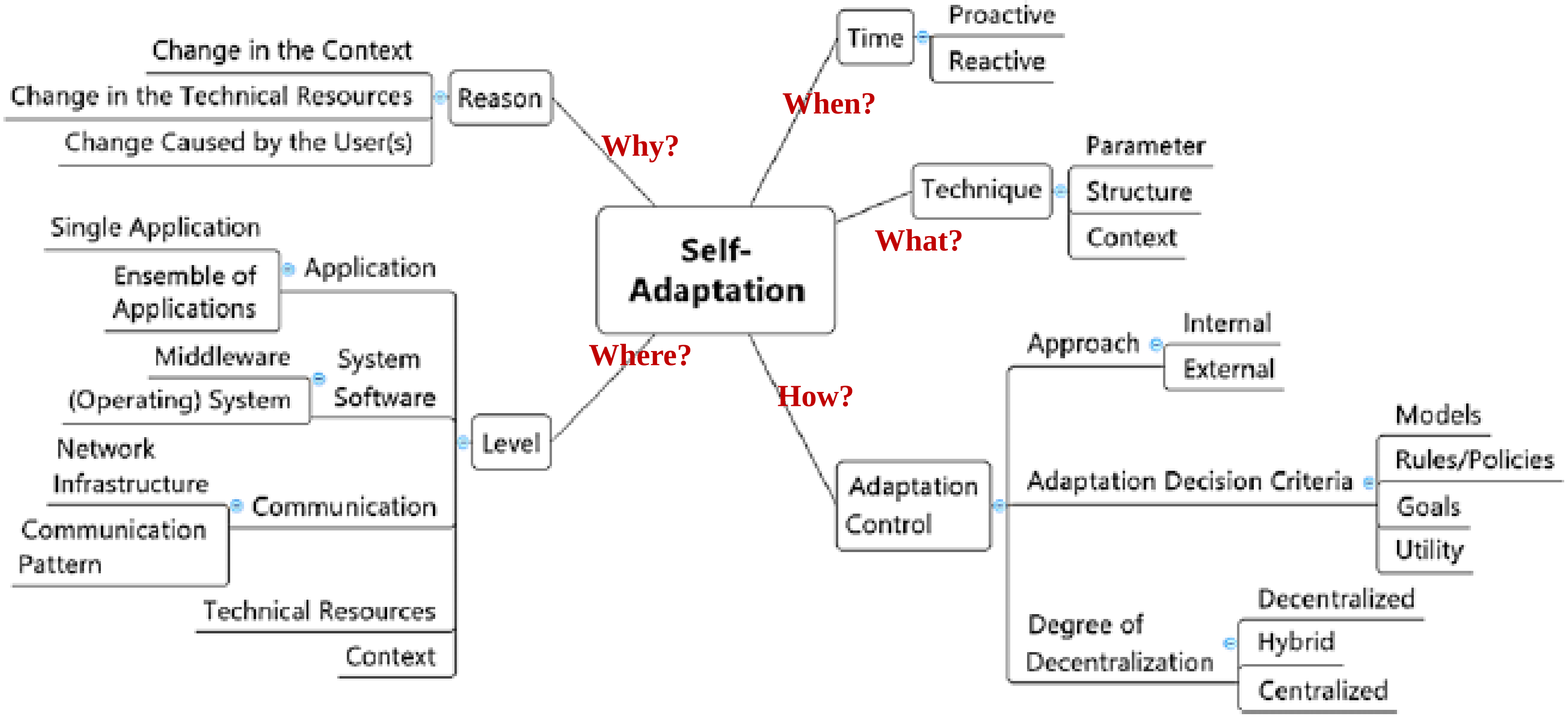
6. From Neuroevolution to Neuromorphic: Evolutionary Spiking Neural Networks

7. The A-IQ Ready View: hybrid computing platforms

Self-Adaptive Computing Systems

A Self-adaptive computing system is a system capable of automatically change its structure, functionality and/or parameters, in response to changes in its operational environment, user demands or self-sensing information. Self-adaptive technologies are needed in the **ubiquitous and pervasive computing** era since unpredictable changes in the environment makes impossible to address at design time all scenarios that can happen at run-time.

Taxonomy of self-adaptation



Triggers for Adaptation

ENVIRONMENTAL AWARENESS: Influence of the environment on the system, i.e., daylight vs. nocturnal, radiation level changes, etc.
Sensors are needed to interact with the environment and capture conditions variations.

USER/EXTERNALLY-COMMANDED: System-User interaction, i.e. user preferences, commands from SoS managers (the boss), etc.
Proper human-machine interfaces are needed to enable interaction and capture commands.

SELF-AWARENESS: The internal status of the system varies while operating and may lead to reconfiguration needs, i.e. chip temperature variation, low battery.
Status monitors are needed to capture the status of the system.

Types of Adaptation

FUNCTIONALITY-ORIENTED: To adapt functionality because the system mission changes, or the data being processed changes and adaptation is required. It may be parametric (a constant changes) or fully functional (algorithm changes)

EXTRA-FUNCTIONAL REQUIREMENTS-ORIENTED: Functionality is fixed, but system requires adaptation to accommodate to changing requirements, i.e. execution time or energy consumption.

REPAIR-ORIENTED: For safety and reliability purposes, adaptation may be used in case of faults. Adaptation may add self-healing or self-repair features. e.g.: HW task migration for permanent faults, or scrubbing (continuous fault verification) and repair.

Self-* Properties

SELF-AWARENESS: ability of a system to be aware of itself states and behaviors, i.e., to monitor its resources, state and behavior.

CONTEXT-AWARENESS: ability of a system to be aware of its operational environment.

Self-aware computer systems are capable of adapting their behavior and resources thousands of times a second to automatically **find the best way to accomplish a given goal** despite changing environmental conditions and demands. These goals can be achieved by **monitoring** the system (*self*) and its environment (*context*).

Self-* Properties

SELF-CONFIGURATION: capability of reconfiguring automatically and dynamically in response to changes.

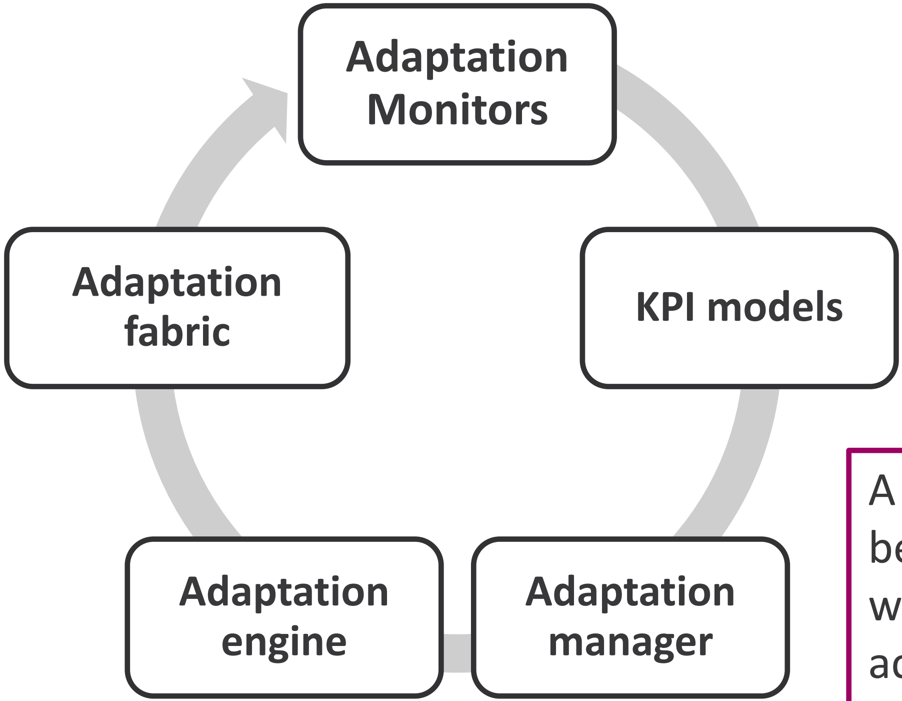
SELF-PROTECTING: capability of detecting security breaches and recovering from their effects.

SELF-OPTIMIZING: capability of managing performance and resource allocation in order to satisfy the requirements of different users. End-to-end response time, throughput, utilization, and workload are examples of important concerns related to this property.

SELF-HEALING: Is the capability of discovering, diagnosing and reacting to disruptions. It can also anticipate potential problems, and accordingly take proper actions to prevent a failure.

Adaptation Loop

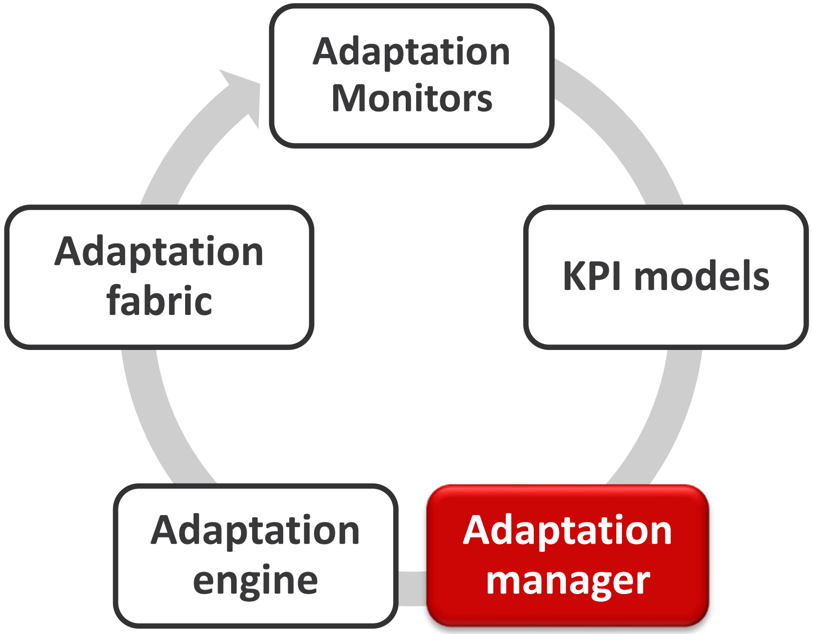
Life-cycle of a self-adaptive system should not be stopped after its development and initial set up. The cycle continues after installation to evaluate the system and respond to changes at all time. **Self-adapting systems** embodies a **close-loop mechanism,** called **the adaptation loop.**



A self-adaptive system evaluates its own behavior and changes its own performance when the evaluation indicates that it is not accomplishing what the system is intended to do, or when better functionality or performance is possible.

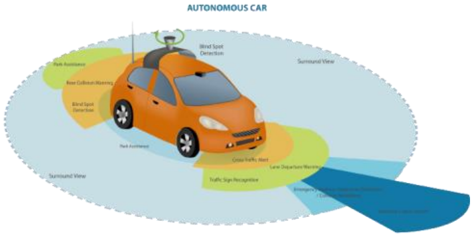
Research Question

Is it possible to build adaptation managers based on Evolutionary Algorithms?



Evolutionary Algorithms can be implemented on-chip and used as black-box optimization tools, enabling self-* properties and continuous-learning capabilities.

Targeting CPS Systems



PRINCIPLES OF EVOLUTIONARY COMPUTATION AND EVOLVABLE HARDWARE

Index of the Presentation

1. Introduction to Self-* Computing Systems

2. **Principles of Evolutionary Computation and Evolvable Hardware**

3. Classic Evolvable Hardware: Application to Image Processing

4. Neuroevolution: Combining Evolutionary Computation and ANNs

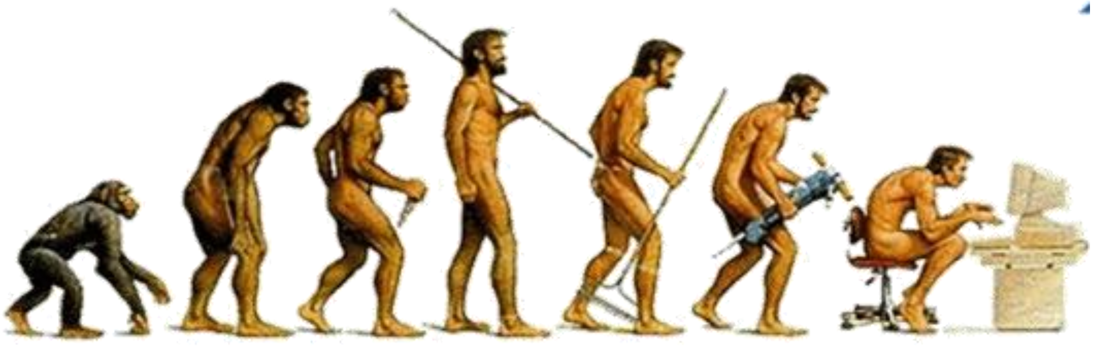
5. Deep Neuroevolution: Evolutionary Computation + Deep Learning

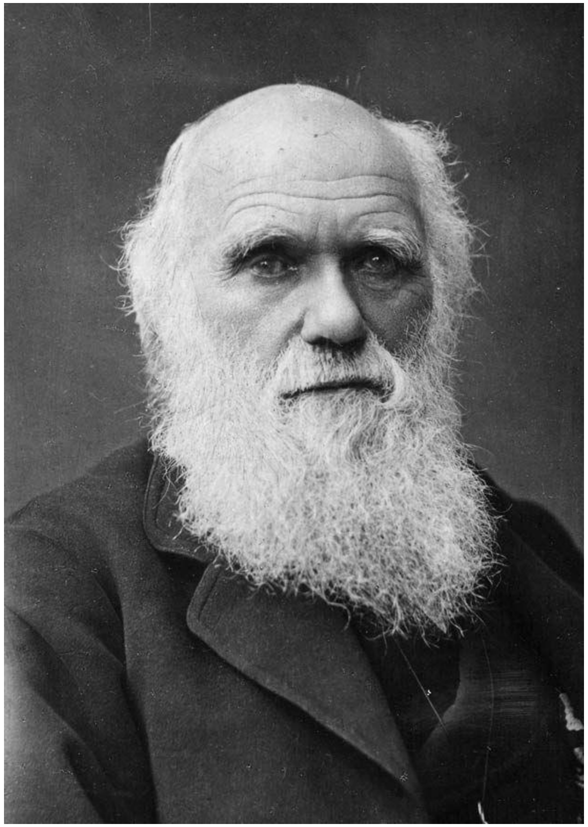
6. From Neuroevolution to Neuromorphic: Evolutionary Spiking Neural Networks

7. The A-IQ Ready View: hybrid computing platforms

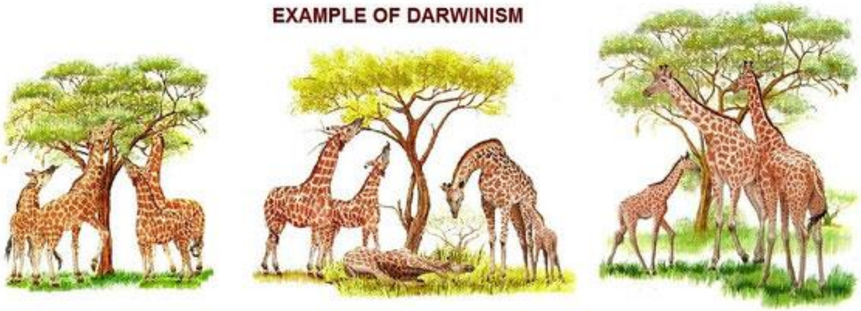
Evolutionary Design: preliminary example

Natural Evolution by Charles Darwin





From an amoeba to a computer engineer: Random changes in our chromosomes, the sexual reproduction and the sexual selection for the adaptation to the environment.



Evolutionary Systems

Evolutionary Systems are systems that rely on the Evolutionary Design Techniques (and by extension**, on the Darwinist evolutionary principles)** during execution time.

One or more populations of individuals **competing** for limited resources

A concept of **fitness** which reflects the ability of the individual to survive and reproduce.

The notion of **dynamically changing populations** due to the birth and death of individuals

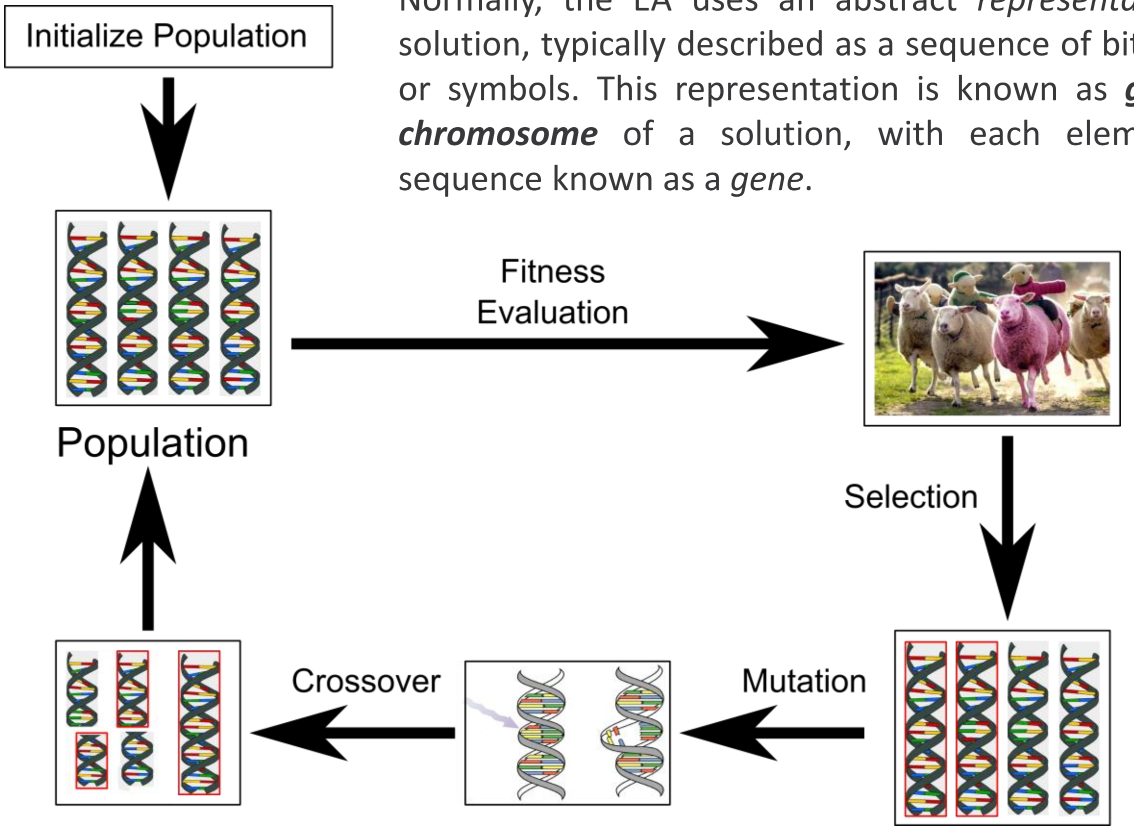
A concept of **variational inheritance**: offspring closely resemble their parents, but they are not identical.

These principles lead naturally to the view of systems that, given particular initial conditions, follow a trajectory over time through a complex evolutionary state-space. This process is guided by an **Evolutionary Algorithm.**

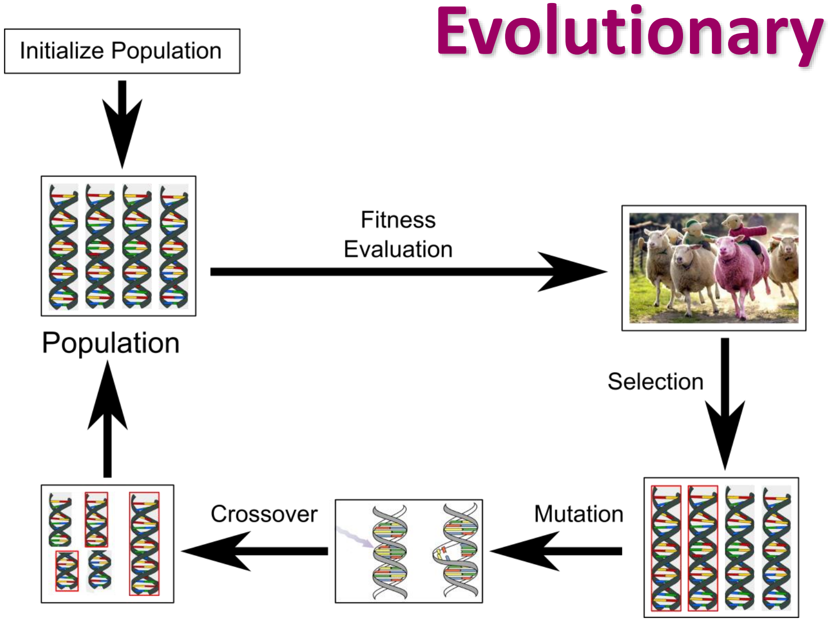
Evolutionary Algorithms (EA)

Evolutionary Algorithms (Eas) are **metaheuristic optimization algorithms** inspired by Darwin's theory of evolution. EAs follow a **progressive search** in which new solutions are explored by combining and modifying previously tested solutions, favoring those which present a more desirable behavior, and discarding those that don't.

Normally, the EA uses an abstract *representation* of the solution, typically described as a sequence of bits, numbers, or symbols. This representation is known as ***genotype*** or ***chromosome*** of a solution, with each element in the sequence known as a *gene*.

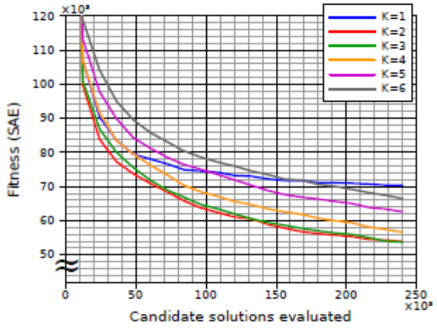


Evolutionary Algorithms



Fitness:

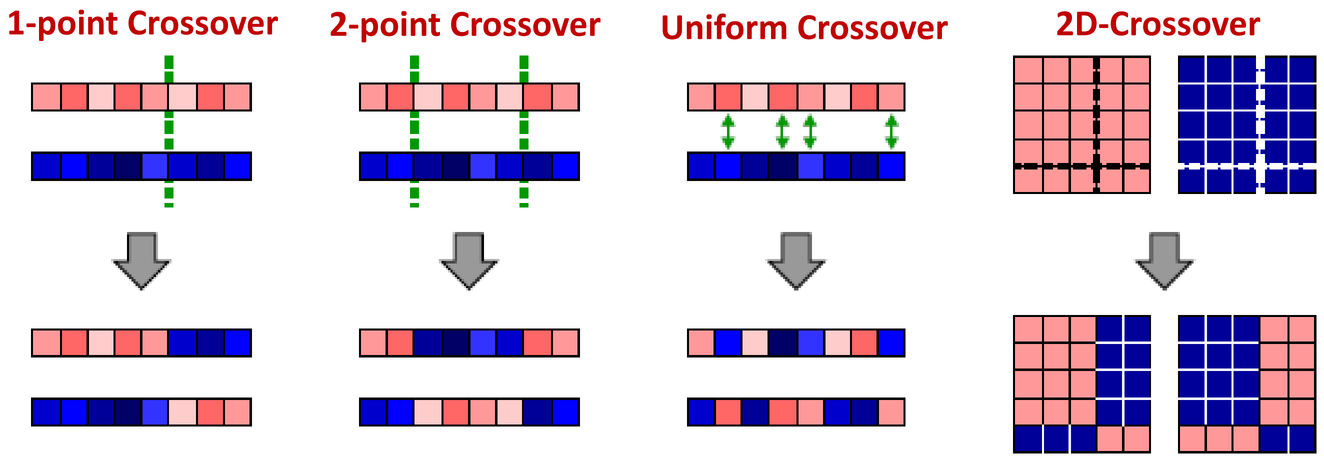
$$\sum | a_{ij} - b_{ij} |$$



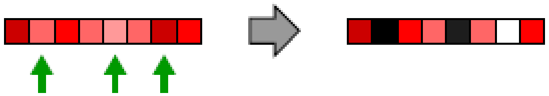
- First, an initial *population* of a certain number of candidate solutions is chosen, typically randomly.
- Then, an iterative process starts in which the following operations are applied to the population:
  - **Selection** of the best individuals in the population based on their fitness, which will be the *parents* for the next generation.
  - **Crossover** or **recombination,** where a new *child* is obtained by combining parts of the genotype of two or more parents.
  - **Mutation** of a genotype, in which some randomly selected genes of the genotype are modified, either by flipping them (in the case of binary genes), adding a random value, or replacing them by a random value.
- Finally, the *evolution is terminated* after either a certain number of iterations (or *generations*) have elapsed, a target fitness value has been reached, or another termination criterion is reached. The best individual in the resulting population will be considered as the result of the evolution.

Evolutionary Algorithms: main operators

- **Crosssover**:

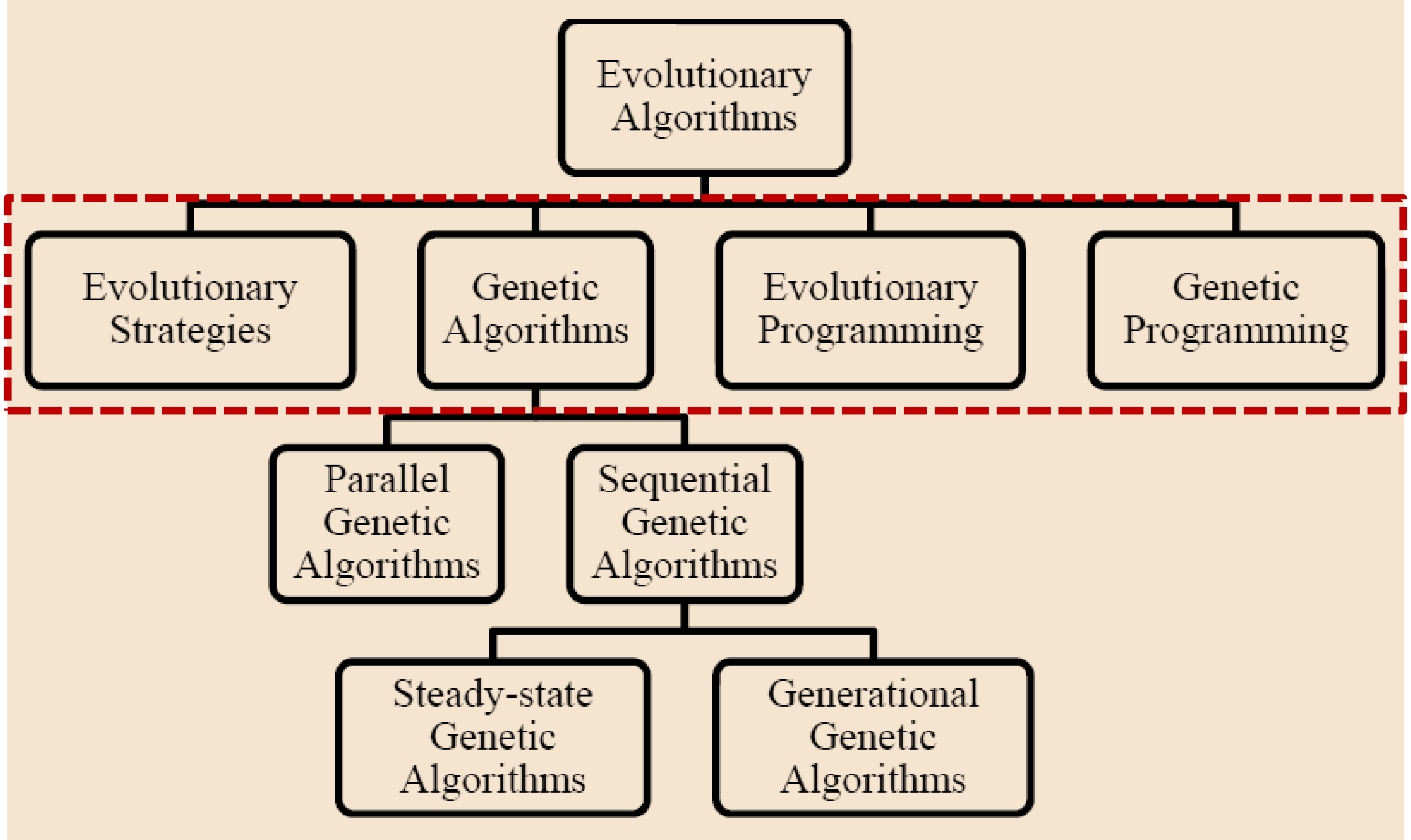


- **Mutation**:



- **Selection**: It may consist in picking the best individuals and discarding the rest (*elitist selection*), picking two individuals at random and choosing the one with best fitness (*tournament selection*), or using uniform random distribution proportional to the fitness of the individuals (*roulette-wheel selection*).

Taxonomy of Evolutionary Algorithms


Evolutionary Algorithms
Evolutionary Strategies
Genetic Algorithms
Evolutionary Programming
Genetic Programming
Parallel Genetic Algorithms
Sequential Genetic Algorithms
Steady-state Genetic Algorithms
Generational Genetic Algorithms

Although similar at the highest level, each of these varieties implements an evolutionary algorithm in a different manner. The differences include almost all aspects of evolutionary algorithms, including the choices of representation for the individual structures, types of selection mechanism used, forms of genetic operators, and measures of performance.

Evolutionary Strategies (ES)

- In Evolutionary Strategies (ES), **the genotype** consists of a sequence of real-valued genes, each of them describing a different parameter of the system to optimize.
- **Mutation** consists in adding a normally distributed random value to all genes in the genotype (this random value might be identically distributed for all genes or have an ellipsoidal shape on R^n).
  - A distinctive feature of this EA is the fact that the standard deviation of this normal distribution may be included as part of the genotype, so that this EA not only evolves the individuals but also the degree in which they may change during the evolution.
- Originally, this EA did not use crossover, relying exclusively on mutation and selection; this is denoted as *(1+λ)-ES*, indicating that each generation has 1 parent and λ children.
- Implementations also exist that perform **crossover** by averaging two individuals; these are known as *(μ+ λ)-ES* or *(μ,λ)-ES* (with *μ* being the number of parents) depending on whether the parents participate in the selection or are systematically discarded.
- **Selection** is deterministic and is implemented in one of two ways:
  - The first allows the N best children to survive, and replaces the parents with these children.
  - The second allows the N best children and parents to survive.

Evolutionary Programming (EP)

- In **Evolutionary Programming (EP)** was developed by Fogel in 1966, and traditionally has used representations that are tailored to the problem domain.
    - For example, in real-valued optimization problems, the individuals within the population are real-valued vectors.
- The **mutation** mechanism consists in adding a normally distributed random value to each gene, but, unlike in ES, the standard deviation of this value is made proportional to the fitness of a solution, so that solutions with a lower (better) fitness will be subject to smaller mutations than those with a higher (worse) one.
- **Crossover** is not used at all.
- This EA variant uses a (μ + μ) strategy in which a population of 2μ individuals is reduced to μ parents by performing a **tournament selection**: each new parent is obtained by randomly choosing two or more individuals of the population and picking the best of them

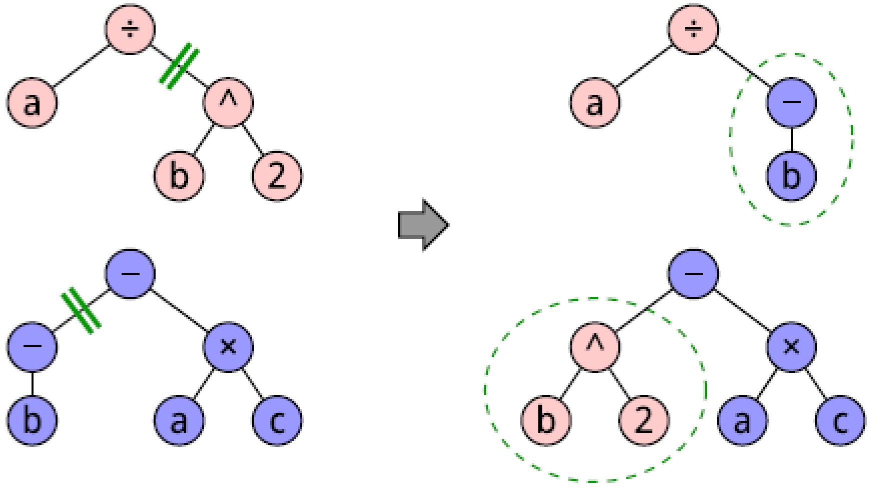
Genetic Algorithms (GA)

- A more flexible variant of EA is the **genetic algorithm** (GA), which has become the most widely used form of EA due to its versatility and relative independence with the problem to optimize.
- Unlike ES and EP, the genotype in GA is typically represented as a *sequence of bits*, with each feature of the system being encoded in a specific group of bits. Representations in the form of sequence of integers is also possible.
  - This encoding provides more flexibility than ES, since it can not only represent real-valued system parameters, but also the choice of a specific option within a range (e.g., indexed functions or modules, multiplexer settings, presence or absence of specific features…)
- Mutation in GA usually consists in randomly selecting a certain number of bits and flipping their value. This more closely approximates biological mutation, which affects a single base pair rather than the whole gene describing a single feature (with each gene typically spanning several thousands of base pairs).
- In addition to mutation, most implementations of GA heavily rely on crossover, unlike ES, which rarely uses it, or EP, which does not use it at all.

The flexibility of this type of EA has made it a very popular choice for most evolvable systems.

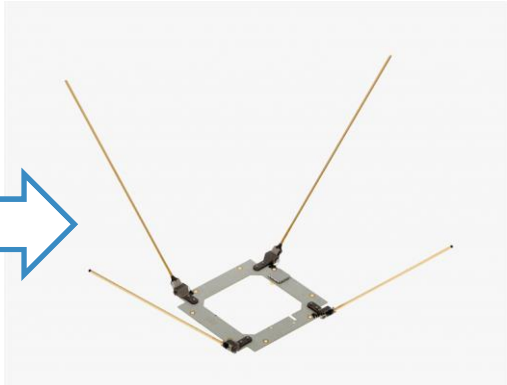
Genetic Programming (GP)

- An EA variant that is often used in the field of evolutionary computation is **genetic programming** (GP).
- GP is specifically oriented to the design of software functions. As such, the genotype is represented as a *tree structure*, with each node in the tree representing a basic function (such as addition, subtraction, multiplication…). This type of structure allows the representation of arbitrary expressions as used in mathematical or computing contexts;
- The crossover mechanism used by GP is very particular to this EA variant: it consists in swapping subtrees between two trees. This results in genotypes potentially growing to arbitrarily large sizes, which would make some solutions impossible to implement in systems with limited resources, so care must be taken to limit the length of the tree (for example, including it as factor for the fitness computation).

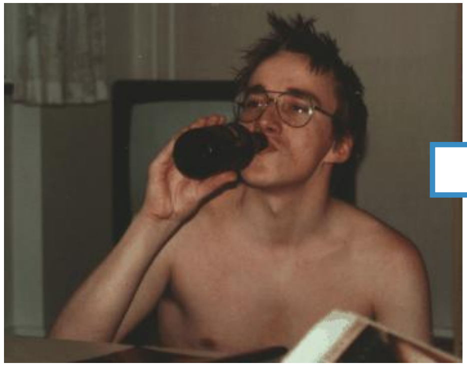


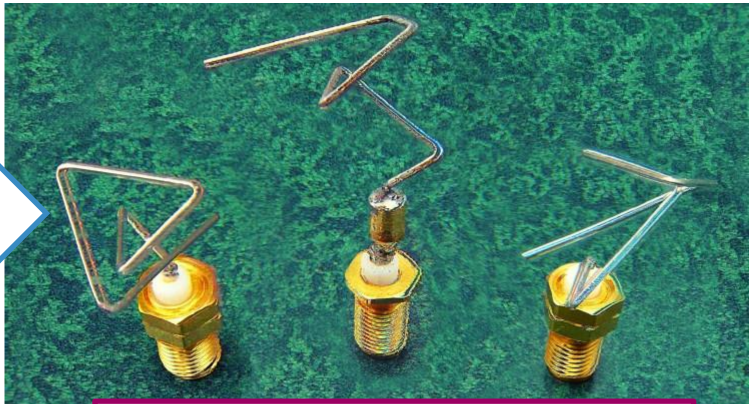
Evolutionary Design: first example

An interesting example by NASA



Reliable and predictable results.



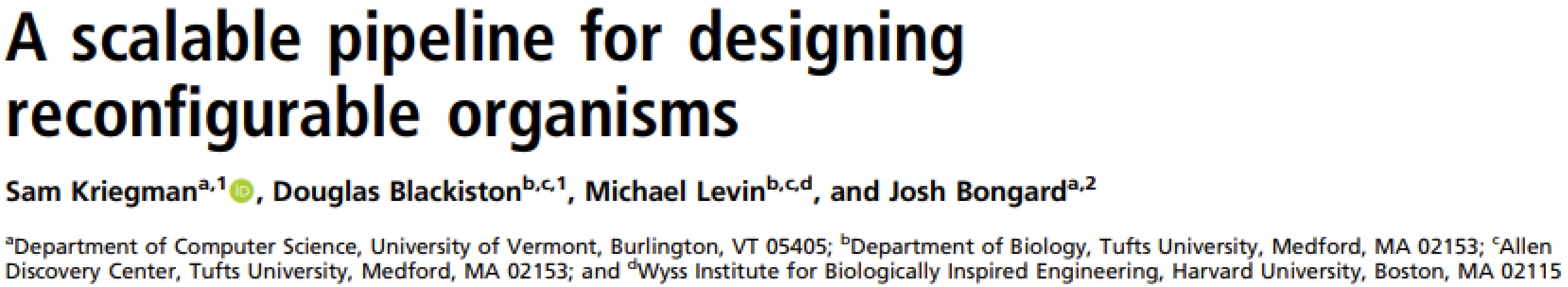


Crazy unexpected *good* results

Rely on evolutionary algorithms to do our work as engineers!

https://en.wikipedia.org/wiki/Evolved_antenna

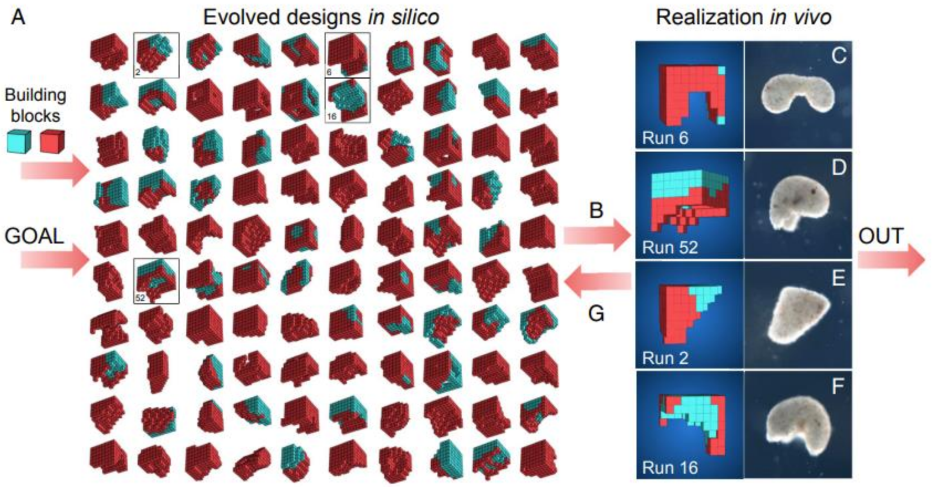
Evolutionary Design: advanced example

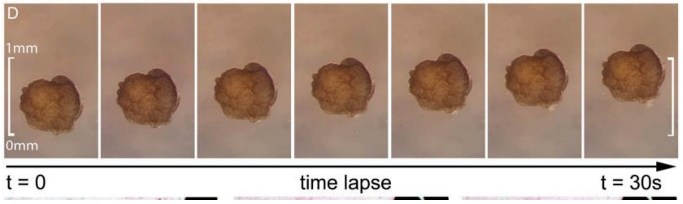


A scalable pipeline for designing reconfigurable organisms

Sam Kriegman[a,1], Douglas Blackiston[b,c,1], Michael Levin[b,c,d], and Josh Bongard[a,2]

[a]Department of Computer Science, University of Vermont, Burlington, VT 05405; [b]Department of Biology, Tufts University, Medford, MA 02153; [c]Allen Discovery Center, Tufts University, Medford, MA 02153; and [d]Wyss Institute for Biologically Inspired Engineering, Harvard University, Boston, MA 02115



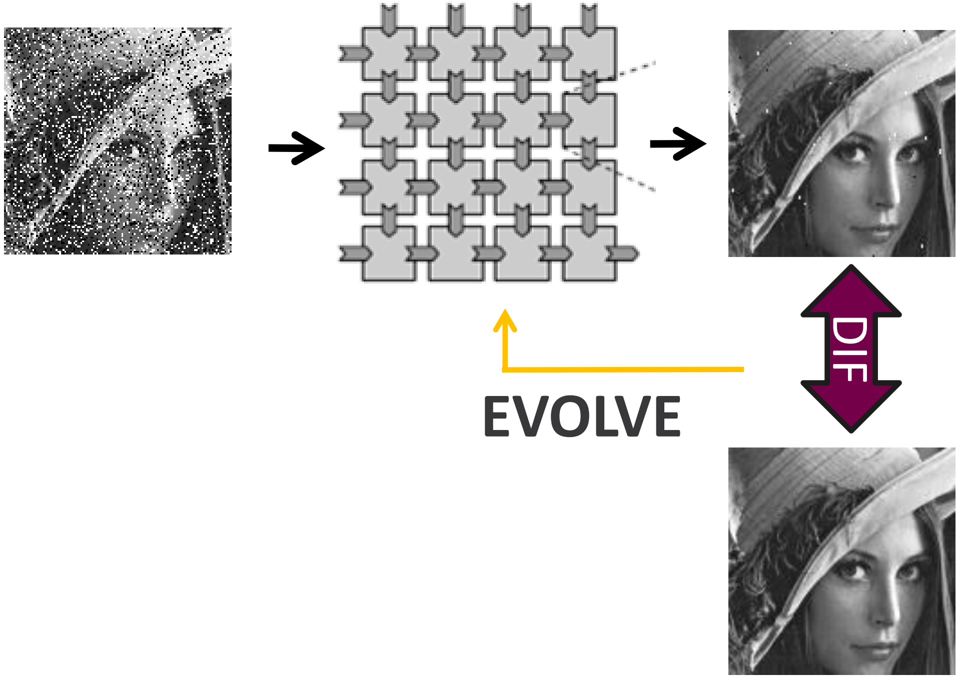


KRIEGMAN, Sam, et al. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 2020, vol. 117, no 4, p. 1853-1859.

Evolvable Hardware Systems

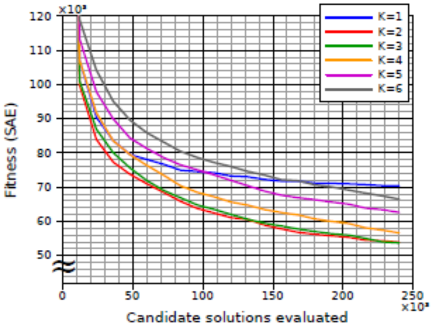
Evolvable hardware provides an unconventional methodology for the design of digital circuits: rather than designing a circuit with knowledge of the problem that needs to be solved, a circuit is "*trained*" by providing an example problem and the desired solution. For this training, an evolutionary algorithm is employed.

The EA would modify random parts of a circuit until its response to a certain input is close enough to the desired output. This permits the automatic design of circuits for problems whose general solution is unknown, but for which it is known what the solution for a specific "training problem" should be.



Fitness:

$$\sum | a_{ij} - b_{ij} |$$

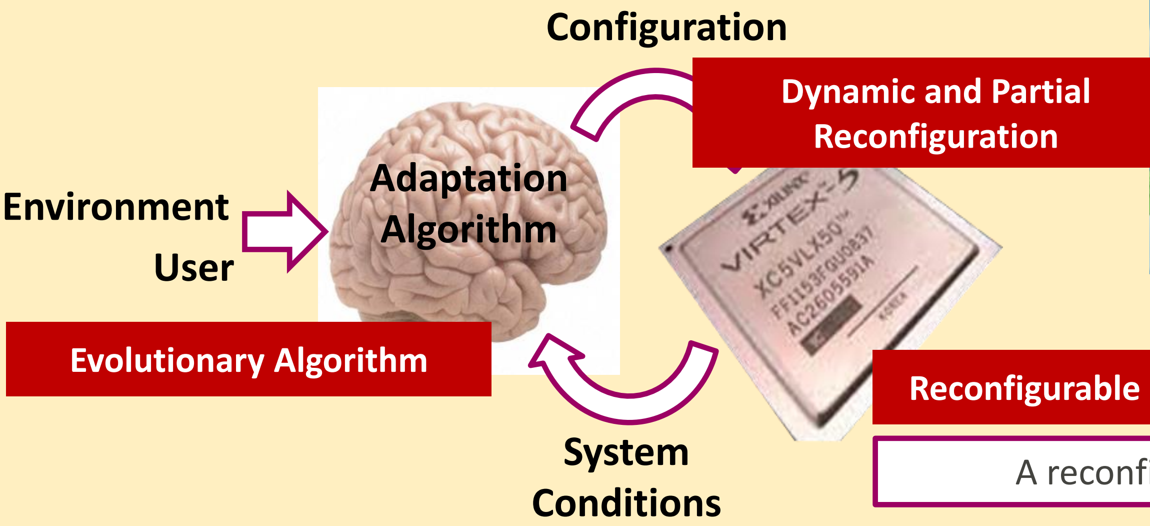


Evolutionary Design: offline and online

Evolutionary Design techniques can be applied offline and online

OFFLINE: As a design tool

ONLINE: Integrating the evolutionary algorithm in the final system (i.e., circuit) enables continues run-time **self-adaptation**.

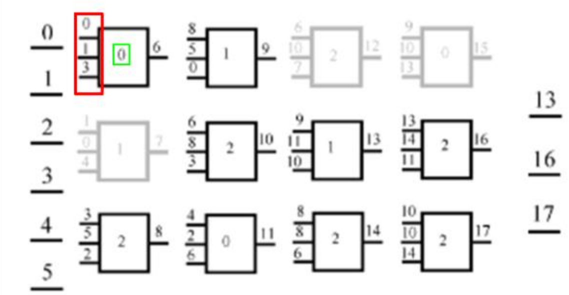


A reconfigurable fabric is required

State of the Art on Evolvable HW

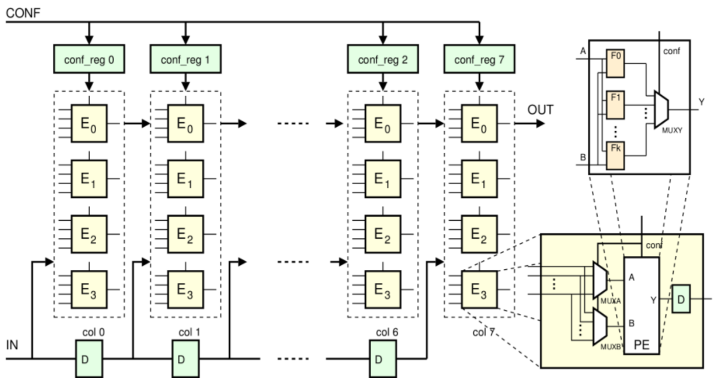
Cartesian Genetic Programming

- Is a form of Genetic Programming for the design of combinatorial hardware



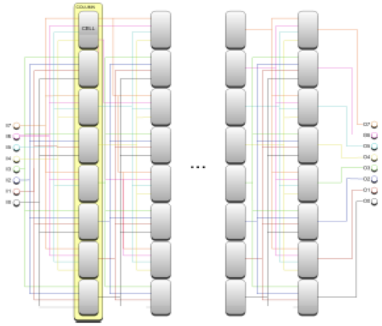
Virtual Reconfigurable Circuits

- VRC Architecture (Brno University)
  - Evolution friendly
  - Processing capability ?



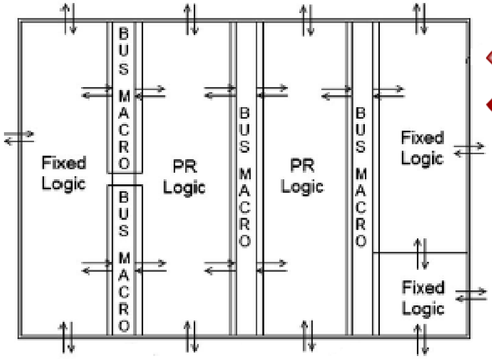
Dynamic Partial Reconfiguration

- HERA (Politecnico di Milano)
  - Based on Cartesian Genetic Programming
  - Parametric Reconfiguration of the LUT Equations



✗ Reduced Flexibility

- Evolvable Neural Network (EPFL)
  - Based on Xilinx Modular Design Flow
  - Adaptable network topology



✗ High Reconfiguration Overhead

CLASSIC EVOLVABLE HARDWARE: APPLICATION TO IMAGE PROCESSING

Index of the Presentation

1. Introduction to Self-* Computing Systems

2. Principles of Evolutionary Computation and Evolvable Hardware

3. Classic Evolvable Hardware: Application to Image Processing

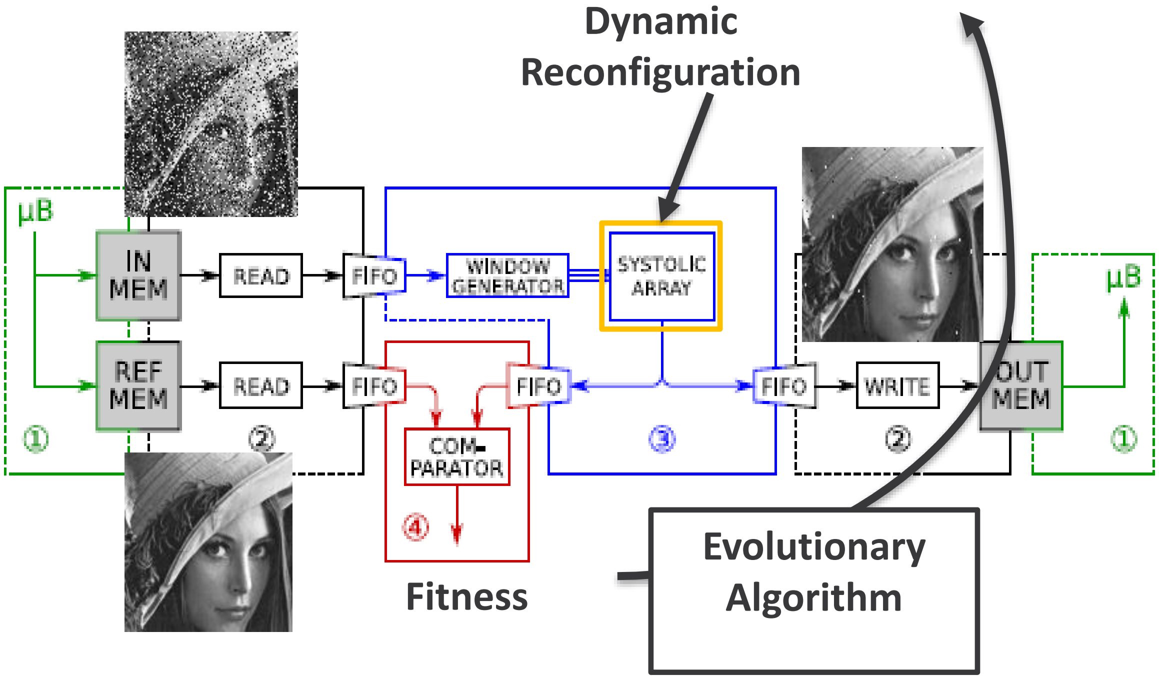
4. Neuroevolution: Combining Evolutionary Computation and ANNs

5. Deep Neuroevolution: Evolutionary Computation + Deep Learning

6. From Neuroevolution to Neuromorphic: Evolutionary Spiking Neural Networks

7. The A-IQ Ready View: hybrid computing platforms

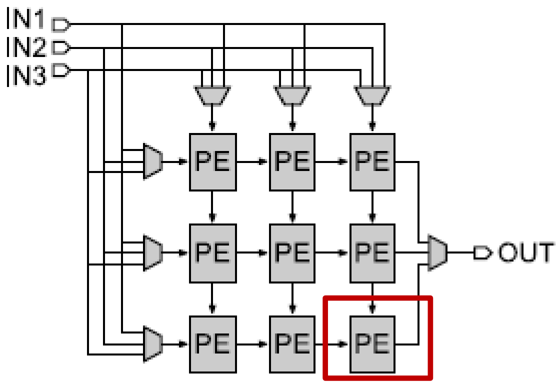
Evolvable Hardware for Self-* Image Processing



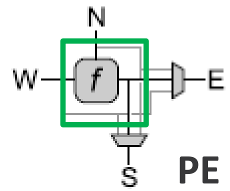
Dynamic Reconfiguration

Evolutionary Algorithm

Fitness

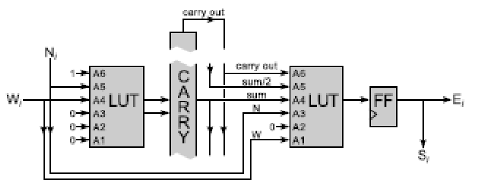


Reconfigurable Systolic Array



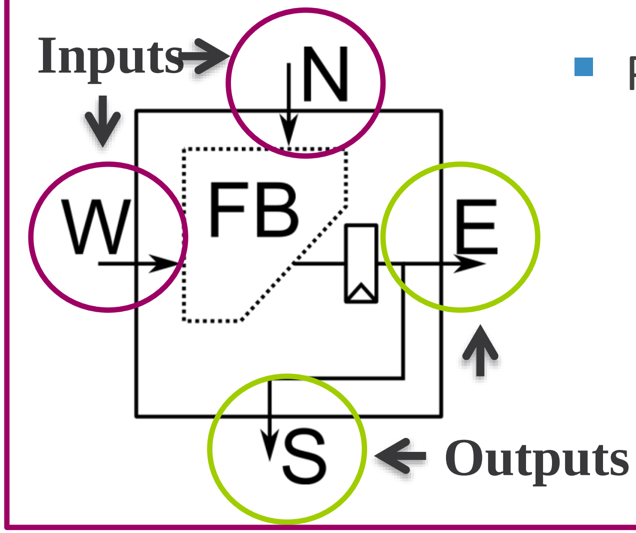
PE

It tries to solve a problem whose model is unknown, and tries to find a solution for the model, given the REFERENCE and a TRAINING input image.



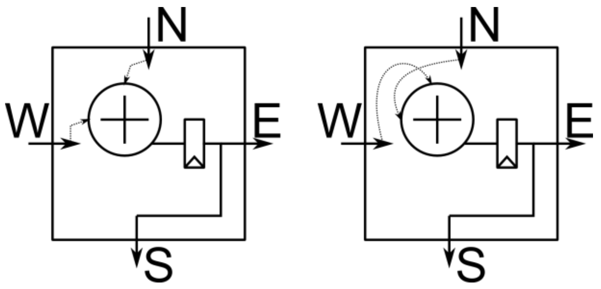
Fragment of a PE

EH Architecture - Processing Elements



- PEs
  - Functional Block (FB)
  - Flip-Flop
  - Routing Logic
  - PE-FB mapping
    - Allows certain data front adaptation

FB	Function
0	x + y
1	x << 1
2	$x +_s y$
3	(x + y) >> 1
4	255
5	x >> 1
6	x
7	max(x,y)
8	min(x,y)
9	$x -_s y$



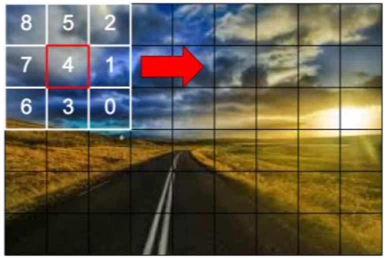
- PE – DPR Library
  - 1 element per routing combination
  - Different PE in:
    - Functionality (FB)
    - Interconnection
  - 16 Different Processing Elements

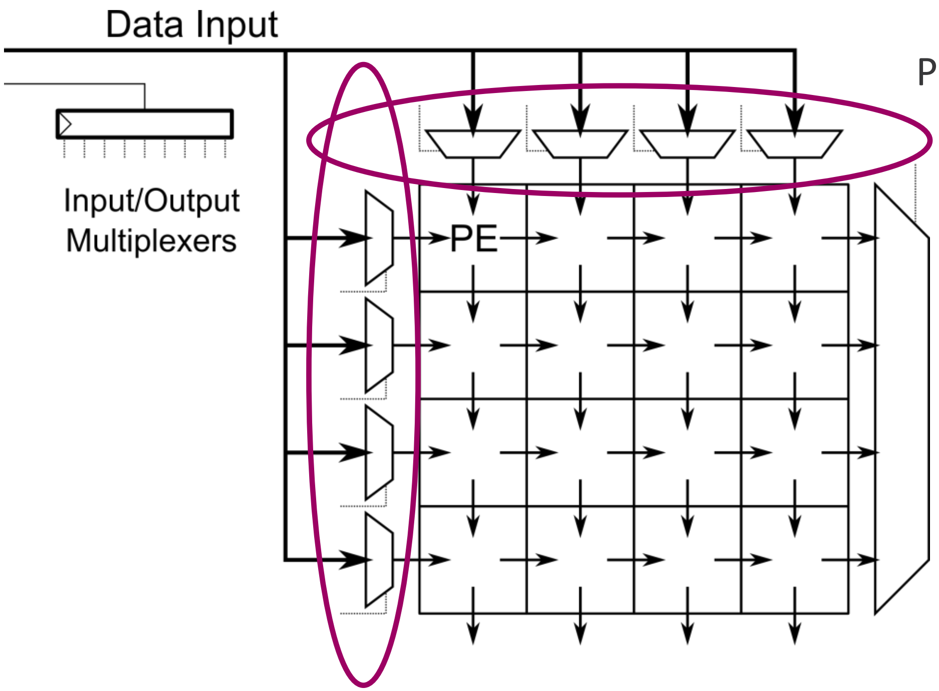
PE	Function
0	N + W
1	N << 1
2	W << 1
3	$N +^S W$
4	$N +^S N$
5	$W +^S W$
6	(N + W) >> 1
7	255
8	N >> 1
9	W >> 1
10	N
11	W
12	max(N,W)
13	min(N,W)
14	$N -_S W$
15	$W -_S N$

Versions with parametric reconfiguration (fine-grain) are also available

Application – Image Filtering

- Input to the Evolvable Array
  - Raster – scan order @ clock rate
  - 3×3 window (9 inputs)
  - Evolution searches the routing “input pixel → PE”

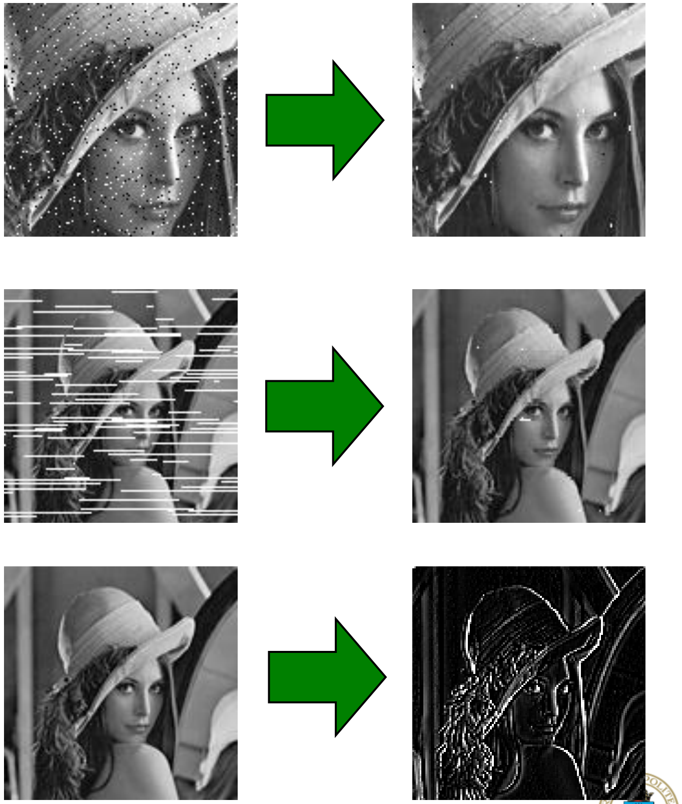




Salt and Pepper noise

Burst Noise

Edge Detection



Evolutionary Framework

Evolutionary framework inspired from Cartesian Genetic Programming

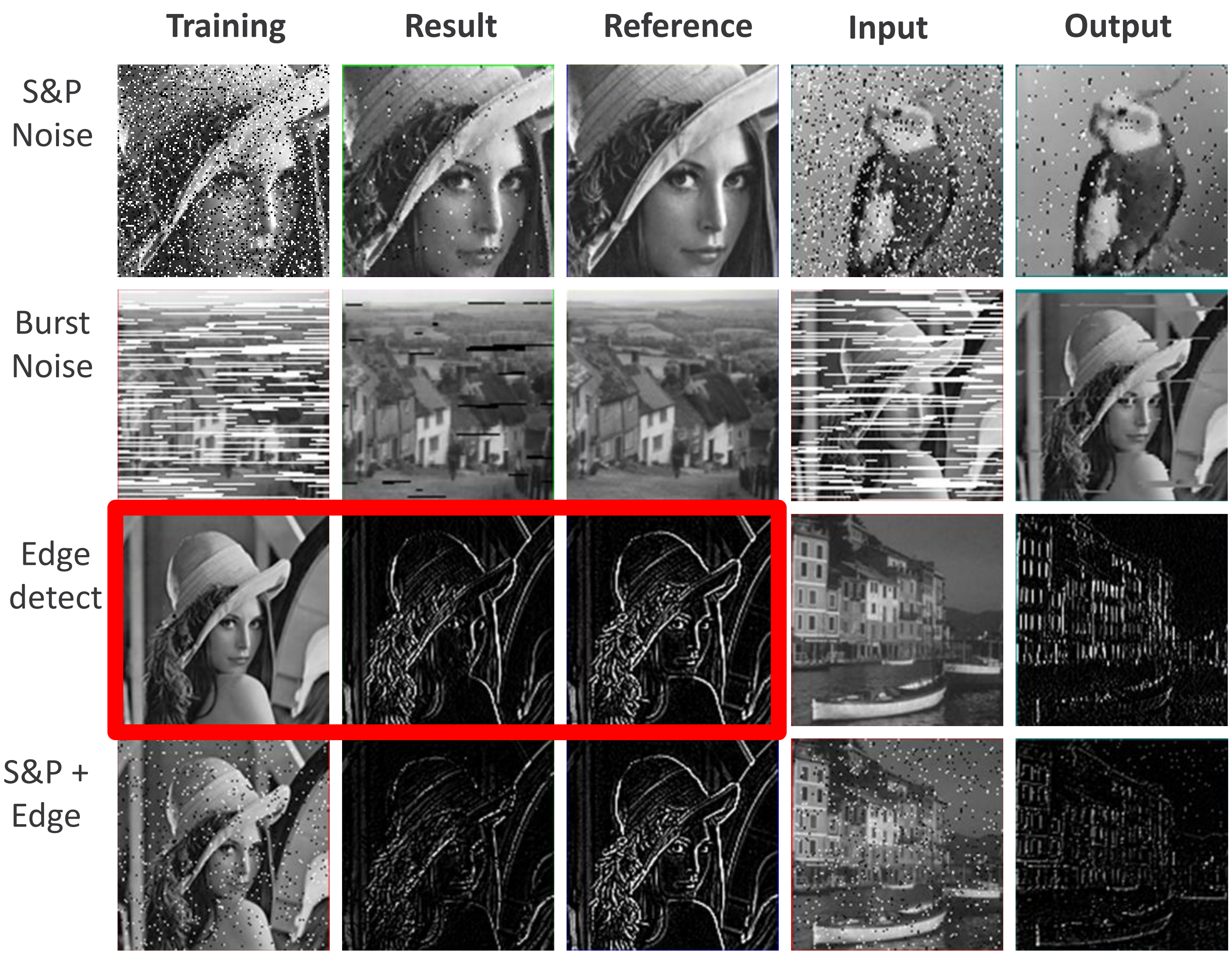
- Inspired from CGP
  - (1+8) Evolution Strategy
- Chromosome
- Random Initial Population
- Selection
  - Parent → Fittest individual
  - Elitism enabled
- Variation operator
  - Mutation
    - Over *k* randomly selected genes
      - Random uniform 0 – 8 for input genes
      - Random uniform 0 – 15 for functionality genes
- New population
  - Parent + Mutants

$$\left\langle InMux_1, \ldots, InMux_{A+B}, PE_{00}, \ldots, PE_{A \cdot B} \right\rangle$$

$$MAE = \frac{1}{RC} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} \left| Rf(r,c) - Filt(r,c) \right|$$

$$SAE = \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} \left| Rf(r,c) - Filt(r,c) \right|$$

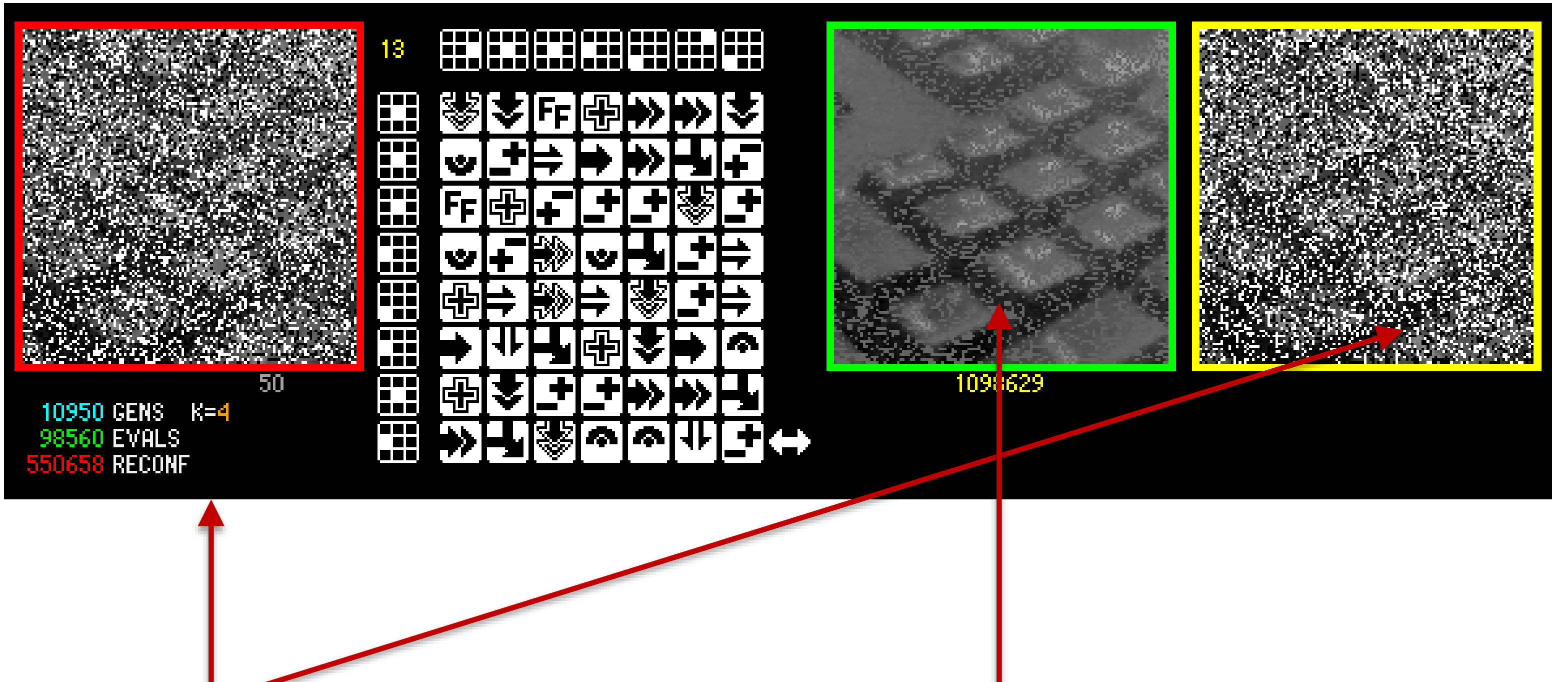
Systems should be adaptable and generalizable



CEI UPM

POLITÉCNICA

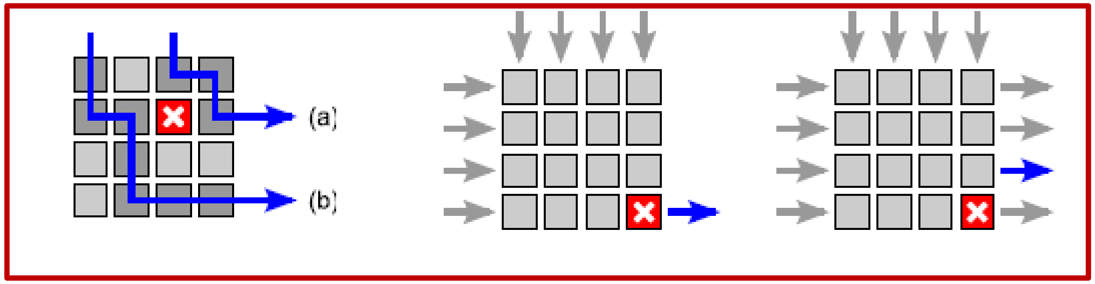
Results of a large array with very noisy reference

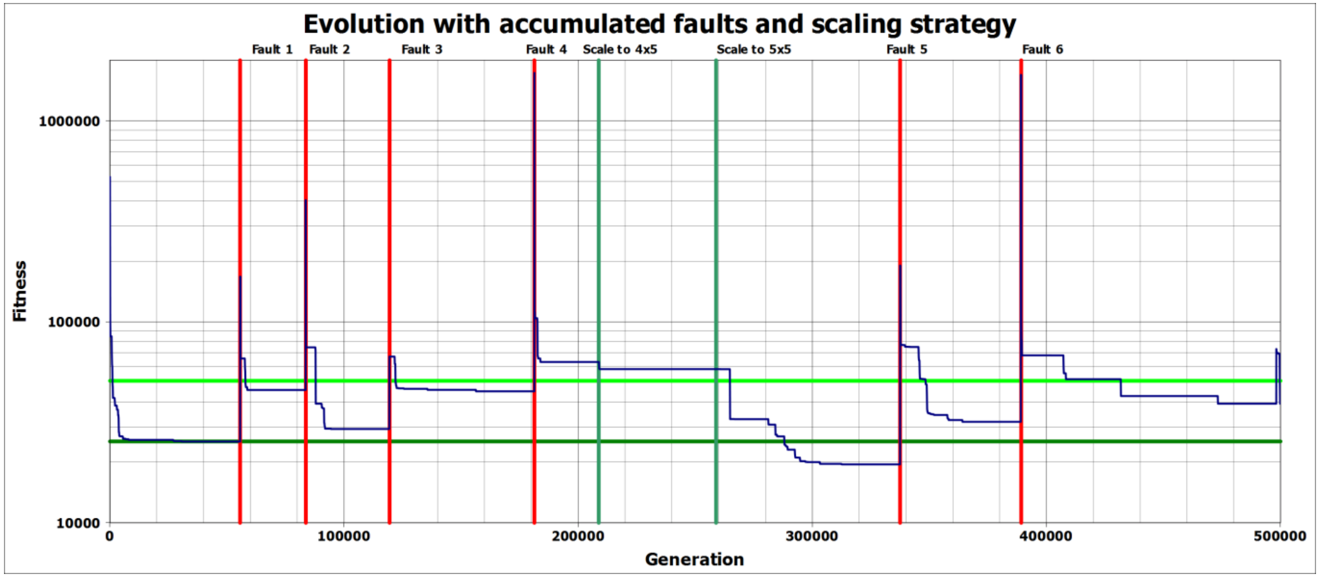


With these two inputs …. We get this output

- Noise-agnostic physically-scalable evolvable hardware
- World record in evaluations/second (> 145.000)
- A good filter, in 1 to 2 seconds, with no DSP eqs. behind!

Evolution for increased fault tolerance (Self-Healing)





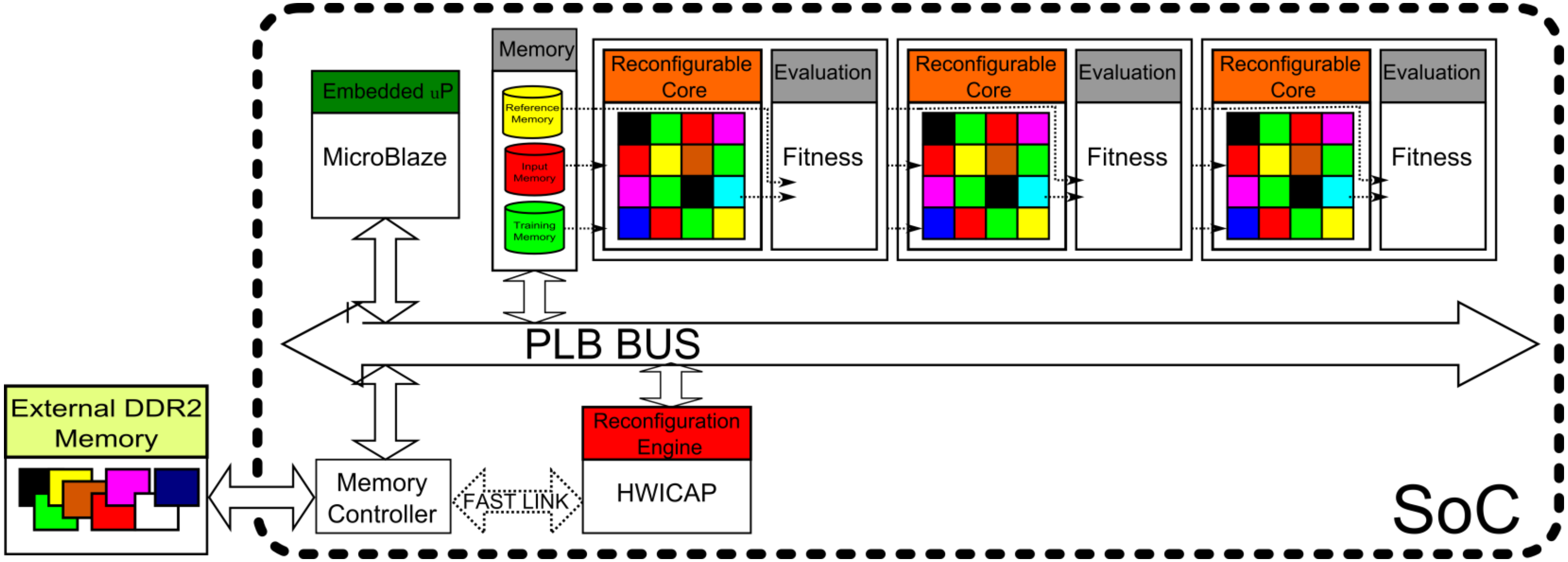
Example of evolution with accumulated faults (threshold at 2x initial fitness)

A 4x4 recovers from 2 faults in average

A 7x7 recovers from 12 faults in average

⇨ Lifetime of the system extended 6 times

Scalable Evolvable Hardware – Redundant Cores



Mission Time

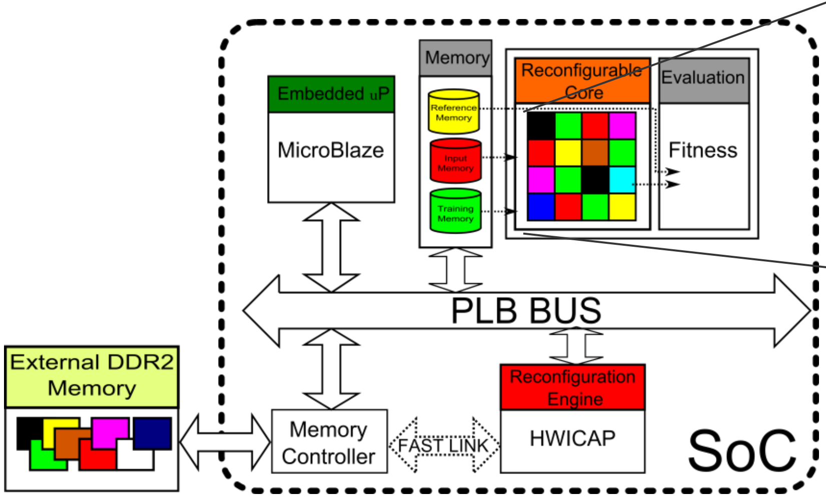
- Independent → Task Level Parallelism
- Parallel → Triple Multiple Redundancy
- Cascade
  - Collaborative
  - Independent
  - Redundant
- Bypass → Self-healing

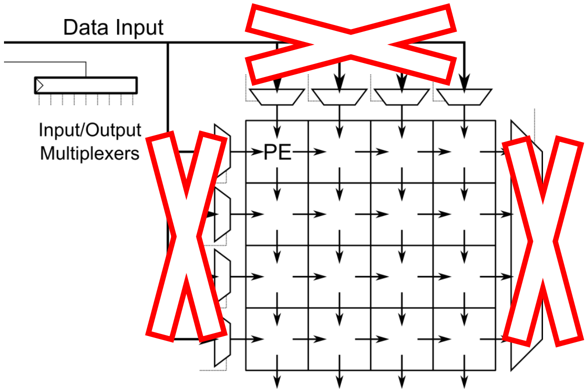
Evolution Time

- Independent → sequential
- Parallel → Accelerate Evol.
- Cascade
  - Separate/Single fitness unit
  - Sequential/Interleaved
- Imitation → Self-healing (Bypass)

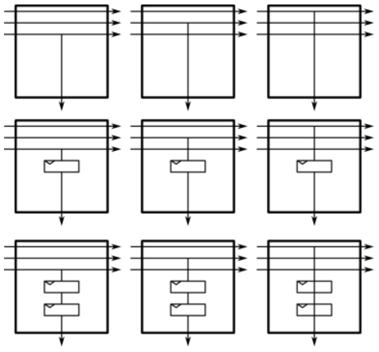
Scalable Evolvable Hardware – Scalable Cores

Single Evolvable Core

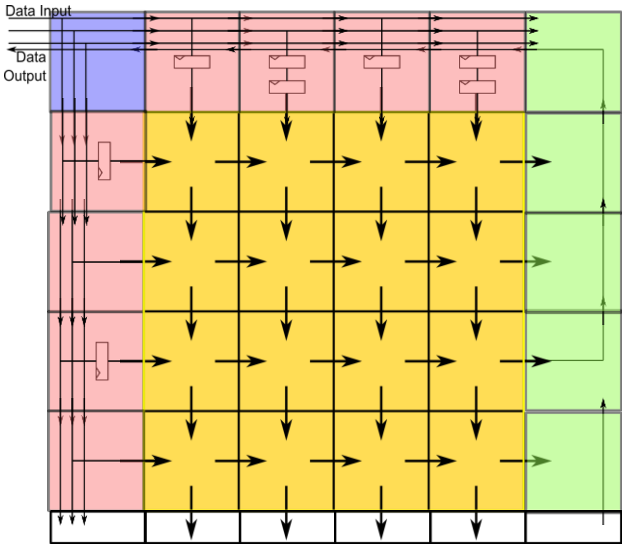




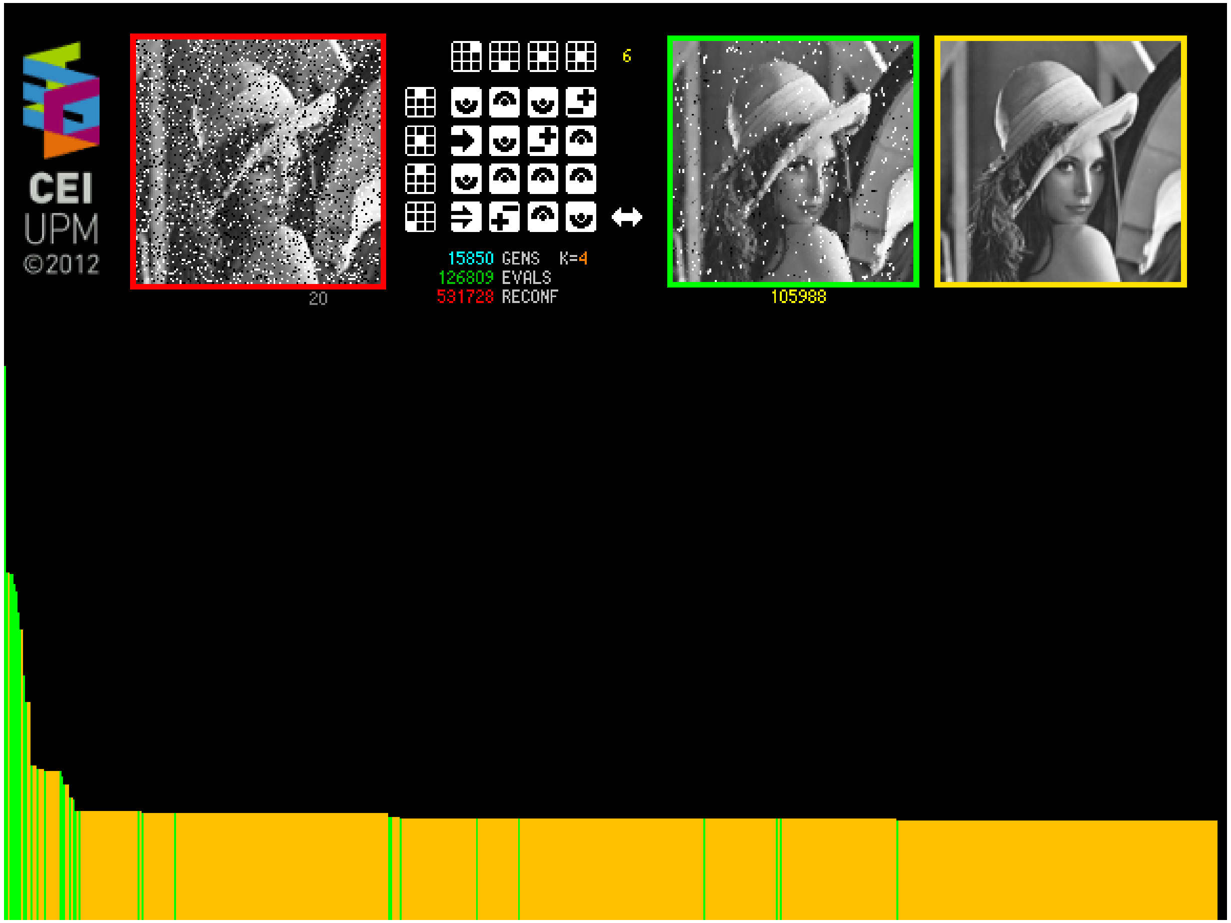
Multiplexers not compatible with scalability



Hardwired Multiplexers



Evolvable Hardware– Demo



```
6
15850 GENS  K=4
126809 EVALS
531728 RECONF
```

- On a Virtex-5
- Autonomous Adaptation!

https://www.youtube.com/watch?v=eVqfQAer_gs

Some References

R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo and L. Sekanina, "Self-Reconfigurable Evolvable Hardware System for Adaptive Image Processing," in IEEE Transactions on Computers, vol. 62, no. 8, pp. 1481-1493

R. Salvador, A. Otero, J. Mora, E. de la Torre, L. Sekanina and T. Riesgo, "Fault Tolerance Analysis and Self-Healing Strategy of Autonomous, Evolvable Hardware Systems," 2011 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 2011, pp. 164-169

A. Otero, R. Salvador, J. Mora, E. de la Torre, T. Riesgo and L. Sekanina, "A fast Reconfigurable 2D HW core architecture on FPGAs for evolvable Self-Adaptive Systems," 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), San Diego, CA, USA, 2011, pp. 336-343, doi: 10.1109/AHS.2011.5963956.

Á. Gallego, J. Mora, A. Otero, R. Salvador, E. de la Torre and T. Riesgo, "A Novel FPGA-based Evolvable Hardware System Based on Multiple Processing Arrays," 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, 2013, pp. 182-191, doi: 10.1109/IPDPSW.2013.56.

Mora, J., Salvador, R. & de la Torre, E. On the scalability of evolvable hardware architectures: comparison of systolic array and Cartesian genetic programming. Genet Program Evolvable Mach 20, 155–186 (2019). https://doi.org/10.1007/s10710-018-9340-5

NEUROEVOLUTION: COMBINING EVOLUTIONARY COMPUTATION AND ANNS

Index of the Presentation

1. Introduction to Self-* Computing Systems

2. Principles of Evolutionary Computation and Evolvable Hardware

3. Classic Evolvable Hardware: Application to Image Processing

4. Neuroevolution: Combining Evolutionary Computation and ANNs

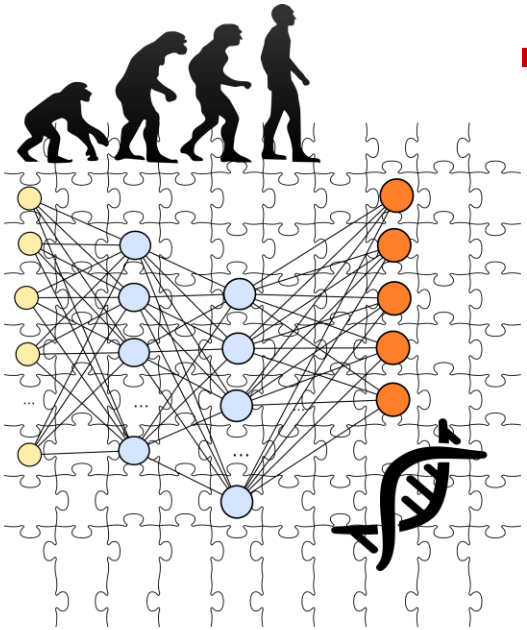
5. Deep Neuroevolution: Evolutionary Computation + Deep Learning

6. From Neuroevolution to Neuromorphic: Evolutionary Spiking Neural Networks

7. The A-IQ Ready View: hybrid computing platforms

Neuroevolution

Evolutionary Algorithms (EA) as the optimization and solution searching tool.



Artificial Neural Networks (ANN) are computational models inspired by the structure and physiology of the human brain, aiming to mimic their natural learning capabilities.

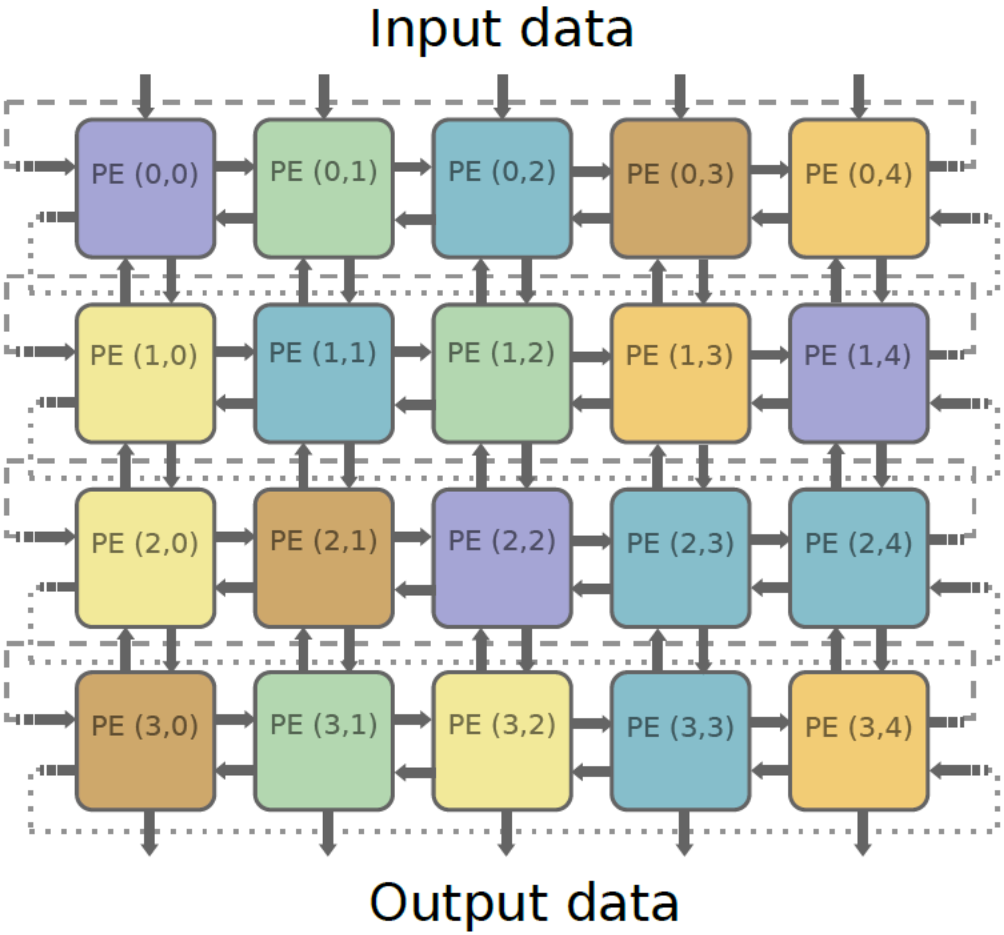
Natural learning and biological evolution are not independent processes. Natural brains are themselves products of natural selection.

Neuroevolution consist in combining ANNs and EAs. It includes techniques to create neural network topologies, weights, building blocks, hyperparameters, and learning algorithms.

CEI UPM

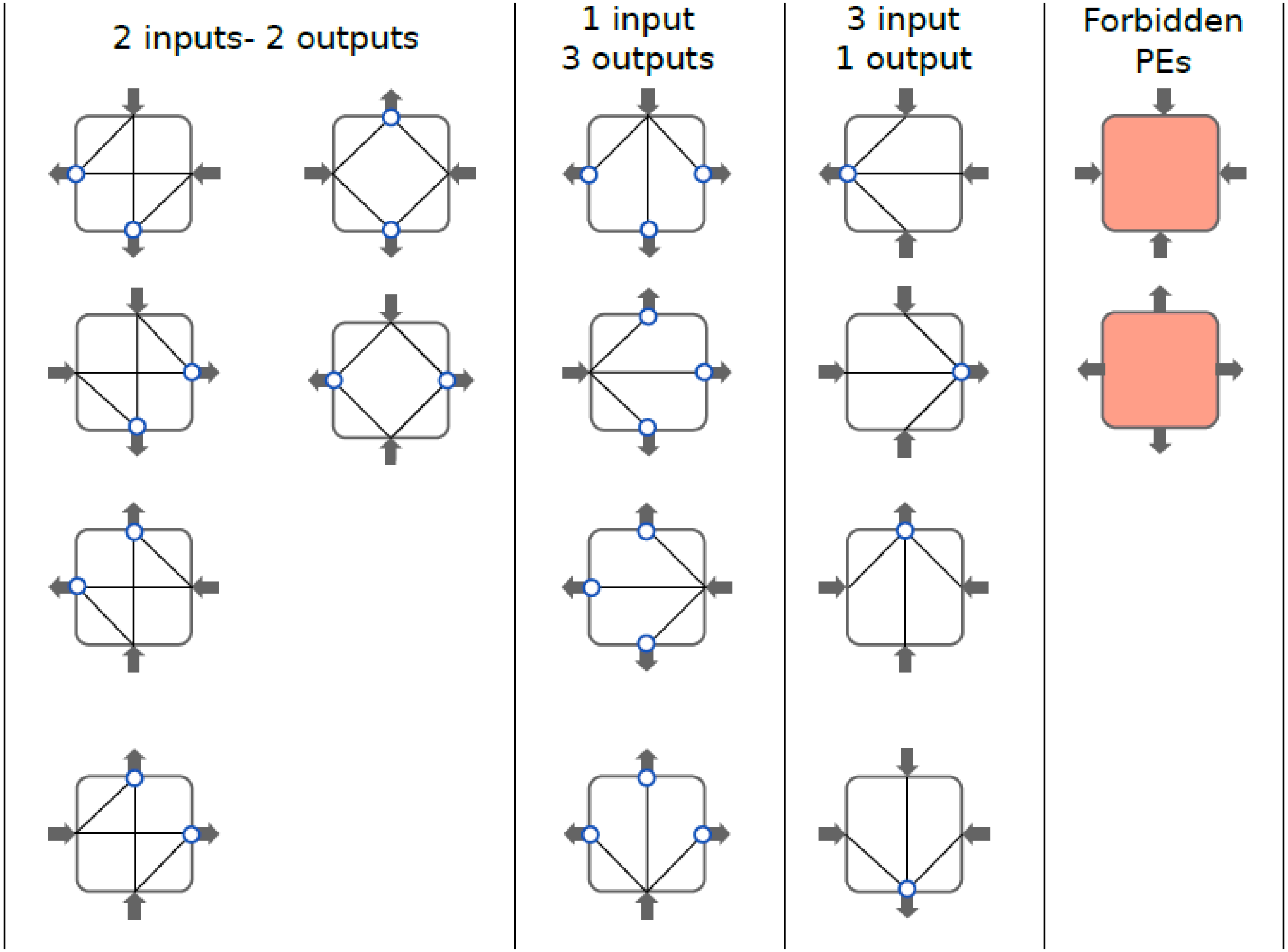
POLITÉCNICA

Block-based Neural Networks (BbNN)



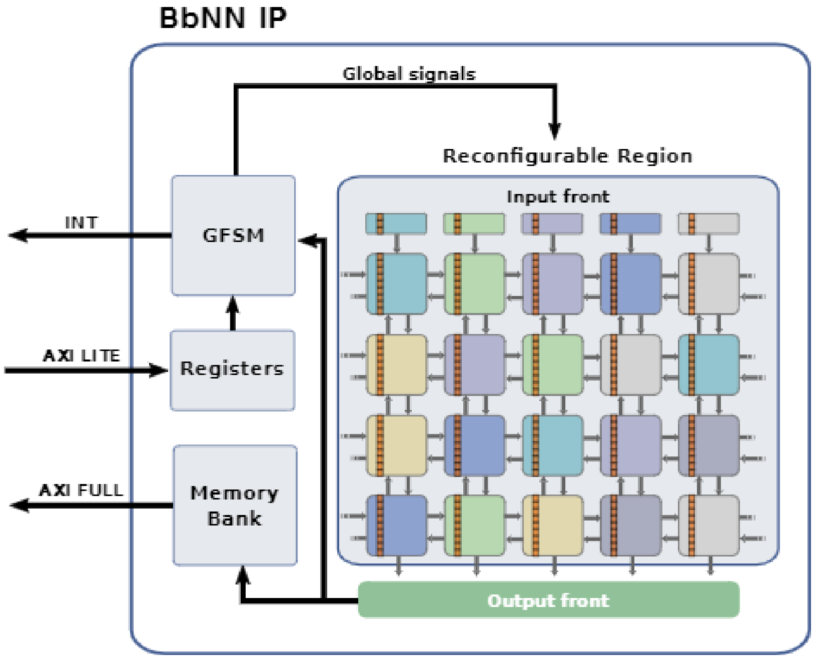
S.-W. Moon, S.-G. Kong, “Block-based neural networks”, in IEEE Transactions on Neural Networks, vol. 12, no. 2, pp. 307–317, 2001,

Original BbNN Proposal



Proposed IP for BbNNs

A hardware-accelerated integrated IP core for neuroevolution that allows training (and re-training) the neural network in an edge computing device, during its whole lifetime.



Neurons are arranged as a two-dimensional array of PEs. The number of inputs of the architecture corresponds to the number of columns.

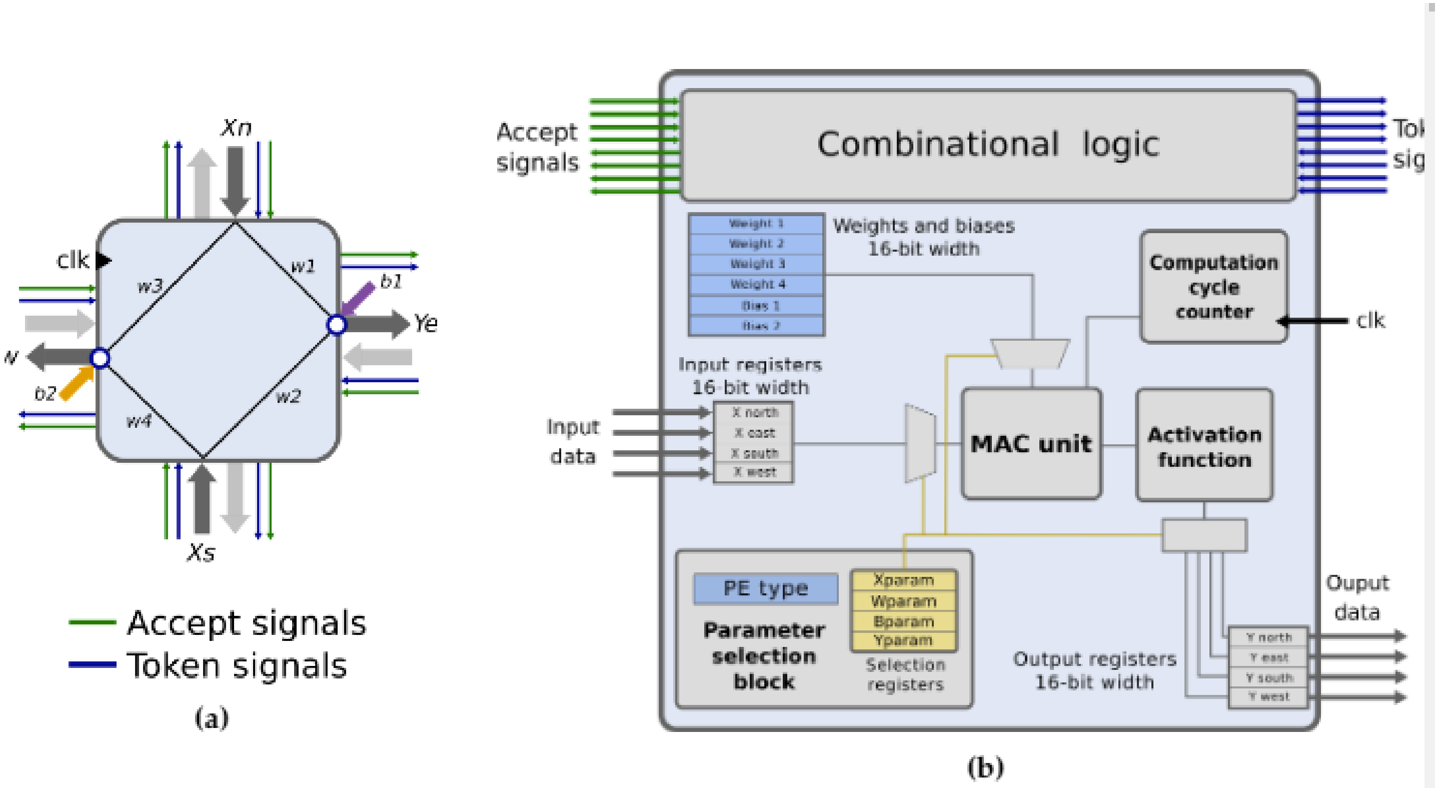
Block-based Neural Network layout

García, A.; Zamacola, R.; Otero, A.; de la Torre, E. A Dynamically Reconfigurable BbNN Architecture for Scalable Neuroevolution in Hardware. Electronics 2020, 9, 803. https://doi.org/10.3390/electronics9050803

Internal Structure of the BBNNs PE

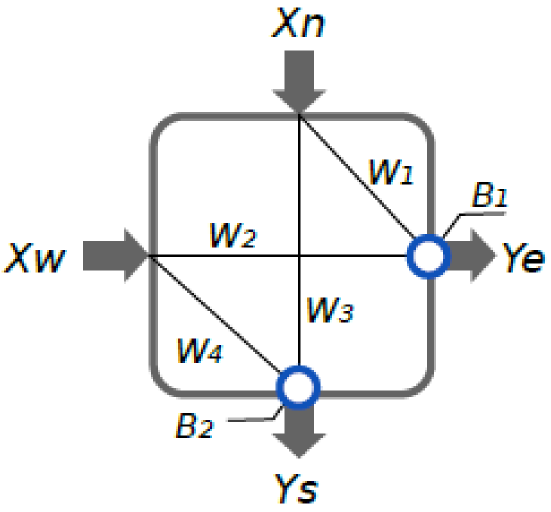
Each PE in the BbNN computes a variable number of outputs with a given number of inputs:

$$y_k = g\left(b_k + \sum_{j=1}^{J} w_{jk} x_j\right), k = 1,2,\ldots K$$



Proposed structure for the BbNN processing element.

PE Control Over 7 Cycles



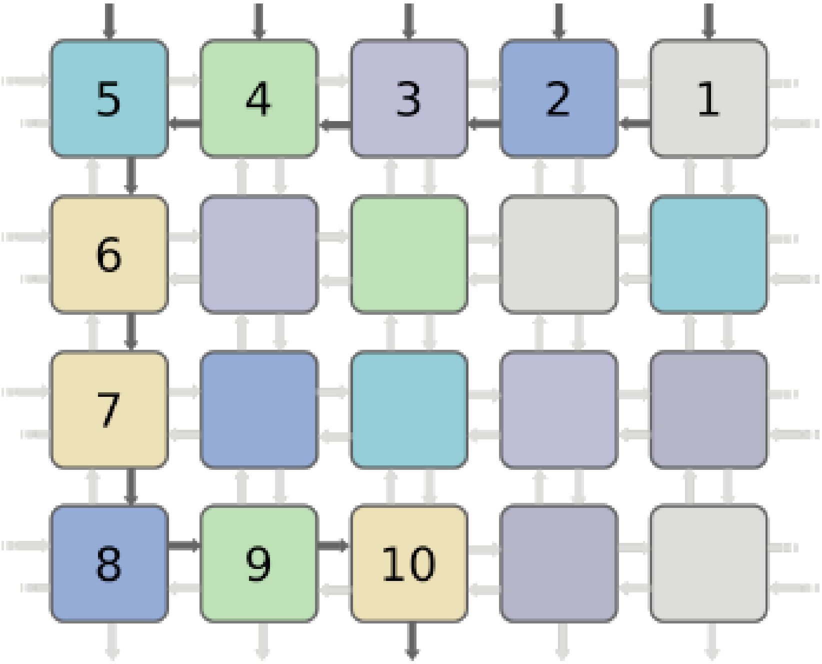
selection registers for selected PE Type

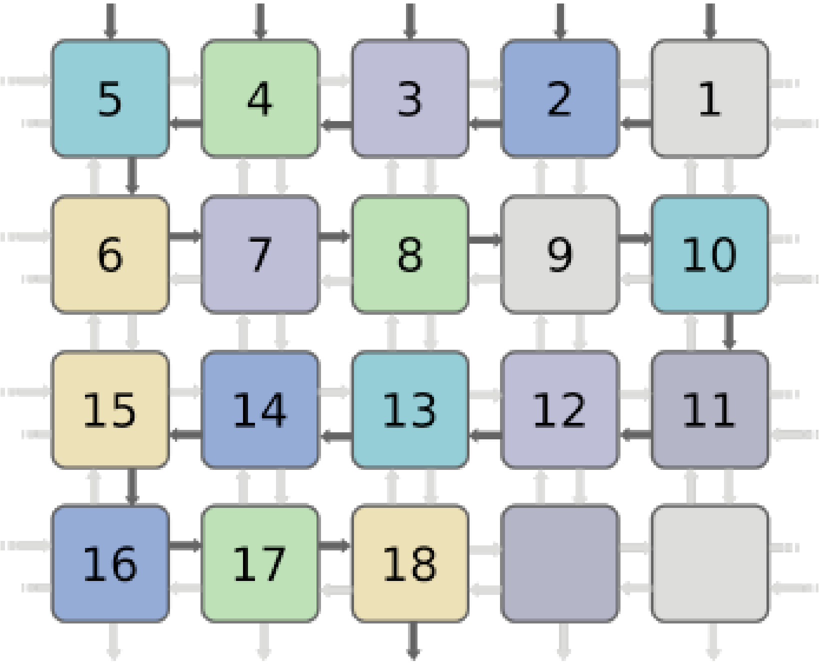
Xparam	001100001100
Wparam	000100101100
Bparam	000110
Yparam	001000100000010001000000

Clock cycle	0	1	2	3	4	5	6
Xsel	--	X_n	X_w	--	X_n	X_w	--
Wsel	--	W_1	W_2	--	W_3	W_4	--
Bsel	--	B_1	B_1	--	B_2	B_2	--
Ysel	--	Y_e	Y_e	--	Y_s	Y_s	--
	Check trigger	$(X_n{*}W_1)$	$(X_w{*}W_2)$	Clear MAC	$(X_n{*}W_3)$	$(X_w{*}W_4)$	Clear MAC

Network Latency

- Different network have different latencies

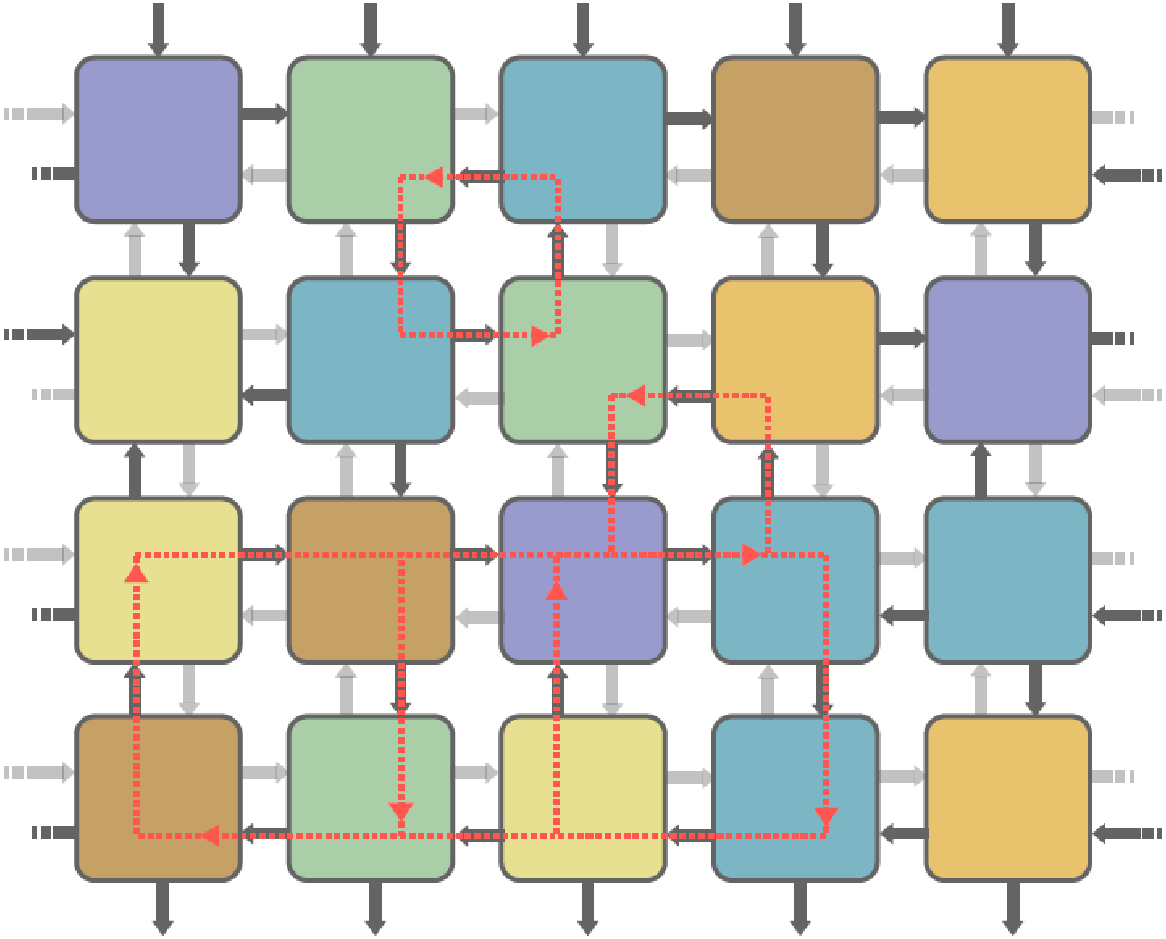




- Different options:
  - Use synchronization signals
  - Let the evolutionary algorithm find the most appropriate latency value
  - Select a random latency value

Loops in BbNN

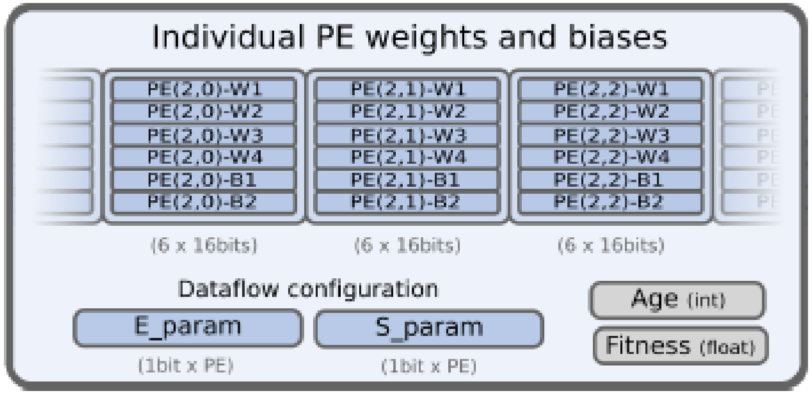
- Loops provide memory-like capabilities



BbNNs for neuroevolution

The hardware-accelerated integrated BbNN IP core for neuroevolution is accompanied by an EA running on a embedded processor, that allows training (and re-training) the neural network in an edge computing device, during its whole lifetime.

Chromosome Representation



Bit value	E_param	S_param
0	←	↓
1	→	↑

The evolutionary algorithm can decide the direction of every link during the training stage, so internal loops may appear.

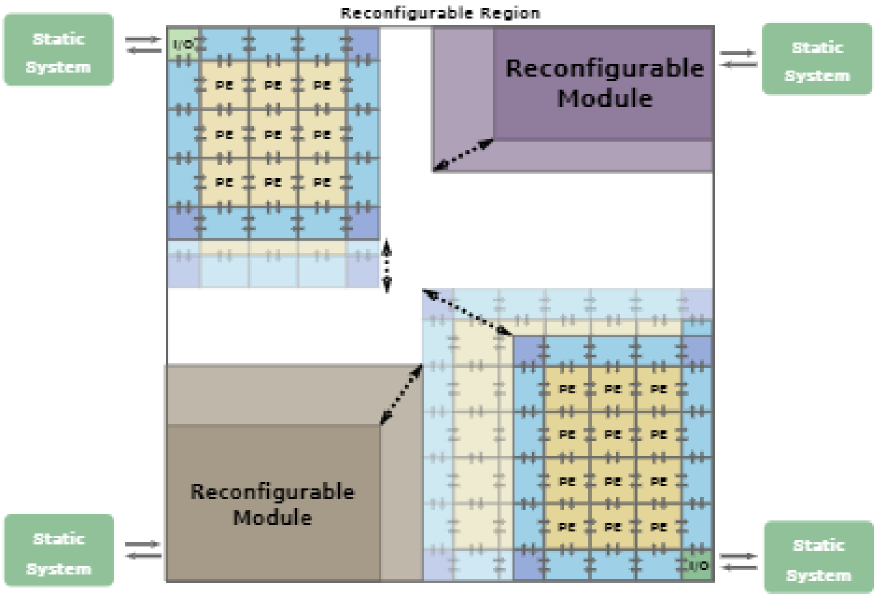
This approach enables the **continuous adaptation of systems** working in dynamic environments. Continuous adaptation is not possible in conventional ANNs that use gradient-based back propagation algorithms for training.

Evolutionary Algorithm

```
Initialization(population);
bestFitness = evaluate(population);
rows = 1;
while bestFitness < targetFitness do
    for chromosome in population do
        offspring = mutation(chromosome);
        bestFitnessMut = evaluate(offspring);
        if bestFitnessMut > (chromosome.fitness) then
            replaceChromosome(chromosome, offspring)
            chromosome.age = 0;
        else
            chromosome.age++;
            if chromosome.age > maxAge then
                remove(chromosome);
    selectBest(population);
    if generationsExtintion > ExtinctionFreq then
        extinction(population);
        generationsExtintion = 0;
    else
        generationsExtintion++;
    if generationsScaling > ScalingFreq then
        composeBbNN(++rows);
        generationsScaling = 0;
    else
        generationsScaling++;
    generations++;
    if generations > maxGenerations then
        break;
```

Concept of Aging to guarantee diversity

Dynamically Scalable BBNNs

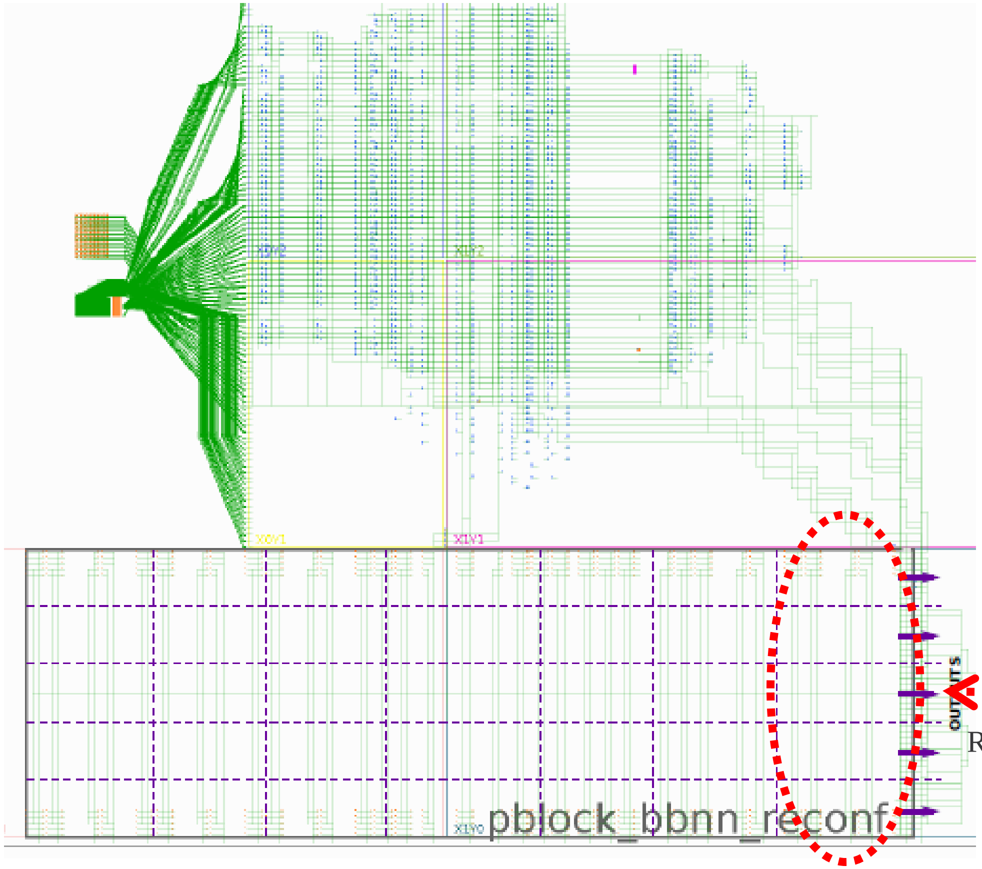


Scalable Implementations of the BBNN

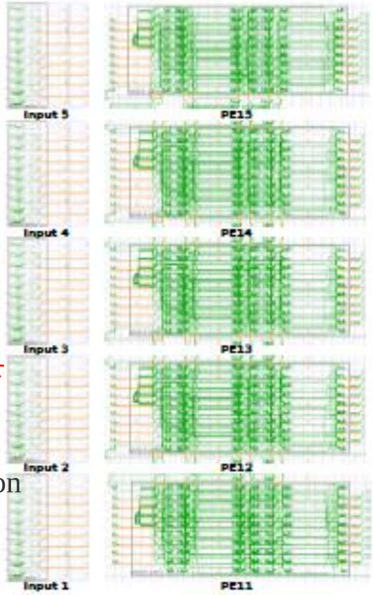
Dynamically scalable BbNN model handles the size of the network as a parameter to be optimized at run-time, instead of being fixed at design-time. This way, the optimization algorithm can find the appropriate size, as a trade-off between the size of the design space under exploration and the capability of the architecture to undertake complex problems.

BbNN Implementation

Implementation in a xc7z020clg400-1 SoC (Digilent Pynq board)

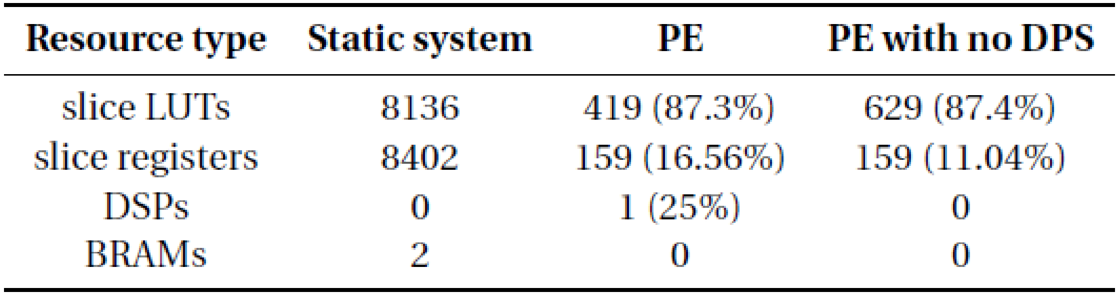


Run-time composition



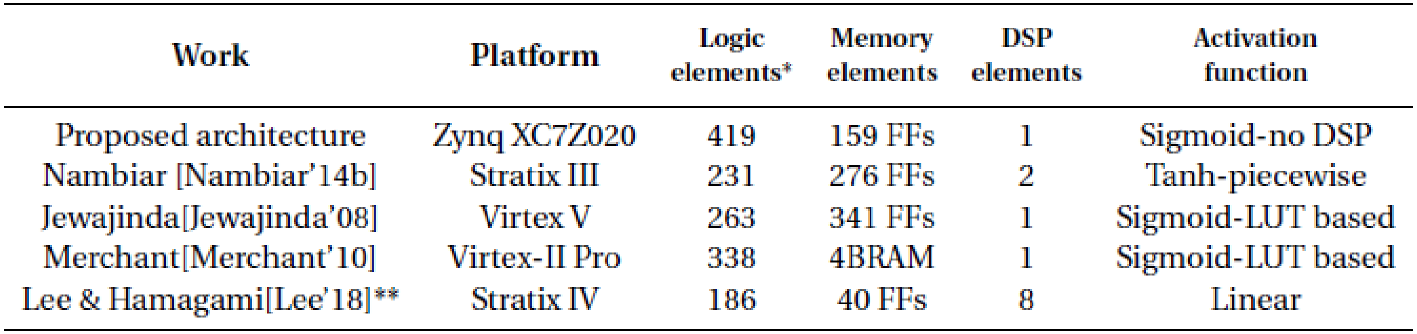
BbNN Resources

PE resources



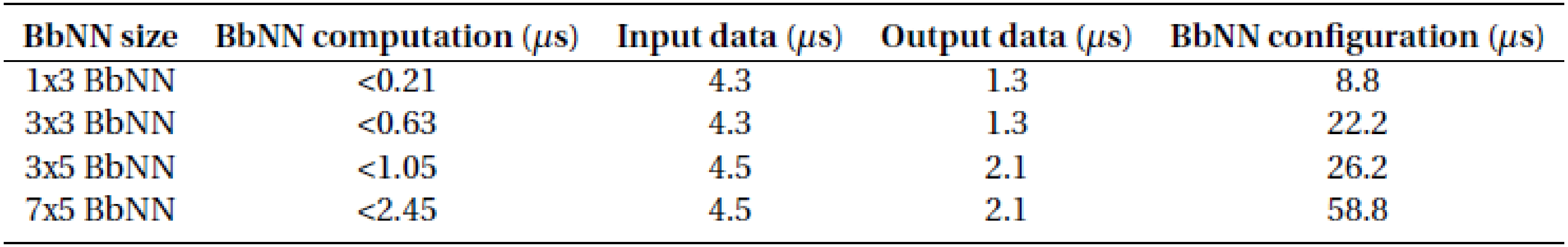
Resource type	Static system	PE	PE with no DPS
slice LUTs	8136	419 (87.3%)	629 (87.4%)
slice registers	8402	159 (16.56%)	159 (11.04%)
DSPs	0	1 (25%)	0
BRAMs	2	0	0

Comparison with other SoA implementations



Work	Platform	Logic elements*	Memory elements	DSP elements	Activation function
Proposed architecture	Zynq XC7Z020	419	159 FFs	1	Sigmoid-no DSP
Nambiar [Nambiar'14b]	Stratix III	231	276 FFs	2	Tanh-piecewise
Jewajinda[Jewajinda'08]	Virtex V	263	341 FFs	1	Sigmoid-LUT based
Merchant[Merchant'10]	Virtex-II Pro	338	4BRAM	1	Sigmoid-LUT based
Lee & Hamagami[Lee'18]**	Stratix IV	186	40 FFs	8	Linear

BbNN Performance



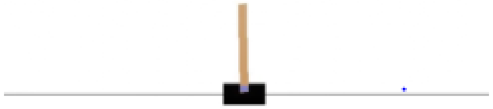
BbNN size	BbNN computation (μs)	Input data (μs)	Output data (μs)	BbNN configuration (μs)
1x3 BbNN	<0.21	4.3	1.3	8.8
3x3 BbNN	<0.63	4.3	1.3	22.2
3x5 BbNN	<1.05	4.5	2.1	26.2
7x5 BbNN	<2.45	4.5	2.1	58.8

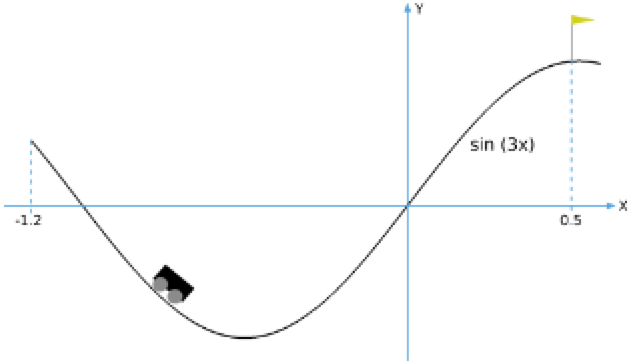
Applications of Neuroevolvable BBNNs

BBNN driven by means of EAs are capable of **lifelong learning in dynamic environments**, as a form of Reinforcement Learning. **Reinforcement Learning** drives agents on how to take actions in an environment, trying to maximize a cumulative reward.

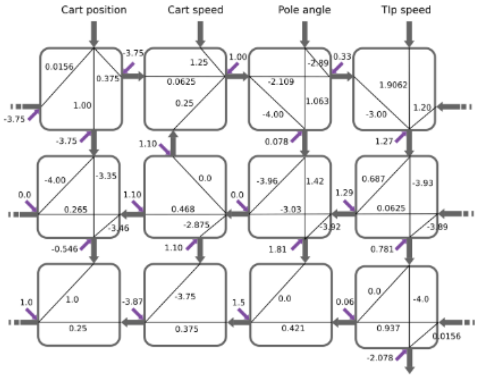
Applied to OpenAI environments

OpenAI





Evolved Controller



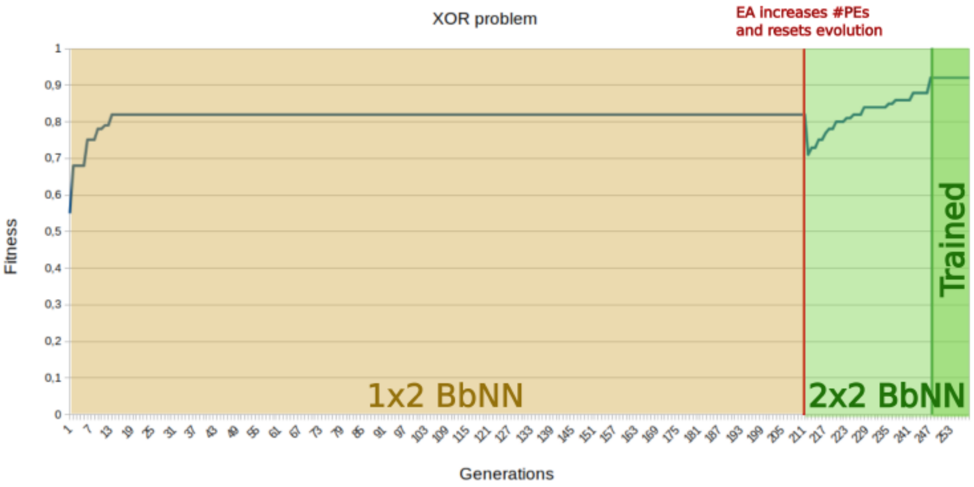
https://www.youtube.com/watch?v=bJnAgLmLHe0&t=381s

Mins. 3:39 – 5:09

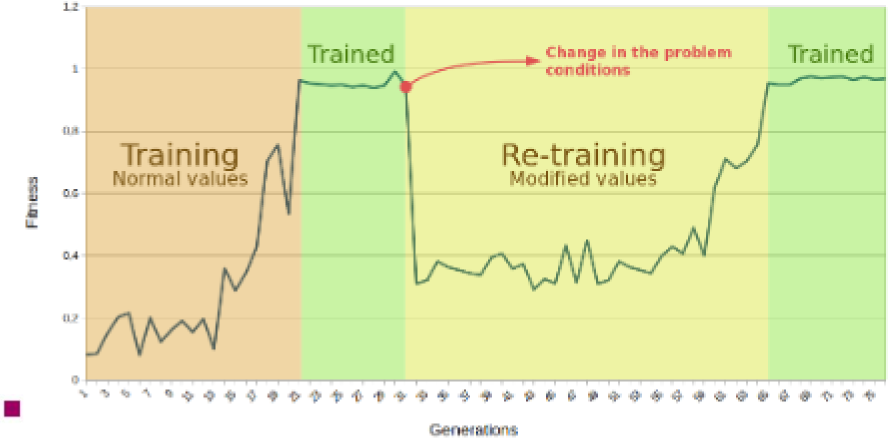
Applications of Neuroevolvable BBNNs

BBNN driven by means of EAs are capable of **lifelong learning in dynamic environments**, as a form of Reinforcement Learning. **Reinforcement Learning** drives agents on how to take actions in an environment, trying to maximize a cumulative reward.

Letting the EA find the BbNN size



Self-adaptivity for Fault-tolerance



DEEP NEUROEVOLUTION: EVOLUTIONARY COMPUTATION + DEEP LEARNING

Index of the Presentation

1. Introduction to Self-* Computing Systems

2. Principles of Evolutionary Computation and Evolvable Hardware

3. Classic Evolvable Hardware: Application to Image Processing

4. Neuroevolution: Combining Evolutionary Computation and ANNs

5. Deep Neuroevolution: Evolutionary Computation + Deep Learning

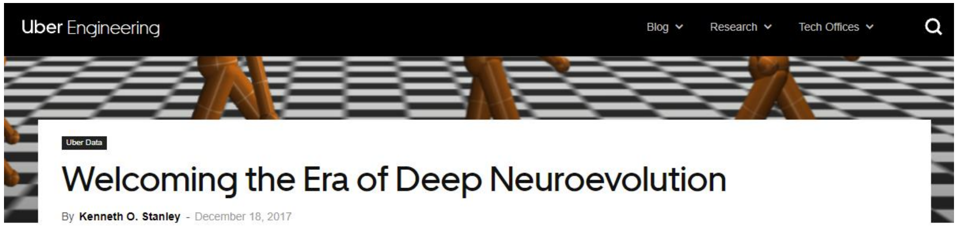
6. From Neuroevolution to Neuromorphic: Evolutionary Spiking

Neural Networks

7. The A-IQ Ready View: hybrid computing platforms

Deep Neuroevolution

Designing a Deep Neural Network (DNN) involves a manual process with relies on the experience of designers. It's a process with a significant manual component, almost a handicraft work. Neuroevolution techniques can be also applied to develop new solutions based on Deep Neural Networks.

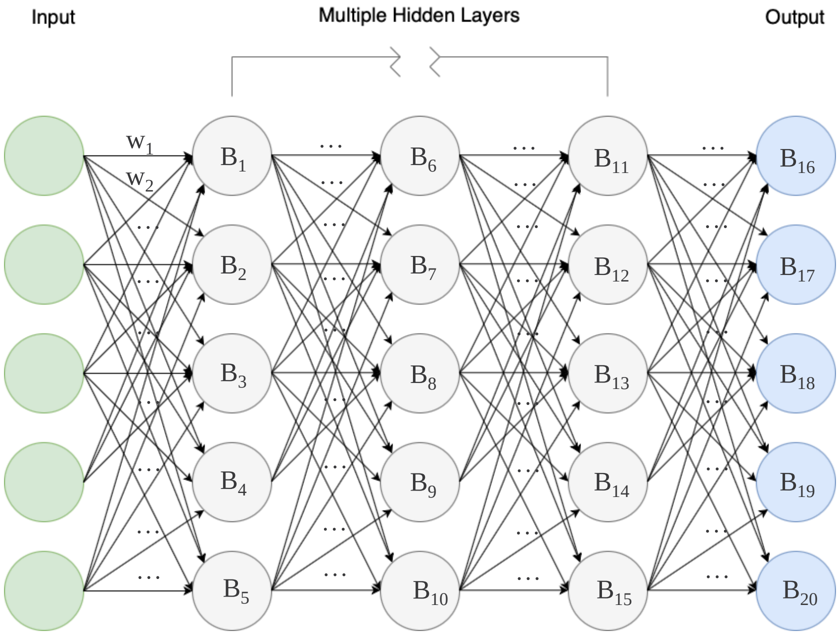


Welcoming the Era of Deep Neuroevolution

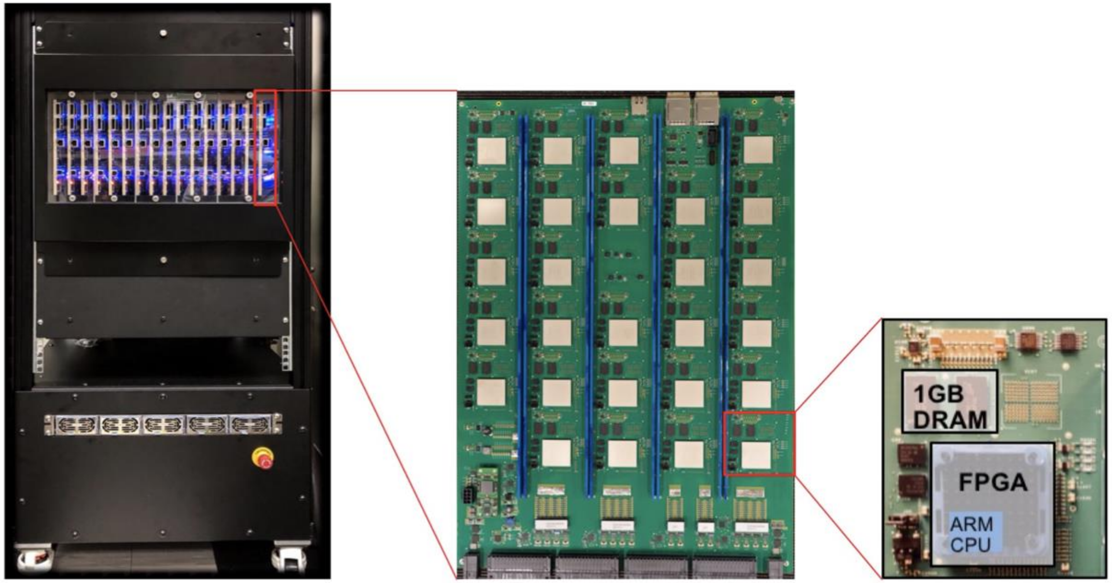
By Kenneth O. Stanley - December 18, 2017

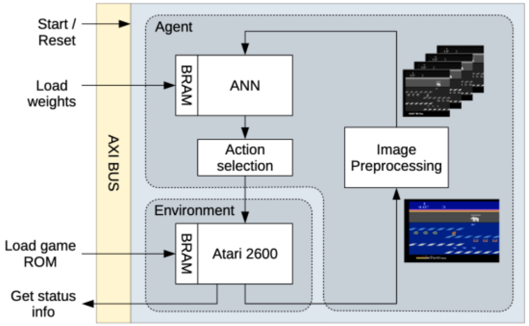
- Use Evolutionary Algorithms to automate the design of Deep Neural networks
- Implement architectures and algorithms that enable online (runtime) DNN training.

Deep Neural Network (DNN)



IBM Neural Computer





IBM's Neural Computer consists of 432 nodes (27 nodes across 16 modular cards) based on Xilinx FPGAs. Each node comprises a Xilinx Zynq system-on-chip — a dual-core ARM A9 processor paired with an FPGA on the same die — along with 1GB of dedicated RAM. The nodes are arranged in a 3D mesh topology, interconnected vertically with electrical connections called through-silicon vias that pass completely through silicon wafers or dies.

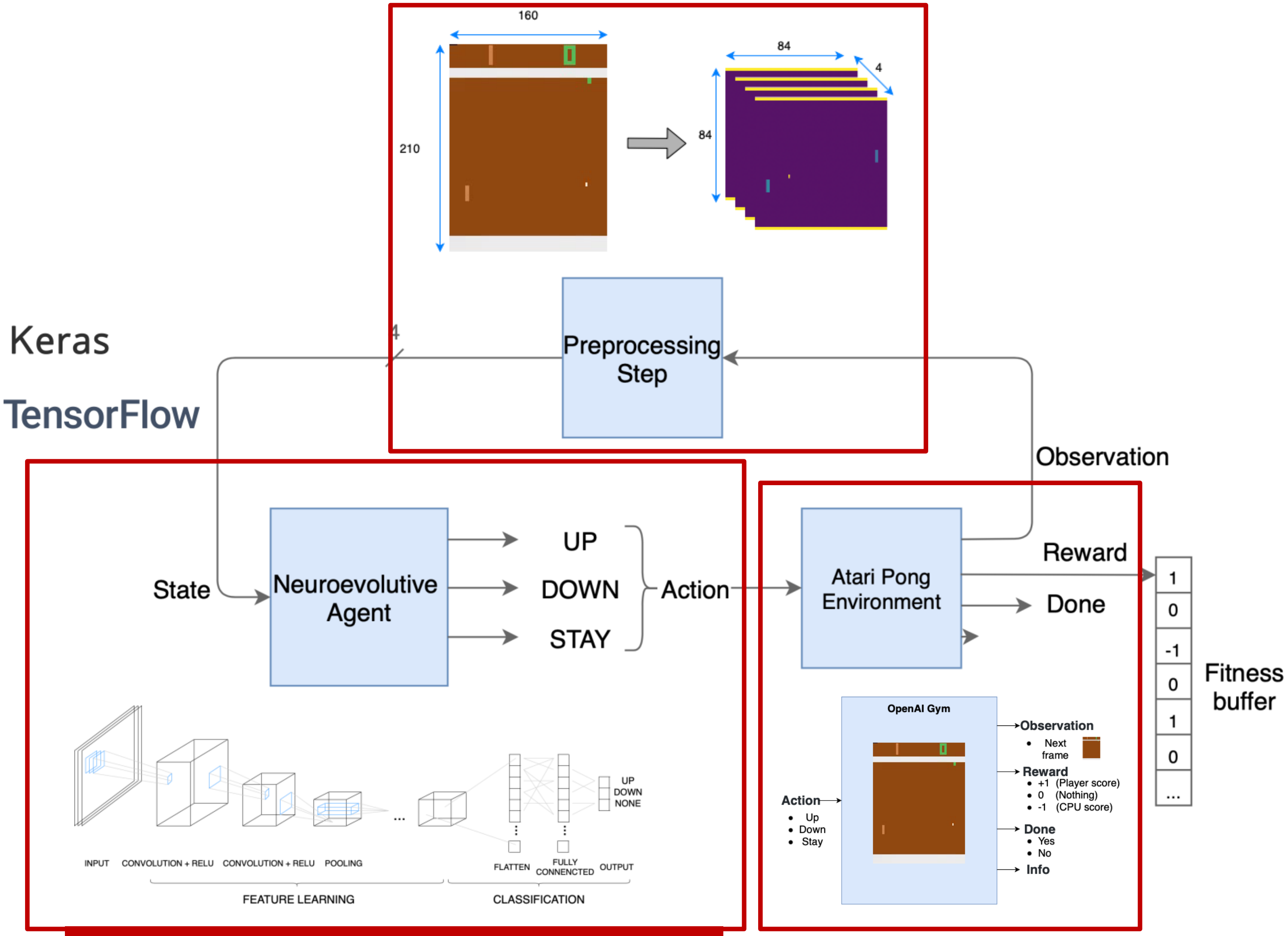
Can this approach work on the embedded domain?

Narayanan, P., Cox, C. E., Asseman, A., Antoine, N., Huels, H., Wilcke, W. W., & Ozcan, A. S. (2020). Overview of the IBM neural computer architecture. arXiv preprint arXiv:2003.11178.

System Architecture

Keras

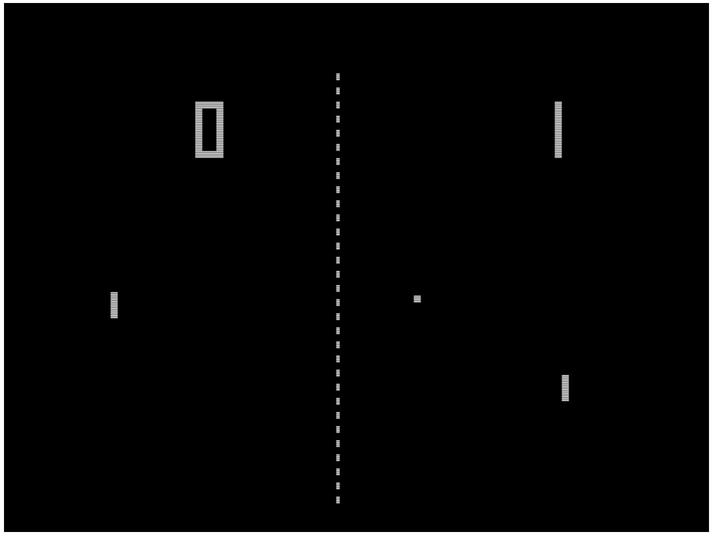
TensorFlow



Convolutional Neural Network

Problem to solve

Atari Pong Game

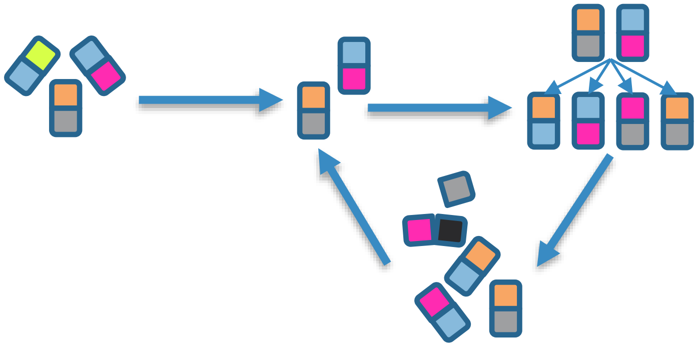


A whole framework has been developed for deploying neuroevolutive algorithms over different hardware architectures (CPU, GPU and FPGA).

OpenAI

Laserna, J., Otero, A., Torre, E.d.l. (2022). A Multi-FPGA Scalable Framework for Deep Reinforcement Learning Through Neuroevolution. In: Gan, L., Wang, Y., Xue, W., Chau, T. (eds) Applied Reconfigurable Computing. Architectures, Tools, and Applications. ARC 2022.

Genetic Algorithm for training



- Evolving the topology, **weights** and/or hyperparameters in a DNN.
- Avoid the handcrafted design of the network topology.
- Enables Continuous learning of the network.

Hyper-parameter	Value
Generations number (G)	250
Population size (N)	1000
Selected individuals (T)	200 (20% of G)
Mutation power (σ)	0.005
Crossover power (ω)	0
Number of elites (E)	1

Fitness Function

1
0
-1
0
1
0
...

Fitness buffer

Option 1

$$\text{fitness} = \sum_{i=0}^{N} F_i$$

0 + 0 + 0 + 0 + 1 + 0 + 0 + 0 + 0 + 1 + 0 + 0 + 0 + 0 + 0 + -1 + ... + 1

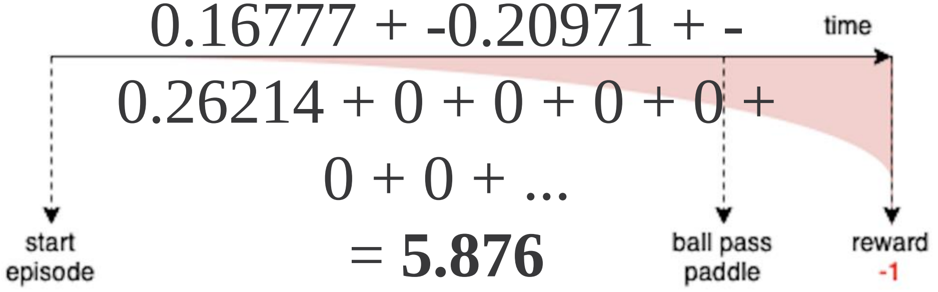
= **20**

Option 2

$$\text{fitness} = \sum_{i=0}^{N} \gamma^i F_i \ \ for\ \gamma = 0.5$$

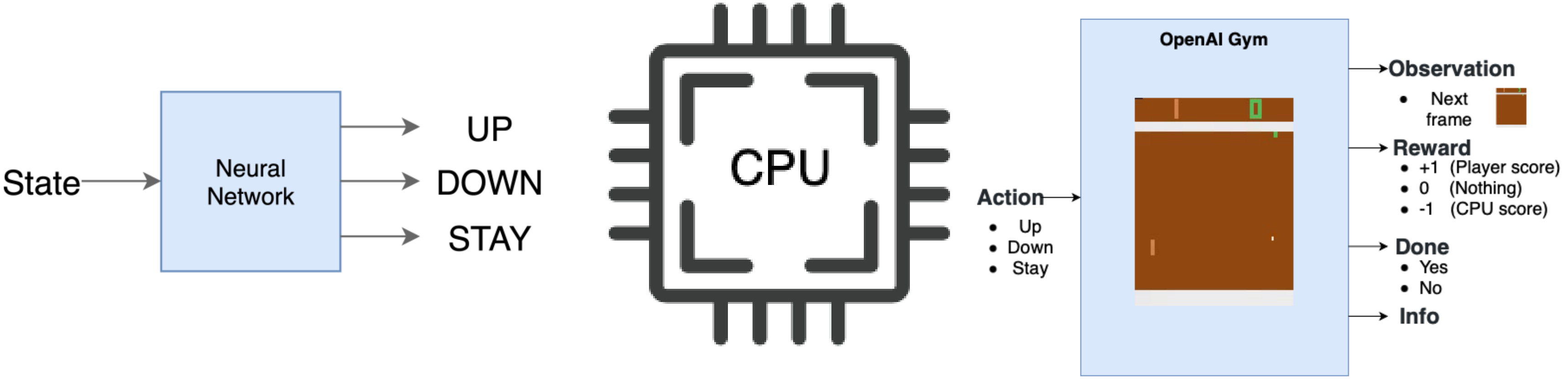
... + -0.04398 + -0.05497, -0.06871 + -0.08589 + -0.10737 + -0.13421 + -0.16777 + -0.20971 + -0.26214 + 0 + 0 + 0 + 0 + 0 + 0 + ...

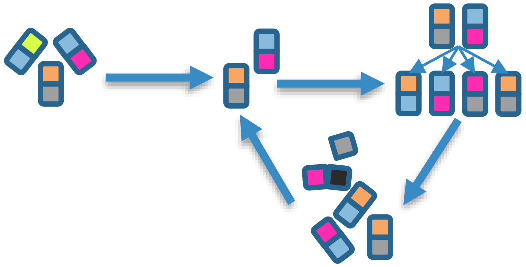
= **5.876**



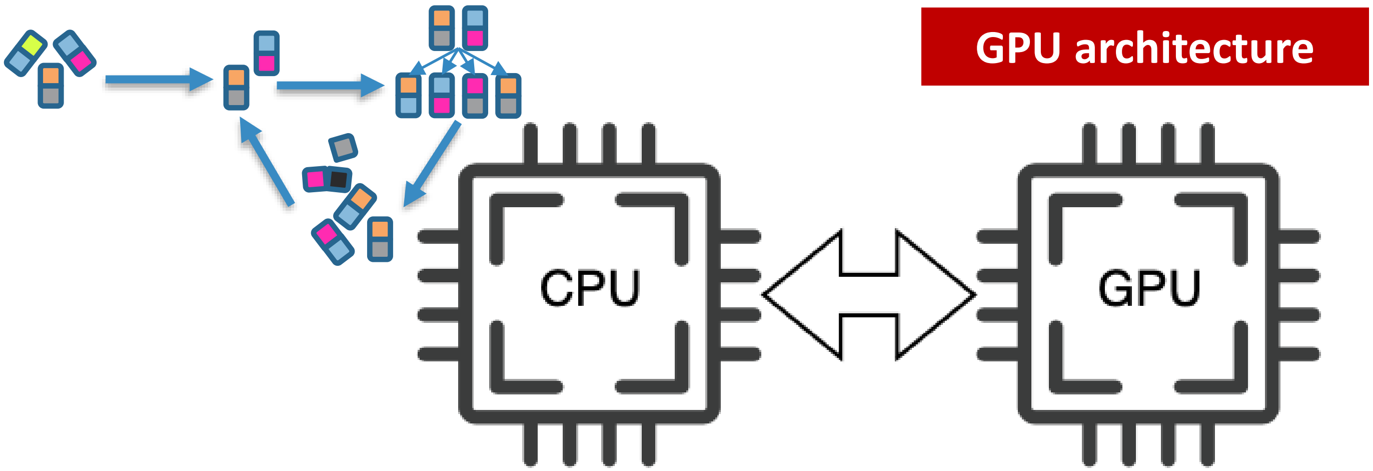
Implementations: CPU-based Solution

CPU architecture

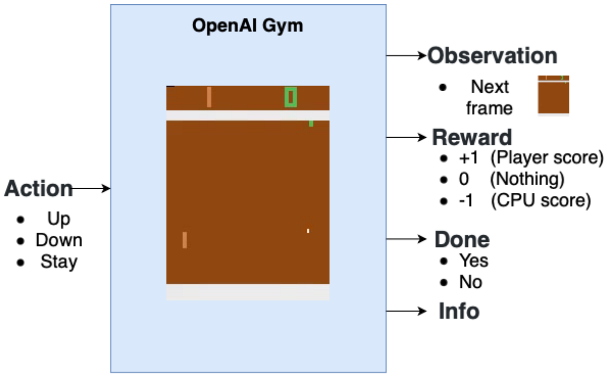




Implementations: GPU-based Solution



GPU architecture



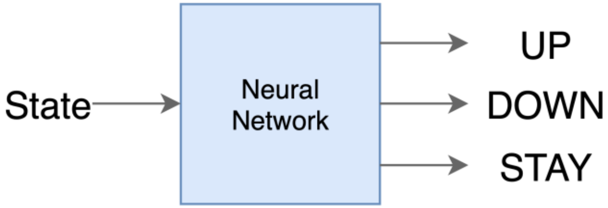
Action
- Up
- Down
- Stay

Observation
- Next frame

Reward
- +1 (Player score)
- 0 (Nothing)
- -1 (CPU score)

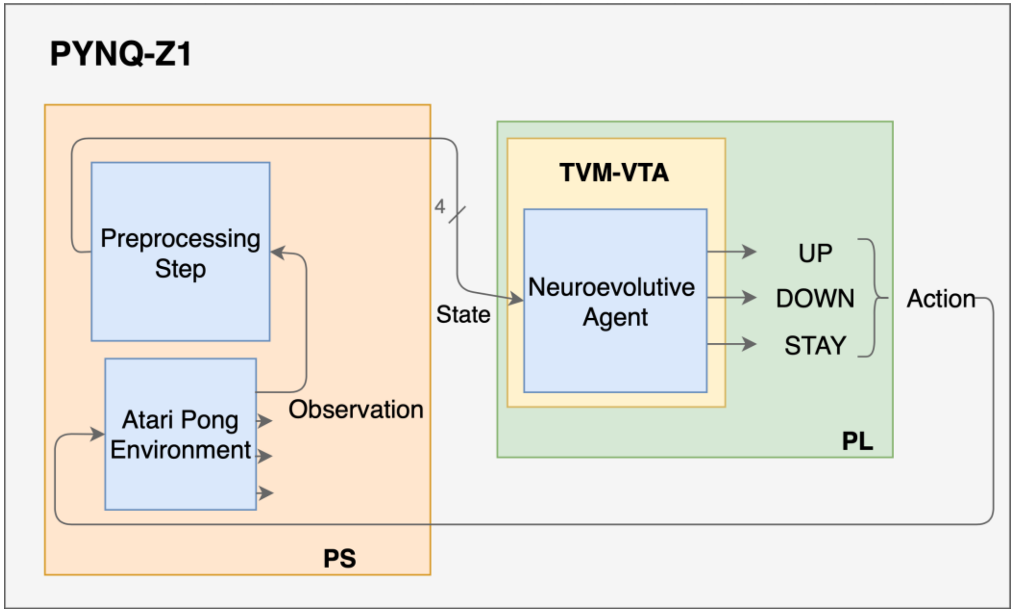
Done
- Yes
- No

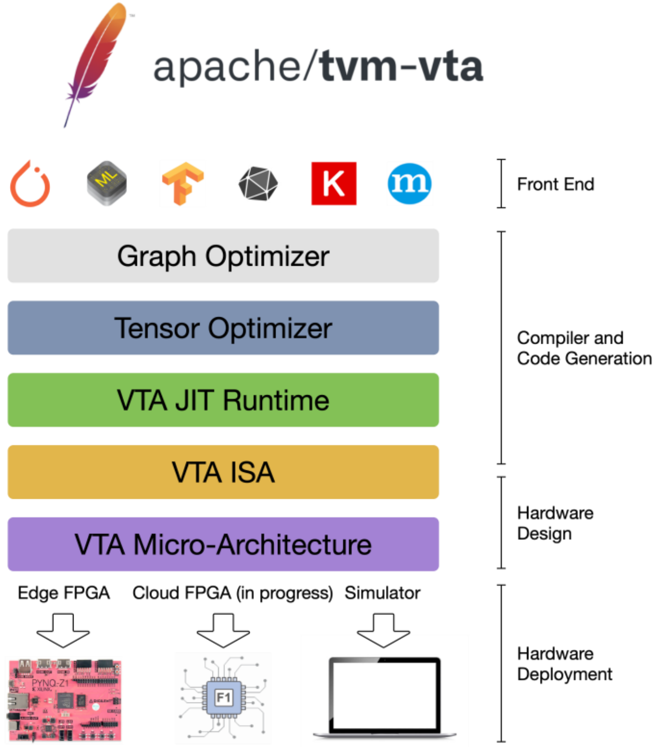
Info



Implementations: FPGA-based Solution

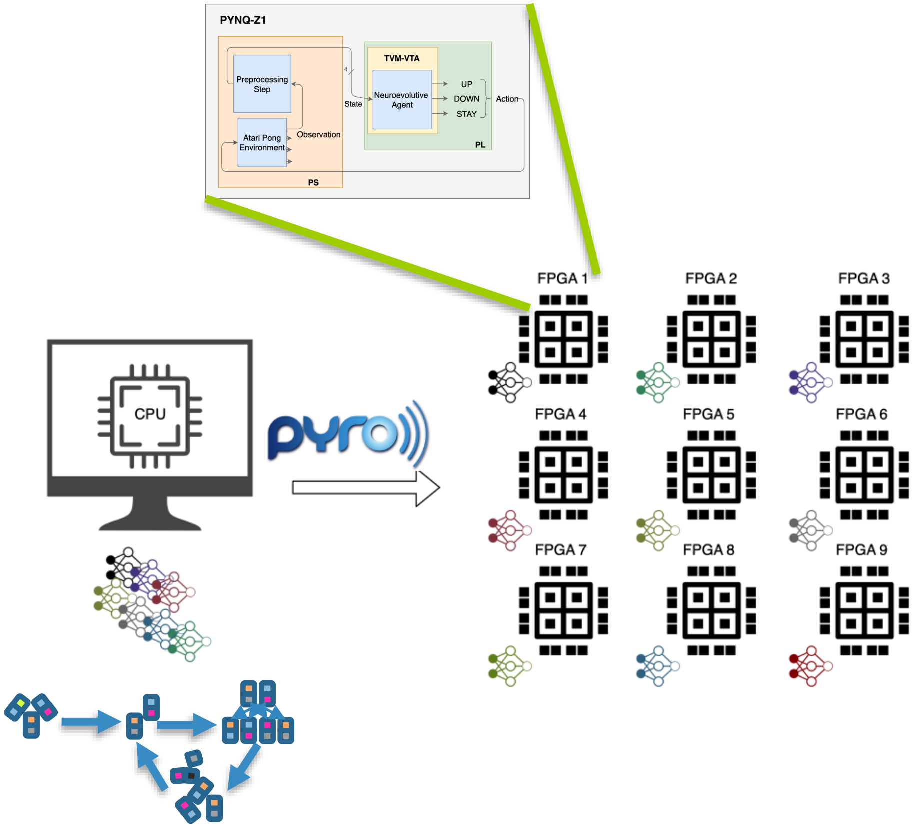
FPGA SoC architecture



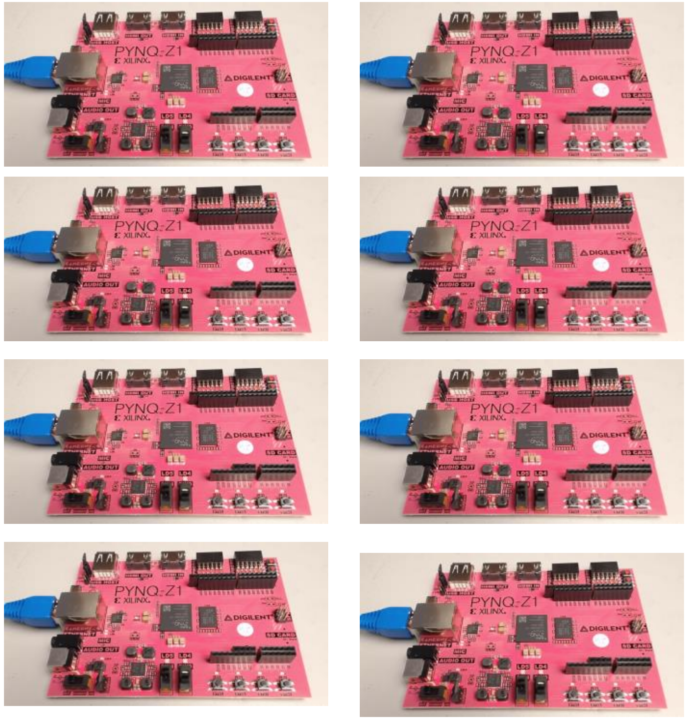


https://tvm.apache.org/vta

Implementations: Multi FPGA Solution



Multi-FPGA Distribution

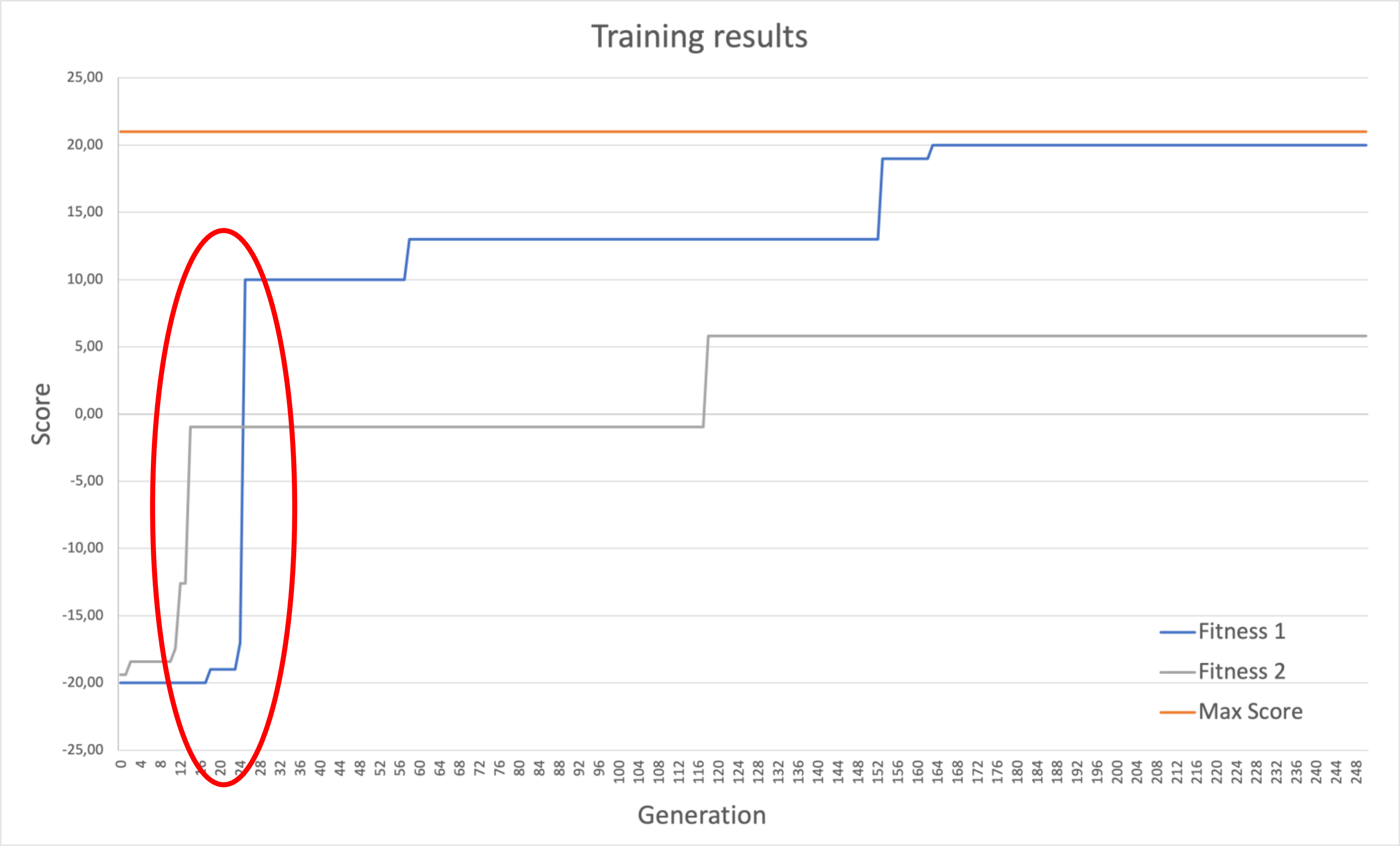


Hardware Details

- *CPU:* Intel Core i5-10400 2.90 GHz
- *GPU:* AMD Radeon RX 570 8 GB
- *VTA:* PYNQ-Z1 (Zynq®-7000 FPGA + dual-core Cortex-A9 processor)

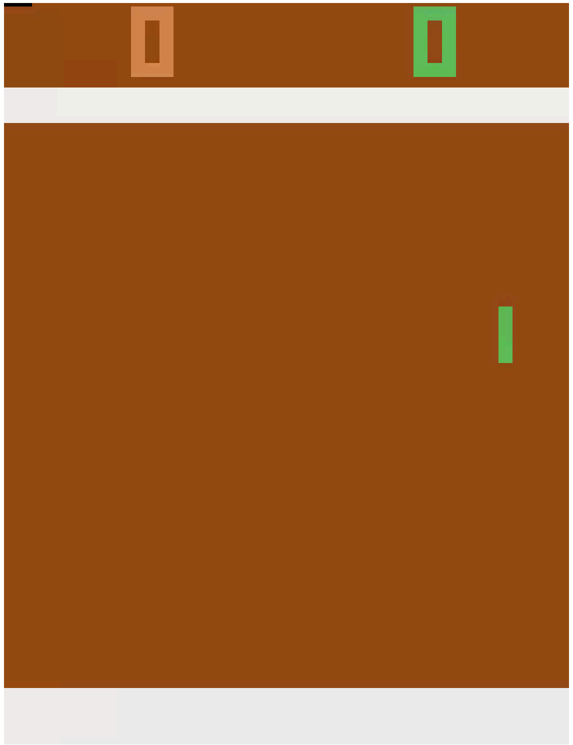
TDP		
CPU	GPU	VTA
65 W	180W	4.7 W

Experimental Results



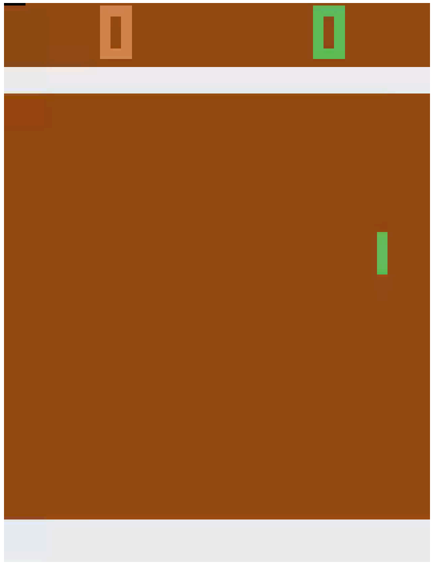
Note: the higher the better

Experimental Results

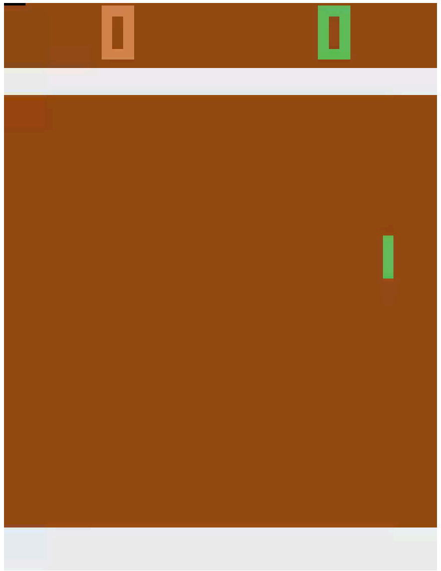


Generation 0 (fitness -21)

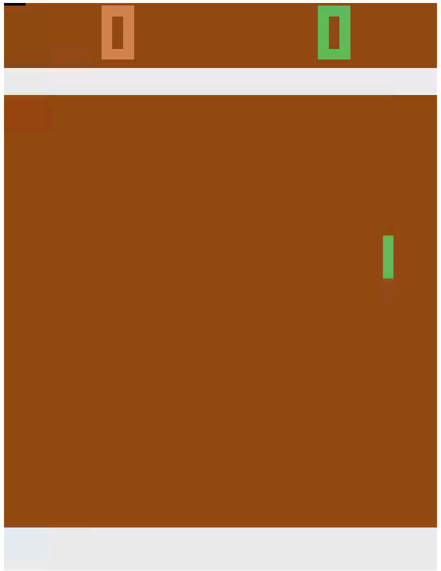
Experimental Results



Generation 25 (fitness 6)



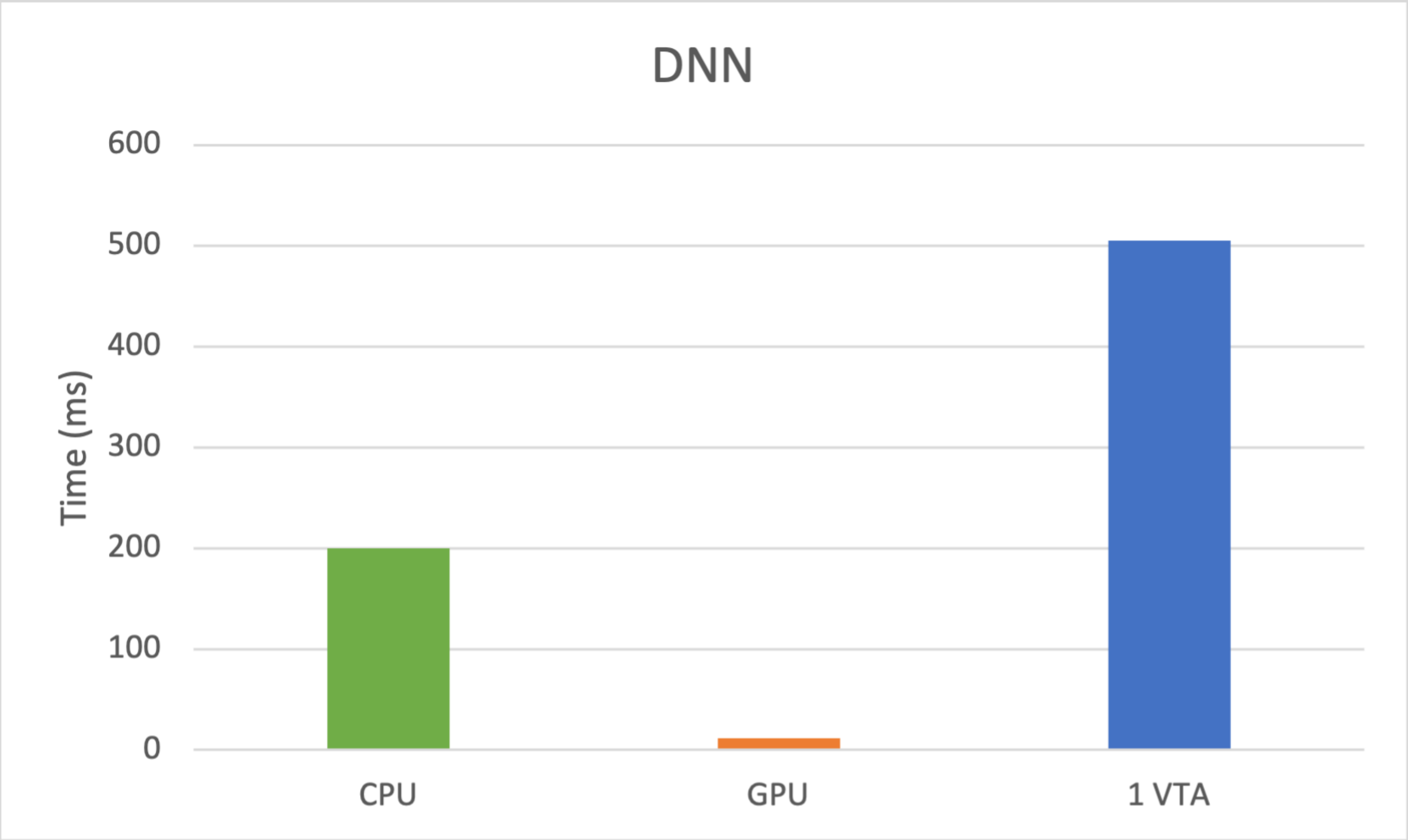
Generation 60 (fitness 13)



Generation 165 (fitness 20)

Experimental Results

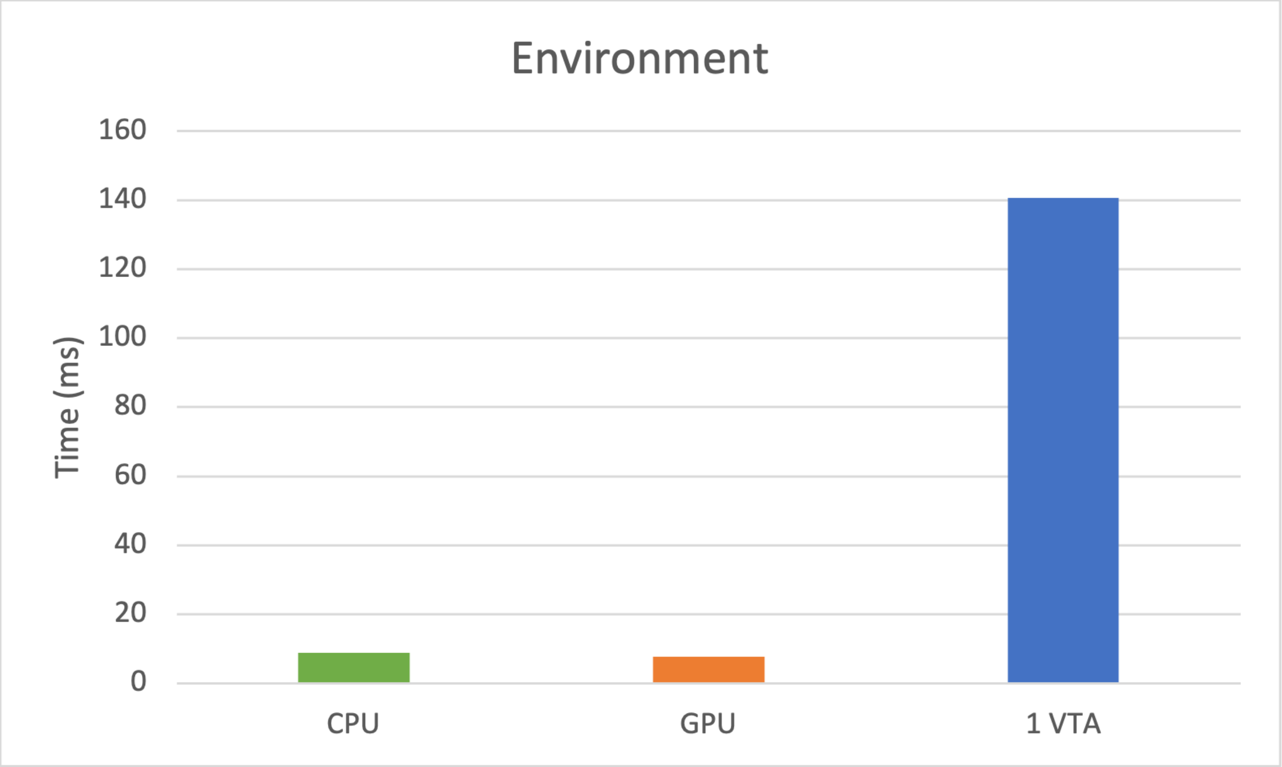
Neural network execution times



Note: the lowest the better

Experimental Results

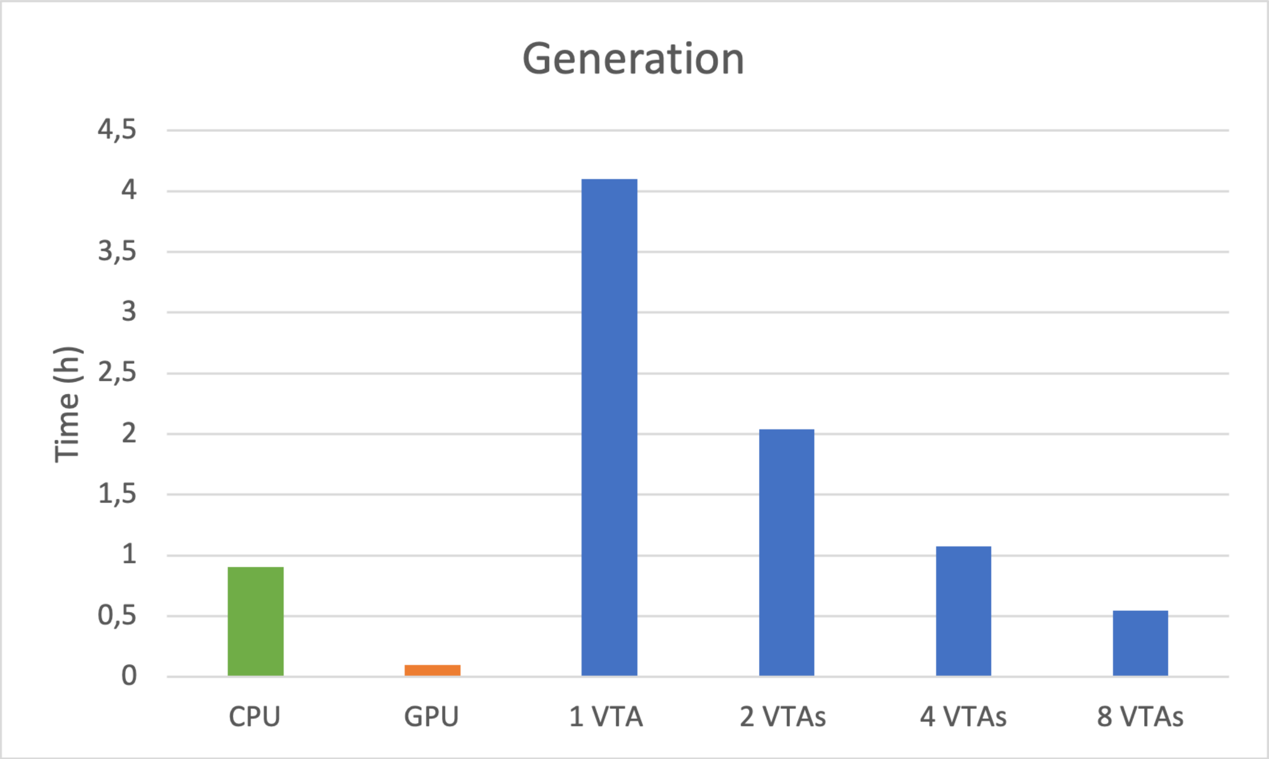
OpenAI Environment execution times



Note: the lowest the better

Experimental Results

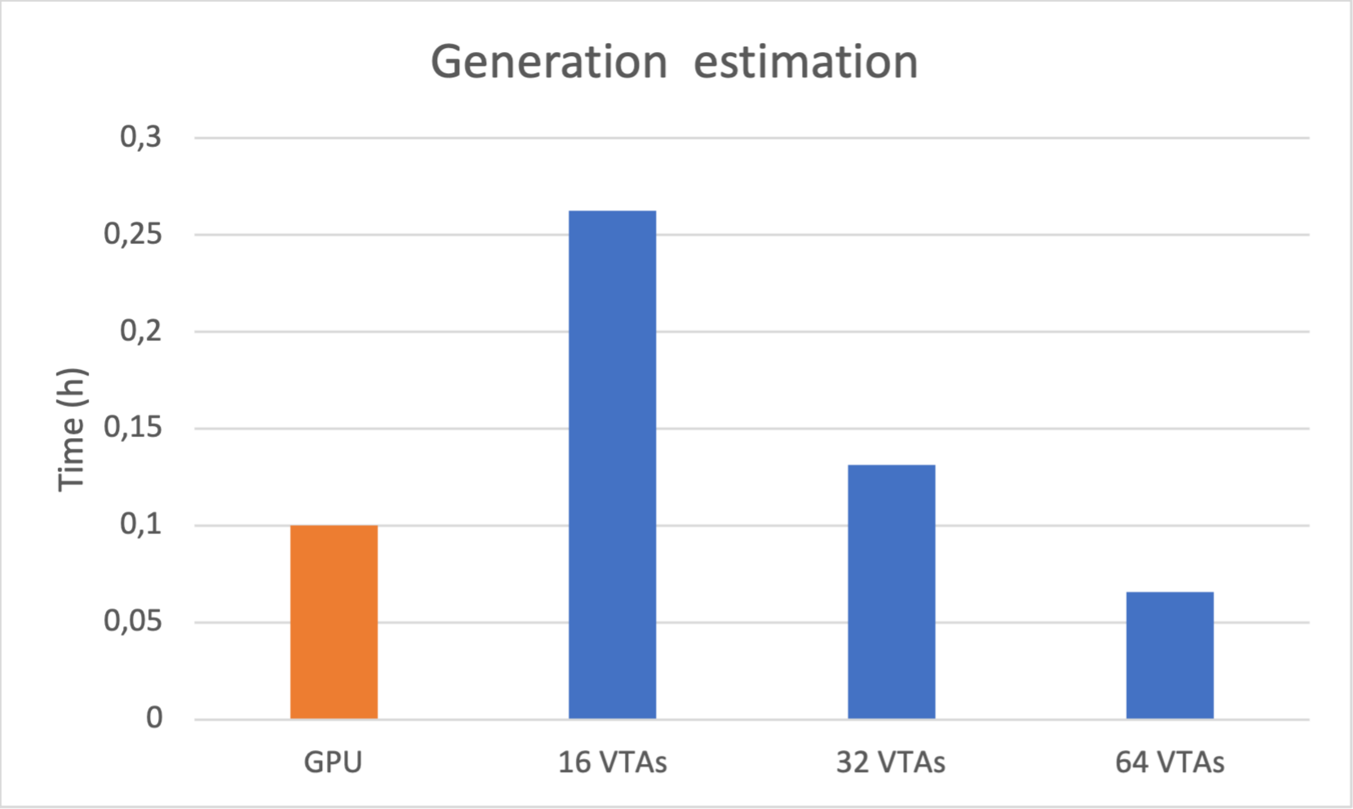
One generation execution times



Note: the lowest the better

Experimental Results

Multiple FPGAs estimated times



Note: the lowest the better

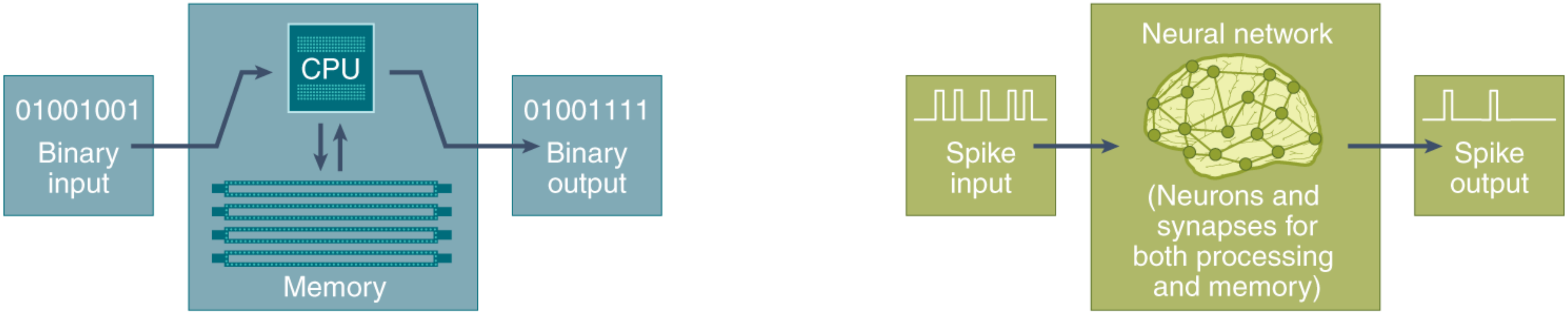
FROM NEUROEVOLUTION TO NEUROMORPHIC: EVOLUTIONARY SPIKING NEURAL NETWORKS

Index of the Presentation

1. Introduction to Self-* Computing Systems
2. Principles of Evolutionary Computation and Evolvable Hardware
3. Classic Evolvable Hardware: Application to Image Processing
4. Neuroevolution: Combining Evolutionary Computation and ANNs
5. Deep Neuroevolution: Evolutionary Computation + Deep Learning
6. **From Neuroevolution to Neuromorphic: Evolutionary Spiking Neural Networks**
7. The A-IQ Ready View: hybrid computing platforms

Neuromorphic Computing Systems

Neuromorphic computing systems are unconventional computing devices that draw inspiration from the human brain. These systems are constructed using artificial neurons and synapses. Within a neuromorphic computer, both data processing and memory storage are controlled by artificial neurons and synapses. Their functionality is determined by the architecture and parameters of the network, and they accept input in the form of spikes, with their timing, intensity, and waveform serving as encoding mechanisms for numerical information.



Von Neumann architecture	versus	Neuromorphic architecture
Sequential processing	Operation	Massively parallel processing
Separated computation and memory	Organization	Collocated processing and memory
Code as binary instructions	Programming	Spiking neural network
Binary data	Communication	Spikes
Synchronous (clock-driven)	Timing	Asynchronous (event-driven)

Benefits of Neuromorphic Computing

Highly parallel operation: Neuromorphic computers operate in a highly parallel manner, with all neurons and synapses potentially working simultaneously. However, their computations are simpler compared to traditional von Neumann systems.

Collocated processing and memory: Processing and memory are closely integrated. Neurons and synapses both handle processing and store values, reducing the processor/memory separation (bottleneck) seen in von Neumann systems, which can slow down performance and consume more energy.

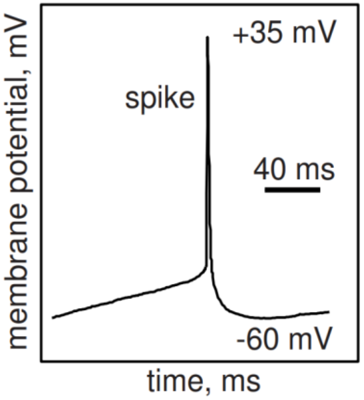
Inherent scalability: Neuromorphic computers are designed to be easily scalable by adding more chips, increasing the number of neurons and synapses. Multiple chips can be combined to create larger networks, as demonstrated in systems like *SpiNNaker* and *Loihi*.

Event-driven computation: Neuromorphic computers use event-driven computation, only performing calculations when data is available. This approach, along with temporally sparse activity, leads to highly efficient computation, as neurons and synapses only work when spikes occur.

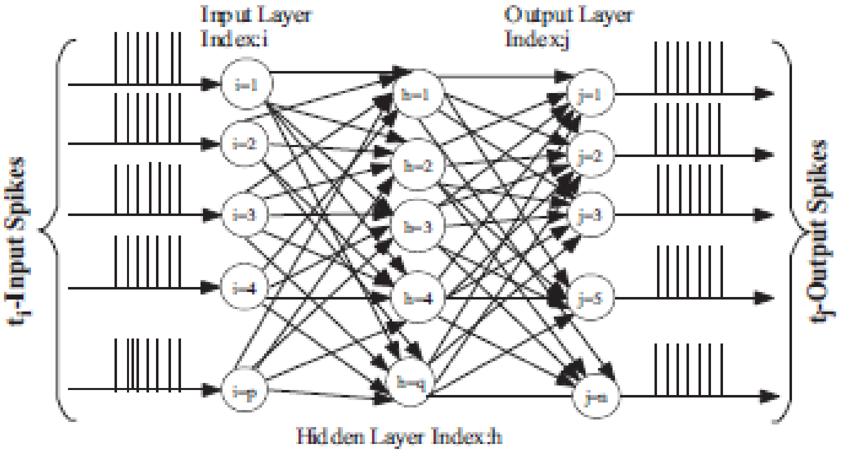
Stochasticity: Neuromorphic computers can incorporate randomness, such as in the firing of neurons, to introduce noise into their operations.

Spiking Neural Networks

- **Spiking Neural Networks (SNNs):**
  - Use specific spiking neuron models
  - Information is encoded in spike times
  - Use synapse strengths as the adjustable parameters of the network



Membrane potential over time in a spiking neuron

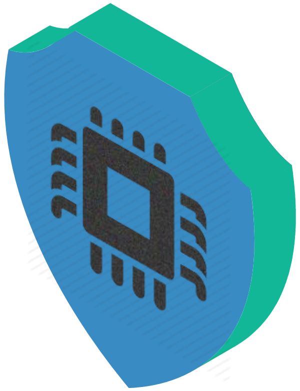


Architecture of a multilayer spiking neural network.

Spiking Neural Networks: Existing Implementations

ASIC implementations

- Multi core: SpiNNaker, TrueNorth, Loihi
- Single core: ODIN
- Low adaptability
- Great computation efficiency and resource allocation



FPGA implementations

- Higher adaptability
- More solutions proposed and research contribution

FPGA Implementations

Ref.	Year	Neuron model	Maximum Neuron number	Target FPGA	Data precision	Learning technique	Dataset	Accuracy
Asmer Hamid Ali et al.	2022	LIF		Xilinx Artix-7	8-bit fixed point	Gradient descent learning	MNIST	92.8 %
Thao N. N. Nguyen et al.	2022	LIF		Xilinx XC7Z020	24-bit fixed point	STDP-based learning algorithm	Caltech-101	95.7 %
Yijun Liu et al.	2022	LIF, IZH	16384	Xilinx XC7K325T	16-bit fixed-point	ANN-SNN weights conversion	MNIST	97,7 % (LIF) 97,81 % (IZH)
Jian Zhang et al.	2022	Custom weighted neuron model		Xilinx Virtex-7 VC707	Fixed-point	Back-Propagation STDP (BP-STDP)	MNIST	95,3 %
Zhen He et al.	2022	LIF		Xilinx ZC706	16-bit fixed-point	Reward-modulated STDP (R-STDP)	MNIST	93 %
Jianhui Han et al.	2020	LIF	16384	Xilinx ZC706	16-bit fixed-point		MNIST	97,06 %
Vasilis Sakellariou et al.	2021	LIF	4096	Xilinx ZYNQ Ultrascale+ MPSoC zcu104		ANN-SNN weights conversion	MNIST	
Jiajun Wu et al.	2021	LIF	6400	Xilinx xc7z035fbg676-2	Floating point / 16-bit fixed-point	STDP	MNIST	95 %
Minitaur	2014	LIF	65K	Xilinx Spartan-6 LX150	16-bit fixed-point		MNIST	92 %
Bluehive	2012	IZH	64K	Altera Stratix IV 230				
Felipe Sanchez et al.	2017	IZH	250K	Zynq 7100 device	Floating point / fixed-point	STDP, ReSuMe	Combinational tasks (XOR benchmark)	~100 %

CEI UPM

POLITÉCNICA

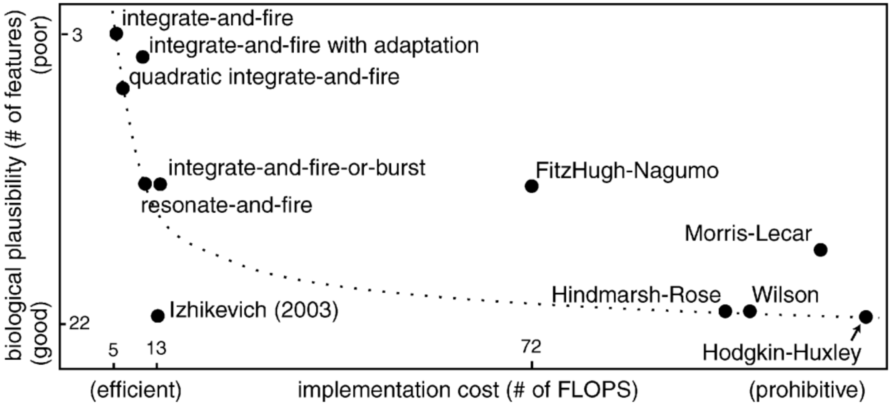
Neuron models

Conductance-based models

- Describe internal variables of the neuron
- Higher implementation cost
- Hodgkin-Huxley model, Morris-Lecar model

Spike models

- Based on modeling the generation of the spike
- Lower implementation cost
- Integrate and Fire (I&F) model, Izhikevich spiking model



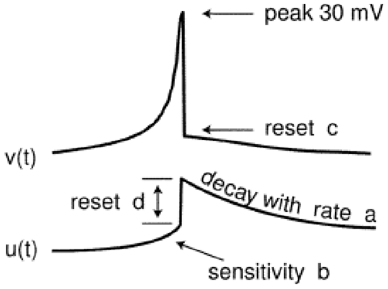
Neuron model	N of FLOPS (1ms)
Hodgkin-Huxley	1200
Morris-Lecar	600
I&F	5
Izhikevich	13

E. M. Izhikevich, "Which model to use for cortical spiking neurons?," in IEEE Transactions on Neural Networks, vol. 15, no. 5, pp. 1063- 1070, Sept. 2004, doi: 10.1109/TNN.2004.832719.

The Izhikevich spiking model

The **Izhikevich Spiking** model combines biological plausibility of HH model with computational efficiency of I&F model

$$v(t)' = 0.04v(t)^2 + 5v(t) + 140 - u(t) + I(t)$$
$$u(t)' = a[bv(t) - u(t)]$$
$$if\ v(t) \geq 30mV \begin{cases} v(t+1) = c \\ u(t+1) = u(t) + d \end{cases}$$

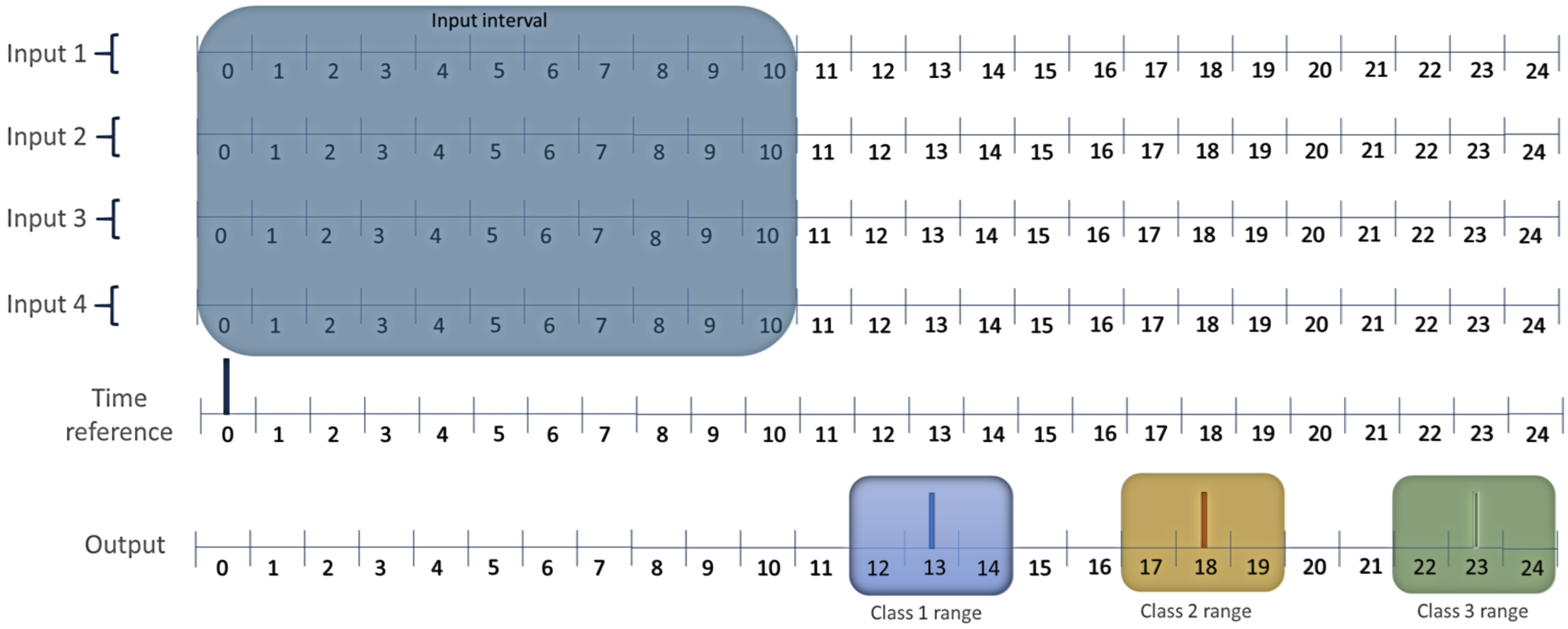


v -> membrane potential

u -> membrane recovery variable

E. M. Izhikevich, “Simple model of spiking neurons,” IEEE Trans. Neural Netw., vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

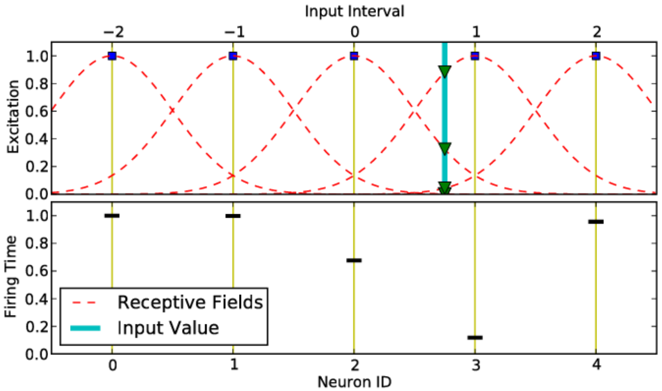
Data Encoding in SNNs



Problem: converting continuous input data into spike times

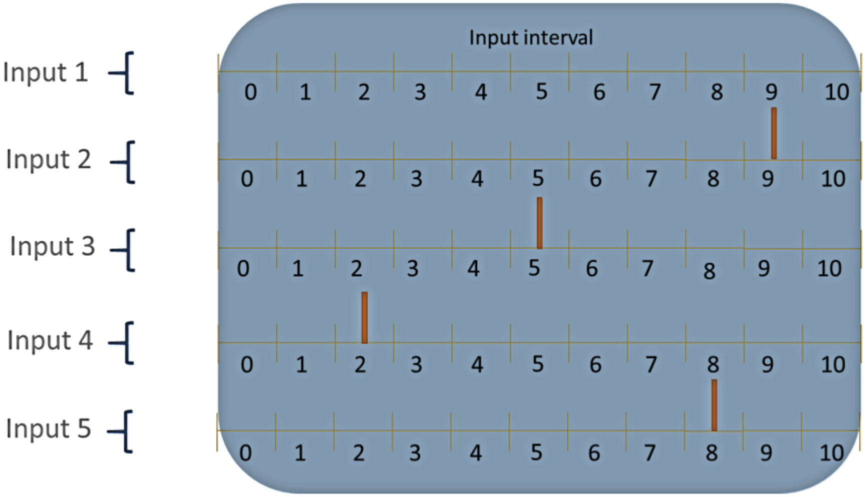
Data encoding in SNNs

Problem: converting continuous input data into spike times



Accuracy can be improved by sharpening the receptive fields or increasing the number of neurons

Solution: Gaussian activation functions with overlapping profiles



```
Higher values at the intersection points indicate stronger
excitation for that variable, and these are transformed
into lower delay times, rounded to the nearest discrete
time step
```

Learning techniques for SNNs

Unsupervised learning

- There is no prior information about the input data classification
- *Spike-timing-dependent plasticity (STDP)*

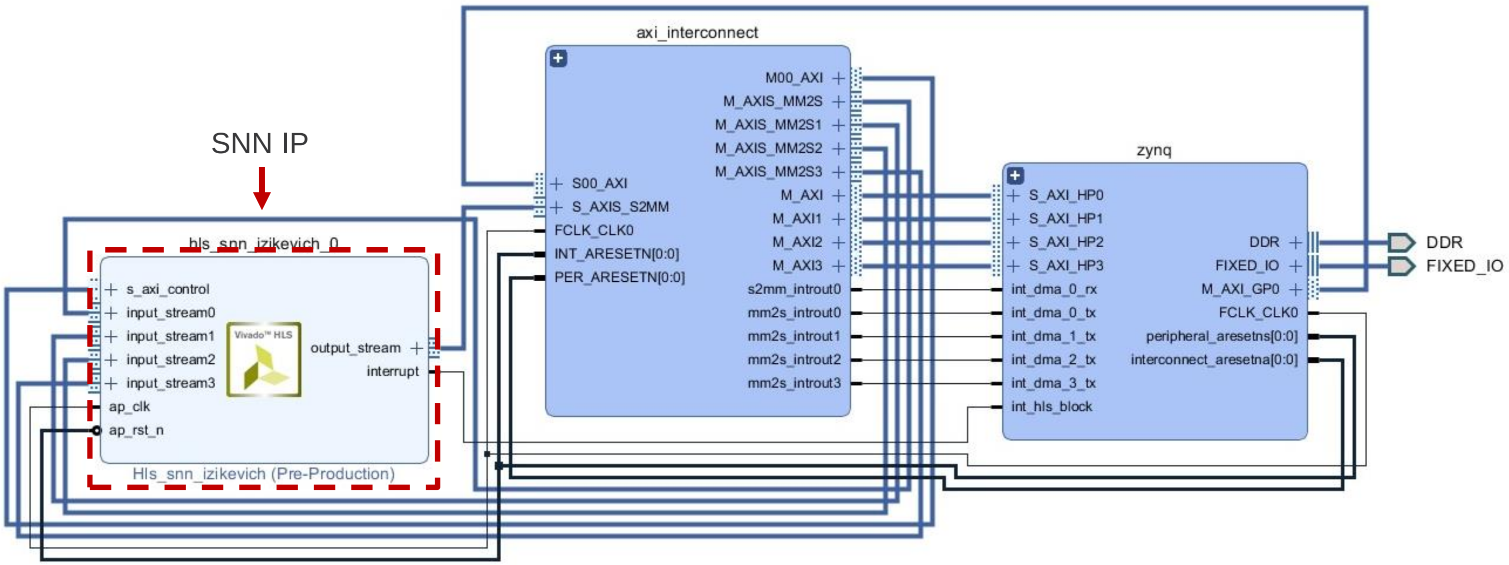
Supervised learning

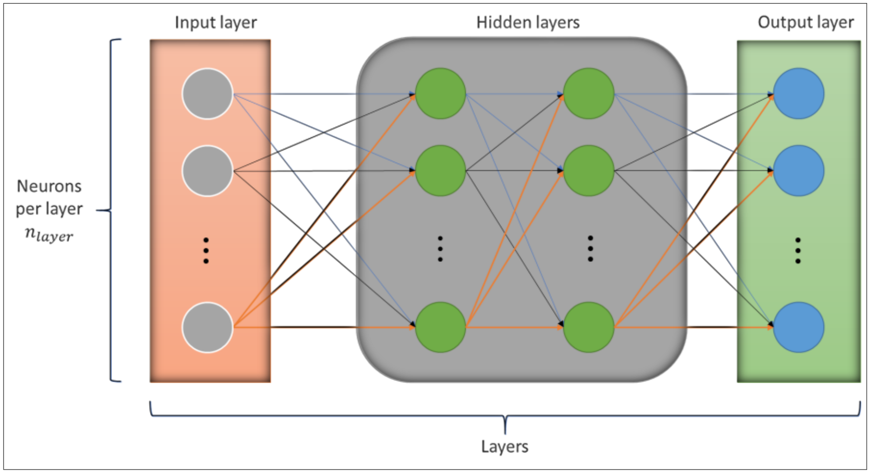
- Require predefined goals in the learning process
- *SpikeProp, ReSuMe, ESC*

Reinforcement learning

- The SNN takes an action based on the state observations of the environment, receiving a reward value in exchange
- Specially appropriated for genetic algorithms

Proposed System implementation for SNNs

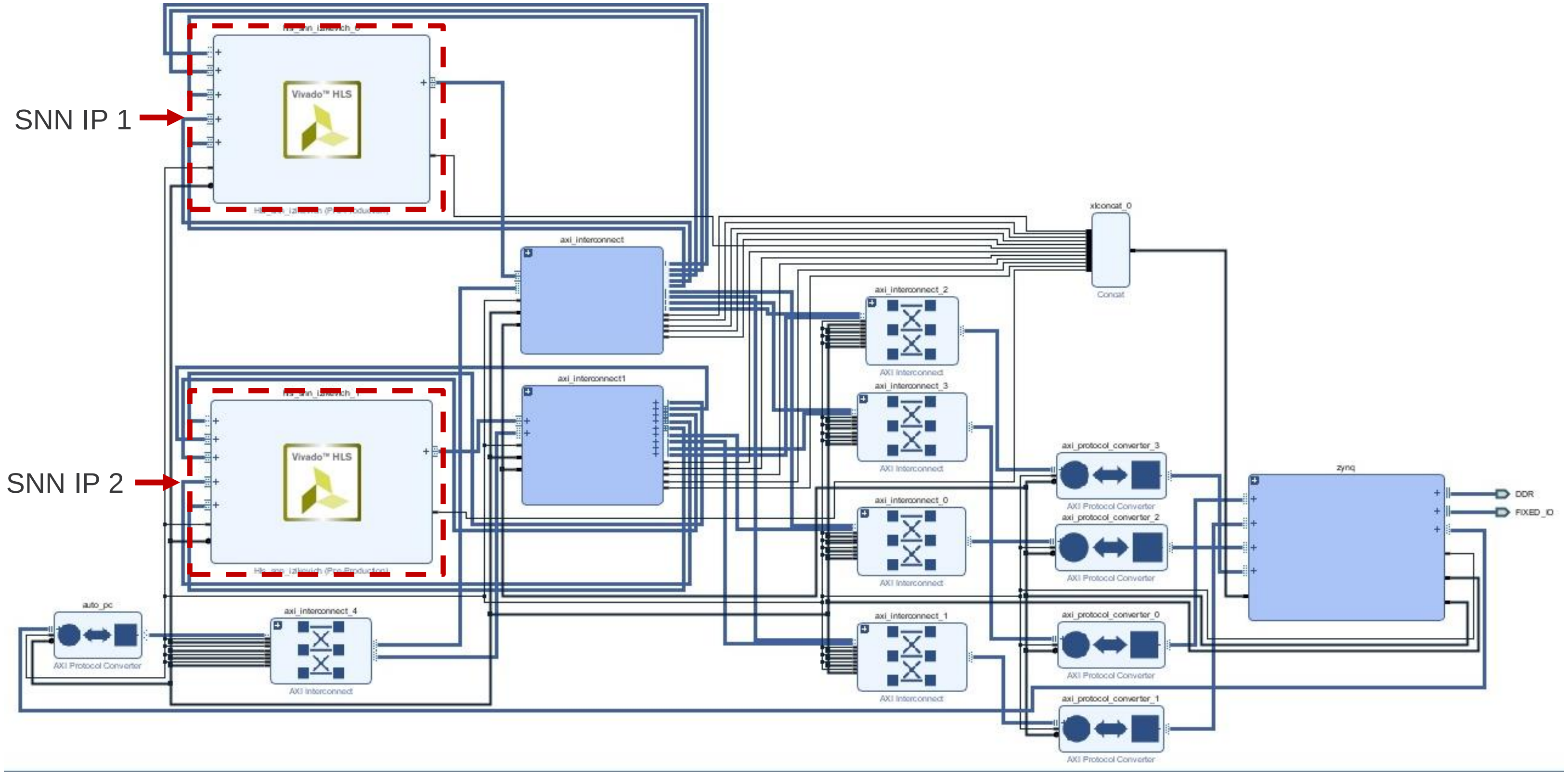




Sanchez, F. G., & Nunez-Yanez, J. (2017). Energy proportional streaming spiking neural network in a reconfigurable system. Microprocessors and Microsystems, 53, 57-67.

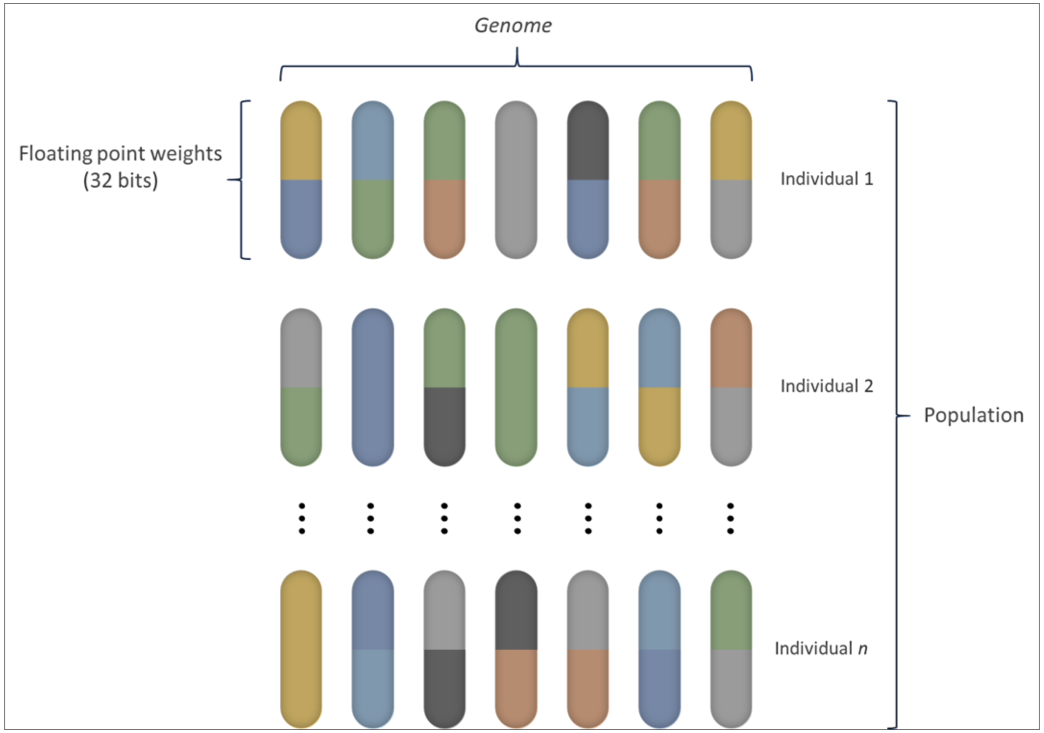
- Feed forward topology with L layers and n neurons per layer
- Targeting a Zynq 7020 device
- Adjustable data width
- Described using HLS

Concurrent solution implementation

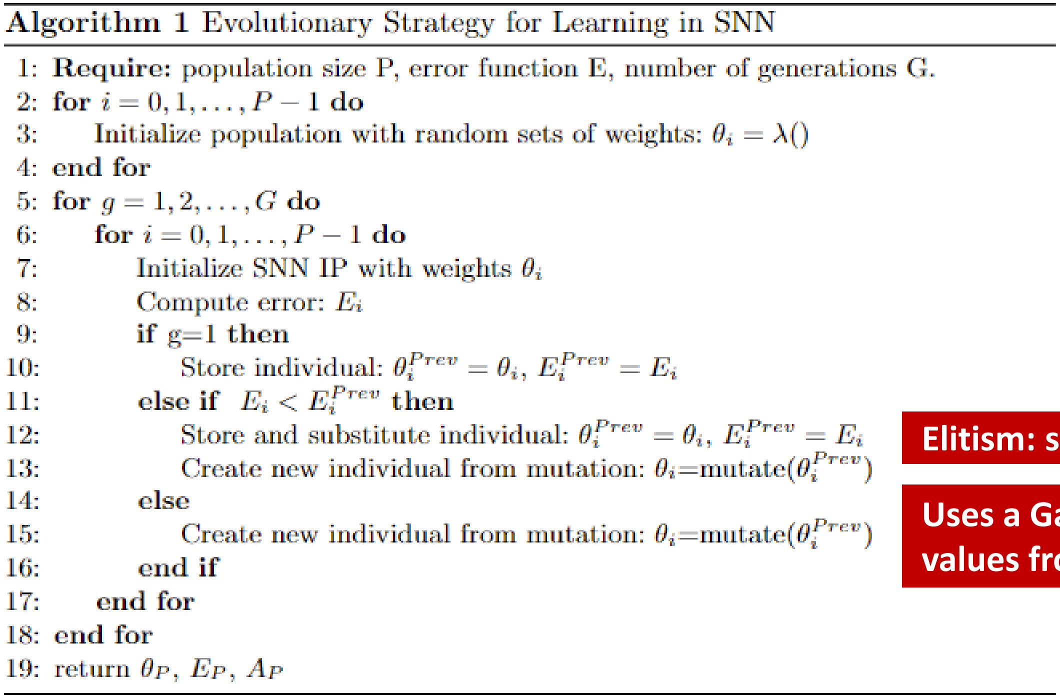


Evolutionary strategy proposed

- Adjustable parameters: network weights (synapse strengths)



Evolutionary strategy proposed

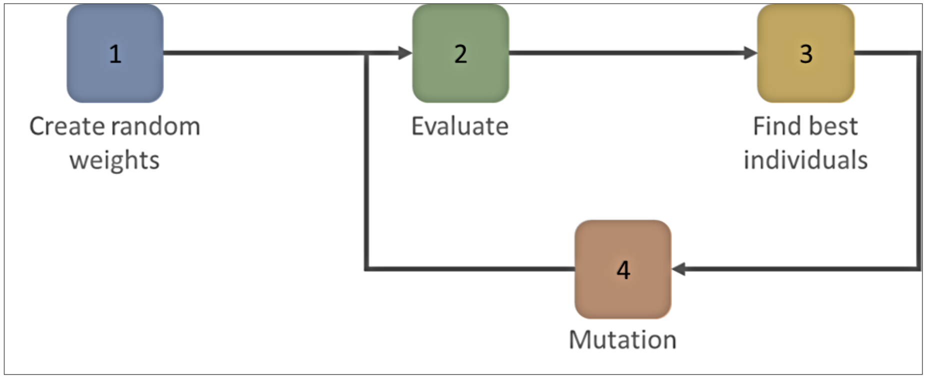


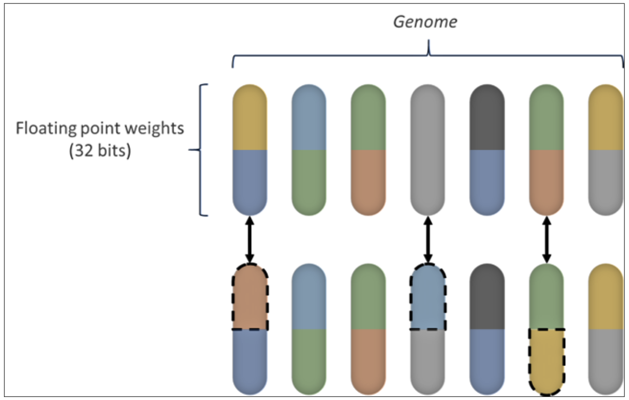
Algorithm 1 Evolutionary Strategy for Learning in SNN

```
Require: population size P, error function E, number of generations G.
for $i = 0, 1, \ldots, P-1$ do
    Initialize population with random sets of weights: $\theta_i = \lambda()$
end for
for $g = 1, 2, \ldots, G$ do
    for $i = 0, 1, \ldots, P-1$ do
        Initialize SNN IP with weights $\theta_i$
        Compute error: $E_i$
        if g=1 then
            Store individual: $\theta_i^{Prev} = \theta_i,\ E_i^{Prev} = E_i$
        else if $E_i < E_i^{Prev}$ then
            Store and substitute individual: $\theta_i^{Prev} = \theta_i,\ E_i^{Prev} = E_i$
            Create new individual from mutation: $\theta_i$=mutate($\theta_i^{Prev}$)
        else
            Create new individual from mutation: $\theta_i$=mutate($\theta_i^{Prev}$)
        end if
    end for
end for
return $\theta_P$, $E_P$, $A_P$
```

Elitism: selection of the best individuals

Uses a Gaussian distribution to generate random values from the previous ones.

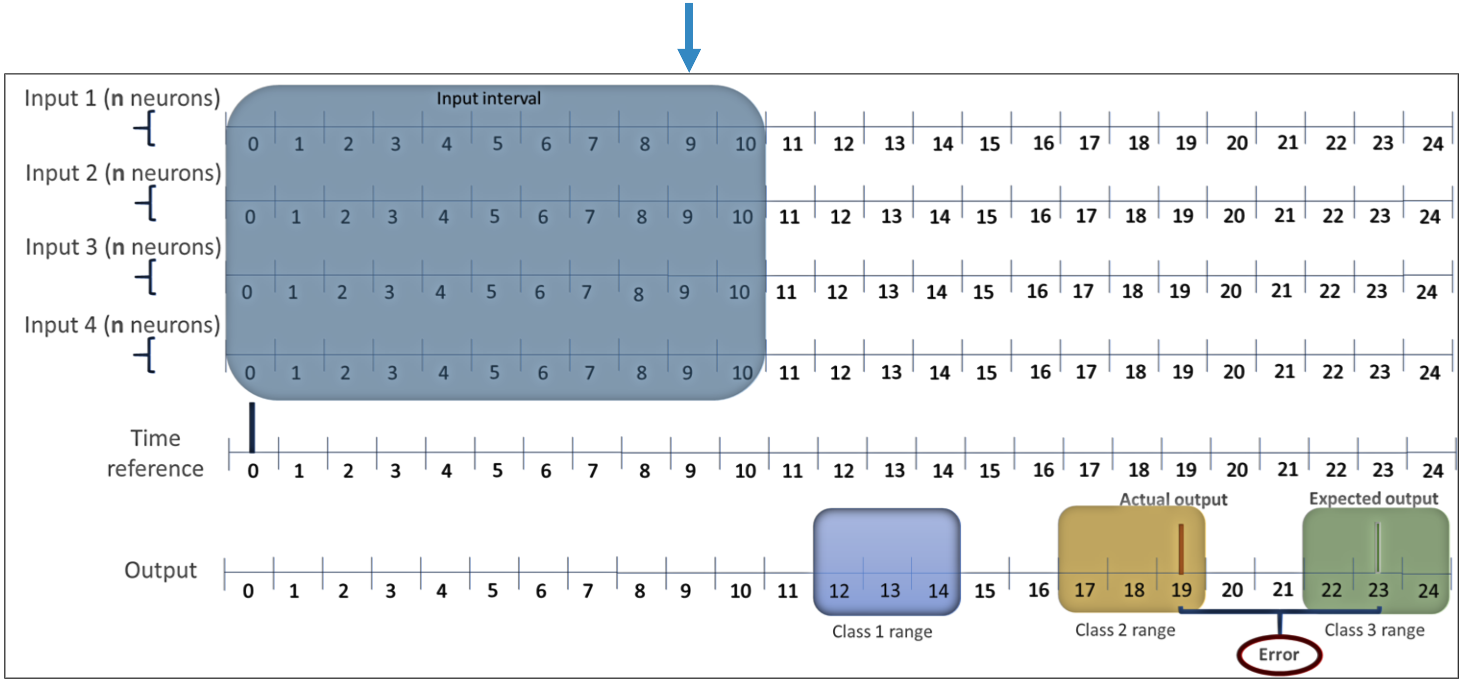




Evolutionary strategy proposed

Classification: accumulated error over the entire dataset

Reinforcement learning: use of a specific fitness function



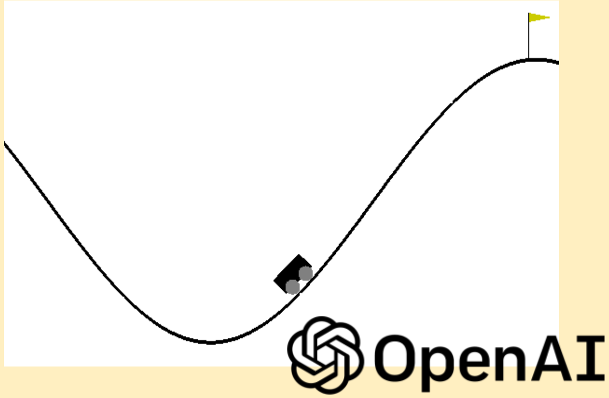
Use cases to Demonstrate the Solution

Supervised learning (Classification)

- Iris dataset
- Breast Cancer Wisconsin dataset
- Pima Indian Diabetes dataset
- Wine dataset

Reinforcement learning

- OpenAI Gymnasium Mountain car environment



Supervised learning: UCI Datasets

Iris

Parameter	Value
Training trials	105 (70%)
Testing trials	45 (30%)
Hidden layers	1
Neurons per layer	17
Hidden neurons	17
Input neurons	17
Output neurons	1
Total neurons	35
Population	20
Trial time (ms)	100

Pima Indian Diabetes

Parameter	Value
Training trials	385 (50.13%)
Testing trials	383 (49.87%)
Hidden layers	1
Neurons per layer	33
Hidden neurons	33
Input neurons	33
Output neurons	1
Total neurons	67
Population	20
Trial time (ms)	100

Breast Cancer Wisconsin

Parameter	Value
Training trials	350 (50.07%)
Testing trials	349 (49.93%)
Hidden layers	1
Neurons per layer	37
Hidden neurons	37
Input neurons	37
Output neurons	1
Total neurons	75
Population	20
Trial time (ms)	100

Wine

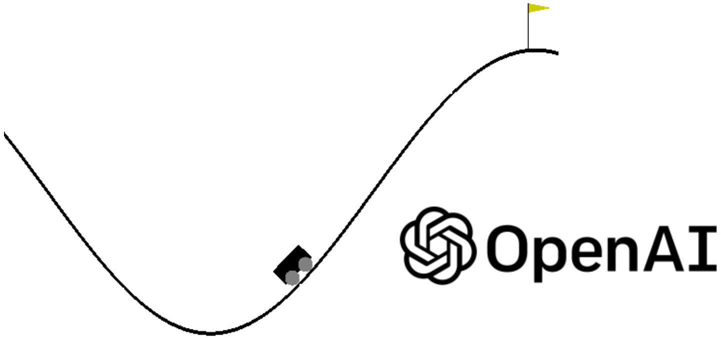
Parameter	Value
Training trials	125 (70.22%)
Testing trials	53 (29.78%)
Hidden layers	1
Neurons per layer	53
Hidden neurons	14
Input neurons	53
Output neurons	1
Total neurons	68
Population	20
Trial time (ms)	100

- Predefined number of trials. Fixed execution duration for every iteration
- Datasets are divided into a training and a testing set.
- Sets are randomly selected, keeping the proportion of classes from the original dataset
- Existence of a target output at every trial

CEI UPM

POLITÉCNICA

Use cases. Reinforcement learning



Mountain car environment

Parameter	Value
Hidden layers	1
Neurons per layer	17
Hidden neurons	17
Input neurons	17
Output neurons	1
Total neurons	35
Population	5
Trial time (ms)	100

Observation Space

Array position	Attribute	Min	Max	Unit
0	position of the car (x-axis)	-1.2 (changed to -0.9)	0.6 (changed to 0.9)	position (m)
1	velocity of the car	-0.07	0.07	velocity (v)

Action Space

Value	Action
0	Accelerate to the left
1	Don’t accelerate
2	Accelerate to the right

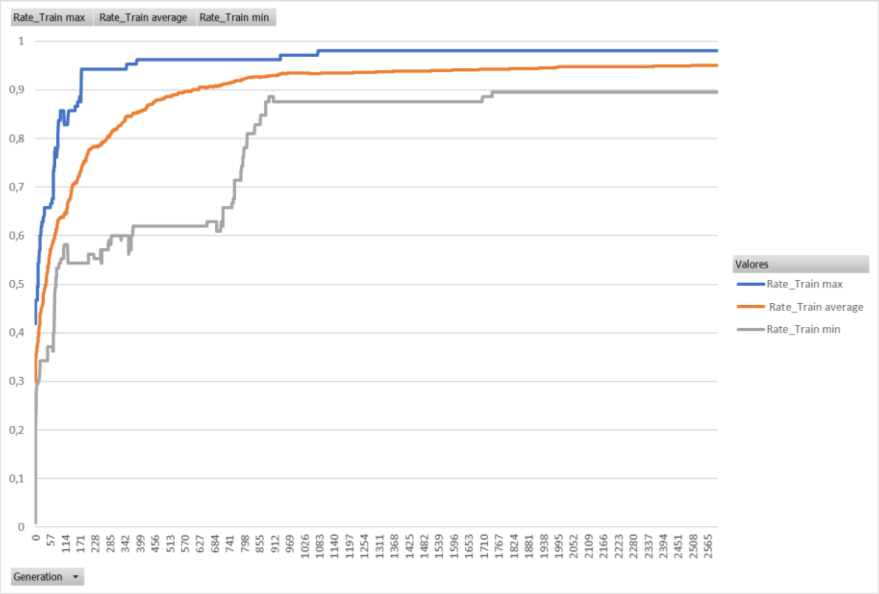
UCI datasets. Results

Resource utilization

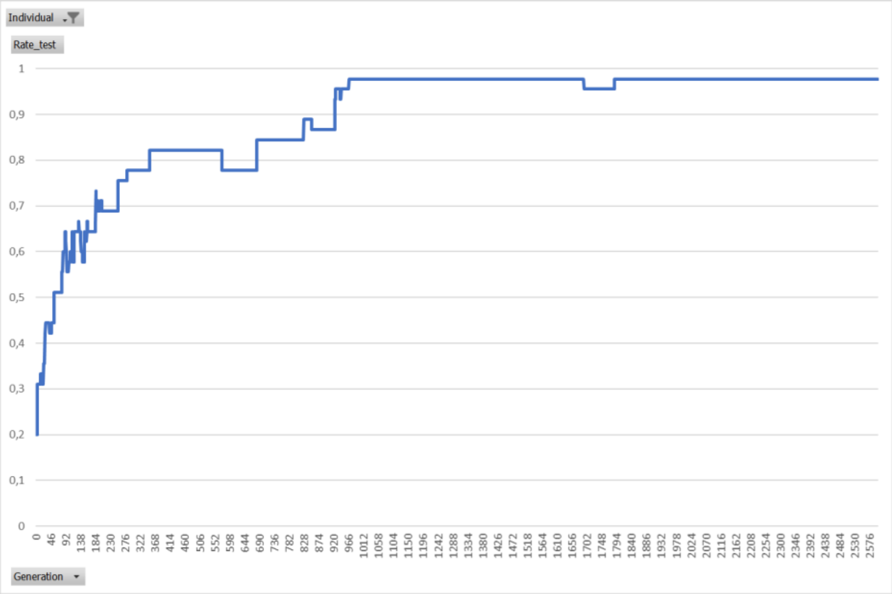
Implementation	Synthesized neurons	LUTs	FFs	Block RAM Tiles	DSPs
Iris (single-IP)	35	13236 (25%)	19608 (18%)	26 (19%)	34 (15%)
Iris (dual-IP)	35 (x2)	26884 (51%)	39441 (37%)	52 (37%)	68 (31%)
Breast Cancer Wisconsin (single-IP)	75	18419 (35%)	27827 (26%)	32 (23%)	49 (22%)
Breast Cancer Wisconsin (dual-IP)	75 (x2)	37179 (70%)	55875 (53%)	64 (46%)	98 (45%)
Pima Indian Diabetes (single-IP)	67	17638 (33%)	26087 (25%)	32 (23%)	44 (20%)
Pima Indian Diabetes (dual-IP)	67 (x2)	35587 (67%)	52401 (49%)	64 (46%)	88 (40%)
Wine (single-IP)	107	22946 (43%)	30610 (29%)	33 (24%)	65 (30%)
Wine (dual-IP) - Estimated	107 (x2)	52124 (98%)	63182 (59%)	66 (47%)	130 (59%)

UCI datasets. Results

Accuracy values



Training accuracy results. Iris dataset



Testing accuracy results. Iris dataset. Best individual

UCI datasets. Results

Accuracy values

Dataset	Training rate (maximum)	Testing rate (selected individual)	Generations until convergence
Iris	98.1%	97.778%	~1000
Breast Cancer Wisconsin	96.857%	95.702%	~100
Pima Indian Diabetes	75.325%	72.063%	~150
Wine	93.600%	88.679%	~2800

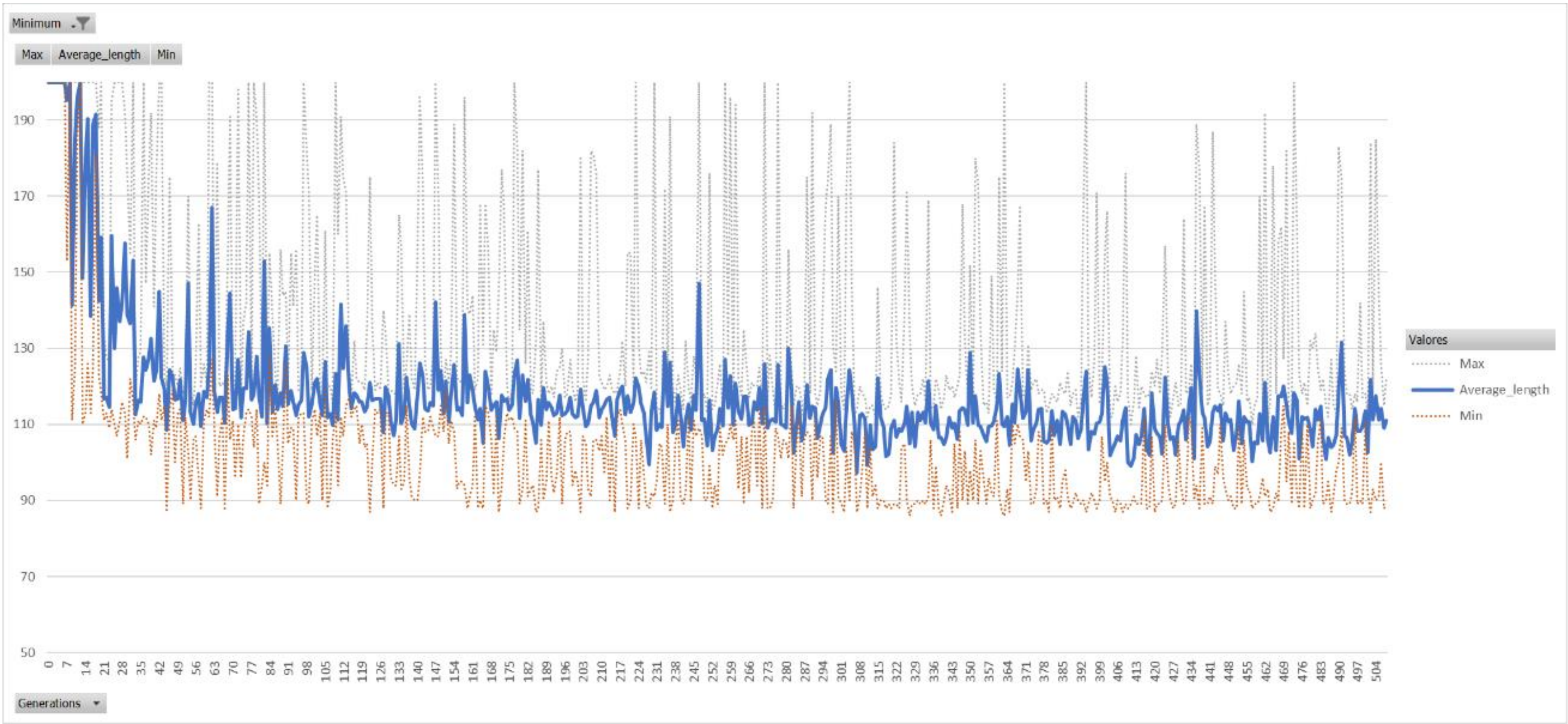
Time results. Seconds for a complete generation (20 individuals):

Dataset	Single-IP	Concurrent solution	Vitis HLS C simulation
Iris	5.013 s	2.752 s	121.262 s
Breast Cancer Wisconsin	25.792 s	12.998 s	1767.579 s
Pima Indian Diabetes	27.214 s	13.841 s	1689.555 s

Results Comparison for the UCI Datasets

Dataset	Algorithm	Neuron type	Hardware implementation	Testing Accuracy (%)
Iris	This work	Izhikevich	Yes	97.78%
	DoB-SNN	LIF	No	97.75%
	SRESN	LIF	No	97.01%
	SpikeProp	LIF	No	96.13%
	SWAT	LIF	No	93.88%
	NIDA	I&F	No	99.30%
	DANNA	I&F	Yes	99.30%
	CS	Izhikevich	No	94.67%
	DE	Izhikevich	No	98.33%
Breast Cancer Wisconsin	This work	Izhikevich	Yes	95.70%
	DoB-SNN	LIF	No	97.35%
	SRESN	LIF	No	97.10%
	SpikeProp	LIF	No	97.04%
	SWAT	LIF	No	95.66%
	NIDA	I&F	No	98.60%
	DANNA	I&F	Yes	98.10%
Pima Indian Diabetes	This work	Izhikevich	Yes	72.06%
	DoB-SNN	LIF	No	76.57%
	SRESN	LIF	No	70.06%
	SpikeProp	LIF	No	77.38%
	SWAT	LIF	No	72.11%
	NIDA	I&F	No	81.00%
	DANNA	I&F	Yes	78.00%
	CS	Izhikevich	No	74.77%
	DE	Izhikevich	No	73.71%
Wine	This work	Izhikevich	Yes	88.679%
	NIDA	I&F	No	99,4%
	DANNA	I&F	Yes	97.2%
	CS	Izhikevich	No	90.78%
	DE	Izhikevich	No	87.44%

Mountain car. Results



Maximum, average and minimum episode duration. Mountain car environment

Mountain car. Results

Best individuals obtained

Generation number	Average episode duration (time steps)	Maximum episode duration (time steps)	Minimum episode duration (time steps)	Standard deviation (time steps)
307	97.3	114	87	11,451
411	99.1	110	89	8,465
228	99.4	129	88	13,922

Time results

Execution mode	Average time per episode (s)
Zynq execution	6705,173
Random internal decision	6707,503

Results comparison for the Mountain car

Algorithm	Average episode duration (time steps)
This work	97.3
Orthogonal decision trees	101.72
Oblique decisión tres	106.02
Closed-form policy	102.61

Future work

IP adaptability

Breaking current dependencies and constrains

Fault-tolerance tests

Using ES and the Izhikevich neuron model

ES adaptations

Improving the ES used with crossover or more complex selection and mutation algorithms

Concurrency

Using several Zynq devices to enable faster computing

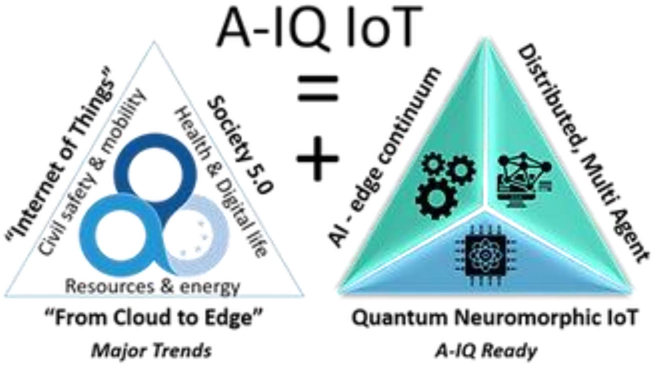
THE A-IQ READY VIEW: HYBRID COMPUTING PLATFORMS

Index of the Presentation

1. Introduction to Self-* Computing Systems
2. Principles of Evolutionary Computation and Evolvable Hardware
3. Classic Evolvable Hardware: Application to Image Processing
4. Neuroevolution: Combining Evolutionary Computation and ANNs
5. Deep Neuroevolution: Evolutionary Computation + Deep Learning
6. From Neuroevolution to Neuromorphic: Evolutionary Spiking Neural Networks
7. **The A-IQ Ready View: hybrid computing platforms**

The A-IQ READY project

- A-IQ Ready EU project addresses two major Trends: “*Internet of Things (IoT) with 1 Billion $ to 1 Trillion $ revenues*” and “*From Cloud to Edge*”.



- A-IQ Ready will apply three backbone for the Society 5.0 disruptive technologies: Quantum Sensor, Neuromorphic Acceleration, AI in Multi-Agent Systems to build the edge continuum as the digital

A-IQ READY receives funding within the Key Digital Technologies Joint Undertaking (KDT JU) - the Public-Private Partnership for research, development and innovation under Horizon Europe – and National Authorities under grant agreement n° 101096658.

The A-IQ READY project

Multi-physics (quantum) sensors

Increase sensing accuracy, reliability and trustworthiness in complex environments with new multi-physics (quantum) sensors

Safe, resilient and prosperous digital society

Increase Europe's competitivity in developing a safe, resilient and prosperous digital society

Open AI Edge Continuum platform

Provide a reference open AI Edge Continuum platform integrating quantum sensors, neuromorphic (cognitive) computing and AI algorithms at the edge

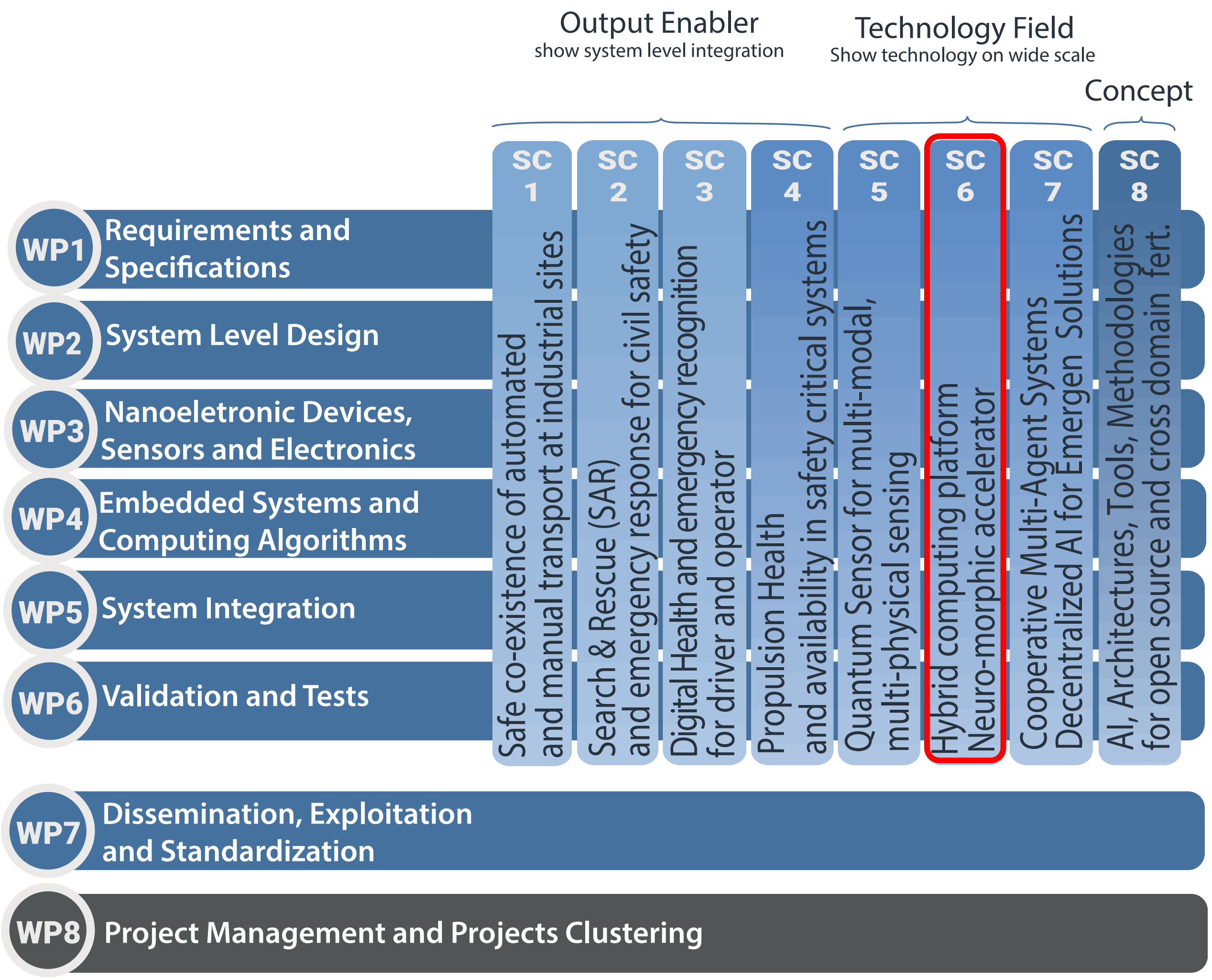
Digital society

Demonstrate the approach to build the digital society

AI methods

Provide AI methods for multi-agent autonomy in uncertain environments

A-IQ READY Project organization



Output Enabler
show system level integration

Technology Field
Show technology on wide scale

Concept

SC 1 Safe co-existence of automated and manual transport at industrial sites

SC 2 Search & Rescue (SAR) and emergency response for civil safety

SC 3 Digital Health and emergency recognition for driver and operator

SC 4 Propulsion Health and availability in safety critical systems

SC 5 Quantum Sensor for multi-modal, multi-physical sensing

SC 6 Hybrid computing platform Neuro-morphic accelerator

SC 7 Cooperative Multi-Agent Systems Decentralized AI for Emergen Solutions

SC 8 AI, Architectures, Tools, Methodologies for open source and cross domain fert.

WP1 Requirements and Specifications

WP2 System Level Design

WP3 Nanoeletronic Devices, Sensors and Electronics

WP4 Embedded Systems and Computing Algorithms

WP5 System Integration

WP6 Validation and Tests

WP7 Dissemination, Exploitation and Standardization

WP8 Project Management and Projects Clustering

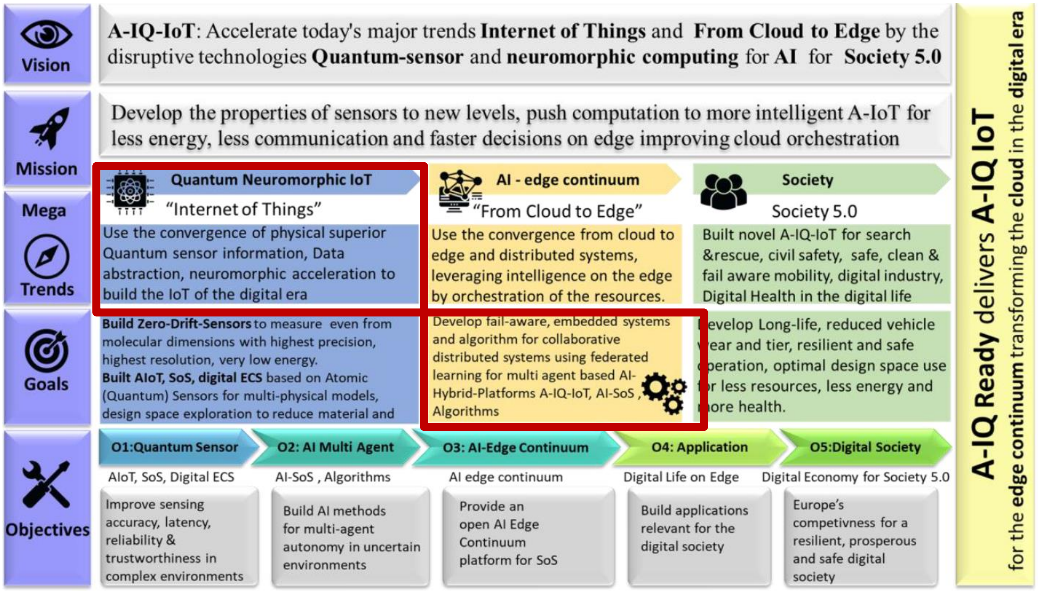
CEI UPM

POLITÉCNICA

A-IQ READY SC 6: Motivation

Hybrid Computing (Quantum Computing & High-Performance Computing)

- Federating intelligence across the value chain and current trends towards edge deployments tend to rely on a **multitude of accelerators**, often deployed within **specialized platform architectures** tied to a single vendor.
- This **fragmentation is detrimental to the performance and efficiency targets** since bridging infrastructure, protocol elements and middleware/software components inevitably add tremendous overheads to any application function that spans multiple platforms.
- Emerging acceleration technologies based on conventional digital computing, novel analogue-mixed signal neuromorphic computing, and high-performance data paths, and deploy these within a standardized system-level architecture with a **fundamentally new software-middleware stack** that enables their integration at any stage in the value chain.



Vision

A-IQ-IoT: Accelerate today's major trends **Internet of Things** and **From Cloud to Edge** by the disruptive technologies **Quantum-sensor** and **neuromorphic computing** for **AI** for **Society 5.0**

Mission

Develop the properties of sensors to new levels, push computation to more intelligent A-IoT for less energy, less communication and faster decisions on edge improving cloud orchestration

Mega Trends

Quantum Neuromorphic IoT

"Internet of Things"

Use the convergence of physical superior Quantum sensor information, Data abstraction, neuromorphic acceleration to build the IoT of the digital era

AI - edge continuum

"From Cloud to Edge"

Use the convergence from cloud to edge and distributed systems, leveraging intelligence on the edge by orchestration of the resources.

Society

Society 5.0

Built novel A-IQ-IoT for search &rescue, civil safety, safe, clean & fail aware mobility, digital industry, Digital Health in the digital life

Goals

Build Zero-Drift-Sensors to measure even from molecular dimensions with highest precision, highest resolution, very low energy. **Built AIoT, SoS, digital ECS** based on Atomic (Quantum) Sensors for multi-physical models, design space exploration to reduce material and

Develop fail-aware, embedded systems and algorithm for collaborative distributed systems using federated learning for multi agent based AI-Hybrid-Platforms A-IQ-IoT, AI-SoS , Algorithms

evelop Long-life, reduced vehicle ear and tier, resilient and safe peration, optimal design space use r less resources, less energy and ore health.

Objectives

O1:Quantum Sensor

AIoT, SoS, Digital ECS

Improve sensing accuracy, latency, reliability & trustworthiness in complex environments

O2: AI Multi Agent

AI-SoS , Algorithms

Build AI methods for multi-agent autonomy in uncertain environments

O3: AI-Edge Continuum

AI edge continuum

Provide an open AI Edge Continuum platform for SoS

O4: Application

Digital Life on Edge

Build applications relevant for the digital society

O5:Digital Society

Digital Economy for Society 5.0

Europe's competivness for a resilient, prosperous and safe digital society

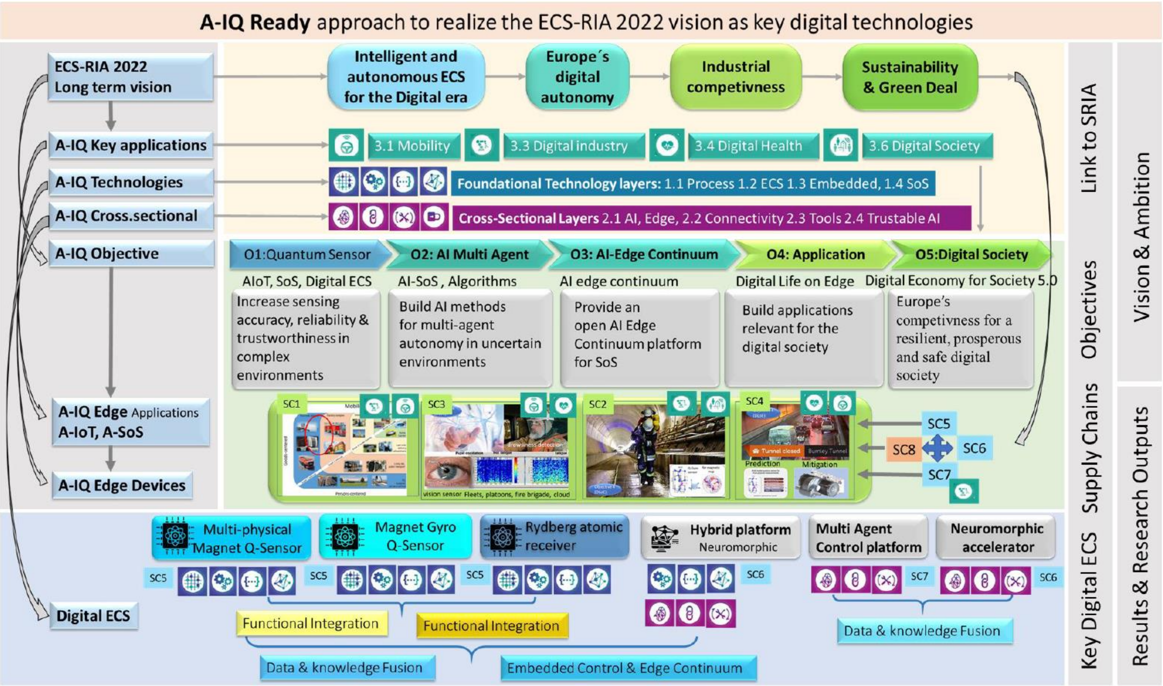
A-IQ Ready delivers **A-IQ IoT** for the **edge continuum** transforming the **cloud** in the **digital era**

A-IQ READY SC 6: Vision

Hybrid Computing (Quantum Computing & High-Performance Computing)

- *Provide* a hybrid computing platform
- *Enable* the seamless deployment of AI algorithms at different stages of the sensor-cloud value chain
- *Ensure* the appropriate power-latency-cost-functionality envelope.

- Delivering optimal power-performance for workloads at each stage of the value chain
- Reducing key metrics of time to insight (especially in real-time contexts) and energy per insight
- Enabling the delivery of AI workloads that span multiple stages of the value chain
- Ensuring security and resilience of data within the compute infrastructure



A-IQ Ready approach to realize the ECS-RIA 2022 vision as key digital technologies

ECS-RIA 2022 Long term vision

Intelligent and autonomous ECS for the Digital era → Europe´s digital autonomy → Industrial competivness → Sustainability & Green Deal

A-IQ Key applications → 3.1 Mobility · 3.3 Digital industry · 3.4 Digital Health · 3.6 Digital Society

A-IQ Technologies → **Foundational Technology layers:** 1.1 Process 1.2 ECS 1.3 Embedded, 1.4 SoS

A-IQ Cross.sectional → **Cross-Sectional Layers** 2.1 AI, Edge, 2.2 Connectivity 2.3 Tools 2.4 Trustable AI

A-IQ Objective

O1:Quantum Sensor	O2: AI Multi Agent	O3: AI-Edge Continuum	O4: Application	O5:Digital Society
AIoT, SoS, Digital ECS	AI-SoS , Algorithms	AI edge continuum	Digital Life on Edge	Digital Economy for Society 5.0
Increase sensing accuracy, reliability & trustworthiness in complex environments	Build AI methods for multi-agent autonomy in uncertain environments	Provide an open AI Edge Continuum platform for SoS	Build applications relevant for the digital society	Europe's competivness for a resilient, prosperous and safe digital society

A-IQ Edge Applications A-IoT, A-SoS

A-IQ Edge Devices

SC1 · SC3 · SC2 · SC4

vision sensor Fleets, platoons, fire brigade, cloud

Tunnel closed · Burnley Tunnel · Prediction · Mitigation

SC5 · SC8 · SC6 · SC7

Digital ECS

Multi-physical Magnet Q-Sensor (SC5) · Magnet Gyro Q-Sensor (SC5) · Rydberg atomic receiver (SC5) · Hybrid platform Neuromorphic (SC6) · Multi Agent Control platform (SC7) · Neuromorphic accelerator (SC6)

Functional Integration · Functional Integration

Data & knowledge Fusion · Embedded Control & Edge Continuum

Data & knowledge Fusion

Link to SRIA · Objectives · Supply Chains · Key Digital ECS

Vision & Ambition · Results & Research Outputs

CEI UPM

UNIVERSIDAD POLITÉCNICA DE MADRID

POLITÉCNICA

A-IQ READY SC 6: Overview

Hybrid Computing (Quantum Computing & High-Performance Computing)

- SC6 is considered as **technology provider**, where fundamental technologies are developed as input for the output enabler SCs.

The **key mission** is to

- integrate emerging non-von Neumann computing technologies & traditional neural network acceleration within a singular programmable platform.

In doing so SC6

- will **realize an acceleration platform** whose operational envelope can be tailored to the varying requirements of device types **across the sensor-cloud value chain**.

Technological Readyness Level

- Start: TRL 2
- End: TRL 4

SC6 Partners: TTTechAuto, IObundle, Universidad Politécnica de Madrid, Universität zu Lübeck, nvision, innatera, insar.sk, synopsys

A-IQ READY SC 6: Development – three primary focus areas

- Hybrid computing platform
  - focus on integrating heterogeneous processing technologies within a single platform architecture
  - focus on a multi-grain reconfigurable overlay architecture, integrated in a RISC-V processor pipeline
  - Communication and interfacing IP to ensure high-performance on throughput
- Middleware
  - Firmware stack to manage mixed-criticality execution along the value chain
  - Middleware for data relay between high-performance computing elements
  - Software stack to orchestrate workloads between conventional compute and ultra-low power always-on accelerators
- Software Tooling
  - SDK and interfacing APIs with standard machine learning frameworks to ensure portability of the developed models
  - develop an accompanying toolchain based on the MLIR representation to support the automatic compilation and mapping of the AI models

A-IQ READY 6: Results

Hybrid Computing (Quantum Computing & High-Performance Computing)

- Integrated platform architecture combining high-performance *datapath* and analogue-mixed signal neuromorphic processors for AI at multiple stages of the sensor-cloud value chain.
- Middleware to enable seamless model deployment across heterogeneous compute resources.
- SDK and requisite software stack for enabling the usability of the developed platforms to execute existing and emerging models.
- Multi-grain reconfigurable overlay architecture, integrated in a RISC-V processor
- Toolchain based on the MLIR
- System-on-Chip Platform, integrating RISC-V processors, CGRAs and communication interfaces

CEI UPM

Centro de
Electrónica
Industrial

UNIVERSIDAD POLITÉCNICA DE MADRID

cei@upm.es

POLITÉCNICA