Microprocessors and Microsystems 125 (2026) 105351

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

GENIO: A Genetic Programming framework for structural design space exploration of HLS image pipelines

José L. Mira [a,*], Carlos E. Hernández [a], Jesús Barba [a], Antonio Caruso [b], Julián Caba [a], M. Soledad Escolar [a]

[a] Technologies and Information Systems Department, School of Computer Science, University of Castilla-La Mancha, Paseo de la Universidad, 4, Ciudad Real, 13071, Castilla-La Mancha, Spain

[b] Department of Mathematics and Physics "Ennio de Giorgi", University of Salento, Piazza Tancredi, 7, Lecce, 73100, Apulia, Italy

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.20020791

Keywords:
Image processing
Edge computing
FPGAs
Structural Genetic Programming
Hyperparameter search

ABSTRACT

In image processing pipelines, the complex interaction between operations and their hyperparameters creates an exponential design space where manual optimization is often infeasible. This challenge is magnified in Field-Programmable Gate Array (FPGA) implementations, where achieving an optimal trade-off between conflicting metrics — specifically output quality and hardware resource utilization (LUTs, Flip-Flops, and BRAM) — is critical for system integration. Standard heuristics often fail to navigate these non-linear dependencies effectively due to the structural complexity of the computational graphs. This paper introduces GENIO, a framework designed for High-Level Synthesis (HLS)-based pipelines. GENIO employs a structured representation of pipeline configurations that captures both topology and parameterization, enabling flexible exploration of compositional design spaces under hardware constraints. Within this framework, a Structural Genetic Programming (SGP) strategy is proposed to efficiently navigate the search space. A comprehensive comparative evaluation against NSGA-II, MOEA/D, Particle Swarm Optimization, and reinforcement learning–based strategies demonstrates that the proposed structural evolutionary approach attains competitive Pareto-front quality while exhibiting markedly faster convergence under identical synthesis budgets. In particular, near-optimal configurations are identified approximately 3.2 times earlier than with dominance-based multi-objective algorithms, highlighting its practical advantages in synthesis-driven optimization environments.

1. Introduction

In computer vision, hyperparameters play a fundamental role in defining how image processing operations are executed. Each stage of a pipeline — such as filtering, binarization, feature extraction, or classification — depends on specific parameter values that directly influence performance. When multiple operations are composed into a complete image processing pipeline, the resulting design space grows combinatorially, making the identification of optimal configurations increasingly challenging [1]. The proper selection of hyperparameters significantly affects both the efficiency and accuracy of computer vision systems. For instance, in image segmentation tasks, parameters such as binarization thresholds, window sizes, smoothing factors, and detection sensitivities strongly determine segmentation quality [2,3].

The rapid growth of the hyperparameter search space has motivated the development of systematic Hyperparameter Optimization (HPO) techniques. Classical approaches such as grid search and random search become inefficient as dimensionality increases, particularly when parameters interact nonlinearly. More advanced strategies, including Bayesian optimization, evolutionary algorithms, and reinforcement learning–based methods, have demonstrated effectiveness in exploring complex search spaces. However, most existing HPO approaches are primarily designed for software-based machine learning workflows and operate on fixed-length parameter vectors, assuming relatively inexpensive evaluation procedures. These assumptions may not hold when optimizing structured image processing pipelines intended for deployment on hardware-constrained embedded platforms [4].

When targeting embedded computing systems, hyperparameter optimization must additionally address limited computational power, memory capacity, and strict energy budgets. In such contexts, optimization becomes inherently multi-objective, requiring a balance between the quality of the output as the target application requires segmentation accuracy and hardware resource utilization [5]. Unlike conventional machine learning environments where computational resources are relatively abundant, embedded vision systems demand hardware-aware optimization strategies capable of accounting for latency, throughput,

* Corresponding author.
E-mail address: joseluis.mira@uclm.es (J.L. Mira).

https://doi.org/10.1016/j.micpro.2026.105351
Received 26 February 2026; Received in revised form 25 September 2026; Accepted 28 September 2026
Available online 8 October 2026
0141-9331/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

and energy consumption simultaneously. Although the optimization process itself is conducted offline, each candidate configuration must be evaluated with respect to the architectural constraints of the target device, typically requiring High-Level Synthesis (HLS), resource estimation, or hardware-in-the-loop analysis. As a consequence, the cost of evaluating candidate solutions is significantly higher, the optimization landscape becomes discontinuous and non-differentiable, and feasibility constraints imposed by the FPGA architecture fundamentally shape the search process.

These challenges are further amplified in real-world deployment scenarios such as embedded vision systems for smart agriculture [6]. Agricultural imaging environments exhibit high variability in illumination conditions, vegetation density, and object diversity [7], resulting in heterogeneous datasets that require robust pipeline architectures capable of maintaining optimal performance across diverse and highly variable operating conditions. At the same time, energy-autonomous IoT platforms — such as solar-powered devices used to process zenithal chromatic trap images [8] — impose strict hardware and energy constraints. In such systems, increased algorithmic complexity directly translates into higher latency, greater hardware utilization, and reduced battery lifetime. Consequently, the underlying optimization problem can be formally characterized as a hardware-constrained, multi-objective structural search task, where both pipeline topology and hyperparameter configurations must be optimized under architectural feasibility conditions.

Genetic Programming (GP) is an evolutionary optimization paradigm inspired by biological evolution, where candidate solutions evolve through selection, crossover, and mutation operators. Due to its ability to explore large, nonlinear, and highly complex search spaces, GP has proven effective in various optimization domains, including parameter tuning in image processing [9]. In contrast to conventional evolutionary multi-objective algorithms such as NSGA-II or MOEA/D, which typically operate on fixed-length encodings, Structural Genetic Programming (SGP) enables modifications to both parameter values and computational graph topology. This property makes it particularly suitable for optimizing structured image processing pipelines, where the search space includes both algorithmic configurations and stage composition.

The main contributions of this work are threefold. First, we propose a template-driven structural representation for HLS-based image processing pipelines, based on a JSON-defined intermediate specification that enables the systematic generation of synthesizable architectures with configurable stages and hyperparameters. Second, we introduce GENIO, a synthesis-aware evolutionary optimization framework that tightly integrates automatic code generation, hardware synthesis feedback, and application-level quality evaluation within a unified exploration loop tailored to FPGA-constrained environments. Third, we provide a comprehensive experimental evaluation against representative evolutionary, swarm-based, and reinforcement learning strategies under identical synthesis budgets, together with an analysis of convergence behavior, scalarization preferences, and parallel scalability in hardware-aware design space exploration.

The rest of the paper is organized as follows. Section 2 provides an overview of the state of the art in genetic algorithms and related optimization techniques applied to image processing and embedded system design. Section 3 describes the overall structure of the proposed framework, including the pipeline encoding strategy, the interaction with the HLS toolchain, the formulation of the genetic algorithm, and the definition of the case study. Section 5 presents the results of the comparative experimental study as well as the analysis of the impact of different scalarization weight configurations within the proposed genetic approach. Section 6 discusses the obtained results and analyzes their implications in the context of hardware-aware pipeline optimization. Finally, Section 7 summarizes the main findings and outlines future research directions.

2. Related work

Hyperparameter Optimization (HPO) has been widely investigated as a means to navigate high-dimensional and often non-convex search spaces. Classical baselines include grid search and random search. Grid search provides exhaustive coverage over a discretized parameter grid but becomes quickly intractable as dimensionality grows. Random search reduces this burden by sampling configurations stochastically and has been shown to be a strong baseline in high-dimensional settings where only a subset of hyperparameters dominates performance [10]. While these approaches remain useful for establishing reference performance, they generally assume that evaluating a candidate configuration is inexpensive, an assumption that is frequently violated in hardware synthesis workflows.

To improve sample efficiency under expensive evaluations, Bayesian optimization and Sequential Model-Based Optimization (SMBO) methods build surrogate models and select candidate configurations via acquisition functions. These ideas have been explored for tuning High-Level Synthesis (HLS) directive configurations, for example, through SMBO-based selection of directive settings [11]. However, HLS-driven design space exploration (DSE) often involves time-consuming compilation and synthesis steps, which restrict the feasible number of evaluations and motivate learning-driven or surrogate-assisted strategies [12]. Moreover, empirical analyses of modern HLS toolchains report complex interactions among directives that influence quality-of-results (QoR) metrics such as area and latency [13], which can complicate surrogate modeling and adaptive sampling in mixed discrete–continuous spaces. Consequently, a growing body of work focuses on hardware-aware DSE methods that explicitly treat synthesis cost as a limiting factor and incorporate model selection, adaptive sampling, and multi-objective reasoning to efficiently approximate Pareto-efficient regions [12,14]. Complementary efforts also leverage multi-fidelity optimization to combine cheaper low-fidelity estimates with higher-fidelity synthesis reports to accelerate the directive-tuning loop [15, 16].

Multi-objective optimization is central in embedded and FPGA-based systems, where designers must balance competing metrics such as performance, area, and energy. Evolutionary multi-objective algorithms (MOEAs) are widely used for approximating Pareto fronts in such settings. NSGA-II [17] and MOEA/D [18] are among the most established MOEAs, representing dominance-based elitist search and decomposition-based optimization, respectively. In addition to the choice of search algorithm, the evaluation of multi-objective solution sets commonly relies on quality indicators, with the hypervolume indicator being a standard unary metric introduced and popularized in the multi-objective evolutionary computation literature [19]. These foundations provide widely adopted tools for algorithmic comparison in conflicting-objective optimization scenarios, including synthesis-aware FPGA DSE.

Evolutionary algorithms more broadly, including Genetic Algorithms (GAs), are frequently used for black-box optimization in discrete or mixed spaces. GAs employ population-based operators such as selection, crossover, and mutation and have been applied to hyperparameter tuning [20] and to computer vision problems such as segmentation [21] and medical imaging [22]. In the hardware domain, evolutionary computation has been used for automated circuit design and topology exploration [23], for evolving HDL-like representations [24], and for model-based optimization of embedded-system design parameters [25]. These studies illustrate the suitability of evolutionary search for complex optimization problems under constraints, although they often target lower abstraction levels or assume fixed computational structures rather than HLS-integrated pipeline generation.

Reinforcement Learning (RL) has also been explored as an alternative paradigm for hyperparameter optimization and design space exploration. Unlike population-based evolutionary methods or surrogate-driven Bayesian approaches, RL formulates optimization as a sequential

decision-making problem under the Markov Decision Process (MDP) framework [26], where an agent interacts with an environment by selecting configuration actions, receiving rewards derived from quality-of-results (QoR) metrics, and updating its policy to maximize long-term cumulative returns.

A key methodological distinction in the context of configurable image processing pipelines is whether the optimizer can explore structure in addition to tuning parameters. Many MOEAs and related metaheuristics operate on fixed-length encodings, which naturally support parametric tuning but constrain the search to a predefined computational graph. Genetic Programming (GP), originally formulated as an evolutionary method that evolves programs rather than fixed vectors [27], provides a natural mechanism for structural search. Practical GP variants and guidance documents further highlight how GP can represent and evolve structured solutions [28]. Related structured representations such as Cartesian Genetic Programming (CGP) represent programs as directed acyclic graphs and have been studied extensively, including discussions of applications and acceleration [29]. These paradigms are particularly relevant when the optimization objective depends on both pipeline topology (e.g., stage selection and composition) and internal hyperparameters.

In FPGA-centric workflows, the need for systematic DSE has motivated a substantial literature on exploring HLS optimization spaces. Surveys on HLS DSE document the breadth of techniques proposed to navigate synthesis option spaces and generate trade-offs from a single behavioral description [30]. Beyond surveys, several tools propose automation strategies for HLS-centric exploration, including multi-objective autotuning frameworks such as OpenTuner [31] and HLS-focused DSE engines that address scalability and compatibility among transformations and directives [32]. Furthermore, recent frameworks emphasize multi-objective and multi-fidelity strategies for directive design, explicitly addressing the contradictions among objectives and the limited budgets available for full synthesis evaluations [16]. These trends reinforce the view that HLS design optimization is commonly performed offline, but must be driven by synthesis-aware QoR metrics and feasibility constraints of the target FPGA.

Finally, hardware-aware optimization has also been actively explored in the Neural Architecture Search (NAS) community, where architecture decisions are explicitly coupled to hardware objectives such as latency. ProxylessNAS is a representative example of searching architectures with hardware objectives in mind [33], and surveys summarize broader trends in hardware-aware NAS across targets and objectives [34]. While these works address structural choices under hardware constraints, they primarily focus on neural network architectures rather than classical HLS-based image processing pipelines composed of algorithmic stages and pragmatic synthesis directives.

In summary, while substantial progress has been achieved in hyperparameter optimization, multi-objective evolutionary search, and hardware-aware design space exploration, existing approaches typically address these dimensions in isolation. Many methods focus exclusively on parametric tuning within fixed structural templates, while others operate at abstraction levels that are not directly integrated with contemporary HLS toolchains. Moreover, topology-level exploration and synthesis-aware evaluation are seldom unified within a single multi-objective optimization framework capable of jointly considering application-level quality metrics and FPGA resource constraints. These limitations highlight the need for methodologies that explicitly integrate structural evolution and application-level output quality metrics within a hyperparameter optimization framework guided by HLS-driven synthesis feedback. This gap motivates the Structural Genetic Programming approach proposed in this work, which targets joint topology–parameter exploration within a hardware-aware, multi-objective FPGA optimization setting.

3. Proposed methodology

The definition of an image processing pipeline refers to a structured and sequential series of operations that are applied consecutively to the data within an image. Following the image acquisition process, these operations aim to modify, enhance, or analyze the information contained in the image. Such pipelines typically consist of multiple stages for manipulating image data, including preprocessing (e.g., noise removal), segmentation and region identification, or classification of image elements.

The focus of this work is the rapid and efficient implementation of image processing pipelines in embedded systems using reconfigurable logic devices. To achieve this goal, the *Vision Library* within the Xilinx/AMD Vitis HLS tool[1] has been chosen as the primary development framework. This library is an adaptation of a subset of standard operations from the OpenCV vision project, specifically modified for compatibility with the Vitis HLS tools to enable efficient operation on FPGAs.

The proposed methodology for optimizing the synthesized pipeline architecture relies on the careful selection of configuration parameters for each stage defining the dataflow processing architecture. Listing 1 presents an example of an image processing pipeline described using C++ and the Vitis Vision library. As shown, the top-level function defines the pipeline stages and the channels between tasks using an OpenCV-like syntax. The output of the pipeline, as well as other metrics such as resource usage, latency, and throughput, depends on the specific parameter values assigned to the pipeline. These parameters include image dimensions, threshold values, the size of the convolution kernel, the standard deviation applied to the Gaussian filter, and more.

```
#include <hls_video.h>
#include <ap_int.h>
#include <ap_axi_sdata.h>
#include <hls_stream.h>
#include "common/xf_common.hpp"
#include "imgproc/xf_gaussian_filter.hpp"
#include "imgproc/xf_sobel.hpp"
#include "imgproc/xf_threshold.hpp"

#define WIDTH  1920
#define HEIGHT 1080
#define NPC1   XF_NPPC1

typedef ap_axiu<8,1,1,1> pixel_t;
typedef hls::stream<pixel_t> stream_t;

void image_pipeline(stream_t& in_stream, stream_t&
     out_stream, int threshold_value) {
    #pragma HLS INTERFACE axis port=in_stream
    #pragma HLS INTERFACE axis port=out_stream
    #pragma HLS INTERFACE s_axilite
         port=threshold_value bundle=CONTROL
    #pragma HLS INTERFACE s_axilite port=return
         bundle=CONTROL

    xf::cv::Mat<XF_8UC3, HEIGHT, WIDTH, NPC1>
         img_rgb(HEIGHT, WIDTH);
    xf::cv::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1>
         img_gray(HEIGHT, WIDTH);
    xf::cv::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1>
         img_blur(HEIGHT, WIDTH);
    xf::cv::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1>
         img_edges(HEIGHT, WIDTH);
    xf::cv::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1>
         img_binary(HEIGHT, WIDTH);

    #pragma HLS DATAFLOW

    xf::cv::axiStrm2xfMat<XF_8UC3, HEIGHT, WIDTH,
         NPC1>(in_stream, img_rgb);
```

[1] https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html.

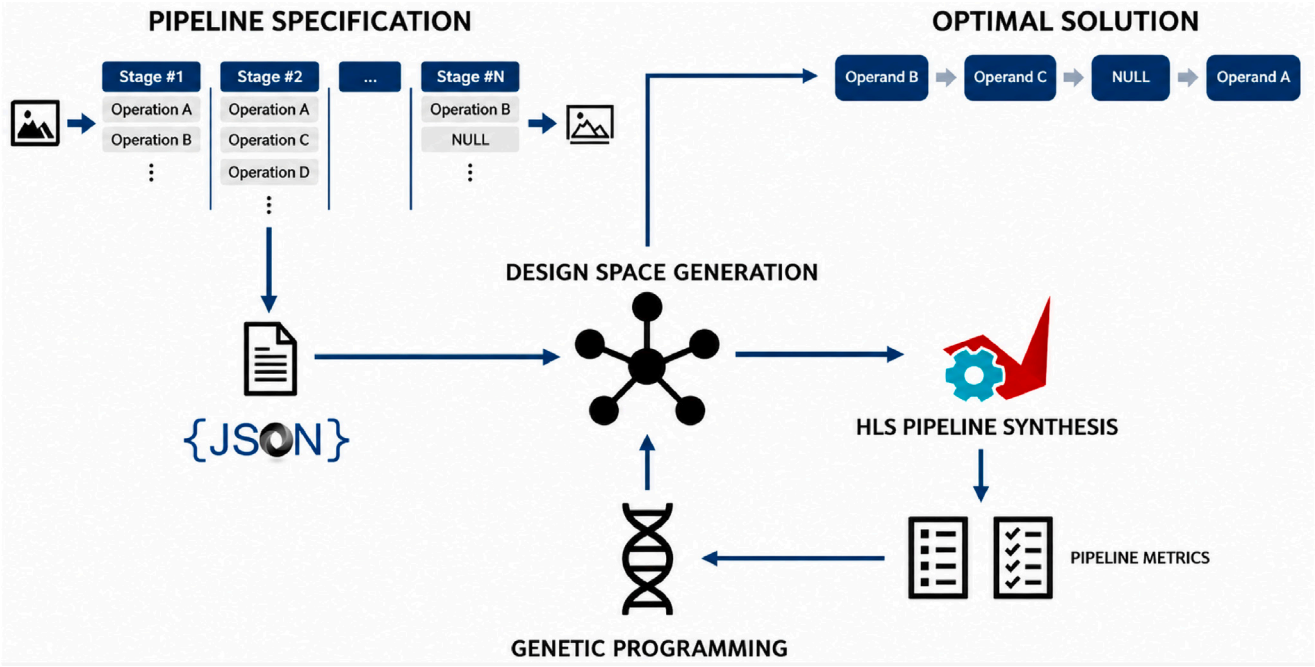


Fig. 1. Proposed optimization workflow for HLS-defined image processing pipelines.

```
32      xf::cv::rgb2gray<XF_8UC3, XF_8UC1, HEIGHT,
            WIDTH, NPC1>(img_rgb, img_gray);
33      xf::cv::GaussianBlur<XF_BORDER_CONSTANT,
            XF_8UC1, HEIGHT, WIDTH, NPC1>(img_gray,
            img_blur, 5);
34      xf::cv::Sobel<XF_BORDER_CONSTANT, XF_8UC1,
            XF_8UC1, HEIGHT, WIDTH, NPC1,
            XF_FILTER_3X3>(img_blur, img_edges);
35      xf::cv::threshold<XF_THRESHOLD_TYPE_BINARY,
            XF_8UC1, HEIGHT, WIDTH, NPC1>(img_edges,
            img_binary, threshold_value, 255);
36      xf::cv::xfMat2axiStrm<XF_8UC1, HEIGHT, WIDTH,
            NPC1>(img_binary, out_stream);
37  }
```

Listing 1: Example of a processing pipeline with Xilinx/AMD Vitis HLS

This work proposes GENIO, an automated approach to efficiently explore the design space defined by the entire set of pipeline configuration parameters using genetic programming techniques. The proposed method is illustrated in Fig. 1.

First, the designer must model the pipeline in a computer-readable format using a custom JSON-based syntax. This description defines the structure of the pipeline as well as the set of admissible operations and hyperparameters for each stage, thereby specifying the design space to be explored.

From this specification, the framework generates an initial set of valid, synthesizable pipeline configurations in C++. Each candidate is evaluated through the synthesis reports produced by the AMD/Xilinx Vitis HLS tool, which provide performance and resource utilization metrics. These metrics are used by an optimization engine to guide the exploration of the design space. In this work, a structural genetic programming approach is employed, where candidate solutions are iteratively evaluated, selected, combined, and mutated to generate new configurations.

This iterative optimization process is repeated across generations, progressively refining both the pipeline structure and its hyperparameter configuration, until a set of high-quality solutions satisfying the design constraints is obtained.

The optimization process is structured as a closed-loop interaction between representation, code generation, and evaluation. Candidate pipelines are instantiated from the template definitions, synthesized using HLS, and assessed based on implementation-level metrics. This feedback is used by the optimization engine to guide the search toward improved configurations.

Throughout the next sections, a detailed view of the methods and techniques integrated into GENIO is provided.

3.1. Pipeline modeling

The input consists of a textual description of the pipeline model, represented using JSON templates. This description specifies the sequence of stages to be executed, as well as the values of the hyperparameters to be applied at each stage, following a formal approach.

The composition list of stages implicitly defines the execution order and potential variations or alternatives at a specific point in the pipeline. Each stage in the list references a unique identifier, which must correspond to the stage definition provided as a library or collection to the developer. For example, in Listing 2, it shows an excerpt of a composition list where two stages are defined. The first stage would process the input data either using operand id 51 or operand id 52, whereas the second stage would process the input data with one of the two operands (i.e. 60 or 61) included under its section. Therefore, the template system provides a clean way to express alternatives or variations within a stage in the pipeline, as well as optionality; there is a special stage id with value 0 that means no action will take place if chosen.

The parameter configuration for a stage is managed through a list of possible values that each parameter can take, or by defining maximum and minimum ranges along with the step interval between them. These configurations are specified for each hyperparameter declared in the stage definition. However, this parameterization is inherently dependent on the selected operation.

The design space associated with each pipeline position is not uniform. Instead, each stage may admit multiple alternative operations, and each operation defines its own design space with potentially different hyperparameter configurations.

For a given pipeline stage, the set of candidate operations is specified in the composition list. Each operation is associated with a distinct set of parameters, whose values can be defined either explicitly as discrete lists or procedurally through parameter ranges. In the latter

case, minimum and maximum bounds, together with step values, define the valid domain for each parameter.

Consequently, different operations within the same stage may expose different numbers and types of hyperparameters. For example, a simple operation may not require any configurable parameter, while others, such as morphological or filtering stages, may define multiple parameters (e.g., kernel size, shape, or threshold values). These parameters are mapped to the corresponding placeholders in the Vitis HLS code templates (e.g., @K_ROWS, @K_COLS, @K_SHAPE), enabling the generation of fully instantiated pipeline configurations.

```
{
    "stages":[
        ...
        [sid: "1", name: "Color space
            conversion",
         operations: [
            {
                "id":"51",
                "min": 0,
                "max": 50,
                "step": 3
            },
            {
                "id":"52"
            }
        ]],
        [sid: "2", name: "Morphologic ops"
            ,
         operations: [
            {
                "id":"60",
                "min":[1, 3],
                "max":[3, 7],
                "step":[1, 2]
            },
            {
                "id":"61",
                "min":[1, 3],
                "max":[3, 7],
                "step":[1, 2]
            },
            {
                "id": "0"
            }
        ]]
        ...
    ]
}
```

Listing 2: Partial example of a composition list using custom JSON templates for pipeline modeling

3.1.1. Stage definition

Similarly to the image processing pipeline, it is also necessary to formally define each operation. For example, Listing 3 shows the structured syntax proposed to capture the metadata for the erode morphological operation and encapsulate the details of the operation itself.

The main elements of the JSON-based language used to create the operand definition template are outlined below:

- **id**: Unique identifier of the operand.
- **name**: Name.
- **description**: Information about the function.
- **input parameters**: Dimensionality and data type of the input for the processing stage.
- **output parameters**: Dimensionality and data type of the output for the processing stage.
- **vitis_hls**: Corresponding Vitis_HLS library function. This section must contain:
  - **function calls**: Vitis HLS code template for C++ generation by GENIO. It should be noted that the format and characteristics of the data processed by the pipeline may change as they propagate through its stages (e.g., image height and width or pixel format). Therefore, the templates may use global pipeline parameters (denoted by the prefix @) to adapt the code generation of the current stage to the data format produced by the previous stage and to propagate the relevant information to the next stage.
  - **mandatory includes**: All the Vitis HLS libraries that must be imported to generate a valid executable specification.
  - **references**: This section includes the constants and variables that govern the operation of each stage, accompanied by a brief description of each.

```
{
    "id": 60,
    "name": "Erode",
    "description": "Computes a local
        minimum over the area of given
        kernel.",
    "input parameters": [
        {
            "shape": ["n", "n"],
            "dtype": "int8"
        }
    ],
    "output parameters": [
        {
            "shape": ["n", "n"],
            "dtype": "int8"
        }
    ],
    "vitis_hls": {
        "function": "xf::erode<
            XF_BORDER_CONSTANT, XF_8UC1,
            @ROWS, @COLS, @K_SHAPE,
            @K_ROWS, @K_COLS, 1>(@input,
            @output, @kernel);",
        "includes": ["#include \"
            xf_erosion.hpp\""],
        "references": {
            "@input": "input image stream
                from previous step",
            "@output": "output image
                stream to next step",
            "@ROWS": "Maximum height of
                input and output image",
            "@COLS": "Maximum width of
                input and output image",
            "@K_SHAPE": "Shape of the
                kernel. The supported
                kernel shapes are RECT,
                CROSS and ELLIPSE.",
            "@K_ROWS": "Height of the
                kernel.",
            "@K_COLS": "Width of the
                kernel.",
            "@kernel": "Erosion kernel of
                size K_ROWS * K_COLS"
        }
    }
}
```

Listing 3: Stage definition template example for erode operation

As it has been introduced before, the proposed syntax allows a mechanism to customize an operand based on the hyperparameter values selected for previous stages during the exploration of the design space. This way, the generated code would be valid and would undergo the compilation step successfully. This mechanism enables seamless integration and manipulation of hyperparameters, ensuring that they can be easily adjusted and fine-tuned to optimize the performance of the processing stages. Additionally, the syntax should support the clear and consistent representation of these hyperparameters, facilitating code readability and maintainability. The reserved character '@' is used to reference any hyperparameter identifier in the code template.

For example, in the Vitis HLS C++ template for the operand erode declared in Listing 3 (see line 18), several parameters are used. Some of them are mandatory and fixed in the framework, such as '@input' and '@output', which refer to the stream channels that will be generated during the C++ executable production. Other parameters, such as '@ROWS' and '@COLS' are propagated from preceding stages where, for example, the dimensions of the input data might be scaled. Finally, other parameters such as '@K_SHAPE', '@K_ROWS', and '@K_COLS' are configuration parameters specific to the current stage, referring to the size of the kernel.

3.2. Search space definition

The design space is defined by the Cartesian product of all possible pipeline structures and the hyperparameter configurations for each stage. This results in a highly complex, multidimensional search space, where each dimension corresponds to a specific design parameter, such as the structure of the pipeline stages, the selection of operations, or the tuning of performance-related hyperparameters. The vastness of this space stems from the combinatorial nature of the design choices available at each stage, compounded by the need to respect various constraints related to compatibility, synthesis feasibility, and performance goals.

An individual solution is generated by instantiating pipeline templates with valid parameters, selected according to a predefined search strategy. GENIO employs a parsing mechanism that processes the template descriptions and generates a set of synthesizable C++ files, each of them representing a unique candidate solution. This set represents the *population* and serves as an approximation of the search space, where each solution is encoded in a C++ file that can be compiled and evaluated for synthesis feasibility. The parsing mechanism is designed to efficiently handle the complexities of the templates and to generate candidate solutions that can be directly synthesized and tested on real hardware or in simulation environments.

3.3. Genetic Programming

Genetic Programming (GP) is an evolutionary computation technique inspired by the principles of biological evolution and natural selection. Unlike traditional optimization methods, which often rely on deterministic or gradient-based approaches, GP employs stochastic search and population-based exploration to evolve solutions over generations. In classical GP formulations, candidate solutions are represented as programs or expression trees whose structure and parameters can be modified through evolutionary operators such as subtree crossover and mutation [35].

In contrast to classical GP formulations the proposed method adopts a template-constrained structural encoding tailored to HLS-based pipeline generation. Structural variability is not achieved through dynamic tree growth, but through controlled selection among predefined alternatives within a structured search space. This approach enables joint exploration of pipeline composition and hyperparameter configurations while ensuring compatibility with synthesis constraints and preserving the validity of generated implementations.

3.3.1. Chromosome codification

Each individual in the population is encoded as a fixed-length structured chromosome derived from the predefined pipeline template. The chromosome length corresponds to the ordered sequence of stage positions specified in the JSON-based representation.

Each gene represents the selection of an operation for its corresponding stage, together with the associated hyperparameter configuration. For every stage position, multiple alternative operations may be defined, including an explicit "no-operation" option. Selecting this option effectively disables the stage during instantiation, enabling different effective pipeline topologies while maintaining a constant chromosome size.

Genetic operators are applied position-wise over this structured encoding. Crossover exchanges subsets of genes between parent individuals, and mutation modifies either the selected operation or its parameter values within the admissible ranges defined in the template. Since all structural alternatives and parameter bounds are specified a priori, every chromosome corresponds to a syntactically valid and synthesizable pipeline configuration.

Under this representation, each chromosome fully defines a candidate solution, including both its effective structural composition and its hyperparameter settings, allowing systematic exploration of the hardware-constrained design space.

3.3.2. Fitness function and metrics

The fitness function in genetic programming is a quantitative measure used to evaluate and rank the quality of candidate solutions in relation to the problem being solved. It serves as the guiding metric for selection, driving the evolutionary process by favoring solutions that perform better according to the defined objectives. In this work, the optimal selection of the hyperparameters that configure the pipeline stages is guided by the simultaneous achievement of two sub-objectives, evaluated using metrics related to the quality of the output image mask and the FPGA resource utilization.

On the one hand, segmentation quality is assessed from the regions identified by the final Connected Components stage of the processing pipeline. This stage groups connected foreground pixels and produces the candidate regions from which bounding boxes are extracted for comparison against the manually annotated ground truth. The resulting bounding boxes provide an application-relevant measure of segmentation performance through the IoU-based QualityScore. The subsequent insect classification stage is not included in the optimization process.

The QualityScore metric (see Eq. (2)) is computed as the average Intersection over Union (IoU) between matched predicted and ground-truth bounding boxes. The IoU between a predicted bounding box B_p and a ground-truth bounding box B_{gt} is defined as:

$$\mathrm{IoU}(B_p, B_{gt}) = \frac{|B_p \cap B_{gt}|}{|B_p \cup B_{gt}|} \tag{1}$$

A predicted region is considered a valid match if its IoU with a reference bounding box exceeds 50%. Let M denote the set of valid matches between predicted and reference bounding boxes. The QualityScore is then computed as:

$$QualityScore = \frac{1}{|M|} \sum_{(B_p, B_{gt}) \in M} \mathrm{IoU}(B_p, B_{gt}) \tag{2}$$

Regions with IoU below this threshold are treated as non-matching and excluded from the score calculation.

Fig. 2 graphically illustrates the comparison between the reference output and the obtained output, which serves as the basis for calculating the quality metric. In the figure, correctly identified regions are highlighted with green boxes, expert-drawn regions without a corresponding match in the output image are shown in yellow, and regions present in the output image but absent in the reference solution are marked in red.

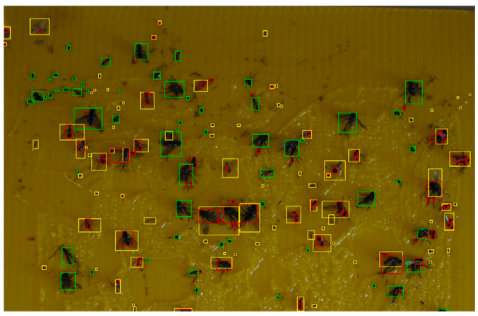


Fig. 2. Bounding-box overlapping analysis between reference and output.

On the other hand, the FPGA resource utilization metric is computed as the weighted average usage percentage of various internal resources, such as LUTs, Flip-Flops, BRAMs, and DSPs. By referencing the available resources on the target board, the utilization percentage for each resource is calculated using Eq. (4). These values are directly extracted from the synthesis reports generated by Vitis HLS.

$$\begin{aligned} r_{ff} &= 1 - \frac{\text{PostImplementationFFs}}{\text{TotalFFs}} \\ r_{lut} &= 1 - \frac{\text{PostImplementationLUTs}}{\text{TotalLUTs}} \\ r_{bram} &= 1 - \frac{\text{PostImplementationBRAMs}}{\text{TotalBRAMs}} \\ r_{dsp} &= 1 - \frac{\text{PostImplementationDSPs}}{\text{TotalDSPs}} \end{aligned} \tag{3}$$

$$\text{HardwareScore} = w_{ff} \cdot r_{ff} + w_{lut} \cdot r_{lut} + w_{bram} \cdot r_{bram} + w_{dsp} \cdot r_{dsp} \tag{4}$$

The weights in Eq. (4) ($w_{ff}, w_{lut}, w_{bram}, w_{dsp}$) serve as the primary mechanism for the designer to communicate hardware-level preferences to the genetic programming engine. While a balanced distribution (e.g., 0.25 each) treats all resources equally, the framework is designed to handle asymmetric weighting to address specific device bottlenecks. For instance, in a system where Block RAM is the primary constraint, a higher w_{bram} coefficient forces the evolutionary process to prioritize structural variants that minimize memory occupancy, even at the cost of higher LUT utilization. This flexibility is essential for FPGA integration, where the 'optimal' design is often dictated by the remaining available resources on a partially occupied chip. By adjusting the hardware weights, the designer effectively steers the framework toward specific implementation styles (e.g. integer arithmetic versus floating/fixed point arithmetic - more DSPs).

Consequently, the evaluation of an individual is determined by a weighted combination of previously mentioned metrics, each addressing a distinct aspect of the solution's performance. This weighted approach enables flexibility in prioritizing specific metrics to emphasize particular objectives, such as computational efficiency or resource utilization, depending on the design requirements. The weights can be adjusted to reflect the application's priorities, ensuring that the optimization process aligns with real-world constraints and goals, as detailed in Eq. (5). By tuning these weights, the methodology accommodates diverse scenarios, from high-performance systems with stringent latency requirements to resource-constrained designs where FPGA utilization is critical.

$$IndividualScore = (\alpha \cdot QualityScore) + (\beta \cdot HardwareScore) \tag{5}$$

The fitness function defined in Eq. (5) employs a weighted-sum scalarization to aggregate quality and resource usage. While multi-objective algorithms such as NSGA-II are widely used to find a full Pareto frontier, a scalarized approach was intentionally selected for this framework to allow for a priori preference integration. In the context of FPGA design, hardware constraints often act as 'hard' boundaries; the weighted sum allows the evolutionary process to prioritize solutions that remain within the physical limits of the target device, such as LUTs or BRAM availability.

3.3.3. Parameters governing genetic optimization

A set of parameters governs the behavior and performance of the genetic algorithm during optimization. These parameters include the encoding method for the genes of each individual, the population size, the selection and crossover strategies used to generate subsequent generations, the mutation rate applied to the genes, and the maximum number of generations. Each gene represents the operation selected for a given pipeline stage together with its associated hyperparameter configuration, forming the basis of the evolutionary search process.

In addition, the initialization of the population is designed to ensure a balanced exploration of the search space. For each stage position, operations are selected uniformly at random from the set of admissible alternatives, independently of the size or dimensionality of their associated parameter space. This prevents operations with larger internal hyperparameter configurations from being overrepresented during initialization. Once an operation is selected, its corresponding parameters are instantiated by sampling valid values within their defined ranges. As a result, the initial population provides an unbiased coverage of structural alternatives, ensuring that all stage types are explored with equal priority at the start of the optimization process.

Each individual is evaluated as described in Section 3.3.2, producing the QualityScore and HardwareScore metrics, as well as their aggregated IndividualScore. Since these metrics operate on different numerical ranges, direct aggregation during the selection phase may lead to unequal contributions. To regulate selection pressure, Min–Max normalization is applied within each generation using the observed metric values of that generation. This normalization is used exclusively for ranking individuals and assigning selection probabilities during reproduction. The reported performance metrics and final results presented in Section 5 correspond to the globally computed, non-normalized evaluation values.

From the population generated in a given generation, the top 50% of individuals, based on their normalized ranking, are selected for reproduction. Genetic crossover probabilities are then assigned proportionally to their normalized fitness values. Parent selection follows a roulette-wheel mechanism, where cumulative probabilities determine the likelihood of selection. A random value in the interval $[0, 1]$ is used to sample parents, ensuring that higher-ranked individuals have a greater probability of contributing to the next generation. Selected individuals remain eligible for subsequent pairings within the same generation.

Once two parents are selected, crossover is performed by exchanging subsets of genes between them. Gene positions are selected randomly, and at least half of the chromosome is inherited from each parent. This position-wise exchange increases population diversity and promotes the exploration of new structural and parametric combinations within the predefined search space. Mutation is subsequently applied by randomly modifying either the selected operation or its associated parameter values within their valid bounds. Since all admissible alternatives are defined in the template specification, every offspring corresponds to a valid and synthesizable pipeline configuration.

3.3.4. Stopping criterion

A stopping criterion is a predefined condition that determines when the genetic algorithm should terminate. It is essential to prevent unnecessary computations and ensure that the algorithm halts once an acceptable solution has been found, or when further iterations are unlikely to result in significant improvements.

It has been established the following condition for such end:

$$\Delta\text{med}(S_n) = \frac{\text{med}(S_{n+1}) - \text{med}(S_n)}{\text{med}(S_n)} \tag{6}$$

where: $\text{med}(S_n)$ represents the median score in generation n and $\Delta\text{med}(S_n)$ denotes the relative increase in the median score between generation n and generation $n+1$.

Table 1
Reference pipeline configuration.

#Stage	Operation	#Stage	Operation
1	Resize (1280 × 720)	5	Dilation (3 × 3 kernel)
2	RGB to Grayscale	6	Erosion (3 × 3 kernel)
3	Histogram Equalization	7	Erosion (3 × 3 kernel)
4	Thresholding (12)	8	Dilation (3 × 3 kernel)
9	Connected Components		

Therefore, the evolution process should be canceled when:

$$\Delta\text{med}(S_n) < \epsilon \tag{7}$$

being ϵ a predefined threshold representing the minimum relative improvement allowed to continue the evolution process. For instance, a typical value could be $\epsilon = 0.01$, representing a 1% improvement threshold.

This stopping criterion ensures that the evolution halts when the relative improvement in the median score between two consecutive generations is below the threshold ϵ. If the median score no longer improves significantly, it can be concluded that the population has converged to a near-optimal solution, and further evolution is unnecessary.

4. Experimental setup

This section describes the experimental protocol used to evaluate the proposed framework. It includes the definition of the case study, the characteristics of the dataset and evaluation environment, and the configuration of the optimization methods considered in the comparative analysis. All experiments are designed to ensure a fair and reproducible assessment under realistic hardware synthesis constraints.

4.1. Case study

This case study focuses on a real-world embedded vision application deployed on an FPGA-based edge platform. The objective is to evaluate the proposed optimization framework under realistic synthesis constraints, considering both application-level segmentation quality and hardware-level resource utilization. The selected scenario reflects the practical challenges of HLS-driven design space exploration, where computational structure, hyperparameters, and synthesis directives jointly impact performance, latency, and area trade-offs.

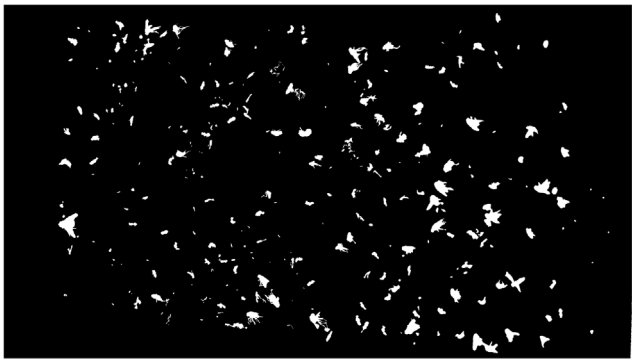
4.1.1. Reference baseline

The proposed framework is evaluated on an embedded image segmentation application for chromatic trap monitoring in agricultural environments. The objective of the application is to segment foreground regions corresponding to pest insects from high-resolution RGB images (3840 × 2160) captured under outdoor conditions. The segmentation stage constitutes the preprocessing step prior to a classification module for insect identification [8].

The target deployment platform is the Avnet Ultra96-V2 FPGA board, representative of resource-constrained edge-computing systems. The reference implementation currently deployed in a prototype system consists of an nine-stage image processing pipeline. The configuration of this baseline pipeline is summarized in Table 1. The stages include spatial resizing, color space conversion, contrast enhancement, global thresholding, morphological filtering operations and image object labeling. Kernel sizes and threshold values were determined through empirical tuning during system development.



(a) Raw input image.



(b) Manually created ground truth mask.

Fig. 3. Dataset image example.

4.1.2. Dataset and evaluation environment

Performance evaluation is conducted on a dataset composed of fifty high-resolution images captured under real outdoor environmental conditions. The dataset includes variations in illumination, wind-induced motion, dust particles, and background clutter, introducing variability in the segmentation task. Ground-truth bounding boxes were manually annotated by domain experts to delineate insect regions and distinguish them from vegetation, trap surfaces, and background artifacts, as illustrated in Fig. 3. The annotations define the reference regions used for IoU-based evaluation.

All experimental evaluations are performed using AMD/Xilinx Vitis HLS version 2023.1.1 to obtain resource utilization and latency estimates for each synthesized configuration. Experiments are executed on a workstation equipped with an Intel Core i7-12700K processor and 64 GB of DDR5 RAM running Ubuntu 22.04.5.

4.2. Evaluated search methods

In this work, five search strategies are compared with the proposed structural genetic programming method to evaluate its performance in a hyperparameter optimization problem. The selected approaches are drawn from evolutionary algorithms, swarm intelligence, and reinforcement learning.

Given that the optimization problem involves balancing segmentation quality and hardware resource utilization, particular attention is paid to whether each method natively supports multi-objective optimization or relies on scalarization. Table 2 summarizes these capabilities.

The configuration adopted for each method is described below:

- **GENIO (SGP)**: The algorithm operates with a population size of 100 individuals over a maximum of 20 generations.

Table 2
Multi-objective optimization capabilities of evaluated algorithms.

Algorithm	Native multi-objective support
Custom Genetic Algorithm	No
NSGA-II	Yes
MOEA/D	Yes
PSO	No
PPO	No
Envelope Q-Learning	Yes

- **NSGA-II**: Non-dominated Sorting Genetic Algorithm II [17] is a widely adopted multi-objective evolutionary algorithm based on fast non-dominated sorting and crowding distance. It explicitly maintains a diverse approximation of the Pareto front by ranking individuals according to dominance levels and preserving diversity through density estimation.
  In this study, NSGA-II was implemented using the `pymoo` framework [36], with a population size of 100 individuals and 20 generations. Binary tournament selection, Simulated Binary Crossover (SBX, 90%, $\eta = 15$), and polynomial mutation ($\eta = 20$) were employed. Survivor selection combined non-dominated sorting and crowding distance to ensure both convergence and diversity. By directly optimizing QualityScore and HardwareScore without scalarization, NSGA-II provides a reference for native multi-objective optimization performance.
- **MOEA/D**: The Multi-Objective Evolutionary Algorithm based on Decomposition [18] tackles multi-objective optimization by decomposing the original problem into a set of scalar subproblems defined by weight vectors. Each subproblem is optimized simultaneously in cooperation with its neighboring subproblems, promoting information sharing during the search process. This decomposition-based strategy encourages a structured and well-distributed approximation of the Pareto front and can improve scalability when the objectives exhibit relatively smooth trade-offs.
  The algorithm was also implemented using `pymoo`. A Das-Dennis decomposition with two objectives and 12 partitions was applied. Each subproblem considered 15 neighboring solutions with a 70% probability of local mating. SBX crossover ($\eta = 20$) and adaptive polynomial mutation were used over 20 generations. This configuration aimed to ensure a uniform approximation of the Pareto front while maintaining computational tractability.
- **PSO**: Particle Swarm Optimization [37] is a population-based optimization method inspired by collective behaviors observed in natural swarms. Each particle represents a candidate solution and updates its position based on its own best-known solution and the best-known solution of the swarm. The update mechanism combines inertia, cognitive, and social components to balance exploration and exploitation.
  In this study, PSO was implemented using `pymoo` with 100 particles initialized through Latin Hypercube Sampling. The inertia coefficient was set to $w = 0.7$, and both cognitive and social coefficients were set to 1.5. Velocity clamping (20%) and adaptive parameter tuning were incorporated to stabilize convergence. Since PSO does not natively support multi-objective optimization, scalarization was applied to combine segmentation quality and hardware efficiency into a single objective function. As the pipeline design space is discrete, whereas PSO updates particle positions using continuous values, a repair mechanism was applied after each position update. Specifically, each floating-point coordinate was truncated to its corresponding integer value, thereby projecting the particle position back into the discrete search space. Since each design variable is defined over a bounded integer domain representing the available structural choices, the resulting integer-valued positions can be directly mapped to valid pipeline configurations.
- **PPO**: Proximal Policy Optimization is a stable policy-gradient algorithm designed to perform reliable and incremental policy updates. It was implemented using `Stable-Baselines3` [38] and interfaced through the `Gymnasium` API [39]. A three-layer multilayer perceptron (100 neurons per layer) was used as the policy network.
  Training was conducted over 20 episodes, using clipped surrogate objectives, generalized advantage estimation, and entropy regularization to ensure stable learning. Because PPO does not natively support multi-objective optimization, scalarization was applied to define a single reward combining segmentation quality and hardware efficiency.
- **Envelope Q-Learning**: Envelope Q-Learning [40] is a multi-objective reinforcement learning method that approximates the Pareto frontier by learning value functions conditioned on preference vectors. Unlike scalarized RL approaches, it preserves vector-valued rewards and applies an envelope operator to maintain Pareto consistency.
  In this work, Envelope Q-Learning was implemented using `mo-gymnasium` [41], enabling direct interaction with a multi-objective environment. Each learned policy corresponds to a specific direction in the objective space, allowing post-training selection based on desired trade-offs. This approach is particularly advantageous when objective weightings are not predetermined.

4.3. Experimental protocol

All methods were evaluated on the same case study under identical application-level evaluation criteria and hardware-synthesis conditions. Population-based methods were executed for 20 generations, whereas PPO and Envelope Q-Learning were evaluated over sequences of 20 individual interaction steps. Since a generation does not necessarily involve the same number of candidate evaluations across methods, a common analysis horizon of 20 recorded generations or evaluation steps was adopted to ensure a consistent basis for comparison. Each method was independently executed ten times. Candidate evaluations that did not produce valid results were retained in the experimental data and represented as missing scores rather than being discarded.

For each independent run, three performance indicators were computed: the best feasible score achieved within the common evaluation horizon, the area under the best-so-far score curve, and the fraction of evaluations yielding a valid score. Prior to the identification of the first feasible solution, the best-so-far curve was assigned a value of zero. This convention represents the absence of a feasible solution and should not be interpreted as an observed candidate score. The resulting distributions were summarized using the median and interquartile range (IQR).

5. Results

This section evaluates the effectiveness of the proposed optimization framework through a comparative study against well-established search methods reported in the hyperparameter optimization literature. The assessment focuses on two complementary objectives: (i) application-level performance, measured through the QualityScore defined in Eq. (2), which evaluates the extent to which the generated output improves the performance of subsequent stages in the processing chain (in this case ROI classification), and (ii) FPGA resource utilization, measured through the HardwareScore defined in Eq. (4). In addition, we analyze the impact of scalarization preferences in Eq. (5) by varying the relative weights assigned to quality and resource usage.

5.1. Comparison of search methods

This section presents the comparative evaluation of the considered optimization strategies. The analysis focuses on their performance in terms of convergence behavior, solution quality, population dynamics, and parallel scalability.

Table 3
Best feasible solutions observed within the common recorded horizon.

Method	Individual score	Hardware score	Quality score	Generation/step
Reference Solution	0.6563	**0.8876**	0.4251	–
GENIO (SGP)	0.6707	0.8778	0.4637	15
NSGA-II	**0.6712**	0.8175	**0.5249**	20
MOEA/D	0.5900	0.7366	0.4433	3
PSO	0.5607	0.7610	0.3604	5
PPO	0.4841	0.7409	0.2273	16
Envelope Q-Learning	0.5157	0.7223	0.3091	20

Table 4
Run-level performance of the evaluated search methods.

Method	Best score [Q1–Q3]	Normalized AUC [Q1–Q3]	Valid evaluations
GENIO (SGP)	0.6483 [0.6364–0.6614]	**0.635** [0.621–0.648]	**92.8%**
NSGA-II	**0.6577** [0.6513–0.6624]	0.573 [0.564–0.583]	56.8%
MOEA/D	0.5511 [0.5256–0.5758]	0.542 [0.523–0.563]	70.1%
PSO	0.5522 [0.5437–0.5607]	0.529 [0.512–0.546]	1.8%
PPO	0.1906 [0–0.4069]	0.057 [0–0.159]	2.5%
Envelope Q-Learning	0 [0–0.1289]	0 [0–0.034]	1%

5.1.1. Convergence analysis

The proposed custom SGP algorithm is evaluated against representative optimization strategies spanning different paradigms: multi-objective evolutionary algorithms (NSGA-II and MOEA/D), swarm intelligence (PSO), and reinforcement learning (PPO and Q-Learning). All methods operate within GENIO framework on the same pipeline encoding and are assessed using identical synthesis-derived metrics to guarantee a fair and controlled comparison.

Table 3 presents the highest-scoring feasible configuration observed for each method within the common horizon of 20 generations. The reported values include the best achieved scalarized objective (IndividualScore), along with the corresponding QualityScore and HardwareScore under the balanced configuration ($\alpha = \beta = 0.5$).

The results indicate that evolutionary-based methods consistently identify competitive trade-off configurations within the allocated evaluation budget. NSGA-II achieves the highest scalarized objective value and the highest segmentation quality. The proposed method obtains the best hardware efficiency among all strategies, achieving a favorable balance between application-level performance and FPGA resource utilization.

PSO, PPO, and Envelope Q-Learning produced lower run-level outcomes under this protocol. Their median fraction of valid candidate evaluations was approximately 1.8% for PSO and 2.5% for PPO, while only one of the ten Envelope runs produced a feasible solution.

To characterize variability across independent executions, Table 4 summarizes the best feasible score, the area under the best-so-far score curve, and the proportion of valid evaluations within the common recorded horizon. Each independent run contributes a single observation to these summaries.

The run-level results reveal a distinction between final solution quality and search trajectory. NSGA-II has the highest median best feasible score, whereas SGP has the highest median area under the best-so-far curve and the highest proportion of valid evaluations. This pattern is consistent with SGP finding competitive solutions earlier in the recorded search, without establishing superiority in final score. The lower success rates of the reinforcement-learning methods also affect their run-level summaries; runs without a feasible configuration are retained rather than discarded. Fig. 4 displays the corresponding median trajectories and interquartile bands.

To provide a more detailed view of the hardware implications of the explored configurations, Fig. 5 reports the evolution of mean FPGA resource utilization for each generation. The figure presents a breakdown of the main hardware resources (LUTs, Flip-Flops, BRAM, and DSPs) across all evaluated methods.

This analysis complements the scalarized HardwareScore by exposing the underlying resource distribution of the synthesized designs.

Table 5
First generation in which each method reaches at least 95% of the best global retrospective score.

Method	Individual score	Hardware score	Quality score	Generation
GENIO (SGP)	0.6415	0.8498	0.4332	5
NSGA-II	0.6499	0.8308	0.4690	16
MOEA/D	–	–	–	–
PSO	–	–	–	–
PPO	–	–	–	–
Envelope	–	–	–	–

While the aggregated score provides a compact evaluation metric, the per-resource breakdown reveals how different optimization strategies navigate the hardware design space and prioritize specific resource types.

The results show that the proposed genetic programming method achieves, on average, the lowest utilization across the individual FPGA resource categories considered in the analysis. The only exception is observed in a limited number of PPO runs, where slightly lower BRAM utilization is occasionally obtained. Overall, however, the proposed method consistently identifies configurations with reduced FF, LUT, BRAM, and DSP requirements, indicating a more effective optimization of hardware resource usage across the explored design space.

The evolution of resource utilization further highlights differences in the search behavior of the evaluated methods. The proposed genetic programming approach exhibits a clear decreasing trend during the first stages of the optimization process, with improvements becoming progressively smaller and beginning to stabilize around generation 10. This behavior suggests that the method is able to rapidly identify resource-efficient regions of the design space before converging toward a stable set of configurations.

NSGA-II also shows a progressive reduction in resource utilization throughout the optimization process, although it does not reach the resource-efficiency levels achieved by the proposed method. In contrast, the remaining strategies exhibit considerably more irregular behavior, with no consistent trend toward lower resource utilization. For some methods, individual generations even contain no candidates producing valid simulation results, further illustrating the difficulty of maintaining a stable search trajectory within the explored design space.

To quantify convergence speed, Table 5 reports the first generation at which each method reaches a feasible configuration with a retrospective score of at least 95% of the best global solution identified across all methods, which is obtained by NSGA-II at generation 20.

The proposed SGP method reaches the predefined performance threshold as early as generation 5, whereas NSGA-II first attains a

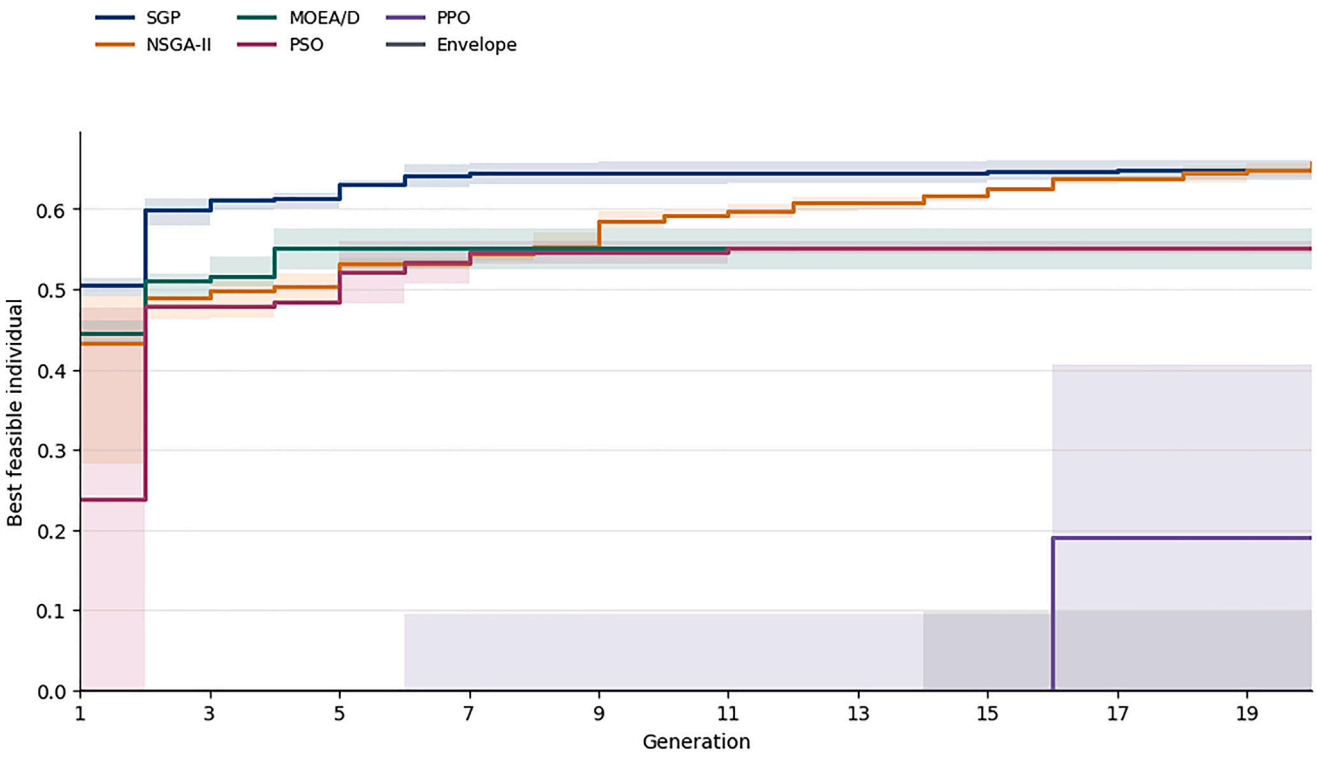


Fig. 4. Median best-so-far retrospective score and interquartile band across runs for each search method.

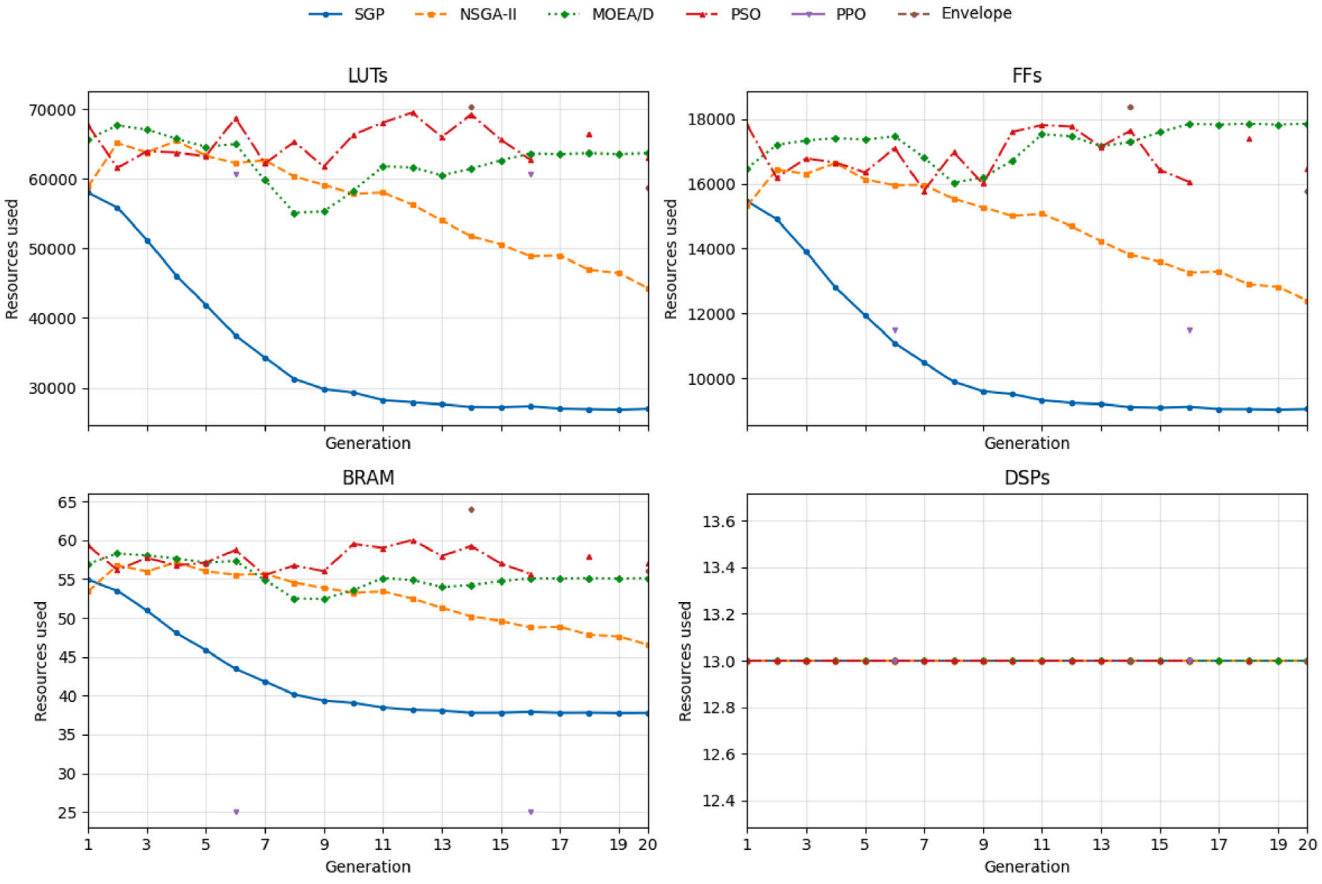


Fig. 5. Average resource utilization per generation for each search strategy under evaluation.

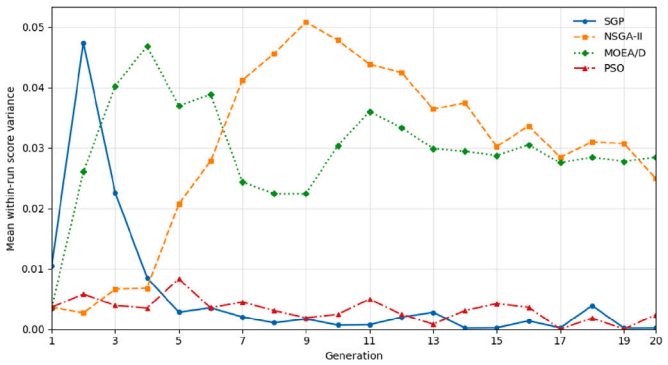


Fig. 6. Evolution of population variance across generations.

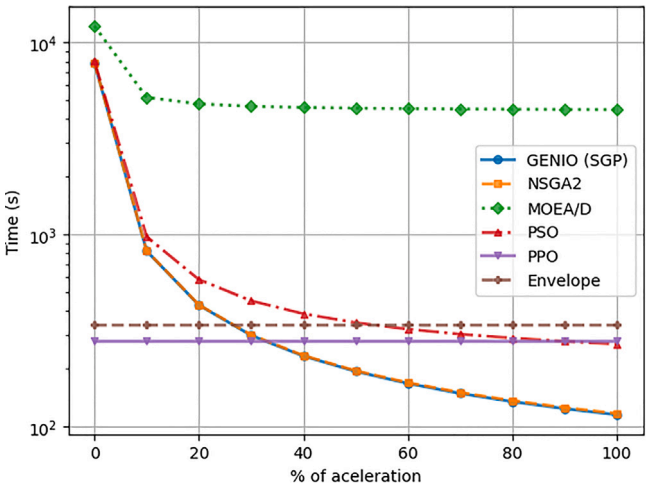


Fig. 7. Runtime scalability of different search strategies under varying degrees of parallel evaluation.

qualifying configuration at generation 16. The remaining methods do not reach the 95% threshold within the evaluated search horizon. Across the independent runs, three of the four executions of both SGP and NSGA-II reach the target, although SGP exhibits an earlier attainment among the successful runs. This behavior is consistent with the ability of the proposed method to exploit promising structural characteristics already present in some of the initial individuals. Given the heterogeneous nature of the search space, randomly generated configurations may already contain relatively efficient pipeline structures. The structured representation and search operators employed by SGP allow these configurations to be preserved and progressively refined, facilitating the early identification of high-quality feasible solutions.

This behavior is particularly relevant in practice, since each candidate evaluation involves computationally expensive hardware synthesis. Under constrained evaluation budgets, identifying competitive configurations at an early stage of the search can substantially reduce the number of evaluations required before high-quality solutions become available. In this respect, SGP reaches the predefined target considerably earlier in the observed search trajectories, whereas NSGA-II continues improving over subsequent generations and ultimately identifies the highest-scoring individual configuration. Therefore, the results reveal a different search behavior between the two strategies: SGP concentrates high-quality solutions earlier in the optimization process, while NSGA-II benefits from a longer search horizon to achieve a slightly higher final score.

Overall, these results suggest that the difficulty of the considered design space arises not only from its size, but also from its structural and discrete nature. Such characteristics can hinder optimization strategies whose search dynamics are less suited to preserving and refining promising structural configurations, reinforcing the importance of explicitly considering the representation of candidate hardware pipelines during design space exploration.

5.1.2. Population diversity analysis

Fig. 6 illustrates the evolution of intra-population variance across generations.

In population-based optimization, within-generation score variance provides an indication of the diversity of the candidate solutions explored during the search. Higher variance generally reflects the simultaneous exploration of regions with different levels of performance, whereas a sustained reduction in variance may indicate an increasing concentration of the population around a narrower region of the design space.

The proposed SGP method exhibits a pronounced increase in variance during the initial generations, followed by a rapid reduction that reaches values close to zero after approximately generation 8. Only small transient increases are observed thereafter, indicating that the population becomes progressively concentrated around solutions with similar performance. In contrast, NSGA-II maintains substantially higher variability throughout the search. Its variance increases during the first half of the optimization process, reaching its maximum around generation 9, and subsequently decreases while remaining clearly above the levels observed for SGP. MOEA/D similarly preserves relatively high variability across most generations, with noticeable fluctuations but no comparable convergence toward near-zero variance.

PSO exhibits consistently low score variance throughout the optimization process. However, this behavior should not necessarily be interpreted as evidence of stable convergence, since the reduced number of valid candidate evaluations obtained by this method limits the diversity represented in the variance estimate. Consequently, the joint analysis of variance and evaluation validity provides a more reliable characterization of the search dynamics.

For SGP, the stabilization of the variance at low values after approximately generation 10 is consistent with the convergence trend observed in the resource-utilization analysis. This behavior could therefore provide complementary information for defining an early-stopping criterion, although variance should be considered together with objective improvement and the proportion of valid evaluations rather than used as an isolated convergence indicator.

5.1.3. Parallel scalability analysis

Since hardware synthesis dominates evaluation cost, scalability under parallel execution is a critical practical consideration. Fig. 7 shows per-generation runtime as a function of parallel evaluation capacity. In this context, the "% of acceleration" represents the proportion of individuals within a generation that can be evaluated concurrently. A value of 0% corresponds to a fully sequential execution, where individuals are synthesized and evaluated one after another. Conversely, 100% represents a fully parallel execution scenario, where each individual can be evaluated independently on a separate physical CPU core.

Each generation consists of two phases: (i) a preparation stage (selection, crossover, mutation, or policy update), and (ii) an evaluation stage, where candidate configurations are synthesized and assessed independently. The runtime breakdown of these two phases for each method is reported in Table 6, highlighting the significant differences in preparation overhead. The evaluation time reported in the table corresponds to the average time required to evaluate a single individual in the design space. This evaluation comprises both the functional assessment of the candidate pipeline and the hardware synthesis stage required to obtain its implementation metrics. Therefore, the differences in computational overhead among the optimization methods arise primarily from their respective preparation stages.

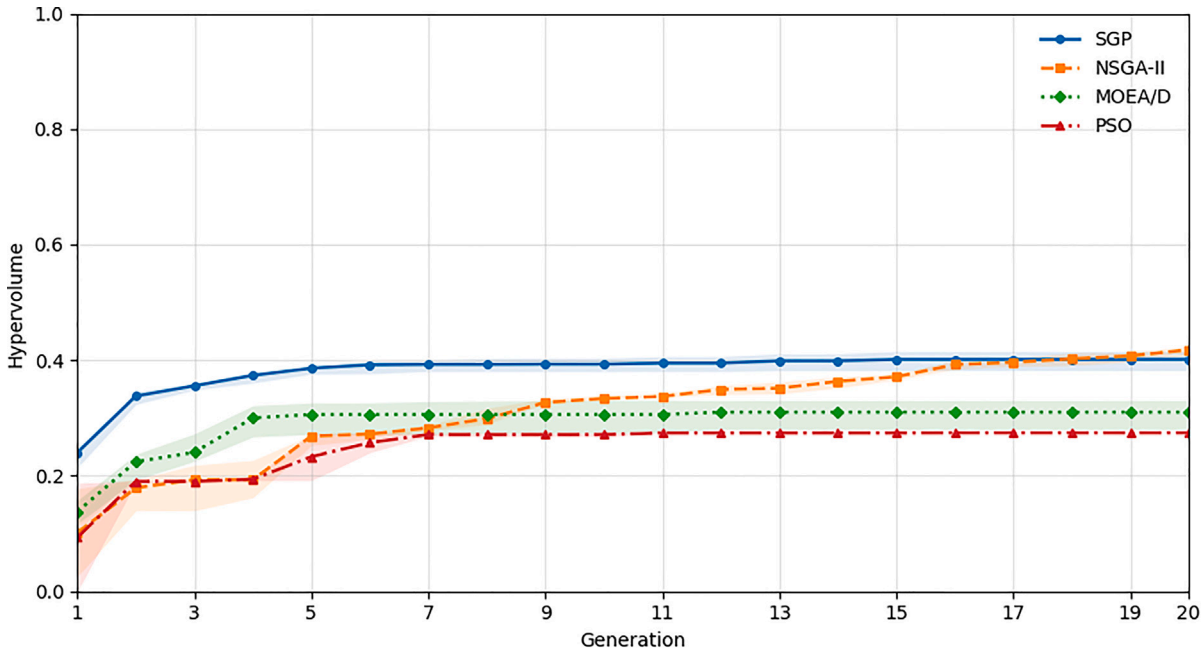


Fig. 8. Hypervolume analysis of different search strategies.

Table 6
Method runtimes.

Method	Preparation (s)	Evaluation (s)
GENIO (SGP)	36.43	78.34
NSGA-II	38.18	78.34
MOEA/D	4402.20	78.34
PSO	190.12	78.34
PPO	199.20	78.34
Envelope	261.80	78.34

In principle, the evaluation of individual candidates is embarrassingly parallel, as each configuration can be synthesized and evaluated independently. However, the extent to which this parallelism can be effectively exploited depends on the algorithmic structure of each method, particularly on synchronization requirements and interdependencies between individuals.

Population-based methods benefit significantly from parallelization because their evaluation phase can be executed concurrently with minimal coordination. As the available parallel capacity increases, runtime decreases sharply until the serial preparation stage becomes dominant, in accordance with Amdahl's Law. Beyond this point, speedup gains diminish due to intrinsic sequential overhead and synchronization costs.

Reinforcement learning methods exhibit limited scalability because policy updates rely on sequential interaction with the environment. This introduces a strong temporal dependency between evaluations, restricting the degree of effective parallelism. Consequently, RL may achieve lower runtime under fully sequential conditions, but is rapidly outperformed by population-based methods once approximately 30% parallelism is available.

Among evolutionary approaches, the proposed method demonstrates the steepest runtime reduction under parallel evaluation, indicating efficient task decomposition and minimal coordination overhead. In contrast, MOEA/D exhibits weaker scalability due to its neighborhood-based structure, where partial dependencies between subproblems introduce additional synchronization constraints that limit fully independent evaluation.

5.1.4. Pareto quality assessment

Fig. 8 presents a comparative analysis of the accumulated Hypervolume (HV) achieved by each population-based search strategy across generations. The Hypervolume indicator is used as a quantitative measure of Pareto set quality [42], as it simultaneously captures convergence toward optimal trade-offs and diversity along the frontier in the bi-objective space defined by *QualityScore* and *HardwareScore*.

Although the proposed method employs a scalarized fitness function for selection, the multi-objective integrity of the search is maintained at the population level. Each generation produces a diverse set of candidate solutions distributed in the bi-objective space, from which we reconstruct the Pareto frontier post-generationally by extracting the non-dominated subset from the cumulative population. For Hypervolume computation, the non-dominated subset of individuals in each generation is extracted, and HV is calculated with respect to a common reference point. This demonstrates that the inherent stochasticity of the Genetic Programming operators maintains sufficient genetic diversity to cover the trade-off space between image quality and hardware footprint, while avoiding the selection pressure stagnation typical for high-dimensional Pareto optimization.

The proposed SGP method achieves the highest Hypervolume during the early stages of the search. Its HV increases rapidly during the first generations, reaching approximately 0.38–0.39 by generation 5 and remaining relatively stable thereafter, with only modest improvements during the second half of the optimization process. This behavior indicates that SGP identifies high-quality trade-off solutions early in the search.

NSGA-II exhibits a slower but sustained improvement throughout the optimization process. Although its Hypervolume remains below that of SGP during most of the search, it progressively closes the gap and reaches comparable values in the final generations, slightly surpassing SGP at the end of the evaluated horizon. In contrast, MOEA/D improves mainly during the first generations and subsequently stabilizes at an HV of approximately 0.31, remaining below both SGP and NSGA-II.

PSO shows a similar early increase but reaches a plateau at approximately generation 7–9, with Hypervolume values around 0.27. No substantial improvement is observed thereafter. This indicates that, under the considered discrete and hardware-constrained design space, PSO explores a more limited portion of the attainable quality–hardware trade-off region than SGP and NSGA-II.

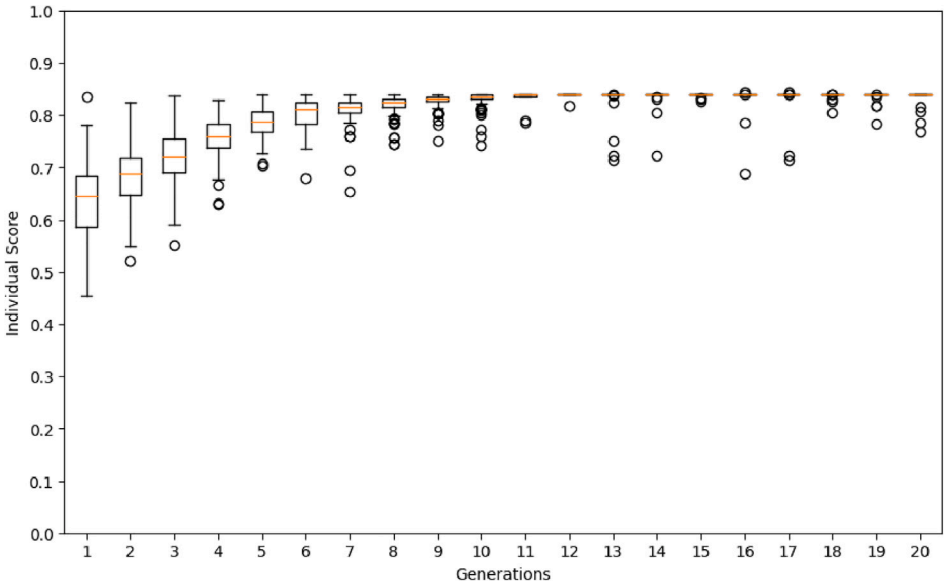


Fig. 9. Overall generation score evolution.

5.2. Effect of preference weights in scalarized fitness

In addition to cross-method comparison, we analyze how the scalarization weights in Eq. (5) influence the optimization outcome. Three representative preference scenarios are considered, each producing a distinct set of pipeline configurations for the chromatic trap segmentation task. The first three scenarios are designed to explore different balances between QualityScore and HardwareScore, whilst the fourth scenario further investigates the internal weighting of the HardwareScore by assigning different priorities to its individual resource-utilization terms.

5.2.1. Segmentation mask quality and resources

In this configuration, alpha and beta are both assigned a value of 0.5 in Eq. (5), ensuring that no term is favored. Similarly, the weights in Eq. (4) are uniformly set to 0.25.

As shown in Fig. 9, the optimization process halts at the 10th generation, where most individuals in the population converge within a narrow variance range.

Figs. 9 and 10 illustrate the convergence dynamics of the population across generations. Under the balanced configuration, convergence is achieved within ten generations, after which the population exhibits stable behavior according to the stopping criterion. The results further indicate that the resource-related objective stabilizes earlier than the quality-related objective, suggesting that reductions in hardware utilization are attained more rapidly than improvements in detection performance within the explored design space.

5.2.2. Segmentation mask quality

This configuration aims to guide the optimization process toward solutions that prioritize outputs better suited for the post-segmentation stages, which extract regions of interest for insect pest identification.

To achieve this, alpha is assigned a value of 0.75 and beta a value of 0.25, resulting in a total score for each individual that is intentionally biased toward the quality of the output mask, aligning with the primary objective of the target application. This weighting scheme places greater emphasis on the quality term while minimizing the effect of resource optimization in the evaluation process.

Figs. 11 and 12 show the behavior of the genetic search in this scenario. As can be observed, there is an impact on the number of generations required to converge due to some instability issues during the exploration process caused by the excessive bias assigned to the quality term in the fitness function (see generations 8 to 10 in Fig. 11). Stability in the evolution and the steadily progressive improvement of the individual metrics is maintained.

5.2.3. Resources optimization

Similar to the previous scenario but oriented toward prioritizing optimal FPGA resource utilization over segmentation mask quality, the fourth experiment assigned a value of 0.25 to alpha and 0.75 to beta. This configuration biases the optimization process in favor of pipelines that perform reasonably well while requiring fewer logic resources.

Figs. 13 and 14 illustrate the evolution of the individual metrics and the overall score throughout the optimization process. Although fewer generations are required compared to the previous scenario, the average resource utilization for the same generation is notably lower. However, it can also be observed that the solutions identified during the search process do not achieve the same level of quality in the output segmentation mask.

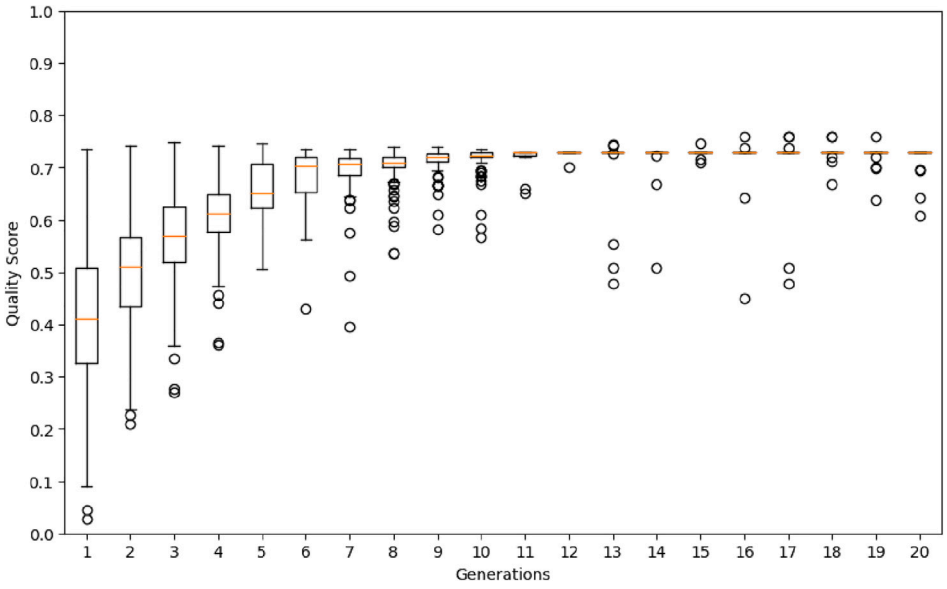
5.2.4. Impact of hardware resource weighting

In this configuration, both α and β in Eq. (5) are set to 0.5, thereby assigning equal importance to the QualityScore and the HardwareScore. In addition to this balanced scalarization, the individual terms of the hardware metric in Eq. (4) are weighted differently in order to analyze the effect of prioritizing specific FPGA resources.

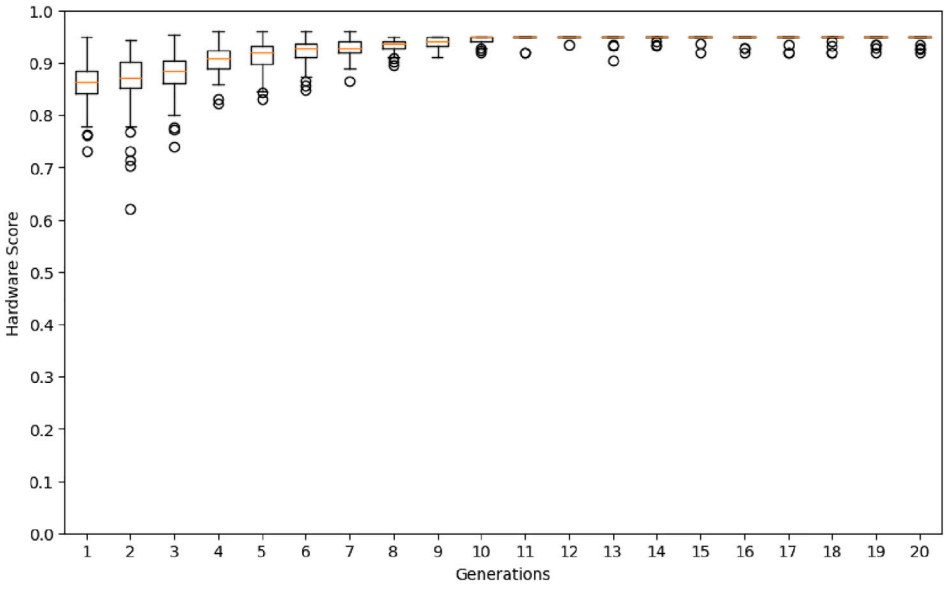
For each experiment, one hardware resource term is assigned a dominant weight of 0.7, while the remaining three terms are assigned a weight of 0.1 each. This preserves a normalized total weight of 1.0 while allowing the optimization process to be biased toward reducing the utilization of a specific resource type. In this way, the analysis evaluates how the proposed method reacts when the designer prioritizes the minimization of LUTs, Flip-Flops, BRAMs, or DSPs within the hardware-aware optimization process.

Fig. 15 shows the evolution of the resource utilization obtained by the best-performing individual at each generation for each hardware-resource weighting configuration. This representation allows us to observe how emphasizing a particular resource affects the hardware profile of the selected solutions throughout the optimization process.

On the other hand, Fig. 16 shows the distribution of the HardwareScore values defined in Eq. (4) across the population for each hardware-resource weighting configuration. Since this analysis is represented through boxplots, it provides a broader view of the optimization behavior by capturing the dispersion, median tendency, and variability of the population rather than focusing exclusively on the best-performing individual.



(a) Quality.



(b) Resources.

Fig. 10. Evolution of the quality and resource terms.

6. Discussion

This section provides insights into the best results achieved across the studied scenarios and conducts a comparative analysis against the metrics obtained from the currently deployed solution, which was designed and optimized by an expert.

We draw conclusions regarding the suitability of each tested configuration, focusing on the segmentation quality of insect detection in actual chromatic trap images and the FPGA platform's resource utilization. Furthermore, the time required for each solution to meet the stopping criterion in each test is also analyzed.

6.1. Expert solution

The manually designed reference pipeline was obtained through an iterative refinement process in which design decisions were progressively adjusted based on experimental evaluation of segmentation outputs. The configuration was validated in field deployments and served as the operational baseline for comparison in this study.

According to the evaluation framework defined in Eq. (5), the reference solution achieved an Individual Score of 0.6563. Decomposing this value into its constituent objectives, the solution attained a HardwareScore of 0.8876, indicating highly efficient FPGA resource utilization, while the QualityScore reached 0.4251, reflecting moderate agreement with the ground-truth annotations under the IoU-based evaluation criterion.

The observed imbalance between hardware efficiency and segmentation quality illustrates the inherent trade-off between competing objectives in hardware-aware pipeline design. Configurations that prioritize conservative resource usage may limit the expressive capacity of the processing stages, thereby constraining segmentation performance. Conversely, improving segmentation quality typically requires additional computational structures, which may negatively affect resource efficiency. Moreover, the combinatorial growth of possible parameter

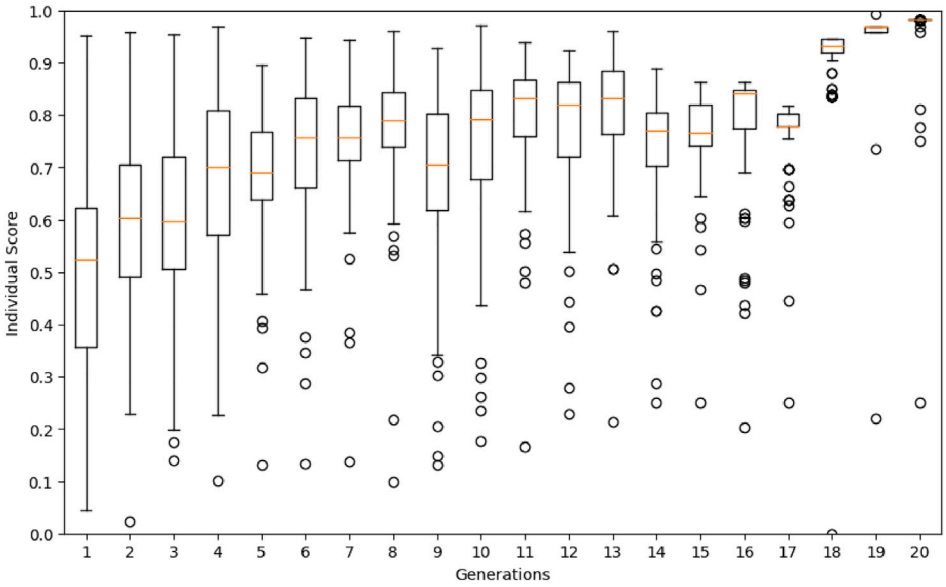


Fig. 11. Overall generation score evolution favoring the quality term.

and structural configurations makes exhaustive manual exploration impractical, reinforcing the need for systematic and automated search strategies.

6.2. Search strategies performance

In terms of overall performance, the results show clear differences in robustness and search effectiveness among the evaluated strategies. Our purposed algorithm and NSGA-II consistently identify the highest-quality configurations, while MOEA/D achieves more moderate performance. In contrast, PSO and the reinforcement learning-based approaches exhibit a much lower proportion of valid evaluations, limiting their ability to explore competitive regions of the design space within the available budget.

The proposed SGP method provides a favorable balance between solution quality, robustness, and evaluation efficiency. Although NSGA-II ultimately identifies the best individual configuration, SGP maintains stronger performance throughout a larger portion of the search and generates valid candidates more consistently. This behavior is particularly relevant in synthesis-driven design space exploration, where unsuccessful evaluations directly increase computational cost.

Regarding convergence, SGP reaches high-quality regions of the search space substantially earlier, while NSGA-II preserves broader exploration for longer and continues improving during later generations. The population-variance analysis supports this behavior, showing a rapid reduction and stabilization of variability for SGP, in contrast to the higher diversity maintained by NSGA-II. Overall, the proposed method is particularly effective under bounded evaluation budgets, where early identification of competitive and feasible configurations is a key requirement.

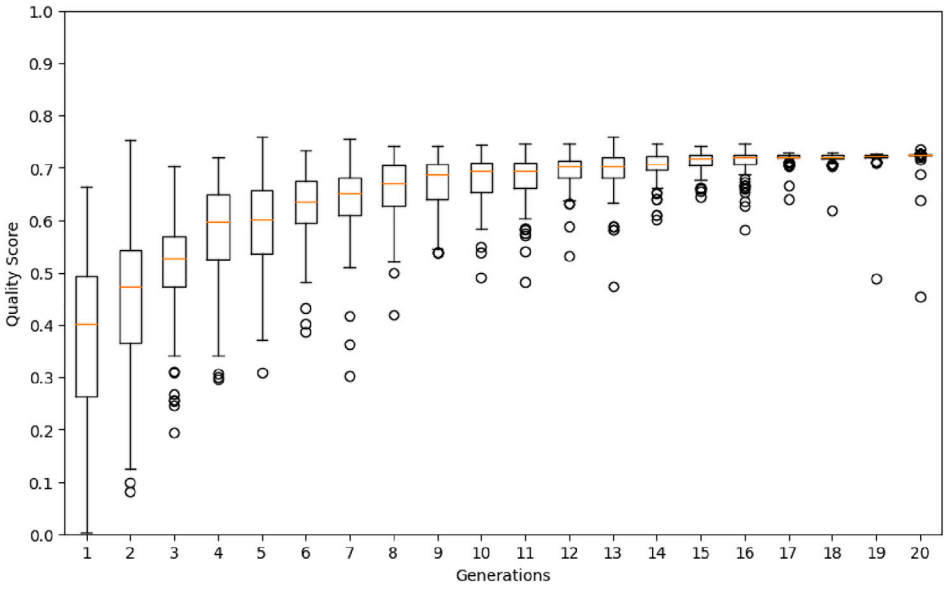
From a parallelization perspective, the experiments highlight that computational scalability is influenced not only by the evaluation phase — which is inherently parallelizable in population-based methods — but also by intergenerational preparation overhead. MOEA/D exhibited the poorest scalability due to its higher computational cost during neighborhood-based coordination and subproblem management, resulting in intergenerational bottlenecks that limited runtime gains under increased parallel capacity. Other population-based approaches, while incurring higher execution times under fully sequential conditions, benefited significantly from moderate levels of parallelism (around 30%), where evaluation tasks could be distributed efficiently. Under such conditions, their generation times became comparable to those of reinforcement learning methods. However, reinforcement learning approaches showed limited scalability improvements, as their inherently sequential policy update mechanisms constrain parallel execution. These findings underline that, in hardware-constrained optimization contexts, algorithm selection must consider not only solution quality and convergence speed but also intrinsic parallel efficiency and preparation overhead, which can critically impact practical deployment feasibility.

6.3. Balanced metrics & Quality and resources

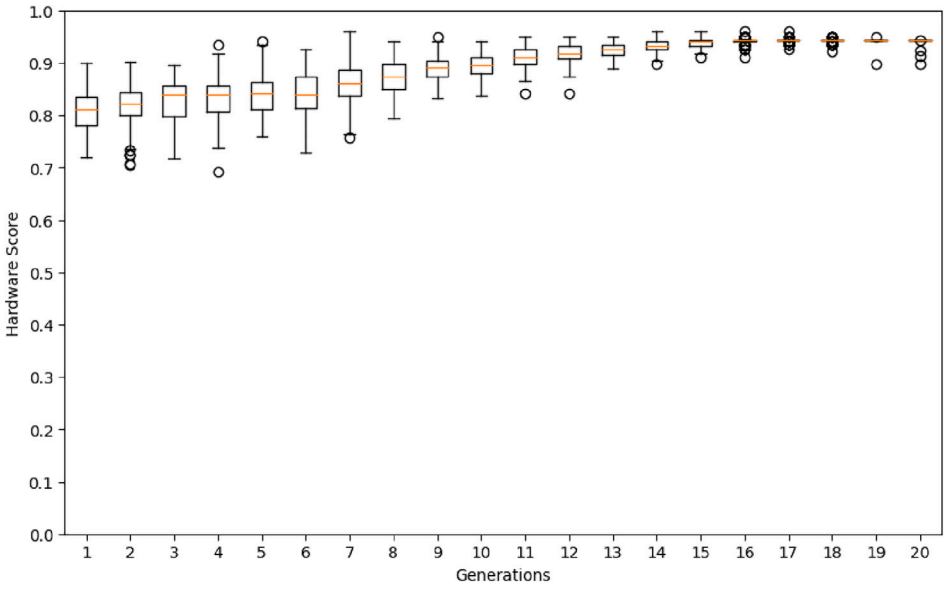
The automated exploration of parameter configurations enables substantially broader coverage of the design space compared to manual iterative procedures. In the present formulation, the search process is guided by a balanced scalarization of the QualityScore and HardwareScore, assigning equal weights to both objectives. Since these metrics quantify distinct and partially conflicting aspects of each candidate configuration, improvements in one objective may negatively influence the other.

Although both QualityScore and HardwareScore are normalized, their effective ranges within the explored solution space do not span the full theoretical interval of the normalization. Empirical analysis shows that, for the considered dataset and pipeline templates, QualityScore values typically range between 0 and 0.72, whereas HardwareScore values lie approximately between 0.70 and 0.96. Consequently, the attainable values are constrained by the structural limits of the design space rather than by the normalization bounds themselves. This mismatch must be considered when interpreting aggregated scores, as the relative contribution of each objective depends not only on its assigned weight but also on its empirical variability.

The difference in dynamic range between the objectives necessitates careful design of the aggregation mechanism. To mitigate scale imbalance, a generation-wise min–max normalization is applied prior to scalarization. This procedure ensures that both objectives contribute proportionally within each generation, preventing dominance effects caused by differing value distributions. However, because normalization is performed locally at the generation level, the effective reference bounds shift over time, potentially reducing global comparability across generations and increasing sensitivity to local optima.



(a) Quality.



(b) Resources.

Fig. 12. Individual metrics favoring quality.

6.4. Imbalances in quality and resources metrics

Another test conducted involved altering the weighting of the Quality and Resources metrics in their contribution to the overall evaluation metric. This weighting adjustment was applied with factors of 25% and 75%, with both scenarios analyzed where the Resources metric is either weighted three times as heavily as the Quality metric or vise versa.

In both runs, the primary difference observed was in the convergence speed of the solution search process. When Quality was weighted more heavily, the population converged at generation 19, whereas assigning greater weight to Resources resulted in convergence at generation 13.

Comparing the progression of the individual Quality and Resources metrics in the case where Quality carries the greater weight Fig. 12 versus where Resources is more heavily weighted Fig. 14 reveals differences in the distribution and ranges of these metrics per generation. Increasing the weight of one metric over the other means the genetic simulation is driven more significantly by feedback from the weighted metric, making it the dominant focus. During the simulation, individual selection is oriented toward improving the overall evaluation and reducing population variability—a key criterion for early stopping. This reduction in variability subsequently impacts the individual metrics comprising the evaluation. Metrics with higher intrinsic variability are inherently more sensitive within the genetic simulation process.

In the considered design space, the empirically observed values of the Quality metric range approximately between 0 and 0.72, whereas the Resources metric typically lies between 0.70 and 0.96. These intervals correspond to globally evaluated, non-normalized scores derived from synthesis and application-level assessment. The difference in effective dynamic range between both objectives influences the sensitivity of the scalarized fitness function to variations in each term. As a result, depending on the assigned preference weights, the evolutionary process may exhibit different convergence behaviors, which explains

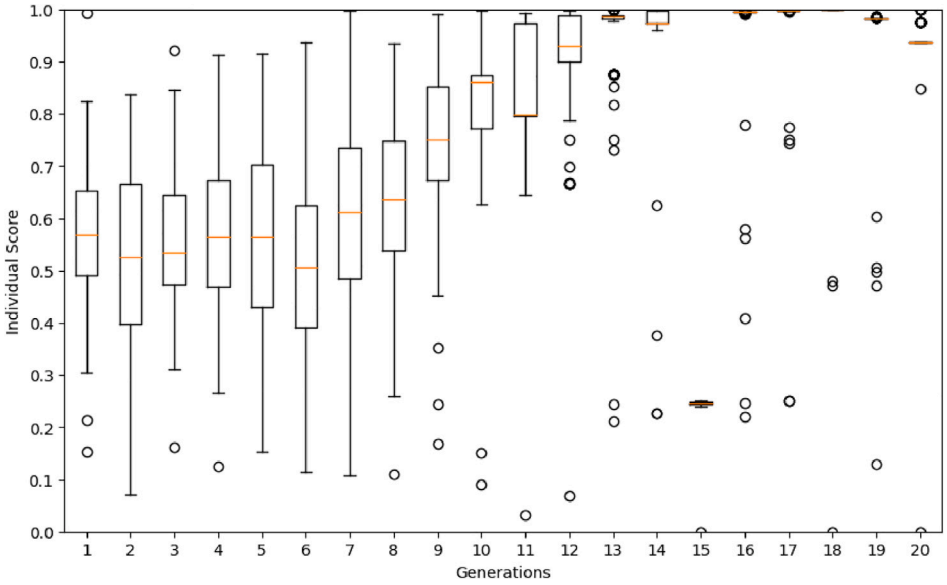


Fig. 13. Population scores favoring resources.

the variation in the number of generations required to satisfy the stopping criterion.

6.5. Analysis of resource-specific hardware weighting

The resource-specific weighting analysis shows that the effect of the hardware weights cannot be interpreted as a direct minimization of a single FPGA resource. Although the weighting configurations modify the selection pressure applied to FFs, LUTs, BRAMs, and DSPs, the resulting search trajectories are also strongly constrained by the feasibility limits of the target device.

This behavior is particularly evident in the Connected Components stage, for which GENIO can select between functionally equivalent BRAM- and LUTRAM-based implementations. The LUTRAM alternative introduces a substantially higher LUT demand and frequently exceeds the logic capacity available on the target FPGA. As a result, these configurations become infeasible and are progressively removed from the population. The resource constraints therefore impose an implicit feasibility pressure that can dominate the finer differences introduced by the individual hardware-resource weights.

Consequently, the weighting mechanism should be interpreted primarily as a means of biasing the search within the feasible region of the architectural space. A higher weight assigned to a given resource does not necessarily imply that the final solution will minimize that resource in isolation, since architectural feasibility and correlated resource consumption also influence the search trajectory. In particular, Fig. 15 shows that LUT-oriented weighting frequently leads to low utilization not only of LUTs, but also of FFs and BRAMs. This suggests that, among feasible configurations, these resources are partially correlated and that reducing logical complexity may favor globally smaller implementations. However, this behavior should be interpreted as an observed association rather than as evidence of a direct causal relationship between LUT reduction and the consumption of the remaining resources.

DSP utilization remains essentially unchanged across the different weighting schemes, indicating that the architectural alternatives considered in this experiment do not substantially modify the operations mapped to DSP resources. The limited differentiation observed for some of the remaining weighting configurations can also be explained by the structure of the explored design space. Most upstream streaming stages do not provide functionally equivalent micro-architectural alternatives with strongly differentiated resource profiles, which restricts the number of meaningful resource-specific trade-offs available to the optimizer.

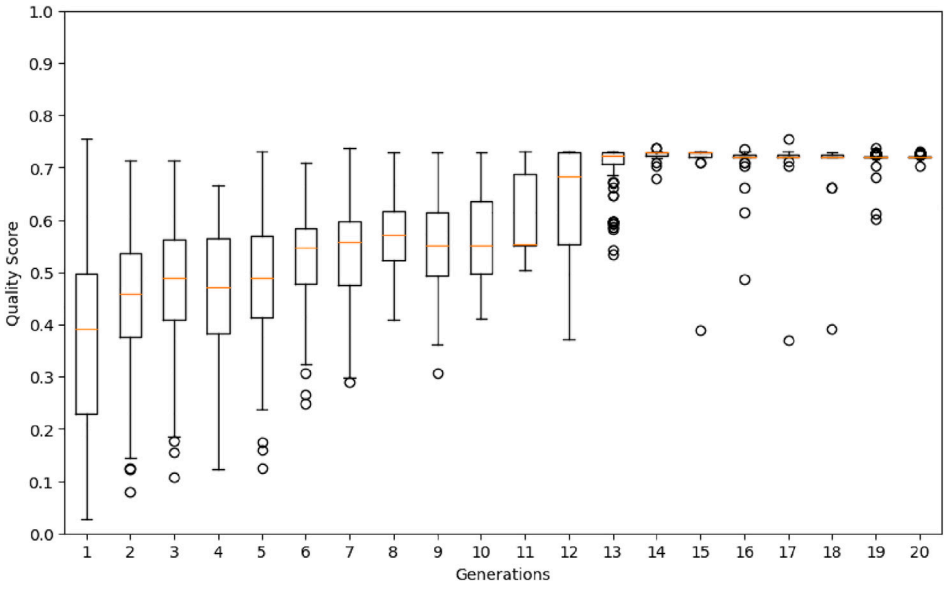
Overall, these results indicate that independent hardware weights are most effective when interpreted as search biases that interact with both architectural alternatives and device-specific feasibility constraints. Their influence is expected to become more pronounced as the design space incorporates a larger number of functionally equivalent implementations with distinct resource profiles. In this context, the weighting mechanism provides the flexibility required to steer the search according to resource bottlenecks and heterogeneous hardware trade-offs.

7. Conclusions

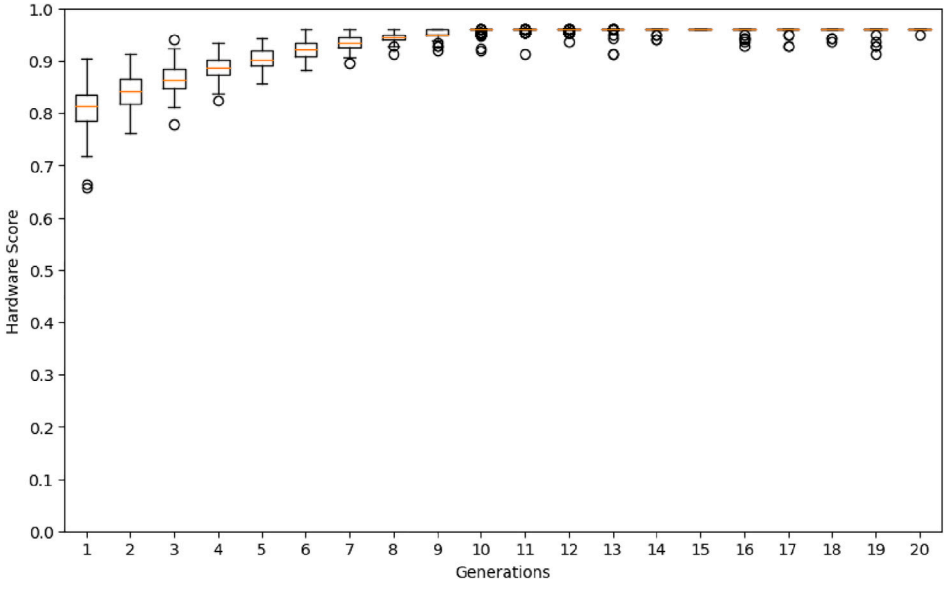
This work has presented GENIO, a synthesis-driven framework for the automated exploration and optimization of image processing pipelines targeting hardware-constrained FPGA platforms. By formalizing the pipeline structure and hyperparameter configuration, the framework enables systematic exploration of a highly combinatorial design space that would be impractical to address through manual tuning.

A comparative evaluation of evolutionary algorithms, swarm intelligence, and reinforcement learning methods was conducted under identical synthesis and evaluation budgets. Results indicate that evolutionary strategies — particularly the proposed structural genetic algorithm and NSGA-II — provide the most effective trade-offs between segmentation quality and hardware resource utilization. While NSGA-II achieved the highest final score, the proposed method demonstrated faster convergence, identifying near-optimal solutions substantially earlier. In contrast, swarm-based and reinforcement learning approaches showed limited effectiveness under constrained evaluation budgets.

The convergence and scalability analyzes further confirm that population-based evolutionary search offers stable optimization dynamics and strong parallel efficiency in synthesis-driven environments. Overall, the results support the suitability of the proposed method for automated, hardware-aware pipeline optimization in embedded vision systems.



(a) Quality.



(b) Resources.

Fig. 14. Individual metrics favoring resources.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

The authors declare that the use of artificial intelligence in this work was strictly limited to language editing and manuscript preparation tasks. During the preparation of this article, ChatGPT 5.6 was used exclusively to improve language clarity and readability. Following the use of this tool, the authors carefully reviewed and revised the content as necessary and assume full responsibility for the final content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is partially funded by the Ministry of Economy and Competitiveness (MINECO) of the Government of Spain under the project OASIS (PID2023-148285OB-C41), jointly with the PERTE-Chip grants for Chairs (UCLM Chair, TSI-069100-2023-0014) from the Ministry of Digital Transformation and Public Function, through the FEDER and by the Regional Government of Castilla-La Mancha under the projects REACH (SBPLY/24/180225/000105), UCLM University Staff Training Scholarship Programme and UCLM Grants for Stays at Universities and Research Centers Abroad (2025).

Data availability

The GENIO framework and all datasets used in this article are openly available on Zenodo at https://doi.org/10.5281/zenodo.20020791.

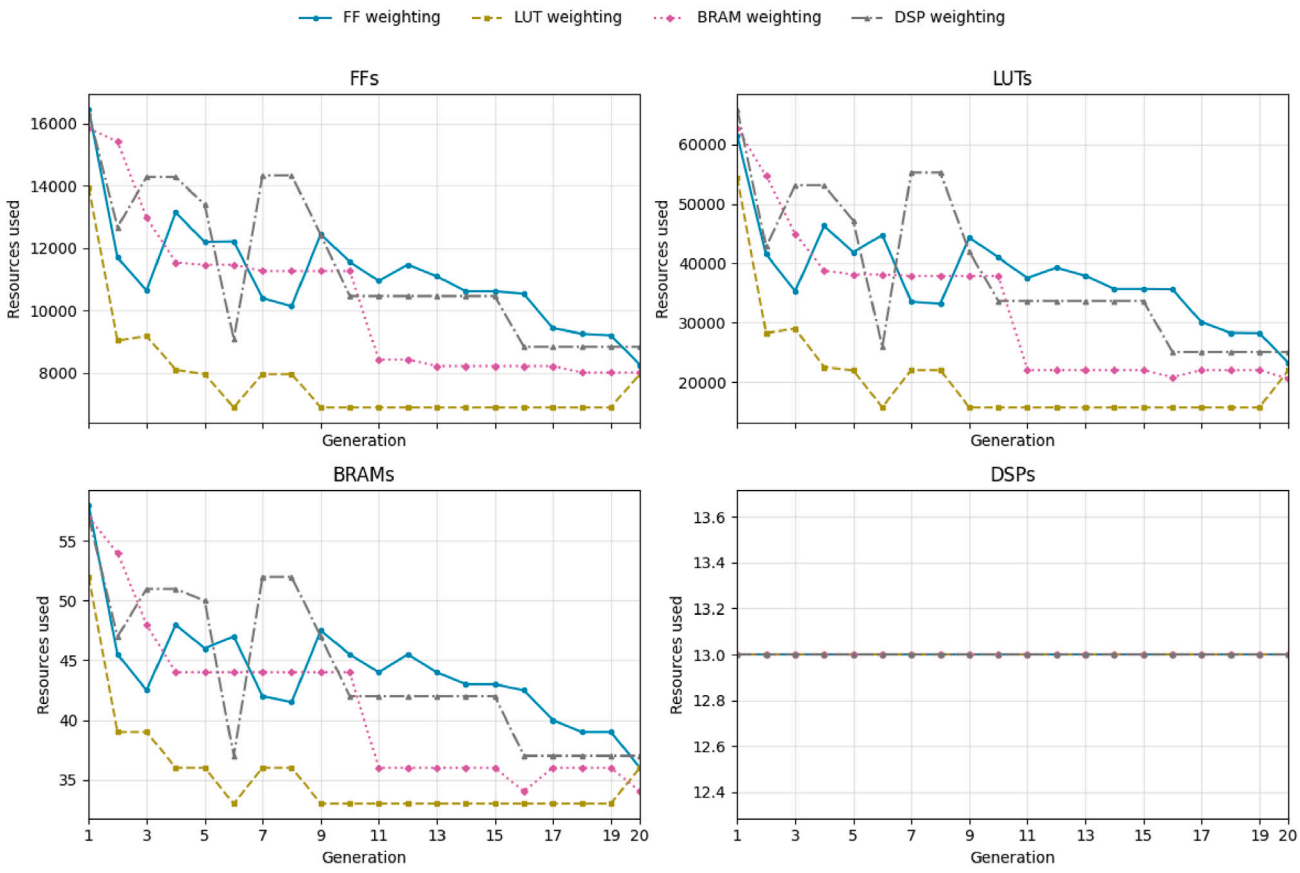


Fig. 15. Evolution of resource utilization for the best-performing individual under different hardware-resource weightings.

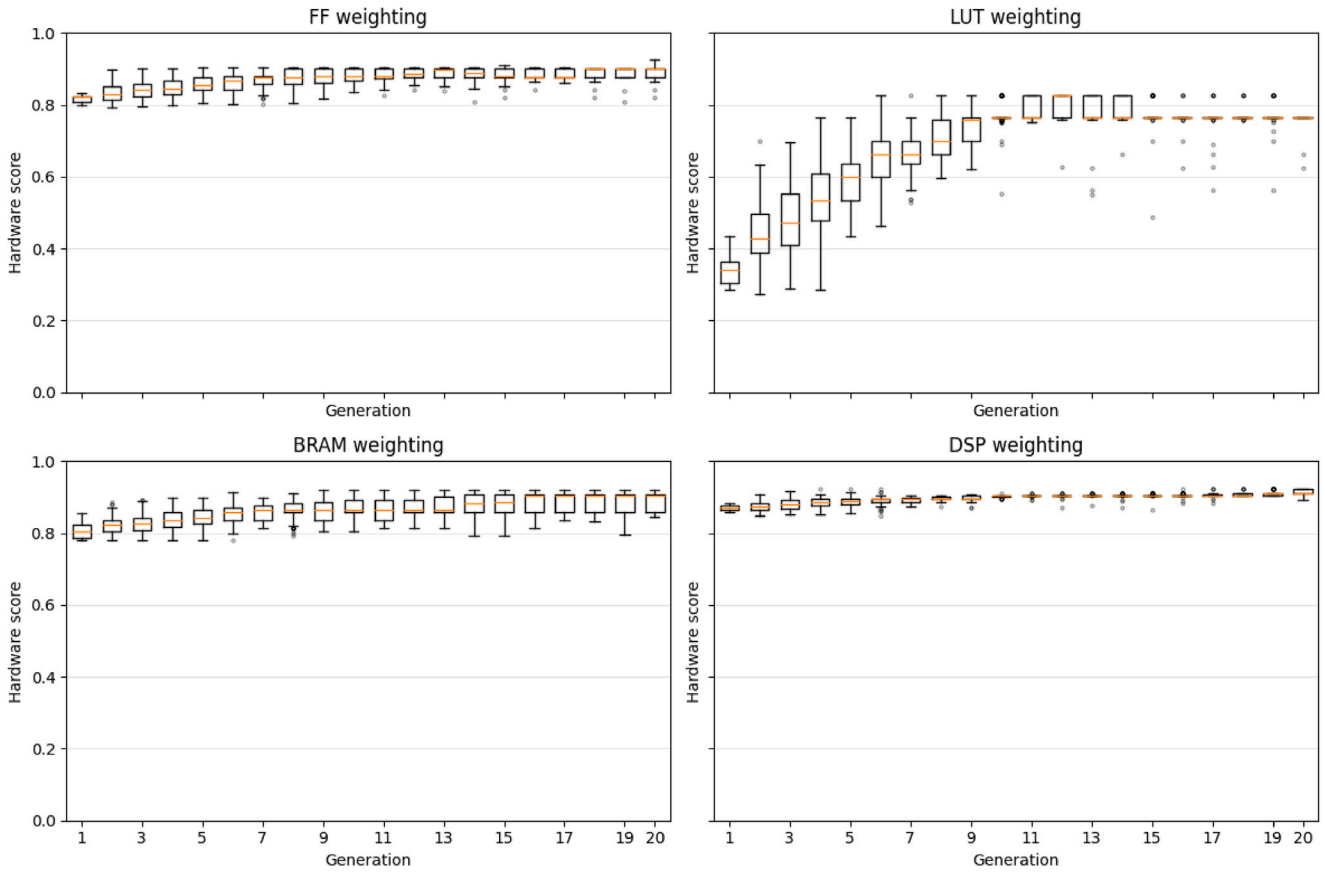


Fig. 16. Distribution of the HardwareScore across the population under different hardware-resource weightings.

References

[1] Moshe Sipper, High per parameter: A large-scale study of hyperparameter tuning for machine learning algorithms, Algorithms 15 (9) (2022) 315.

[2] Abir Chowdhury, Malik Magdon-Ismail, Bülent Yener, Quantifying error contributions of computational steps, algorithms and hyperparameter choices in image classification pipelines, 2019, arXiv preprint.

[3] Gabriel F.C. Campos, Sérgio M. Mastelini, Gustavo J. Aguiar, Rafael G. Mantovani, Lucas F. Melo, Sérgio Barbon, Machine learning hyperparameter selection for contrast limited adaptive histogram equalization, EURASIP J. Image Video Process. 2019 (1) (2019) 59.

[4] Diana Marculescu, Dimitrios Stamoulis, Ermao Cai, Hardware-aware machine learning: Modeling and optimization, in: Proceedings of the International Conference on Computer-Aided Design, ICCAD, 2018.

[5] Sergey Alyamkin, et al., Low-Power computer vision: Status, challenges, and opportunities, IEEE J. Emerg. Sel. Top. Circuits Syst. 9 (2) (2019) 411–421.

[6] Daniel I. Patrício, Renan Rieder, Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review, Comput. Electron. Agric. 153 (2018) 69–81.

[7] Yuzhen Lu, Sierra Young, A survey of public datasets for computer vision tasks in precision agriculture, Comput. Electron. Agric. 178 (2020) 105760.

[8] Jose L. Mira, et al., Benchmarking of computer vision methods for Energy-Efficient High-Accuracy olive fly detection on edge devices, Multimedia Tools Appl. 83 (2024) 81785–81809.

[9] Bertram Nickolay, Bernd Schneider, Stefan Jacob, Parameter optimisation of an image processing system using evolutionary algorithms, in: Computer Analysis of Images and Patterns, CAIP 1997, in: Lecture Notes in Computer Science, Vol. 1296, Springer, 1997, pp. 637–644.

[10] James Bergstra, Yoshua Bengio, Random search for Hyper-Parameter optimization, J. Mach. Learn. Res. 13 (10) (2012) 281–305.

[11] Charles Lo, Paul H.W. Chow, Model-Based optimization of High-Level synthesis directives, in: 2016 26th International Conference on Field-Programmable Logic and Applications, FPL, IEEE, 2016, pp. 1–10.

[12] Md Rubel Ahmed, Toshiaki Koike-Akino, Kieran Parsons, Ye Wang, AutoHLS: Learning-Driven design space exploration for High-Level synthesis, 2024, arXiv preprint.

[13] Alexandros Koronis, et al., Reverse-engineering optimization techniques of High-Level synthesis: Practical insights into accelerating applications with AMD-Xilinx vitis, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2025) Early Access.

[14] Quentin Gautier, et al., Sherlock: A Multi-Objective design space exploration framework, ACM Trans. Des. Autom. Electron. Syst. (2022).

[15] Charles Lo, Paul H.W. Chow, Multi-Fidelity optimization for High-Level synthesis directives, in: 2018 28th International Conference on Field-Programmable Logic and Applications, FPL, IEEE, 2018, pp. 272–279.

[16] Qi Sun, Tinghuan Chen, Siting Liu, Jianli Chen, Hao Yu, Bei Yu, Correlated Multi-objective Multi-fidelity optimization for HLS directives design, in: 2021 Design, Automation & Test in Europe Conference & Exhibition, DATE, 2021.

[17] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[18] Qingfu Zhang, Hui Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731.

[19] Eckart Zitzler, Lothar Thiele, Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.

[20] Hamad Alibrahim, Simone A. Ludwig, Hyperparameter optimization: Comparing genetic algorithm against grid search and Bayesian optimization, in: 2021 IEEE Congress on Evolutionary Computation, CEC, 2021, pp. 1551–1559.

[21] Sébastien Chabrier, Christophe Rosenberger, Bruno Emile, Hélène Laurent, Optimization-Based image segmentation by genetic algorithms, EURASIP J. Image Video Process. 2008 (2008) 842029.

[22] Ujjwal Maulik, Medical image segmentation using genetic algorithms, IEEE Trans. Inf. Technol. Biomed. 13 (2) (2009) 166–173.

[23] Y. Jiang, J. Ju, X. Zhang, B. Yang, Automated analog circuit design using genetic algorithms, in: 2009 3rd International Conference on Anti-Counterfeiting, Security, and Identification in Communication, ICASID, 2009, pp. 223–228.

[24] Junichi Mizoguchi, Hitoshi Hemmi, Katsunori Shimohara, Production genetic algorithms for automated hardware design through an evolutionary process, in: Proceedings of the First IEEE Conference on Evolutionary Computation, 1994, pp. 661–664.

[25] J. Northern, M. Ribeiro, GENIE: A genetic algorithm model based integrated simulation framework for design of embedded systems, in: 2007 IEEE Region 5 Technical Conference, 2007, pp. 223–227.

[26] Martin L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley & Sons, New York, 1994.

[27] John R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.

[28] Riccardo Poli, William B. Langdon, Nicholas F. McPhee, A Field Guide to Genetic Programming, Lulu Enterprises, 2008.
[29] Julian F. Miller, Cartesian Genetic Programming, Springer, 2011.
[30] Benjamin C. Schafer, Zizhong Wang, High-level synthesis design space exploration: Past, present, and future, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (10) (2020) 2624–2638.
[31] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, Saman Amarasinghe, OpenTuner: An extensible framework for program autotuning, in: Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques, PACT, ACM, 2014, pp. 303–316.
[32] X. Jun, H. Ye, Y. Liang, et al., AutoScaleDSE: Scalable and portable design space exploration for high-level synthesis, ACM Trans. Reconfigurable Technol. Syst. 16 (1) (2023) 1–24.
[33] Han Cai, Ligeng Zhu, Song Han, ProxylessNAS: Direct neural architecture search on target task and hardware, in: International Conference on Learning Representations, ICLR, 2019.
[34] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, Naigang Wang, Hardware-Aware neural architecture search: Survey and taxonomy, in: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, 2021, pp. 4563–4570.
[35] Wolfgang Banzhaf, Artificial intelligence: Genetic programming, in: Neil J. Smelser, Paul B. Baltes (Eds.), International Encyclopedia of the Social & Behavioral Sciences, Pergamon, Oxford, 2001, pp. 789–792.
[36] Julian Blank, Kalyanmoy Deb, Pymoo: Multi-Objective optimization in Python, IEEE Access 8 (2020) 89497–89509.
[37] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4, 1995, pp. 1942–1948.
[38] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, Noah Dormann, Stable-Baselines3: Reliable reinforcement learning implementations, J. Mach. Learn. Res. 22 (268) (2021) 1–8.
[39] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al., Gymnasium: A standard interface for reinforcement learning environments, 2024, arXiv preprint.
[40] Runzhe Yang, Xingyuan Sun, Karthik Narasimhan, A generalized algorithm for multi-objective reinforcement learning and policy adaptation, 2019, arXiv preprint.
[41] Florian Felten, Lucas N. Alegre, Ann Nowé, Ana L.C. Bazzan, El Ghazali Talbi, Grégoire Danoy, Bruno C. da Silva, A toolkit for reliable benchmarking and research in Multi-Objective reinforcement learning, in: Advances in Neural Information Processing Systems, NeurIPS 2023, 2023.
[42] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, Vasco G. da Fonseca, Performance assessment of multiobjective optimizers: An analysis and review, IEEE Trans. Evol. Comput. 7 (2) (2003) 117–132.



Jose Luis Mira completed the graduate studies in computer science at the University of Castilla La-Mancha and the master degree in artificial intelligence in the International University of La Rioja in 2020 and 2022 respectively, he is currently pursuing the Ph.D. degree at the Department of Technologies and Information Systems in the University of Castilla-La Mancha in Ciudad Real, Spain. His research interests include, artificial Intelligence on the edge systems, high-level syntesys, reconfigurable computing, machine learning and computer vision.



Carlos Hernández is an Electronics Engineer and holds a Master's degree in Industry 4.0, as well as a Master's degree in Microelectronic Systems Design based on Open Architectures. He is currently a Ph.D. student and researcher with the ARCO Research Group at the University of Castilla-La Mancha (UCLM). His research interests focus on the development of hardware accelerators for RISC-V-based systems, reconfigurable hardware, MPSoC design, and High-Performance Computing (HPC) applications.



Jesus Barba received the MS and Ph.D. degrees in Computer Engineering Diploma from the University of Castilla-La Mancha (UCLM), Spain, in 2001 and 2008 respectively. He is working as Associate Professor with the Department of Information and Systems Technology (TSI) since 2001 and member of the ARCO research group, located at the School of Computer Science (UCLM, Spain. Among the open research lines and interests it is worth mentioning the following: Low-cost and low power reconfigurable systems for ubiquitous computing, Reconfigurable Computing platforms for AI algorithms, Heterogeneous Distributed Embedded Computing and High-level Synthesis Tools.



Antonio Caruso received the M.S. degree (cum laude) and Ph.D. degree from the University of Pisa, Pisa, Italy, both in computer science. In 2005, he joined the Mathematical and PhysicsDepartment, University of Salento, Lecce, Italy, as an Assistant Professor. Prof. Caruso was a recipient of the Innovation Award from Italian–Canada in 2017. He has been a member of several Program Committees of conferences and workshops and his research has been published in international journals and conference proceedings mainly in the area of mobile distributedsystems, Security, Internet of Things, Smart Environment, Wireless Sensor Networks. He is currently the site leader for the University of Lecce of theTEBAKA project (PON-MISE) an AgriTech project that use AI/ML and remote sensing (UAV and Satellite Images) for improving quality and productivity of crops.



Julián Caba received the M.S. degree in Computer Science and the Ph.D. from the University of Castilla-La Mancha (UCLM), Spain, in 2009 and 2018, respectively. He won the Ph.D. category in the Xilinx Open Hardware Contest in 2017. He is currently an Assistant Professor with the TSI Department, UCLM. His current research interests include hardware verification methodologies, embedded systems, high-level synthesis, run-time reconfigurable systems and heterogeneous distributed systems.



Soledad Escolar holds a Ph.D. in Computer Science and Technology from the University Carlos III de Madrid in 2010 with honors. In September 2015, she joined the University of Castilla-La Mancha (UCLM) as a post-doctoral researcher and in 2020 she took possession of the position of Associate Professor at the same university. Since 2021, she has been appointed as the Deputy Director of Academic Planning and Smart ESI at the School of Computer Science. Her research career has focused mainly on the study of a wide range of problems related to distributed systems, more specifically, sensor networks and their evolution toward the Internet of Things, specifically application portability in heterogeneous environments, communication protocols, synchronization, mobility, models, and energy efficiency management algorithms in energy-harvesting devices.