https://doi.org/10.31891/2307-5732-2026-363-13
УДК 004.51

DANYLO CHUMACHENKO
Odesa Polytechnic National University, Odesa, Ukraine
https://orcid.org/0009-0000-1477-534X
email: chumachdk@stud.op.edu.ua
VIRA LIUBCHENKO
Odesa Polytechnic National University, Odesa, Ukraine
https://orcid.org/0000-0002-4611-7832
email: lvv@op.edu.ua

A MULTI-OBJECTIVE OPTIMIZATION MODEL FOR DESIGNING THE SOFTWARE ARCHITECTURE OF INTERNET-OF-THINGS SYSTEMS

Modern Internet of Things (IoT) systems span edge, fog, and cloud layers and must satisfy several conflicting non-functional requirements, such as responsiveness, security, reliability, and resource efficiency. Earlier work (CAL and MCAL) increased the transparency of architectural decisions by scoring and ranking individual patterns using a weighted sum, but implicitly treated the architecture as a loose set of independent patterns and compressed all requirements into a single scalar index. In this paper, IoT architecture design is reformulated as a multi-objective pattern-portfolio selection problem. Each candidate architecture is represented as a binary portfolio of patterns and evaluated by an additive vector of objective functions corresponding to selected quality attributes, while linear constraints capture practical engineering limits: implementation budget, pattern dependencies, incompatibilities, and portfolio size. The resulting multi-objective integer linear programming model yields a discrete Pareto set of non-dominated portfolios that makes trade-offs explicit and auditable. A small case study with two antagonistic objectives, responsiveness and security, illustrates the effect of the reformulation and compares it with the classic weighted-sum approach. The scalar model produces markedly different "optimal" portfolios for different weight vectors and can hide feasible compromise solutions that balance competing requirements. In contrast, the Pareto-based model exposes the entire family of efficient portfolios under the same data and constraints, allowing architects and stakeholders to select an option that matches their priorities, constraints, and acceptable risk, providing a clear basis for subsequent a-posteriori ranking or sensitivity analysis when project preferences change. The formulation is readily extensible to additional objectives and larger pattern catalogs, keeping feasibility rules explicit, and reusable across IoT projects, thus supporting systematic architectural decision-making in complex distributed environments.

Keywords: *Internet of Things; software architecture; architectural patterns; multi-objective modeling; Pareto front; non-functional requirements; decision support; pattern portfolios.*

ЧУМАЧЕНКО ДАНИЛО, ЛЮБЧЕНКО ВІРА
Національний університет «Одеська політехніка»

МУЛЬТИЦІЛЬОВА МОДЕЛЬ ОПТИМІЗАЦІЇ ДЛЯ ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ

Сучасні системи Інтернету речей охоплюють рівні edge, fog та хмари й мають одночасно задовольняти низку конфліктних нефункціональних вимог, зокрема вимоги до швидкодії та безпеки. Попередні підходи CAL та MCAL підвищували прозорість архітектурних рішень, використовуючи зважену суму для оцінювання та ранжування окремих патернів, але неявно розглядали архітектуру як вільну сукупність незалежних патернів і стискали всі вимоги в один скалярний індекс. У цій роботі проєктування архітектури формулюється як задача багатокритеріального вибору портфеля патернів. Кожна кандидатна архітектура подається у вигляді бінарного портфеля, який оцінюється адитивним вектором цільових функцій для вибраних нефункціональних вимог, тоді як лінійні обмеження відображають бюджет, залежності та несумісності між патернами, а також кардинальність портфеля. Відповідна багатокритеріальна модель цілочисельного лінійного програмування породжує дискретну множину Парето з недомінованих портфелів. Невелике прикладне дослідження з двома антагоністичними вимогами, швидкодією та безпекою, демонструє наслідки такої реформуляції. Підхід із зваженою сумою дає різні рекомендовані портфелі для різних векторів ваг і не виявляє більш збалансований портфель, який також задовольняє всі обмеження. Натомість багатокритеріальна модель явно показує всю сукупність недомінованих портфелів за незмінних вхідних даних та обмежень, надаючи архітекторам і зацікавленим сторонам можливість обрати той варіант патернів, який найкраще відображає їхні реальні пріоритети та прийнятний рівень ризику.

Ключові слова: *Інтернет речей; програмна архітектура; архітектурні патерни; багатокритеріальне моделювання; фронт Парето; нефункціональні вимоги; підтримка прийняття рішень; портфелі патернів.*

Стаття надійшла до редакції / Received 16.01.2026
Прийнята до друку / Accepted 12.02.2026
Опубліковано / Published 26.03.2026

Introduction

The rapid proliferation of Internet-of-Things (IoT) systems has significantly increased the complexity of software architecture design. Modern IoT solutions typically span multiple computational layers (edge, fog, and cloud) and operate under strict constraints related to latency, security, reliability, scalability, and resource consumption. In such systems, architectural decisions play a decisive role in determining whether non-functional requirements (NFRs) are met throughout the system lifecycle.

A common way to address NFRs at the architectural level is to select and combine architectural patterns. Patterns encapsulate proven design knowledge and provide reusable solutions to recurring architectural problems. However, in realistic IoT projects, an architecture rarely consists of a single pattern. Instead, it emerges as a portfolio of patterns, whose combined behavior determines the overall system quality. Importantly, many NFRs are inherently conflicting: for example, mechanisms that improve security or safety often introduce additional latency and computational overhead, thereby degrading responsiveness. As a result, architectural design becomes a complex decision-making problem involving trade-offs among competing quality attributes.

Traditional decision-support approaches in software architecture often rely on scalar aggregation, most notably weighted-sum models, to rank architectural alternatives. In earlier work, such methods were successfully applied to increase transparency in pattern selection by assigning expert-based scores to individual patterns and aggregating them into a single integral index [1]. While this approach simplifies decision-making, it has two fundamental limitations. First, it implicitly treats architectural patterns as independent choices rather than as interacting elements of a coherent portfolio. Second, collapsing multiple quality attributes into a single scalar value obscures trade-offs and makes the final recommendation highly sensitive to the choice of weights, which must be fixed a priori and often reflect subjective assumptions rather than concrete project realities.

Recent research in software architecture and optimization has increasingly emphasized multi-objective approaches as a more faithful representation of real-world design problems. Instead of forcing a single "optimal" solution, multi-objective optimization aims to identify a set of Pareto-optimal alternatives, each representing a different compromise among conflicting objectives. Such an approach aligns naturally with architectural decision-making, where stakeholders may have different priorities and risk tolerances, and where understanding the cost of improving one quality attribute at the expense of another is often more valuable than obtaining a single ranked solution.

In this paper, the design of an IoT software architecture is reformulated as a multi-objective pattern-portfolio selection problem. Each candidate architecture is represented as a binary portfolio of architectural patterns, evaluated by a vector of objective functions corresponding to selected non-functional requirements. Linear constraints are introduced to capture practical engineering considerations, including implementation budget, pattern dependencies, incompatibilities, and overall architectural complexity. Under these assumptions, the problem is expressed as a multi-objective integer linear programming (MOILP) model, whose solution is a discrete Pareto set of non-dominated architectural portfolios.

Architectural challenges and solutions

Designing software architecture is fundamentally a complex decision-making challenge, particularly when balancing conflicting non-functional requirements. It is well known that improving one attribute often comes at the cost of another [2]. Traditionally, architects relied on scalar aggregation methods to simplify these decisions. However, recent literature suggests that these approaches can be misleading, as they tend to obscure the real trade-offs and introduce bias through static weighting [3]. In response, the field is moving toward combinatorial search and multi-objective approaches. These methods explicitly model conflicts, allowing for the identification of Pareto-optimal solutions rather than forcing a single, potentially compromised alternative [4, 5].

Our previous research in this domain focused on establishing a structured methodology for selecting architectural patterns, using expert-driven scoring and weighted-sum models [1]. While this work successfully improved the transparency of the selection process, it was limited by the very scalar assumptions that are now being questioned. Building on those foundations, this paper takes the next logical step. We advance the approach by reformulating the problem as a multi-objective optimization, shifting the focus from simply ranking individual patterns to generating comprehensive portfolios that reveal the necessary design compromises.

Problem Formulation and Multi-Objective Decision Making

This paper aims to propose and validate a multi-objective integer linear programming model for selecting portfolios of architectural patterns in IoT systems, enabling explicit exploration of trade-offs between conflicting NFRs and overcoming the limitations of traditional weighted-sum approaches.

Software architecture design is a decision-making process aimed at satisfying a set of functional and, critically, NFRs [6]. In IoT systems, these NFRs (such as reliability, safety, responsiveness, and usability) are often in conflict [2]. For example, enhancing safety and security by introducing additional checks and encryption (e.g., the Reference Monitor pattern) almost always increases latency and resource consumption, thereby directly conflicting with the Responsiveness requirement [7].

Traditional methods based on scalar aggregation hide these trade-offs behind static weights assigned by experts. This leads to a single solution that is heavily biased by a priori preferences and does not allow the architect to explore alternative design options [8].

This paper moves away from the scalar approach and treats the architecture not as a set of individual patterns, but as a unified portfolio. The task is to select a subset of patterns from a general catalog that best balances the conflicting NFRs.

At its core, this is a combinatorial optimization problem (COP), as the solution space consists of 2^Npossible pattern combinations, where N is the total number of candidate patterns [9]. Given that we aim to balance M simultaneously conflicting objective functions corresponding to NFRs, the task is accurately classified as a multi-objective combinatorial optimization problem (MOCOP) [10].

The goal of such an approach is not to find a single solution, but to approximate the Pareto-optimal set (or Pareto front). The Pareto front represents a set of non-dominated solutions (pattern portfolios). This front explicitly visualizes the trade-offs, allowing decision-makers (architects, project managers) to select a portfolio that best aligns with the project's specific priorities [11].

To mathematically formulate the MOCOP task described, we introduce the following notation.

Pattern Catalog is a finite, predefined catalog of N candidate architectural patterns relevant to the target domain (IoT), $P = \{p_1, p_2, \ldots, p_N\}$.

A set of Quality Attributes is the set of M non-functional requirements selected for balance, $K = \{k_1, k_2, \ldots, k_M\}$. In the context of this work

$$K = \{Reliability, Safety, Usability, Responsiveness, Security\}.$$

Contribution Matrix W is an $N \times M$ matrix, where the element $w_{j,k}$ represents the quantitative, normalized score of the contribution of the j-th pattern (p_j) to the k-th quality attribute (k_m). These values are a key model input, derived from an expert knowledge base as detailed in our prior work [2]. The process of quantifying NFRs is itself a complex research challenge [12].

Cost Vector $C = (c_1, c_2, \ldots, c_N)$ is a vector where c_j represents the estimated cost (e.g., implementation complexity, man-hours, computational resources) of implementing the pattern p_j.

Portfolio Vector is a binary vector that represents a solution $X = (x_1, x_2, \ldots, x_N)$, where

$$x_j = \begin{cases} 1, & if\ patern\ p_j\ is\ selected\ for\ the\ portfolio \\ 0, & otherwise \end{cases}. \quad (1)$$

Key Model Assumptions

The formulation of the objective functions is based on the fundamental assumption of linearity and additivity. The model assumes that the total contribution of a portfolio X to a quality attribute k (i.e., $F_k(X)$) is the simple sum of the contributions ($w_{j,k}$) of all selected patterns ($x_j = 1$).

This assumption implies

- independence: the contribution of each pattern is independent of the presence or absence of other patterns in the portfolio.
- lack of interactions: the model, in its basic form, does not account for interaction effects, neither positive (synergy) nor negative (interference or conflict) [13].

For example, the combined use of Rate-Limiting and Caching patterns might produce a synergistic effect for Responsiveness that is greater than the sum of their individual contributions. Conversely, two security patterns might prove redundant or conflict, degrading overall performance.

Modeling such interactions requires more complex (e.g., quadratic) objective functions and is a direction for future research. For the current work, an additive model is adopted as an effective approximation, enabling the use of standard MOO methods to demonstrate the trade-off analysis.

Objective Functions Formulation

At the heart of the proposed decision-making framework lies the challenge of quantifying the architectural suitability of any given pattern portfolio X. Unlike trivial design tasks where a single metric might suffice, software architecture is inherently multidimensional. To capture this complexity, we formulate the evaluation process not as a search for a single value, but as the simultaneous maximization of a vector of objective functions.

For a defined set of M quality attributes (non-functional requirements, K), we establish a corresponding vector $F_k(X)$. The overarching design goal is to identify portfolios that push this vector towards its theoretical maximum:

$$\text{maximize } F(X) = \left(F_k(X)\right)_{k \in K} = \left(F_1(X), F_2(X), \ldots, F_M(X)\right). \quad (2)$$

While we acknowledge that software patterns can exhibit complex, non-linear interactions in reality, a linear approximation serves as a necessary and robust baseline for automated decision support. Under this assumption, the quality of a portfolio for a specific attribute k is calculated as the cumulative sum of the individual contributions of its constituent patterns. If a pattern p_j is selected (i.e., $x_j = 1$), its pre-determined impact score $w_{j,k}$ is added to the total. Formally, for each quality attribute $k \in K$, the objective function is defined as:

$$F_k(X) = \sum_{j=1}^{N} w_{j,k} \cdot x_j, \forall k \in K. \quad (3)$$

This formulation offers two distinct advantages. First, it ensures high interpretability: an architect can easily trace why a portfolio received a particular score. Second, it maintains the problem within the domain of Linear Programming, allowing for efficient processing even as the catalog size N grows.

Modern IoT systems, which typically span constrained edge devices, fog layers, and cloud infrastructure, demand a rigorous balance of specific conflicting qualities. For the experiments conducted in this study, we define $M = 5$ critical objective functions that represent the most common trade-offs in IoT design. The vector of objective functions F(X) can be expressed as a linear transformation of the decision vector X using the transpose of the contribution matrix W:

$$F(X) = W^T \cdot X = \begin{bmatrix} F_{Rel}(X) \\ F_{Safe}(X) \\ F_{Usab}(X) \\ F_{Resp}(X) \\ F_{Sec}(X) \end{bmatrix}. \quad (4)$$

Where each element corresponds to the total score for a specific NFR defined in the set K.

This linear and decomposable formulation is highly flexible, allowing for the addition or removal of objective functions without altering the model's core structure. It provides a robust foundation for applying multi-objective optimization algorithms to find the set of non-dominated solutions.

The objective functions must be optimized subject to a set of linear constraints. These constraints reflect real-world design, technical, and budgetary limitations, thereby defining the feasible solution space F. A portfolio X is considered a possible solution only if it satisfies all defined constraints. The model incorporates several types of constraints to define this space.

The first is the Complexity Budget Constraint, which reflects resource limitations. Each pattern p_j has an associated cost c_j (e.g., estimated person-hours, implementation complexity), and the total cost of the selected portfolio must not exceed the maximum project budget C_{max}:

$$\sum_{j=1}^{N} c_j \cdot x_j \leq C_{max} \,. \tag{5}$$

The model also enforces Architectural Rules to ensure design logic. These rules primarily consist of Dependencies, where one pattern (p_a) requires another (p_b), modeled as $x_a \leq x_b$; and Incompatibilities, which represent mutually exclusive choices (p_c, p_d) for the same sub-problem, modeled as $x_c + x_d \leq 1$. Furthermore, Portfolio Cardinality Constraints are used to manage overall architectural complexity by setting a lower (L) and upper (U) bound on the total number of selected patterns:

$$L \leq \sum_{j=1}^{N} x_j \leq U \,. \tag{6}$$

Given that all objective functions and these constraints are linear functions of the binary decision variables x_j, the formulated mathematical model is precisely classified as a Multi-Objective Integer Linear Programming (MOILP) problem.

The Concept of Pareto Optimality

As objective functions conflict in Multi-Objective Optimization (MOO) problems, a single solution that simultaneously maximizes all M quality attributes generally do not exist. Instead, the concept of Pareto dominance is used to compare solutions.

A solution X_a dominates X_b if it is no worse than $X_b$across all M objective functions and is strictly better in at least one objective. Formally, $X_a \succ X_b$ if

$$\forall k \in K: F_k(X_a) \geq F_k(X_b) \quad \wedge \quad \exists k' \in K: F'_k(X_a) > F'_k(X_b) \,. \tag{7}$$

A solution $X^*$is Pareto-optimal (or non-dominated) if no other feasible solution $X \in F$ exists that dominates it. In other words, X^* is Pareto-optimal if any improvement in one quality attribute (e.g., $Reliability$) must necessarily lead to a degradation in at least one other attribute (e.g., $Responsiveness$).

The goal of an MOO task is not to find a single solution, but to identify the complete set of all Pareto-optimal solutions, known as the Pareto Set **(**P_S**)**:

$$P_S = \{X^* \in F \mid \nexists X \in F: X \succ X^*\} \,. \tag{18}$$

The projection of this set into the M-dimensional objective space is called the Pareto Front **(**P_F**)**. The Pareto Front represents the trade-off surface, which explicitly visualizes the cost of choice for the architect. In the context of an MOILP problem, the Pareto Front is a discrete set of points rather than a continuous curve.

Case study & comparative evaluation

To validate the proposed model's performance and visually demonstrate its advantages over traditional methods, we will use a small illustrative example. Using a simplified dataset (a small number of patterns and objectives) allows us not only to trace the calculation process step by step but also to clearly visualize the fundamental differences in the results produced by the two different approaches. This example simulates the real-world architectural decision-making process in miniature, where conflicting requirements must be balanced under limited resources, and where trade-offs are rarely clean or comfortable. In practice, engineers typically work with partial information, tight deadlines, and competing stakeholder expectations, so even a compact toy scenario can feel surprisingly close to reality. By keeping the setup intentionally small and readable, we can focus on the intuition behind each choice, see how priorities shape the outcome, and explain the results in a way that remains transparent, reproducible, and easy to interpret for both researchers and practitioners.

Let's consider a simplified task of selecting architectural patterns. Assume we have a catalog of $\mathrm{N} = 5$ available patterns. Our goal is to optimize the architecture for $\mathrm{M} = 2$ key non-functional requirements (NFRs): Responsiveness (which we aim to maximize) and Security (which we also aim to maximize). These two quality attributes were chosen deliberately: they are a classic antagonistic pair, as strengthening security (through additional checks, encryption) almost always negatively impacts responsiveness.

Each pattern P_j is characterized by three parameters:

1) implementation cost (e.g., estimated person-hours), C_j;
2) a quantitative score of the pattern's contribution to the system's overall responsiveness (W_j, Resp);
3) a quantitative score of the pattern's contribution to the system's overall security (W_j, Sec).

These parameters can be summarized in Table 1 for clarity.

Table 1

Input data for the illustrative example (patterns and their attributes).

Pattern ID (P_j)	**Pattern Name**	**Cost (C_j)**	**Resp. Score (W_j, Resp)**	**Sec. Score (W_j, Sec)**
p1	Caching	5	9	1
p2	Firewall	5	1	9
p3	Authenticator	4	2	8
p4	Load Balancer	6	7	3
p5	Logger	3	4	5

Furthermore, should introduce a realistic constraint that mimics a project budget: the total cost of the selected pattern portfolio cannot exceed the Complexity Budget Constraint $C_{max} = 10$. This means the development team has only 10 conditional units of complexity available to implement the chosen architecture.

Method 1. Weighted-Sum (Integral) Approach. Its essence is to collapse a multi-dimensional decision problem into a one-dimensional one. This is achieved by assigning a weight coefficient to each objective function (each NFR) that reflects its relative importance. All objective functions are then summed into a single integral function F_{WS}:

$$F_{WS}(X) = w_{\text{Resp}} \cdot F_{\text{Resp}}(X) + w_{\text{Sec}} \cdot F_{\text{Sec}}(X)\ . \quad (9)$$

The main problem with this method is that the architect must define these weight coefficients a priori, that is, before seeing any results or trade-offs. As we will show, this subjective choice has a decisive impact on the final result.

To illustrate how sensitive the weighted-sum method is to the choice of weights, consider a simple example in which the architect initially assumes that responsiveness and security are equally important and assigns them the exact weights, $w_{\text{Resp}} = w_{\text{Sec}} = 0.5$. The scalar objective function is then:

$$F_{WS}(X) = 0.5 \cdot F_{\text{Resp}}(X) + 0.5 \cdot F_{\text{Sec}}(X)\ . \quad (10)$$

By enumerating all feasible portfolios of patterns under the cost constraint $C(X) \leq 10$, we obtain a single optimal solution: Portfolio A = $\{p_2, p_3\}$. Its total cost is $5(p_2) + 4(p_3) = 9 \leq 10$. The combined responsiveness is $F_{\text{Resp}} = 1(p_2) + 2(p_3) = 3$, and the combined security is $F_{\text{Sec}} = 9(p_2) + 8(p_3) = 17$. The integral score becomes:

$$F_{WS}(X) = 0.5 \cdot 3 + 0.5 \cdot 17 = 1.5 + 8.5 = 10.0\ . \quad (11)$$

Although the numerical weights are balanced, the resulting portfolio is strongly skewed: it provides excellent security (17 points) but catastrophically low responsiveness (3 points).

Now change only the project priorities while keeping all other data exactly the same. Suppose responsiveness becomes much more important than security, and the architect updates the weights to $w_{\text{Resp}} = 0.8$and $w_{\text{Sec}} = 0.2$. The scalar objective is now:

$$F_{WS}(X) = 0.8 \cdot F_{\text{Resp}}(X) + 0.2 \cdot F_{\text{Sec}}(X)\ . \quad (12)$$

Recomputing the optimum over the same set of feasible portfolios yields an entirely different recommended architecture: Portfolio B = $\{p_1, p_5\}$. Its cost is $5(p_1) + 3(p_5) = 8 \leq 10$; the combined security is $F_{\text{Sec}} = 1(p_1) + 5(p_5) = 6$; the combined responsiveness is $F_{\text{Resp}} = 9(p_1) + 4(p_5) = 13$. The corresponding integral score:

$$F_{WS}(X) = 0.8 \cdot 13 + 0.2 \cdot 6 = 10.4 + 1.2 = 11.6\ . \quad (13)$$

Thus, a purely subjective adjustment of the weights, without any change in the underlying patterns, scores, or constraints, produces an entirely different portfolio. This shows that the weighted-sum method does not reveal the trade-offs among objectives; instead, it yields a single solution that is essentially an artifact of the chosen weights.

Method 2. Multi-Objective Approach. The limitations of the weighted-sum method in the previous subsection arise from treating a genuinely multi-objective decision as if it were one-dimensional. A small change in the weights, made a priori and without seeing the alternatives, led to two different and even conflicting portfolios. In Method 2, we keep exactly the same input data, constraints, and search space as before, but we adopt a different decision criterion: instead of collapsing the objectives into a single scalar score, we treat responsiveness and security as two separate objective functions, $F_{\text{Resp}}(X)$ and $F_{\text{Sec}}(X)$, and evaluate every portfolio X by the vector $(F_{\text{Resp}}(X), F_{\text{Sec}}(X))$.

In a multi-objective (Pareto-based) setting, the goal is no longer to identify one best portfolio, but to compute the set of non-dominated portfolios. A portfolio $X_1$is said to dominate X_2 if it is at least as good in all objectives and strictly better in at least one. A portfolio is non-dominated if there is no other feasible portfolio that dominates it. The set of all non-dominated portfolios forms the Pareto set; its image in the $(F_{\text{Resp}}, F_{\text{Sec}})$ plane is the Pareto front, which can be interpreted as a list of the best attainable trade-offs between responsiveness and security.

For the toy problem in Table 1, the Pareto front can be obtained by simple exhaustive enumeration. We consider all portfolios that satisfy the cost constraint $C(X) \leq 10$, compute their total responsiveness $F_{\text{Resp}}(X)$ and security $F_{\text{Sec}}(X)$, and then filter out all dominated portfolios. The remaining non-dominated portfolios are reported in Table 2 and analyzed in the discussion section that follows; they represent the complete set of best trade-offs achievable under the given budget and quality constraints.

Table 2

Pareto-optimal set obtained by exhaustive enumeration

Solution ID	Portfolio (X)	Total Cost	Resp. Score (F_{Resp})	Sec. Score (F_{Sec})
A	$\{p_2, p_3\}$	9	3	17
B	$\{p_1, p_5\}$	8	13	6
C	$\{p_3, p_4\}$	10	9	11

Discussion and Comparison

Comparing the results of the two methods highlights the fundamental advantages of the multi-objective formulation. The weighted-sum approach produces a single portfolio that depends entirely on the chosen weights. In the example, equal weights on responsiveness and security lead to portfolio $A = \{p_2, p_3\}$, which almost maximizes security (17 points) but yields very low responsiveness (3 points). When the weights are shifted in favour of responsiveness, the method returns an entirely different solution, portfolio $B = \{p_1, p_5\}$, which offers excellent responsiveness (13 points) but only modest security (6 points). In both cases, the architect receives one recommendation without any information about alternative compromises that might better match the project context.

The Pareto-based multi-objective approach restructures the decision process. Instead of a single scalar score, each portfolio is evaluated by the vector of objective values, and dominated portfolios are discarded. Table 2 shows that, under the same data and constraints, this procedure yields a compact set of non-dominated portfolios, *A*, *B*, and *C*. Together, they form a spectrum of trade-offs between responsiveness and security: *A* concentrates on security, *B* on responsiveness. At the same time, $C = \{p_3, p_4\}$ achieves an intermediate, more balanced combination (9 points of responsiveness and 11 points of security) within the same budget. The architect can now make an a posteriori choice among these alternatives, guided by the actual business priorities of a concrete project rather than by guessed a priori weights.

From this perspective, the weighted-sum method can be interpreted as a mechanism that implicitly selects extreme points of the Pareto front. Portfolios *A* and *B*, obtained for different weight vectors, are not incorrect; they correspond to corners of the trade-off surface. However, the scalar aggregation hides the fact that these solutions are just two members of a broader family of Pareto-optimal portfolios. In particular, the balanced solution *C* is entirely invisible to any of the considered weighted-sum scenarios: no linear combination of the objectives with fixed weights identifies $\{p_3, p_4\}$ as optimal, even though this portfolio may be more attractive for many realistic IoT projects where neither responsiveness nor security can be sacrificed completely.

The case study, therefore, confirms that scalar aggregation is structurally limited as a decision-support tool for architectural design. By compressing multiple quality attributes into a single index, it conceals the nature of the trade-offs and introduces bias through the choice of weights. The proposed multi-objective model, in contrast, explicitly constructs the Pareto set and exposes all efficient portfolios available under the given constraints. This turns the optimization procedure into a transparent aid for architectural decision-making: instead of prescribing one supposedly best architecture, it provides a well-defined set of alternatives from which stakeholders can select the option that best aligns with their requirements and risk appetite.

Conclusions

This paper reformulates the design of IoT software architecture as a pattern-portfolio selection problem under conflicting non-functional requirements. Instead of ranking individual patterns with a scalar weighted-sum, the proposed approach represents each candidate architecture as a binary portfolio and evaluates it by an additive vector of objective functions, while enforcing linear feasibility constraints that capture practical engineering limits. Under these assumptions, the task becomes a multi-objective integer linear programming problem whose output is a discrete Pareto set of non-dominated portfolios, making trade-offs explicit and auditable. The comparative case study with two antagonistic objectives demonstrates the structural weakness of scalar aggregation: small changes in the weight vector can yield qualitatively different "optimal" portfolios, while concealing feasible compromises that may better align with realistic project priorities. In contrast, the Pareto-based formulation exposes a compact family of efficient portfolios, including an intermediate, more balanced option. This shifts decision support from prescribing a single architecture to providing a transparent set of defensible alternatives for stakeholders to choose from.

Future work can make the method more practical by considering how patterns interact, by adding more realistic constraints such as tier limits, technology compatibility, deployment, and maintenance restrictions, and by testing the approach on larger pattern catalogs while keeping the computation fast enough for projects. It is also worth handling uncertainty because costs and quality scores are often only estimates, so sensitivity checks and scenario-based evaluation can help find portfolios that remain good when inputs change. After the Pareto set is built, a simple second step can help choose one final architecture. For example, TOPSIS can rank Pareto optimal portfolios using normalization and stakeholder weights, then select the option closest to the ideal solution and farthest from the worst solution. This turns the Pareto front into a clear shortlist or one recommendation.

References

1. Chumachenko D.K., Liubchenko V.V. (2025). A seven-step method for selecting architectural patterns for IoT. *Applied Aspects of Information Technology*, 8(2), 178–190.
2. Orellana C., Cereceda-Balic F., Solar M. (2024). Enabling design of secure IoT systems with trade-off-aware architectural tactics. *Sensors*, *24*(22), 7314.
3. Aled Williams, Yilun Cai. (2025). Insights into Weighted Sum Sampling Approaches for Multi-Criteria Decision-Making Problems.
4. Di Pompeo D., & Tucci M. (2023). Multi-objective Software Architecture Refactoring driven by Quality Attributes. *arXiv* preprint arXiv:2301.07500
5. Lust T., & Teghem J. (2010). The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, 17(2), 167-186.
6. Dongmo C. (2024). A Review of Non-Functional Requirements Analysis Throughout the SDLC. *Computers*, 13(12), 308.
7. Capobianco F., Zhou Q., Basu A., Jaeger T. (2024). *TALISMAN: Tamper Analysis for Reference Monitors.* In Proceedings of the Network and Distributed System Security (NDSS) Symposium, 2024.
8. Karl F., Pielok T., & Moosbauer J. et al. (2022). Multi-Objective Hyperparameter Optimization in Machine Learning — An Overview. *ACM Transactions on Evolutionary Learning and Optimization*, 3(4), 1–50.
9. Agh H., Azamnouri A., & Wagner S. (2024). *Software product line testing: a systematic literature review.* Empirical Software Engineering, 29, 146.
10. Peres F., & Castelli M. (2021). *Combinatorial Optimization Problems and Metaheuristics: Review, Challenges, Design, and Development.* Applied Sciences, 11(14), 6449.
11. García-Reddy F., Bhattacharya P., & Kolesnikov S. (2023). *Multi-objective Software Architecture Refactoring driven by Quality Attributes.* arXiv preprint arXiv:2301.07500.
12. Meijer W., Trubiani C., & Aleti A. (2024). *Experimental evaluation of architectural software performance design patterns in microservices.* Journal of Systems and Software, 218, Article 112183.
13. Kaltenecker C., Mühlbauer S., & Grebhahn A. et al. (2023). Performance evolution of configurable software systems: an empirical study. Empirical Software Engineering, 28(6), 152.