A Surrogate-based Approach for Fast Multi-objective Architectural Refactoring Optimization

J. Andres Diaz-Pace[1], Daniele Di Pompeo[2], and Antonela Tommasel[1,3]

[1] ISISTAN, CONICET-UNCPBA, Tandil, Argentina
[2] SPENCER Lab, University of L'Aquila, L'Aquila, Italy
[3] Johannes Kepler University Linz, Austria

Abstract. Software model optimization is a process that generates architecture alternatives aimed at improving quantifiable non-functional properties of software systems, such as performance and reliability. Multi-objective evolutionary algorithms are commonly used to explore the search space and help designers identify trade-offs among competing non-functional properties (e.g., through a Pareto front). However, such algorithms face efficiency challenges in complex software models and large design spaces, since evaluating the fitness (i.e., the quality) of each architecture requires analysis tools that become computationally expensive when repeatedly invoked during the search process. In this paper, we explore the construction of surrogate models based on regression techniques to approximate the outputs of these analysis tools at significantly lower computational cost, while maintaining reasonable output accuracy. Our experimental results suggest that surrogate models provide savings of up to 30% in computational time and maintain the Pareto front quality provided by evolutionary algorithms. Also, we observed some differences in the architectural models produced by our approach. Overall, surrogate models constitute a promising approach for scaling multi-objective architecture optimization to larger spaces and complex architectural models.

Keywords: Software Architecture, Search-Based Software Engineering, Multi-objective Optimization, Surrogate Models, Machine Learning

1 Introduction

A software architecture is a high-level abstraction of a system that defines its internal structure in terms of components, connectors, properties and main functions. An architecture plays a crucial role in determining the blueprint of the software system and the non-functional properties that it should satisfy, such as maintainability, scalability, or performance, among others [26, 25]. Software architectures have been exploited to analyze non-functional properties of software systems [1, 5, 27]. These efforts normally rely on predefined analysis models (or solvers), such as Layered Queuing Networks (LQNs) for performance [15] or rule-based detection for performance antipatterns [10]. Beyond analysis, architectural

refactoring has emerged as a complementary activity aimed at systematically modifying the architecture structure (*e.g.*, by redistributing components, introducing replicas, or restructuring communication paths) to improve its quality attributes [31, 30, 26]. For example, Zhao et al. [30] analyzed the energy efficiency of microservices of different architectural decompositions, and Ni et al. [26] improved the performance of an architecture using rule-based refactoring.

When competing properties must be optimized together, multi-objective techniques—such as genetic algorithms [25, 24]—are well suited to this task, particularly when the properties can be expressed as numerical quantities. However, non-functional properties are often challenging to optimize due to the complexity of the architectural models, the computational cost of running the analysis solvers, and the large search space [1, 26]. A common strategy to mitigate computational cost is to impose time budgets on the optimization process [14, 3]. Although effective for execution time, such constraints may hinder the exploration of the design space, affecting the quality of architectural alternatives.

In this work, we address the high computational cost of evaluating the optimization objectives via solvers, which hampers multi-objective architectural optimization in practice. We propose a novel *approach based on surrogate models to estimate non-functional properties of architectures*, approximating the behavior of the solvers with cheaper Machine Learning models. In particular, our approach leverages regression techniques to build surrogate models based on features extracted from the architectures handled by the optimization engine.

We evaluate our approach on *CoCoME* [16], which is a well-known case study in the literature. Furthermore, we compare the surrogate results with those achieved by a state-of-the-art search-based technique [12]. Through this investigation, we aim to answer the following research questions:

- **RQ1:** Can surrogate models reduce the overall execution time of multi-objective architectural refactoring optimization while maintaining solution quality?
- **RQ2:** Are there differences in the structural characteristics of the architecture candidates generated by a surrogate-assisted approach and those generated by a standard search-based optimization engine?

Our results show that surrogate models can significantly reduce the evaluation time of non-functional properties ($\approx 30\%$) during the optimization process, leading to faster convergence. A trade-off we observed here is less variability of architectural models in the exploration of the design space, as outlined by quality indicators employed to evaluate quality of Pareto fronts. With regard to the architectural models, we noticed that surrogate models lead to different models (in the Pareto front) than those generated by a standard evolutionary search.

2 Related Work

Although multi-objective optimization techniques, such as evolutionary algorithms, can yield good-quality and diverse solutions in different engineering domains, a main drawback is their execution time until reaching convergence.

Along this line, the definition of stopping criteria for the optimization process is a common strategy explored in the literature. Arcuri and Fraser [2, 3] showed

that fixed time budgets influence search-based test generation, while Luong et al. [23] proposed a cost-aware Bayesian optimization method that explicitly incorporates the evaluation cost of candidate solutions. In the context of software architecture optimization, Diaz-Pace et al. [14] examined how time budgets affect multi-objective refactoring search. Their findings indicate that, although budget-aware search reduces overall evaluation time, it may also lead to suboptimal architectural trade-offs depending on the search strategy, highlighting the sensitivity of architecture-level optimization to budget constraints.

Surrogate models offer a complementary strategy by learning fast approximations of expensive evaluation functions. Their effectiveness has been demonstrated in surrogate-assisted evolutionary algorithms [17], where predictions replace a portion of fitness evaluations to accelerate convergence on expensive multi-objective problems. Recent research has examined surrogate-based optimization specifically for system architecture problems, which are characterized by hierarchical, mixed-discrete, and multi-objective design spaces. Prior work has proposed Bayesian optimization with specialized structure [7], as well as methods that explicitly model hidden constraints such as solver failures or infeasible geometries to improve search performance [8]. Also, Bussemaker et al. [6] evaluated how well surrogate-assisted optimizers approximate the true Pareto front under fixed evaluation budgets, how efficiently they progress under convergence criteria, and how robust they remain in the presence of hidden constraints.

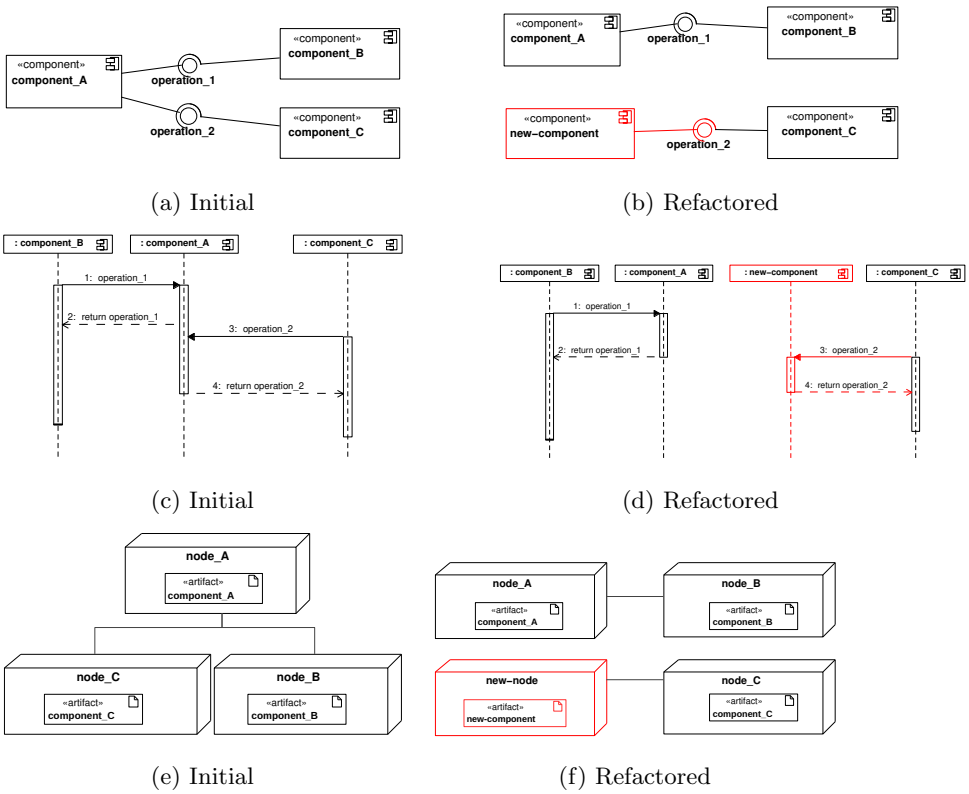
Despite their success in other domains, surrogates remain comparatively underexplored in software engineering. Only a few works have attempted to introduce surrogate models in architecture optimization. The closest one is [28], which trained surrogates to approximate performance and modifiability metrics in tree-based architectural search. While the results demonstrate the feasibility of predicting non-functional values from architectural features, the design of the approach had practical limitations for "live" optimization scenarios. In particular, the surrogates were trained offline on precomputed datasets and had no influence on the optimization process.

In contrast to these earlier efforts, this work integrates surrogate models directly into the evolutionary architecture-refactoring algorithm. Our surrogates are trained incrementally on solver evaluations generated during search, and their predictions guide exploration by replacing a portion of costly solver invocations. This design allows the optimization to scale to larger design spaces while maintaining the quality of the solutions being explored.

3 Approach

The multi-objective optimization process establishes a mapping between candidate architectural models and a multi-valued response surface, where each point represents the values of the non-functional properties of interest for an architecture. From a data-driven perspective, this process generates new pairs of architectural models and objective values, progressively forming a dataset that becomes the foundation for surrogate modeling.

Our work departs from an architectural refactoring optimization framework [11], in which a genetic algorithm (GA) iteratively transforms architecture mod-



(a) Initial (b) Refactored

(c) Initial (d) Refactored

(e) Initial (f) Refactored

Fig. 1: Example of action for selecting an operation to be moved to a new component to be deployed on a new node (*motn*). The elements added by the refactoring action, across several UML views for the same architecture, are in red.

els using predefined refactoring actions and evaluates each candidate through solver-based analyses of non-functional properties. The framework, called `EASIER`, is specifically designed for capturing architectural models using evolutionary algorithms and treating non-functional properties as objectives being evaluated on those models. Architectures are typically expressed as UML models, which are converted into quantitative models for analysis. These analysis models are usually supported by specialized solvers. NSGAII is the default GA supported by `EASIER` to find a set of Pareto-optimal architecture. Essentially, NSGAII involves the following steps: initialize a random population of candidate architectural models), evaluate fitness of individual architectures on multiple objectives (solver-based assessment of non-functional properties), perform a non-dominated sorting of these individuals, calculate a crowding distance for diversity, generate offspring via GA operators, and merge populations to select the best individuals. These steps are repeated for a fixed number of iterations. The GA operators are predefined refactoring actions for architectural models. For the non-functional properties, we consider competing objectives such as performance, reliability, energy and cost, each one with its corresponding analysis model.

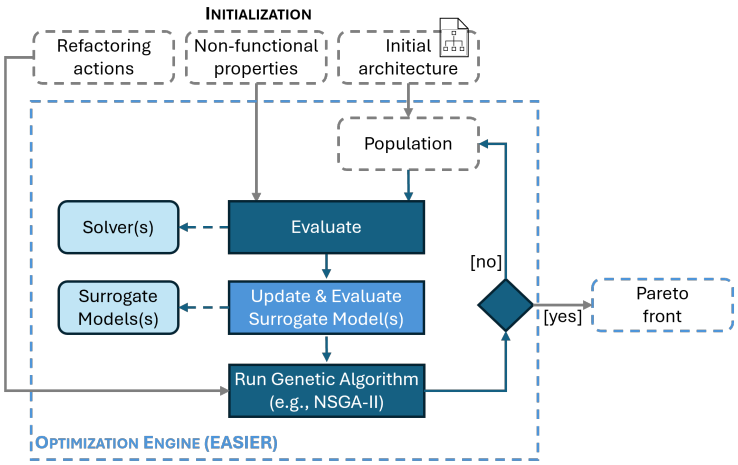


Fig. 2: Workflow of the proposed approach integrating surrogate models into a genetic algorithm engine for architectural refactoring optimization (`EASIER`).

The refactoring catalog employed in this work comprises actions designed to enhance a variety of non-functional properties of a software system [12]. These actions involve: cloning a node (`clone`) to reduce utilization of an existing platform device, re-deploying a component to another node (`rede`) to reduce the load of the source node, transferring the logic of a given operation from one component to another (`move`), and selecting an operation to be moved to a new component to be deployed on a new node (`motn`) also to reduce load of the original component and node. An example of the `motn` action is given in Figure 1. A feasibility engine [1] ensures the validity of the sequences of refactoring actions applied to the architectures.

We extend this base approach by integrating surrogate models (SMs) into the evaluation phase, as illustrated in Figure 2. Our workflow begins with the initialization of a population of candidate solutions, each representing an architectural alternative of the software model. The fitness of each architecture with respect to a given objective can be computed either by invoking the corresponding solver, or by querying an SM. When an SM is used, the objective values are predicted from features extracted from the architecture. In each GA iteration, only a subset of the architectural population is evaluated using the solvers to obtain ground-truth objective values, while the remaining architectures are evaluated by the SMs. An initial set of surrogates is trained offline and injected into the optimization engine before the search begins. To provide flexibility, we maintain one SM per objective, allowing each surrogate to evolve independently based on the distribution and difficulty of its corresponding objective.

As SMs inevitably introduce prediction errors, and because the GA may explore regions of the design space that were not represented in the initial training data, the SMs can be incrementally updated through the search. At predefined iterations, their training sets are augmented with samples from the most recent

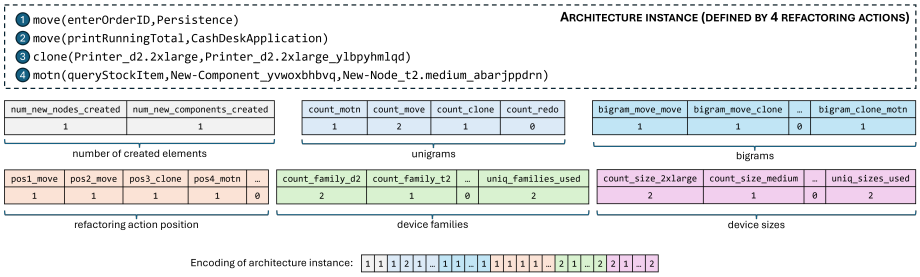


Fig. 3: Encoding of a sequence of refactoring actions leading to a candidate architecture into feature vector with different feature categories.

solver evaluations, as proposed in [28]. This periodic re-training helps maintain the relevance and accuracy of the surrogates during the optimization process.

The approach relies on UML to model the software architectures under optimization. Specifically, as shown in Figure 1, we utilize component diagrams to represent static relationships among software components, sequence diagrams to capture dynamic system behavior (*e.g.*, sequences of operations), and deployment diagrams to model the hardware architecture. Since standard UML does not natively support performance modeling, we leverage established UML profiles [21, 11, 15]. For deployment nodes, we selected a set of representative Amazon EC2 instances[4], namely: `d2.2xlarge`, `m5ad.xlarge`, `t2.medium`, `t2.micro`, to capture diverse cost-performance trade-offs [12]. Instance costs were taken from the Amazon EC2 Price History dataset[5].

3.1 Feature Encoding of Architectural Models

Our SMs require a representation of each architecture of a population to a format (*i.e.*, features) suitable for regression. Since each architecture is defined by a sequence of refactoring actions (applied to an initial architecture), it cannot be processed directly by the SMs. Thus, we construct a tabular, numeric representation in which each candidate sequence is converted into a fixed-length feature vector describing its refactoring and architectural characteristics. This encoding is defined once at the onset and remains stable in their features across all subsequent populations, considering that refactoring actions might introduce new nodes or components in some architectural models.

To encode the sequence structure in a way that is both compact and expressive, we borrow ideas from natural language processing [19, 29] and bio-molecular sequence analysis [20]. In those fields, sequences (e.g., words in a sentence or amino acids in a protein) are described not only by the individual elements that occur but also by the short-range patterns in which they appear. In our context, the encoding includes *action unigrams*, which count how many times each refactoring action occurs, and *action bigrams*, which count how often one

[4] Amazon EC2 Instances Carbon Footprint Estimator: `https://docs.google.com/spreadsheets/d/1DqYgQnEDLQVQm5acMAhLgHLD8xXCG9BIrk-_Nv6jF3k`

[5] Amazon EC2 Spot Price History: `https://zenodo.org/doi/10.5281/zenodo.5880792`

action immediately follows another. While unigrams reflect the overall distribution of refactoring types, bigrams capture local dependencies that often carry architectural meaning. For example, a `clone` followed immediately by a `motn` suggests a distinct pattern that contributes to fault tolerance and load distribution, compared to the same actions appearing far apart in the sequence. By incorporating short-range co-occurrence patterns, we obtain a lightweight but informative description of how refactoring actions interact to lead to specific architecture configurations. For instance, Figure 3 shows how a sequence with four actions is mapped to a feature vector. Note that teatures are grouped into different categories, namely: elements created (nodes and components from UML models), action counts (unigrams), bigrams, positional action counts, and device families and sizes (for AWS characteristics of nodes), among others.

To retain information about the temporal structure of the transformation process, we additionally encode which actions appear at each position of the sequence. As some refactoring actions can introduce new architectural elements, the encoding also includes indicators quantifying the structural expansion of the architecture, such as the number of newly created components and nodes. For instance, having more components or nodes often correlates with horizontal scaling, functional decomposition, increased architectural complexity or potential points of failure, among other aspects.

A representation challenge is that node identifiers in the architectural model often embed hardware information in non-uniform strings (*e.g.*, `StoreServer_d2.2xlarge`) To retain this contextual information without relying on raw identifiers, we normalized node names by extracting two stable tokens: the hardware *family* (*e.g.*, `t2`, `m6i`, `d2`) and the *size* class (*e.g.*, `medium`, `xlarge`). These tokens are mapped to a finite vocabulary determined at initialization, and their occurrences are tracked across the refactoring actions.

3.2 Incremental Regression Models

Once the architecture candidates are encoded, their non-functional objective values computed by the solvers are seen as regression targets for the SMs. Regression is a type of supervised learning in which the goal is to predict continuous values based on input data. Since we deal with multiple objectives, our approach internally creates a single regressor per objective.

We support incremental model training (see Figure 2) to reflect the dynamic nature of the GA optimization process. For the SMs, an initial regression model is learned using a fixed number of populations. The idea is to adjust the SM with new architecture candidates while still retaining patterns extracted from previous instances. Thus, every k iterations, the model can be updated with a fraction of the new data available from each population. Gradient boosting techniques (e.g., *XGBoost*) are appealing in this scenario, because the underlying estimators (tree ensembles) can be updated with newly-trained estimators for incoming data.

4 Evaluation

The research methodology is an empirical study based on a well-known case-study from the literature, on which we performed different experiments. We

evaluate our approach by formulating two research questions that guided our experimental design. We aim to assess how surrogate models can support an efficient multi-objective optimization of architectural models.

• **RQ1:** *Can surrogate models reduce the overall execution time of multi-objective architectural refactoring optimization while maintaining solution quality?*

This question investigates whether integrating SMs into the optimization process yields meaningful reductions in execution time compared to exclusively relying on solver-based analyses. Execution time was selected the definitive metric to ensure findings remain independent of specific hardware resource allocations and the fluctuating economics of cloud environments, offering a direct picture of the optimization's efficiency. Furthermore, the question examines the effect of surrogate models on the quality of the Pareto solutions, using indicators such as hypervolume and inverted generational distance to determine how closely the obtained front approximates the reference one.

• **RQ2:** *Are there differences in the structural characteristics of the architecture candidates generated by a surrogate-assisted approach and those generated by a standard search-based optimization engine?*

Unlike the previous question that looks at the objective space, this question focuses on the architectural models being explored during the optimization process, and how surrogate models shape the use of certain refactoring actions for the architecture candidates.

4.1 Experimental Setup

We compared a standard multi-objective architecture optimization setup using *NSGA-II* as the underlying GA with our surrogate-assisted approach, in which regression models approximate solver evaluations[6]. The surrogates were implemented using *XGBoost*, which supports incremental model updates. The regression models were generated and consumed by the optimization engine using a FastAPI server. Once pre-trained, the SMs provided predictions of objectives values for each population. In each optimization iteration, 50% of the candidate architectures were evaluated with the solver, while the remaining architectures were assigned objective values predicted by the SMs. The SMs were re-trained at pre-specified iterations

Regarding the SMs, we used two strategies: a pre-trained model (without re-training), and a periodic re-training mode in which the models were re-trained every k steps. The former strategy was used as a baseline for estimating the execution time reduction of SMs, while the latter served to evaluate the potential benefits of updating SMs during the optimization process. To evaluate the impact of re-training SMs, two frequencies (k step) were tested: every 2 and 5 iterations.

Experiments were conducted on a well-known case study from the software architecture optimization literature. We evaluated performance along two dimensions: computational time and quality indicators for the resulting Pareto front (**RQ1**). We also analyzed the structural characteristics of the architecture

[6] We chose to use NSGA-II because it is widely used in the literature. Recently, NSGA-III has shown advantages over NSGA-II only with a large set of objectives (15+) [13].

candidates generated by both approaches, focusing on the refactoring actions applied to the architectural models (**RQ2**).

The GA configuration followed common literature settings [30, 1, 26]: a population of 12 architecture candidates, chromosomes (*i.e.*, a sequence of refactoring actions) of length 4, and 102 iterations per run. Each configuration was executed 31 times on each system to mitigate variability[7]. Furthermore, the optimization process was configured to evaluate 7 objectives, involving performance, reliability, energy, price and number of changes, among others. Therefore, the found Pareto fronts include solutions, i.e., alternative architectures, that represent different trade-offs among these objectives.

For deployment nodes, we selected a set of representative Amazon EC2 instances[8], namely: `d2.2xlarge`, `m5ad.xlarge`, `t2.medium`, `t2.micro`, to capture diverse cost-performance trade-offs [12]. Instance costs were taken from the Amazon EC2 Price History dataset[9]. It is worth noting that we selected the AWS instances based on the availability of datasets, which to the best of our knowledge, are the only ones that provide detailed information on both performance and energy consumption for a variety of instance types[10].

Experiments were carried out on a cluster of 3 Dell PowerEdge C6525 servers, each equipped with 2 AMD EPYC 7282 2.80GHz CPUs and 512 GiB of RAM[11]. The target case-study was *CoCoME (CCM)*, which is a reference system for non-functional model-based analyses [16]. It describes a trading system consisting of several stores, each containing one or more cash desks. A cash desk is equipped with hardware and software elements required to serve customers (*e.g.*, a cash box, printer, bar bode scanner). *CCM* captures operational scenarios such as scanning products, processing payments, and managing stock replenishment.

For the initial training of the surrogates (one per objective), we took populations from two *EASIER* iterations ($\approx$ 32 architectures). Although this is a small sample size, the SMs can be adjusted quickly with periodic updates from the optimization process. *XGBoost* was configured with 500 estimators and a learning rate of 0.1. This pretraining setting is not computationally expensive (3 min on average in total) for *XGBoost*.

4.2 Metrics

To evaluate the quality of the Pareto fronts produced by the optimization approaches, we rely on standard multi-objective quality indicators. In particular, we employ Hypervolume (HV) [9, 4], Inverted Generational Distance (IGD+) [18].

[7] These parameters reflect typical choices in architecture optimization studies, considering that each population member represents a full software architecture model and each evaluation involves running computationally intensive analysis tools.

[8] Amazon EC2 Instances Carbon Footprint Estimator: `https://docs.google.com/spreadsheets/d/1DqYgQnEDLQVQm5acMAhLgHLD8xXCG9BIrk-_Nv6jF3k`

[9] Amazon EC2 Spot Price History: `https://zenodo.org/doi/10.5281/zenodo.5880792`

[10] These values can easily changed whether other datasets become available. More important, using these AWS datasets do not limit generalizability of the approach.

[11] The replication package is available at `https://doi.org/10.5281/zenodo.17846462`

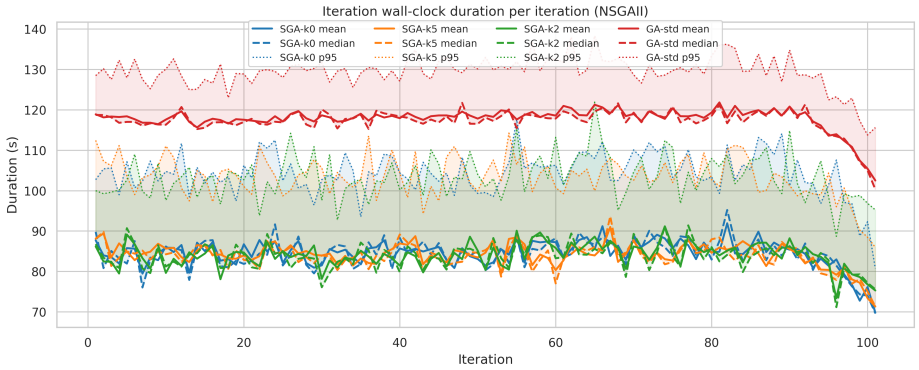


Fig. 4: Per-iteration execution time of the standard GA and the surrogate-based GA across both case studies. *GA-std*: plain genetic algorithm without surrogate. *SGA-k0*: pre-trained surrogate model (no re-training). *SGA-k2*, *SGA-k5*: surrogate re-trained every 2 and 5 iterations, respectively.

These indicators were selected due to their prevalence in prior literature and their complementary nature [22]. For the `IGD+` indicator that requires a reference front, we constructed it by merging the non-dominant solutions from all runs. For `HV` that requires a nadir point, such a reference point was defined as the worst value observed per objective across all executions.

5 Results

5.1 RQ1: Reductions in overall execution time

A comparison of execution times between standard GA and surrogate-based GA is presented in Figure 4. We remark that we have considered the execution time as primary cost metric for our evaluation, as it is a direct measure of the computational efficiency of the optimization process. Furthermore, the execution time is a critical factor in determining the practical applicability of evolutionary-based optimization methods in real-world scenarios. The standard GA exhibited the highest execution time in the first iteration, which then stabilized as iterations progressed. This behavior was expected due to various optimization steps that occurred throughout the optimization process. In contrast, the surrogate-based GA did not exhibit that behavior and achieved a substantial speedup across all iterations, reducing the overall execution time by $\approx 30\%$.

As expected[12], GA-std achieves the best median performance for both indicators across 31 independent runs. In particular, GA-std attains the highest HV, indicating superior coverage of the objective space. Similarly, GA-std yields the lowest IGD+ value, suggesting better proximity to the reference Pareto front and improved distribution of solutions. Among the SGA variants, SGA-k2 shows the closest performance to GA-std, while SGA-k5 consistently exhibits lower HV and higher IGD+ values, indicating reduced Pareto front quality.

[12] The quality indicators are expected to be better for the standard genetic algorithm.

Table 1: Pairwise comparisons between GA-std and SGA variants using Wilcoxon signed-rank tests with Holm correction. p_{Holm}: Holm-corrected p-value; d_z: Cohen's effect size. *GA-std*: plain genetic algorithm without surrogate. *SGA-k0*: pre-trained surrogate model (no re-training). *SGA-k2*, *SGA-k5*: surrogate re-trained every 2 and 5 iterations, respectively.

Comparison	Indicator	p_{Holm}	d_z	Indicator	p_{Holm}	d_z
GA-std vs SGA-k0	HV	0.289	-0.28	IGD+	1.000	-0.19
GA-std vs SGA-k2	HV	0.546	-0.20	IGD+	1.000	0.01
GA-std vs SGA-k5	HV	0.193	-0.44	IGD+	0.471	-0.30

For the comparisons, the variants considered the same group under different scenarios (the quality indicators). Since data was not normally distributed, we used a Friedman test to check for differences and Wilcoxon signed-rank as a post-hoc test to identify differences. The Friedman test revealed a significant difference among variants for HV ($\chi^2 = 9.50$, $p = 0.023$), but not for IGD+ ($p = 0.218$). Pairwise comparisons between GA-std and SGA variants using Wilcoxon with Holm correction (due to repeated measures) are reported in Table 1. No pairwise comparison reached statistical significance after correction ($\alpha = 0.05$). However, effect sizes provide additional insights. For HV, GA-std shows a moderate advantage over SGA-k5 ($d_z = -0.44$) and smaller effects over SGA-k0 and SGA-k2. For IGD+, effect sizes are small, with a moderate trend favoring GA-std over SGA-k5 ($d_z = -0.30$).

The results indicate that GA-std consistently outperforms SGA variants in terms of both coverage (HV) and proximity/diversity (IGD+). While these differences are not statistically significant under multiple comparison correction, the observed effect sizes suggest that the performance gap—particularly between GA-std and SGA-k5—is practically meaningful. Overall, the SGA variants do not provide improvements over the standard GA in this setting, and in some configurations (*e.g.*, SGA-k5) may lead to a degradation in Pareto front quality.

Summary. Answering **RQ1**, the surrogate-assisted approach provides substantial runtime savings by reducing expensive solver calls, while remaining competitive in solution quality when re-training is frequent. Thus, surrogates with frequent re-training can preserve quality of Pareto fronts despite relying on approximate evaluations. Furthermore, quality indicators indicate that differences in Pareto-front quality are limited: `HV` shows only a global trend without significant Holm-corrected pairwise contrasts, and `IGD+` shows no significant differences. Overall, `GA-std` remains a strong baseline, but `SGA-k2` achieves comparable quality, especially for `IGD+`. This aspect is relevant because surrogates reduced execution time by $\approx 30\%$, highlighting a practical quality-time tradeoff.

5.2 RQ2: Characterization of architectural models

To analyze differences in the architectural models between the standard and surrogate-assisted GAs, we used the refactoring actions, described in Section 3, as proxies for understanding the corresponding software architecture.

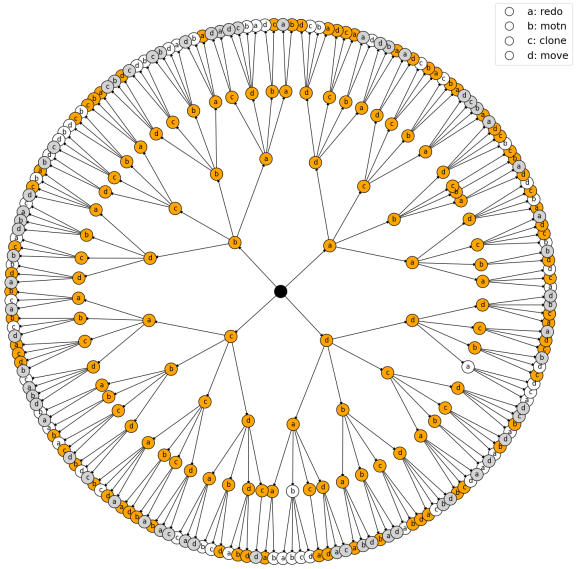


Fig. 5: Sequences of refactoring actions for architectures in the Pareto fronts of both GAs. White nodes indicate sequences only generated by the standard GA, while gray nodes indicate sequences only generated by the surrogate-assisted GA ($k = 5$). Yellow nodes correspond to intersecting sequences from algorithms.

For *CCM*, we considered all the sequences of refactoring actions of the architectures observed in the Pareto fronts of both GAs. For simplicity, we omitted the parameters of the actions and retained only their types. The action sequences were all arranged in a (prefix) tree, as schematially shown in Figure 5. A significant intersection of sequences was observed and, interestingly, the main differences between the two algorithms lie in the last refactoring action of the sequences. This trend was similar, regardless of the k step (for re-training) chosen for the surrogate models.

While the standard GA employed actions `motn` (move operation to new component on new node) and `clone` (clone node) more frequently, the surrogate-assisted GA relied more on actions `redo` (re-deploy existing component) and `move` (relocate operation to existing node). The different action usage profiles indicate differences in the resulting architectural models, and it can also affect the achievement of particular non-functional objectives. For example, since the `motn` and `clone` actions tend to expand the system infrastructure, the standard GA may favor solutions with scalability and robustness (probably involving a higher resource consumption). On the other hand, the preference of the surrogate-assisted GA for actions that re-distribute system components suggests changes that lead to efficient and less costly solutions. As a result, the SMs could contribute to architectures that better satisfy energy, power or price objectives, but lower gains for performance or reliability objectives, when compared to a pure GA algorithm. It should be noticed that these architectural aspects are difficult to assess when considering only quality indicators for the Pareto front.

Summary. Our evidence for answering **RQ2** shows that a significant portion of the sequences explored by both GAs are shared, which means that the surrogate models do not drastically alter the search space exploration in terms of architectural modifications. Nevertheless, the variations in the usage of refactoring actions (towards the end of the sequences) tell that the two types of GAs seem to explore slightly different areas of the design space. Furthermore, these action profiles can influence the achievement of specific non-functional objectives.

5.3 Lessons Learned

Our results highlight a practical trade-off: surrogate models reduce execution time by decreasing expensive invocations to analysis solvers, while Pareto-front quality remains broadly comparable but can vary across indicators (**RQ1**). At the same time, standard and surrogate-assisted GAs exhibit relatively similar patterns of refactoring-action usage across sequence positions, except for the last action of the sequence, consistent with the findings of **RQ2**. The architectural solutions explored by the surrogate-assisted GA are not necessarily a subset of those explored by a standard GA. Although more experimentation is needed, our results suggest that SMs can lead to solutions with alternative tradeoffs.

The predictive performance of the SMs depend mainly on having a more or less regular (e.g., normal) distribution of the series of objective values, which makes them treatable with non-linear regression. For *XGBoost*, in particular, we were able to obtain good results with a relatively small number of estimators, which made (re-)training quite fast. Having a fixed set of features also contributed to keeping the execution times low, because it makes the feature set independent from the sizes of the architectural models. Increasing the number of estimators, e.g., to achieve better predictive performance or to compensate for irregular objective distributions, would likely increase the SM training times.

As for the encoding strategy, it proved to be appropriate for the *CoCoME* predictions for different architectural models, by relying on unigrams and bigrams about components, nodes and refactoring actions, regardless of the size or structural variations of the architectural models. Based on initial results with another case study (`TrainTicket`), we believe that the encoding strategy can be successfully extrapolated to other systems.

Overall, surrogate-assisted optimization is promising, especially under constrained computational budgets, while motivating further study of surrogate accuracy and role of action usage profiles in the search through the design space.

6 Threats to validity

We identified several threats to internal, construct, and external, and conclusion validity in our experimental evaluation.

Construct validity. A threat arises from the lack of feature selection or feature importance analysis, as we included all features from our encoding schema without filtering them. Irrelevant or redundant features might have introduced noise into the regression models, while key features might not have been emphasized appropriately. In addition, we split the population instances into halves to decide which ones were tackled with the solvers and which ones were predicted with the

SMs, but we did not try alternative sampling ratios. These factors could have affected the predictive accuracy of surrogates or their training times.

External validity. The generalizability of our results is limited by several experimental assumptions. Our evaluation relies on architectural models expressed in a particular modeling notation. Although UML notation is widely used in performance and reliability analysis, other modeling languages might allow different architectural structures, refactoring primitives, or analytic solvers. As a result, the behavior of SMs might vary when applied to another setting.

A further threat arises from the limited refactoring space used by the optimization engine. Architecture candidates were generated using a predefined set of four refactoring actions and a fixed sequence length, which stem from common architecture refactorings. Finally, the datasets used are relatively small compared to typical machine learning benchmarks. Surrogate behavior, accuracy trends, and sampling effects may differ in larger or noisier datasets, especially those produced by long-running or industrial-scale optimization processes.

Internal validity. Parts of our machine learning pipeline involve inherent nondeterminism. Although we controlled randomness through repeated runs and fixed seeds, different configurations and executions might lead to variations in training/test splits, sampling strategy or surrogate performance, potentially affecting our conclusions. Also, differences in the execution environment (*e.g.*, hardware, engine configuration) might affect timing measurements.

Conclusion validity. We did not evaluate human factors such as how architects interpret or trust surrogate predictions and the derived architectural models, which might influence the practical adoption of surrogate-based optimization.

7 Conclusion

In this work, we investigated the feasibility of using surrogate models to accelerate multi-objective quality-attribute optimization of architectural models. The central idea was to interleave expensive solver-based evaluations with fast, but potentially less accurate, surrogate predictions, thereby reducing computation time while preserving the effectiveness of the optimization process. Our study confirms that surrogate models can be integrated into the architecture optimization process with minimal overhead and that they offer meaningful efficiency gains in realistic settings, albeit with less variability on the Pareto fronts. The execution time reductions are important, as the computational cost is a typical concern for multi-objective optimization, and also because in the architecture domain these reductions can enable practitioners to explore larger design spaces (with a given budget). Furthermore, the search space (*i.e.*, the structure of the architecture candidates) explored by the surrogate-assisted GA seems to differ slightly from that taken as the baseline. More experiments should be performed to confirm this trend and the effects of surrogates on the objective values. Also, we would like to study whether surrogates can have a potential impact on the monetary cost and resource consumption of the optimizations.

We envision several directions for future work for our approach. Incorporating drift detection over incoming data batches would make it possible to

selectively re-train the surrogates upon data distribution changes, thus improving the performance of the current approach. Exploring alternative encodings for architectural models and refactoring actions (*e.g.*, graph-based or embeddings techniques) may yield more informative feature spaces for surrogate modeling. Finally, the decision of which candidates to evaluate with solvers can be enhanced through adaptive sampling strategies that account for uncertainty, diversity, or predicted model reliability.

Data Availability. For the surrogate implementation, the datasets, and experimental results see `https://github.com/danieledipompeo/easier-surrogate`.

Acknowledgments. This research was funded in whole or in part by the Austrian Science Fund (FWF): 10.55776/COE12. Also, this work was partially supported by PICT-2021-00757 project (Argentina).

References

[1] Arcelli, D., Cortellessa, V., D'Emidio, M., Di Pompeo, D.: EASIER: an evolutionary approach for multi-objective software architecture refactoring. In: 18th IEEE International Conference on Software Architecture, ICSA, pp. 105–114 (2018)
[2] Arcuri, A., Fraser, G.: On Parameter Tuning in Search Based Software Engineering, LNCS, vol. 6956, p. 33–47 (2011)
[3] Arcuri, A., Fraser, G.: Parameter tuning or default values? an empirical investigation in search-based software engineering. Empirical Software Engineering **18**(3), 594–623 (2013), ISSN 1382-3256, 1573-7616
[4] Beume, N., Naujoks, B., Emmerich, M.: Sms-emoa: Multiobjective selection based on dominated hypervolume. Eur. J. Oper. Res. **181**(3), 1653–1669 (2007)
[5] Busch, A., Fuchss, D., Koziolek, A.: Peropteryx: Automated improvement of software architectures. In: IEEE International Conference on Software Architecture Companion, , Hamburg, Germany, March 25-26, 2019, pp. 162–165 (2019)
[6] Bussemaker, J.H., Bartoli, N., Lefebvre, T., Ciampa, P.D., Nagel, B.: Effectiveness of surrogate-based optimization algorithms for system architecture optimization. In: AIAA Aviation 2021 forum, p. 3095 (2021)
[7] Bussemaker, J.H., Saves, P., Bartoli, N., Lefebvre, T., Lafage, R.: System architecture optimization strategies: dealing with expensive hierarchical problems. Journal of Global Optimization **91**(4), 851–895 (2025)
[8] Bussemaker, J.H., Saves, P., Bartoli, N., Lefebvre, T., Nagel, B.: Surrogate-based optimization of system architectures subject to hidden constraints. In: AIAA AVIATION FORUM AND ASCEND 2024, p. 4401 (2024)
[9] Cao, Y., Smucker, B.J., Robinson, T.J.: On using the hypervolume indicator to compare pareto fronts: Applications to multi-criteria optimal experimental design. Journal of Statistical Planning and Inference **160**, 60–74 (2015), ISSN 03783758
[10] Cortellessa, V., Di Pompeo, D.: Analyzing the sensitivity of multi-objective software architecture refactoring to configuration characteristics. Information and Software Technology **135**, 106568 (2021), ISSN 09505849
[11] Cortellessa, V., Di Pompeo, D., Stoico, V., Tucci, M.: Many-objective optimization of non-functional attributes based on refactoring of software models. Inf. Softw. Technol. **157**, 107159 (2023)
[12] Cortellessa, V., Di Pompeo, D., Tucci, M.: Exploring sustainable alternatives for the deployment of microservices architectures in the cloud. In: 21st IEEE International Conference on Software Architecture, ICSA, pp. 34–45 (2024)

[13] Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Trans. Evol. Comput. **18**(4), 577–601 (2014)
[14] Diaz-Pace, J.A., Di Pompeo, D., Tucci, M.: On the role of search budgets in model-based software refactoring optimization **33**(1), 18 (2025), ISSN 1573-7535
[15] Franks, G., Al-Omari, T., Woodside, M., Das, O., Derisavi, S.: Enhanced modeling and solution of layered queueing networks. IEEE Trans. Softw. Eng. **35**(2) (2008)
[16] Herold, S., Klus, H., Welsch, Y., Deiters, C., Rausch, A., Reussner, R., Krogmann, K., Koziolek, H., Mirandola, R., Hummel, B., Meisinger, M., Pfaller, C.: CoCoME - The Common Component Modeling Example, LNCS, vol. 5153, p. 16–53 (2008)
[17] Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evol. Comput. **23**(3), 343–367 (2015)
[18] Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y.: Modified distance calculation in generational distance and inverted generational distance. In: Evolutionary Multi-Criterion Optimization, p. 110–125 (2015)
[19] Jurafsky, D., Martin, J.H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Third edition draft edn. (2024)
[20] Leslie, C.S., Eskin, E., Noble, W.S.: The spectrum kernel: A string kernel for svm protein classification. Pacific Symposium on Biocomputing pp. 564–75 (2001)
[21] Li, C., Altamimi, T., Zargari, M.H., Casale, G., Petriu, D.C.: Tulsa: A tool for transforming UML to layered queueing networks for performance analysis of data intensive applications. In: 14th QEST, LNCS, vol. 10503, pp. 295–299 (2017)
[22] Li, M., Yao, X.: Quality evaluation of solution sets in multiobjective optimisation: A survey. ACM Comput. Surv. **52**(2), 1–38 (2020), ISSN 0360-0300, 1557-7341
[23] Luong, P., Nguyen, D., Gupta, S., Rana, S., Venkatesh, S.: Adaptive cost-aware bayesian optimization. Knowl. Based Syst. **232**, 107481 (11 2021), ISSN 0950-7051
[24] Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Proceedings of the First Joint WOSP/SIPEW Int. Conf. Perform. Eng., p. 105–116 (2010)
[25] Meedeniya, I., Buhnova, B., Aleti, A., Grunske, L.: Architecture-Driven Reliability and Energy Optimization for Complex Embedded Systems, LNCS, vol. 6093, p. 52–67 (2010)
[26] Ni, Y., Du, X., Ye, P., Minku, L.L., Yao, X., Harman, M., Xiao, R.: Multi-objective software performance optimisation at the architecture level using randomised search rules. Inf. Softw. Technol. **135**, 106565 (2021)
[27] Rago, A., Vidal, S., Diaz-Pace, J.A., Frank, S., van Hoorn, A.: Distributed quality-attribute optimization of software architectures. In: Proceedings of the 11th SB-CARS, p. 1–10 (2017)
[28] Titov, V., Pace, J.A.D., Frank, S., van Hoorn, A.: Architecture optimization using surrogate-based incremental learning for quality-attribute analyses. In: 22nd IEEE International Conference on Software Architecture, ICSA, pp. 278–288 (2025)
[29] Wang, S.I., Manning, C.D.: Baselines and bigrams: Simple, good sentiment and topic classification. In: Proceedings of the 50th ACL, pp. 90–94 (2012)
[30] Zhao, Y., Matteis, T.D., Bogner, J.: How does microservice granularity impact energy consumption and performance? A controlled experiment. In: 22nd IEEE International Conference on Software Architecture, ICSA, pp. 84–95 (2025)
[31] Zimmermann, O.: Architectural refactoring: A task-centric view on software evolution. IEEE Softw. **32**(2), 26–29 (2015)