Speeding up Quality-attribute Analysis in Architecture Optimization using ML-based Surrogate Models

EMMANUEL A. TASSONE, Facultad de Matemática, Astronomía, Física y Computación - CONICET, Argentina
J. ANDRES DIAZ-PACE, ISISTAN Research Institute – UNCPBA University & CONICET, Globant, Argentina
ANTONELA TOMMASEL, ISISTAN Research Institute – UNCPBA University & CONICET, Johannes Kepler University, Argentina, Austria
VADIM TITOV, University of Hamburg, Germany
SEBASTIAN FRANK, University of Hamburg, Germany
ANDRE VAN HOORN, University of Hamburg, Germany

Automated architecture optimization has been commonly addressed through multi-objective search techniques. While these techniques are helpful for assisting architects in navigating large sets of alternative designs, they pose efficiency challenges in large design spaces. This is mainly because evaluating architecture alternatives often requires model transformations and the execution of specialized analysis tools (e.g., LQN solvers for performance), which are computationally expensive when invoked repeatedly during the search process. In this context, surrogate models, which approximate the behavior of the quality-attribute solvers at a lower computational cost, offer a promising way to scale up architecture optimization. In this paper, we adapt an architecture exploration framework based on tree search by combining solver-based and surrogate-based evaluations with incremental learning techniques. Our surrogates are built from data generated during the search process through regression models that exploit both the structure of the architecture instances and the architecture transformations (tactics) applied by the optimization engine. We compare single-output and multi-output regression variants across two case studies from the literature (*ST+* and *CoCoME*), focusing on the accuracy-cost tradeoff and the effects of surrogate design on this tradeoff. Experimental simulations with datasets generated from the case studies demonstrated that surrogate models can achieve good reductions in computation time while still maintaining a reasonable level of accuracy in quality-attribute predictions. Furthermore, the choice between single-output or multi-output regression depends on factors such as the size and complexity of the architecture system, as well as the number, type, and interactions among the objectives. Overall, the proposed approach fosters surrogate modeling as an effective strategy to make architecture exploration more efficient and to enable the analysis of larger design spaces.

Additional Key Words and Phrases: Software architecture optimization, tree search, Machine learning surrogates, Active learning, Quality attributes, Architecture evaluation

Authors' addresses: Emmanuel A. Tassone, emmanuel.tassone@unc.edu.ar, Facultad de Matemática, Astronomía, Física y Computación - CONICET, Argentina; J. Andres Diaz-Pace, ISISTAN Research Institute – UNCPBA University & CONICET, Globant, Tandil, Argentina, andres.diazpace@isistan.unicen.edu.ar; Antonela Tommasel, ISISTAN Research Institute – UNCPBA University & CONICET, Johannes Kepler University, Tandil, Argentina, Austria, antonela.tommasel@isistan.unicen.edu.ar; Vadim Titov, University of Hamburg, Hamburg, Germany; Sebastian Frank, University of Hamburg, Hamburg, Germany; Andre van Hoorn, University of Hamburg, Hamburg, Germany.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
Manuscript submitted to ACM

ACM Reference Format:
Emmanuel A. Tassone, J. Andres Diaz-Pace, Antonela Tommasel, Vadim Titov, Sebastian Frank, and Andre van Hoorn. 2018. Speeding up Quality-attribute Analysis in Architecture Optimization using ML-based Surrogate Models. 1, 1 (October 2018), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

To support the exploration of alternative software architectures to satisfy a set of quality-attribute requirements, several search-based approaches that employ multi-objective architecture optimization have been proposed [1, 2, 4, 16, 31, 41]. These optimization approaches involve an iterative process with certain rules for automatically refactoring an initial architecture and deriving alternative architectures. Although these approaches are useful for helping architects navigate a large design space, they often face efficiency challenges. Specifically, the generation and evaluation of architecture alternatives usually relies on predefined analysis models (or solvers), such as Layered Queuing Networks (LQNs) for performance [23] or rule-based detection for performance antipatterns [16]. These solvers transform each architecture into an internal representation and then evaluate it, either analytically or through simulations, to compute different metrics such as response time, number of antipatterns (for performance), cost (for modifiability), or probability of failure (for availability). These metrics quantify the quality-attribute properties of an architecture that constitute the optimization objectives. However, the transform-and-evaluate process above can take considerable computational resources (e.g., from 30 secs up to 2 min per architecture), particularly when executed several times during the optimization cycles and on non-trivial architecture instances. In some cases, evaluating certain objectives (e.g., performance via simulation) is significantly more expensive than others. When expensive objectives dominate the evaluation process, expanding the set of explored architectures becomes difficult, limiting the applicability of architecture optimization approaches in practice.

This challenge has been faced by other engineering disciplines [30], mostly in cases of multi-objective optimization of numeric variables. In this context, *surrogate models* have been shown to be effective in reducing the computational costs of computational research. evaluating complex objective functions. By a surrogate model, we refer to an approximation of the original model or function, such as the computation of an optimization objective, which is cheaper to construct — oftentimes following a *data-driven* strategy. In multi-objective architecture optimization, surrogate models can approximate the outputs of quality-attribute solvers through Machine Learning (ML) techniques such as regression [3, 11, 32], based on features to characterize the architectures. Recently, some works have tackled this direction to learn performance models and make inferences from architectural configurations [29, 51]. We argue that the effective use of surrogates in architecture optimization depends on addressing three main issues: i) how to represent architectures through informative features, ii) the choice of an appropriate regression technique for prediction, and iii) how to couple surrogate within the optimization process.

In prior work [47], we explored ML-based surrogate models for the architecture optimization problem using an approach called IAL, which combined Incremental learning, Active Learning and multi-output regression. The incremental part of the approach is intended to reflect the dynamic nature of an optimization in which data come in batches, as the cycles of search (iterations) of the optimization proceed. This means that a regression model is learned with an initial batch of architectural data and progressively updated as new data (from the optimization iterations) are available. The active learning part, in turn, is intended to sample a fraction of a data batch to invoke the solvers and update (i.e., re-train) the regression model, as a mechanism to control the computational cost allocated to the solvers. In initial experiments with two architecture case studies and gradient boosting algorithms, we observed positive results for the

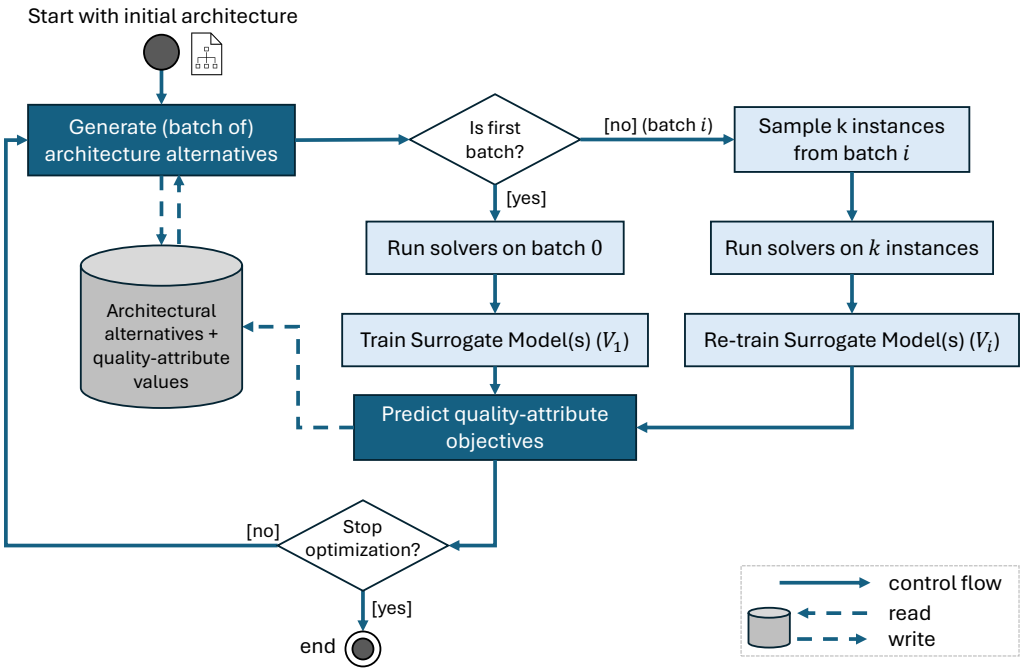


Fig. 1. Integrating quality-attribute solvers and surrogate models in an architecture optimization process.

tradeoff between computational time savings and prediction accuracy when compared with ground truth datasets. However, since we relied on multi-output regression, all objectives were treated as a single unit for sampling and surrogate (re-)training. This ignored the fact that some objectives can be more expensive to evaluate than others or might evolve differently during the optimization. Furthermore, we used random sampling for active learning, but we did not perform a systematic evaluation of this strategy in the prediction errors of the model.

In this paper, we extend the `IAL` approach by introducing individual surrogate models per objective, allowing each regression model to evolve independently and adapt to the specific characteristics of its target objective, if necessary for the optimization problem. We further conduct a detailed evaluation along two dimensions: i) a comparison between multi-output and single-output regression using the `CatBoost` algorithm [9, 40], and ii) a Monte Carlo analysis [36, 42] to study how uncertainty in active learning affects surrogate construction and prediction stability. Our experimental results showed that using multiple single-output regressors yields slightly more accurate and computationally efficient results than using a single multi-output regressor, although the results are dependent on the architectural problem and number of objectives at hand. Finally, Monte Carlo methods provided a stochastic framework for quantifying uncertainty through repeated random sampling, making them well-suited for assessing the robustness of surrogate modeling [44].

The rest of the article is organized as follows. Section 2 provides background information on multi-objective architecture optimization, with an emphasis on the quality-attribute evaluation and transformation of architectures. Section 3 describes how surrogates are modeled, built and incrementally updated using the *IAL* strategy, which considers one surrogate per objective and random sampling for active learning. Section 4 presents the evaluation methodology

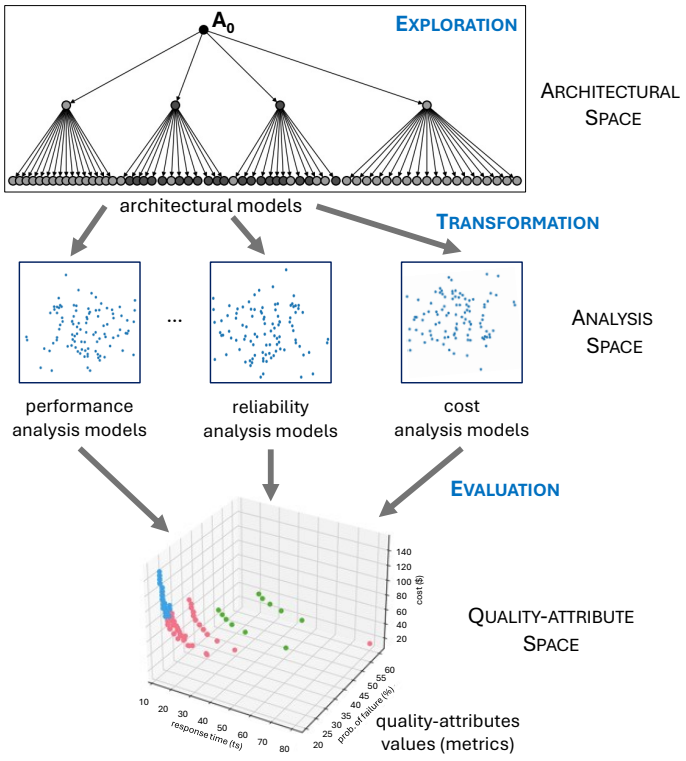


Fig. 2. Exploration, transformation, and evaluation phases in a multi-objective optimization process for software architectures.

using `CatBoost` as the regression algorithm and Monte Carlo simulations. Section 5 reports on the results of experiments with case study datasets. Section 6 discusses threats to validity while Section 7 covers related work. Finally, Section 8 gives the conclusions and outlines future work.

2 THE ARCHITECTURE OPTIMIZATION CYCLE

A typical architecture optimization process works in iterations and involves three phases, namely: i) exploration (or generation), ii) transformation, and iii) evaluation, as illustrated in Fig. 2. The cycle begins with an initial architectural model (A_0) that is fed into an optimization engine. In the *search phase*, the engine applies predefined refactoring operations (or architectural tactics [6]) to generate a set of candidate architectures. Each generated architecture is an alternative for satisfying the quality-attribute objectives. To quantify the degrees of satisfaction, in the *transformation phase*, each architecture alternative is mapped to multiple analysis models to be evaluated by quality-specific solvers. Then, in the *evaluation phase*, the assessment of the architectures leads to quantitative metrics such as response time, modifiability cost, or failure probability in the quality-attribute space. The cycle continues and newly generated candidates become inputs for further exploration, transformation, and evaluation. The process terminates when a stopping criterion (e.g., a maximum number of search iterations) is met, and the generated architectures are returned as output. This style of architecture optimization is exemplified by the *SQuAT* framework [41], which employs a tree-based expansion heuristic to explore architecture alternatives.

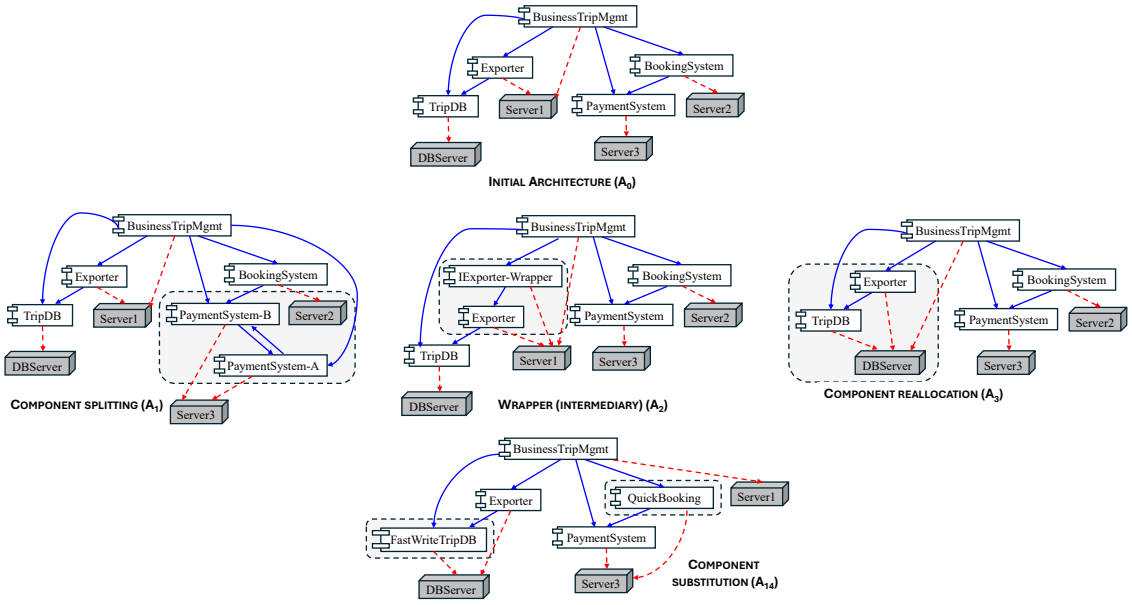


Fig. 3. Example of applying tactics to the initial ST+ architecture A_0 to derive alternatives A_1, A_2, A_3 and A_{14}.

To understand how the *search phase* works, Fig. 3 shows an example of the ST+ system [41], in which different operations (tactics) are applied to an initial architecture A_0 to derive four different alternatives. ST+ is a trip management system in which users can search and book trips with different payment options. The tactics for A_1 and A_2 are oriented to tackle modifiability, while the tactics for A_3 and A_{14} are geared towards improving performance. In A_1, a tactic divides the responsibilities of *PaymentSystem* into two interacting components. In A_2, another tactic splits the *Exporter* component into an interface (*IExporterWrapper*) and its implementation (*Exporter*) to enable different exportation mechanisms. In A_3, a tactic allocates the *Exporter* and *BusinessTripMgmt* to the same sever as the *TripDB* to improve transactions to the database. In A_14, components *BookingSystem* and *TripDB* and replaced by faster implementations as indicated by*QuickBooking* and *FastWriteTripDB*, respectively.

In *SQuAT* [41], we seek to return a set of architecture alternatives having different tradeoffs with respect to the quality-attribute objectives. To this end, the search iteratively expands a tree rooted at the initial architecture employing tactics. For each newly-generated alternative, the solvers compute their quality-attribute metrics. The tree grows both in breadth and depth as specified by configuration parameters. It should be noted that the tree expansion is partial, as our aim is not necessarily to perform a full, combinatorial search in the design space. Furthermore, the search, as currently implemented by *SQuAT*, is driven by tactics rather than by the evaluation results returned by the solvers.

The focus of this work is on the *transformation and evaluation phases*, which are responsible for estimating the quality-attribute properties of the candidate architectures. Estimating such properties using analysis models is a widespread practice in the software architecture domain [15]. For instance, performance and reliability estimations can be achieved using UML models annotated according to the MARTE [26] and DAM [10] profiles, respectively. A transformation is generally necessary to enable a quantitative analysis. For example, LQNs [23] are a typical formalism for producing performance estimations, while Petri Nets [37] and Markov Chains [39] can be employed for reliability estimations.

2.1 Problem formulation

We consider multi-objective architecture optimization in the context of the PAD (Predictable Architecture Design) framework [5], and assume an architectural space for a family of systems that encompasses all possible architectural configurations in terms of a (finite) set of design decisions. Let $AS = \{A_0, A_1, A_2, ..., A_n\}$ be an *architectural space* with n architectural configurations in which each A_i corresponds to a (valid) configuration that results from a sequence of predefined decisions $A_i =< d_{1i}, d_{2i}, ..., d_{mi} >$. Each $d_{ji} = 1$ if the decision was made for configuration A_i or 0 otherwise.

In this work, we restrict possible decisions d_{ji} to architectural tactics [6]. As mentioned above, a tactic is a design operation or refactoring that changes parts of an architectural configuration to improve one or more quality attributes. Thus, the configurations in AS are linked to each other through the tactics applied. AS can be seen as a directed acyclic graph in which each node represents a configuration, while an edge between two nodes (A_i, A_l) captures a tactic leading from A_i to A_l. A distinctive node A_0 refers to the initial configuration.

We require a configuration A_i to be assessed with respect to multiple quality attributes (objectives) by means of quantitative evaluations (e.g., model-based predictions). Let $QAS =< O_1, O_2, ..., O_k >$ be a quality-attribute space with k objectives in which each O_k represents a quality metric (e.g., latency, failure probability or cost) for some architectural configuration. That is, an evaluation function $E : AS \rightarrow QAS$ maps a configuration to a multi-valued vector in $\mathfrak{R}^k$. For simplicity, we assume that E also includes the transformation phase and the mapping of configurations A_i to their counterparts in the analysis space. Specifically, E is the function that we can replace with a surrogate model when computationally costly.

3 APPROACH

The optimization cycle can be seen as a relation between alternative architecture configurations and a multi-valued response surface, which assigns numeric scores to each configuration with respect to the quality-attribute objectives. From a data-driven perspective, the execution of the optimization generates a dataset comprising both architecture configurations and vectors of quality-attribute values, which becomes a key asset to enable surrogate modeling, as hinted in Fig. 1. The different parts of our surrogate-based approach are described below.

3.1 Surrogate modeling

In general, given a primary, expensive model M that computes some outputs, the idea behind a surrogate is to construct a cheaper model $\widehat{M}$ as a curve fit to the available data generated by M, so that outputs can be predicted using $\widehat{M}$ instead of M. This strategy is based on the assumption that, once computed, $\widehat{M}$ will be faster than M and maintain reasonable fidelity (or accuracy) when making predictions for unseen data. In our architecture optimization problem, the evaluation function E that encompasses the solvers is M, and thus we aim at constructing $\widehat{E}$ via regression techniques [3, 32]. Regression is a type of supervised learning in which the goal is to predict a continuous value (output) based on input data. Since we deal with multiple objectives, either single- or multi-output regression can be applied. The latter is useful if the objectives are correlated [11].

3.2 Feature representation

In ML terminology, a feature is an individual property or attribute that characterizes instances of a dataset. Since architecture configurations are often complex, multi-dimensional objects, we need to create a tabular representation for them to be used in a regression model. A common representation is to use numeric features; thus, configurations

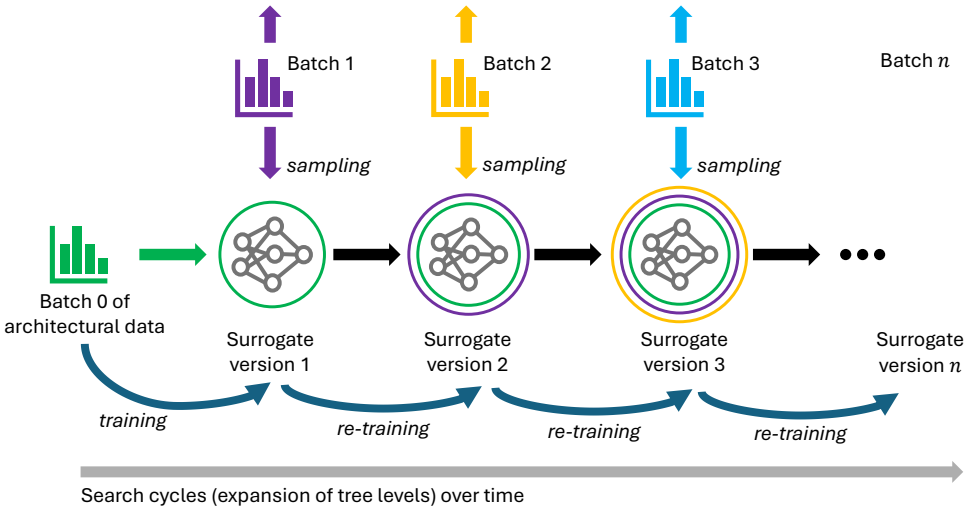


Fig. 4. Incremental learning of a surrogate based on sampling from batches. Each batch is the result of a new level in the search tree.

become vectors in a dataset. We identify two sources of information for creating these architecture vectors: i) the tactics for deriving architecture alternatives, and ii) the structure of the architectures themselves.

First, the optimization engine produces alternatives by applying sequences of tactics starting from the initial architecture. Thus, an architecture instance A_i is represented by a *sequence of tactics* $S_i =< T_{1i}, T_{2i}, ..., T_{mi} >$, which comes from the shortest path between the root node and a particular node in the search tree (see Fig. 2). For regression, each tactic, defined by its name and parameters, is encoded into a numeric format using a one-hot encoding schema [24].

Second, since architectures can be naturally modeled as graphs (e.g., using a component-and-connector view), we compute *graph embeddings* [52] to capture the structural information of architecture instances. Embeddings map entities, edges, and subgraphs into compact, lower-dimensional numeric vectors while preserving topology.

3.3 Incremental and active learning for regression

When building a surrogate model, dealing with a multi-dimensional space (like the quality-attribute space) is challenging due to the number of instances that need to be covered to deliver the predictions. Furthermore, in an architecture search process that proceeds cycles, like SQuAT, not all instances are available at once, but rather they are generated in batches. Each batch corresponds to expanding the search tree one level. A way around this issue is to construct and update a series of surrogate versions as the search generates batches and new architectural information is acquired. Along this line, our ML strategy combines *incremental and active learning* to adapt the surrogate to the nature of the architecture optimization process, as outlined in Fig. 4. In this strategy, called IAL, one or more regressors are re-trained with samples evaluated with the solvers in every search cycle (tree level).

Incremental learning permits using (new) input data to further train the (current) model. In this way, we can adjust our regressors with new architectural instances (e.g., when exploring a new tree level) without forgetting patterns extracted from previous instances. Active learning, in turn, is a type of semi-supervised ML [24] in which the model is trained using both labeled and unlabeled data. In our case, labeled data correspond to architecture instances whose quality-attribute evaluations are determined by calling the solvers (i.e., the oracle). Active learning samples training

(labeled) instances based on their likelihood to improve the model performance. With every search cycle, the model is adjusted based on the cumulative labeled data and its accuracy is expected to increase.

The IAL strategy is described in Algorithm 1. It takes an initial architecture X_0 along with the quality-attribute solvers E as the main inputs. Additional parameters are the maximum depth T for the search tree and a sampling factor $1 - 100\%$ for the active learning. The architectural instances are assumed to be generated in incremental batches $X_1, X_2, ..., X_T$ by the SQuAT engine. The evaluation of the architectures in the incremental batch X_t, either returned by the solvers or (predicted by) the regression models, is captured by $\widetilde{Y_t}$. The algorithm begins by asking SQuAT to search for the first level of architecture alternatives and then building a first version of the surrogate on the first batch (lines $1 - 3$). The surrogate can be configured to work with either a collection of individual regressors (one per objective) or a multi-output regressor handling all objectives as a whole.

Algorithm 1 Incremental Active Learning (IAL) strategy

Inputs: X_0 : initial architecture
E : transformation and evaluation function (solvers)
T : maximum number of search levels (batches)
SF : sampling factor
Outputs: $\widehat{E}$: surrogate for E
$< \widetilde{Y_1}, \widetilde{Y_2}, ..., \widetilde{Y_T} >$: objective values for batches $< X_1, X_2, ..., X_T >$

{Process the first batch}
1: $X_1 \leftarrow searchAlternatives(X_0)$ {Generate initial batch (SQuAT - level *1*)}
2: $\widetilde{Y_1} \leftarrow E(X_1)$ {Invoke solvers on initial batch}
3: $\widehat{E} \leftarrow buildSurrogate(X_1, \widetilde{Y_1})$ {Train initial surrogate}

{Iterate over the remaining batches}
4: **foreach** $t \in [2..T]$ **do**
5: $X_t \leftarrow searchAlternatives(X_{t-1})$ {Generate t-th batch (SQuAT - level t)}
6: $X_t^S \leftarrow sample(X_t, \widetilde{Y_t}, SF)$ {Select SF% of random instances}
7: $\widetilde{Y_t^S} \leftarrow E(X_t^S)$ {Invoke solvers on sample}
8: $\widehat{E} \leftarrow adjustSurrogate(\widehat{E}, X_t^S, \widetilde{Y_t^S})$ {Re-train surrogate}
9: $\widetilde{Y_t} \leftarrow \widehat{E}(X_t - X_t^S) \cup \widetilde{Y_t^S}$ {Predict with new surrogate}
10: **end for**

11: **return** $\widehat{E}, < \widetilde{Y_1}, \widetilde{Y_2}, ..., \widetilde{Y_T} >$ {Note that some $\widetilde{Y}$ values are predicted while others are computed by the solvers}

For each of the following search cycles (i.e., batches), IAL generates a new tree level with architectural alternatives (batch Xi), and proceeds to sample a fraction of Xi to be evaluated by the solvers (lines $5 - 6$). The SF parameter controls the number of instances for re-training the surrogate, which also affects accuracy and computational costs. The closer SF to 1, the higher the number of solver calls. Ideally, SF should take a small batch fraction for the approach to be cost-effective. We used random sampling for our experiments. Other heuristics such as the *jacknife* method [20] are possible, although they tend to be computationally expensive, defeating the purpose of our surrogates with respect to the solvers.

Based on the sample, the current surrogate is re-trained by incorporating the new batch of data into the existing model (line 8). In this way, the surrogate evolves over time without discarding previously learned estimators. The

updated version of the surrogate is used to evaluate the architectural instances in the batch (line 9). Once the predefined stopping criterion (e.g., maximum number of iterations) is reached, both the final surrogate and the quality-attribute values computed across all batches are returned.

4 EXPERIMENTAL DESIGN

To assess the cost-accuracy tradeoff of our surrogate modeling approach, we consider two case studies of multi-objective architecture optimization from the literature and perform comparative simulations using regression models to make predictions of the quality-attribute values. We also compared single- and multi- output regression variants. These simulations took the form of controlled experiments based on datasets, in which all architectures were evaluated beforehand (i.e., in a supervised mode) with quality-attribute solvers. Since the generation of architecture alternatives in `SQuAT` is decoupled from their quality-attribute evaluations, the choices about using solvers or surrogates to compute the objective values affect only the results in the quality-attribute space, but it does not affect the search process nor the generation of architectural alternatives.

The research questions are the following:

- **RQ#1**: Is it better to use single- or multi- output regression for the surrogate model? What is the effect of random sampling?
- **RQ#2**: How is the performance of the surrogate model affected by the sampling factor?
- **RQ#3**: What is the computational cost of using the surrogate models (with a given sampling factor) in combination with the solvers?

The processing of the case studies for the construction of the surrogate models was implemented with an ML pipeline. The case studies were treated as input datasets for the pipeline, which involved pre-processing for obtaining the features and model building for the regressors. We used gradient boosting as the regression technique, since it was the best-performing model in our initial experiments [47]. Furthermore, we opted for `CatBoost` [40] over `XGBoost` as the regression model, given its excellent performance, built-in treatment of categorical features, support for incremental learning and low training times[9]. The experiments were carried out with *Jupyter* notebooks and *Python* scripts, which are available for reproducibility[1].

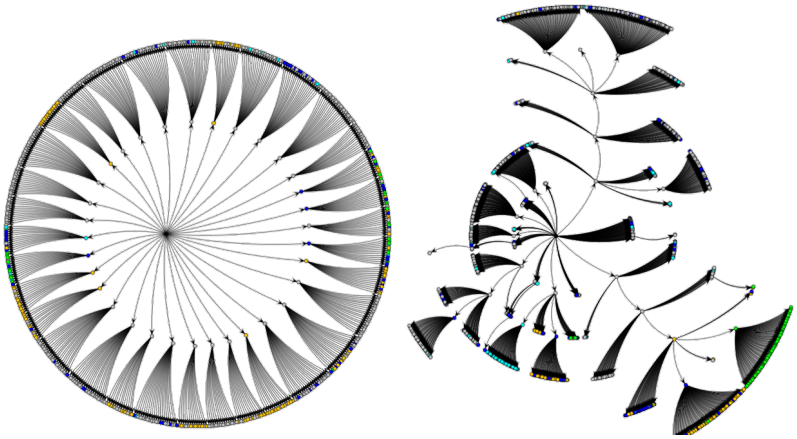
4.1 Case studies

We based our study on two datasets, *ST+* [41] and *CoCoME* [27], both provided by the `SQuAT-Vis` toolkit [22]. Table 1 summarizes their main characteristics. The case studies differ in both the size and structure of their search spaces. The *ST+* search tree is balanced and contains 454 architecture alternatives, while the `CoCoME` search tree has an irregular shape with 1192 alternatives. The irregularity of `CoCoME` is due to the fact that not all tactics apply to every architecture instance. Figure 5 illustrates the search trees resulting from running `SQuAT` on `ST+` (a) and `CoCoME` (b) with two search and five cycles, respectively. The initial architecture is located in the center (root), and each branch is a sequence of applied tactics. In these trees, the root corresponds to the initial architecture, each branch represents a sequence of tactics, and each node denotes an architecture alternative reachable via that sequence. Each architecture instance is linked to a Palladio (PCM) [7] model in Eclipse-EMF format, which captures structural and dynamic aspects of the architecture.

[1]Github site: https://github.com/andresdp/architecture-surrogates

Table 1. Overview of the case study datasets

Name	Sequence Length	Objectives	Components	Alternatives
CoCoME	5	8	55	1192
ST+	2	4	9	454



(a) ST+ (2 tree levels) (b) CoCoME (5 tree levels)

Fig. 5. Graphical representations of the search trees (architecture spaces) for the two case studies [19] .

The case studies also differ in domain complexity. ST+ models a relatively simple trip management system, while CoCoME represents a more complex commercial trading system. Their optimization objectives address both *modifiability* and *performance*. For modifiability, the goal is to keep the effort of adding new features low. In ST+, the scenarios involve: (*m1*) adding a new payment option and (*m2*) adding a security token. In CoCoME, the scenarios are to (*m1*) add a new payment option, (*m2*) add a cash desk for premium customers, (*m3*) add a service to withdraw money, and (*m4*) add a logging service. For performance, the goal is to maintain low response times under specific conditions. In ST+, these conditions include (*p1*) a 10% increase in usage and (*p2*) a node failure. In CoCoME, the scenarios are defined as (*p1*) a 10% increase in usage, (*p2*) a heavy usage increase of 50%, (*p3*) a node failure, and (*p4*) a shift in customer behavior, where customers purchase more items per transaction on average.

In both datasets, various tactics have been applied. For improving modifiability, the SQuAT engine can i) split components and ii) add a wrapper; while for improving performance, SQuAT can i) select component alternatives, ii) re-deploy components, iii) scale resources, iv) move distributed components, and v) change resource capacity. These tactics were taken from the literature. During the search process, the tactics were systematically applied to the architectural configurations until a predetermined tree depth was reached.

Figure 6 illustrate the distributions of objective values across the different levels of analysis for both datasets. Among the modifiability objectives for CoCoME, *m1* and *m2* remain tightly concentrated near zero with only a few extreme negative outliers, suggesting that most architectural alternatives remain relatively easy to extend, but certain tactic applications can severely increase modification costs. *m3* introduces broader variability, offering a finer gradient of modifiability outcomes that can guide optimization decisions more effectively. In contrast, *m4* is highly skewed, with

many alternatives performing poorly. On the performance side, the objectives show greater dispersion and more extreme tail behaviors. Both *p1* and *p3* exhibit wide ranges with multiple outliers, indicating high sensitivity of response times. These objectives are the most discriminative for optimization, as they clearly separate good and poor-performing architectures. *p2* is generally stable around zero but occasionally spikes to very high values. Similarly, *p4* is concentrated near zero yet produces a few extreme cases.

ST+ shows a simpler and more compact distribution of objectives compared to CoCoME, but still exhibits meaningful contrasts between modifiability and performance. For modifiability, both objectives (*m1* and *m2*) exhibit moderate spread centered around zero. Their interquartile ranges span from slightly negative to slightly positive values, indicating that most architectural alternatives involve low to moderate costs of change. Extreme negative outliers are present, but they are less pronounced than in *CoCoME*, suggesting that ST+ offers a more balanced trade-off space for modifiability. The performance objectives show stronger variability. *p1* has its central mass tightly clustered around zero, but level 1 introduces several strong positive outliers, showing that even in the earliest iterations some architectures already experience significant performance degradation. As generation proceeds to level 2, the distribution widens and positive values become more common, reflecting that deeper transformations expose additional bottlenecks. In contrast, *p2* remains highly skewed: most configurations cluster close to zero, but sporadic large spikes appear, indicating rare but severe vulnerabilities to failures.

These differences show that ST+ provides a controlled benchmark with clearer decision boundaries, while CoCoME captures a more complex search space with overlapping and interdependent objective behaviors, thereby being a more demanding evaluation setting for surrogate models. Furthermore, the objective distribution can influence the performance of single- or multi-output regressors.

4.2 Pre-processing

The two datasets encoded each architecture instance as a set of features describing the applied refactoring actions, along with the corresponding performance and modifiability metrics (quality-attribute objectives) used as regression targets. In ST+, the representation comprises two modifiability objectives (*m1* and *m2*), two performance objectives (*p1* and *p2*), and a sequence of up to two applied tactics (*op1* and *op2*). The CoCoME dataset includes four modifiability targets (*m1* to *m4*), four performance targets (*p1* to *p4*), and sequences of up to five tactics (*op1* to *op5*). The tactic sequences were taken from [22] and encoded using a one-hot scheme [24]. Finally, since the targets varied in scale, we applied Z-standardization to normalize values.

To complement the tactic-based encoding, we also computed graph embeddings that capture the structural information of the architecture instances (derived from the PCM models). These embeddings can be appended to the tactic features or used as a standalone representation. In particular, we adopted the *Feather-N* approach [43], which efficiently summarized vertex characteristics at multiple scales by combining vertex statistics with random walks through the so-called characteristic functions. Transition probabilities from these random walks define the function weights. For a set of architecture graphs, node embeddings are aggregated into fixed-length vectors, one vector per architecture, using a pooling mechanism. This approach embeds both the structure and the attributes of vertices while remaining computationally efficient. Compared to alternatives such as `graph2vec` [38], *Feather-N* scales better to larger graphs and captures richer structural information via its multi-scale statistics, making it especially suitable for representing software architectures that vary widely in size and complexity. In practice, it required less than a minute to compute embeddings for our datasets and provided competitive accuracy in the regression tasks. Once trained, the embeddings

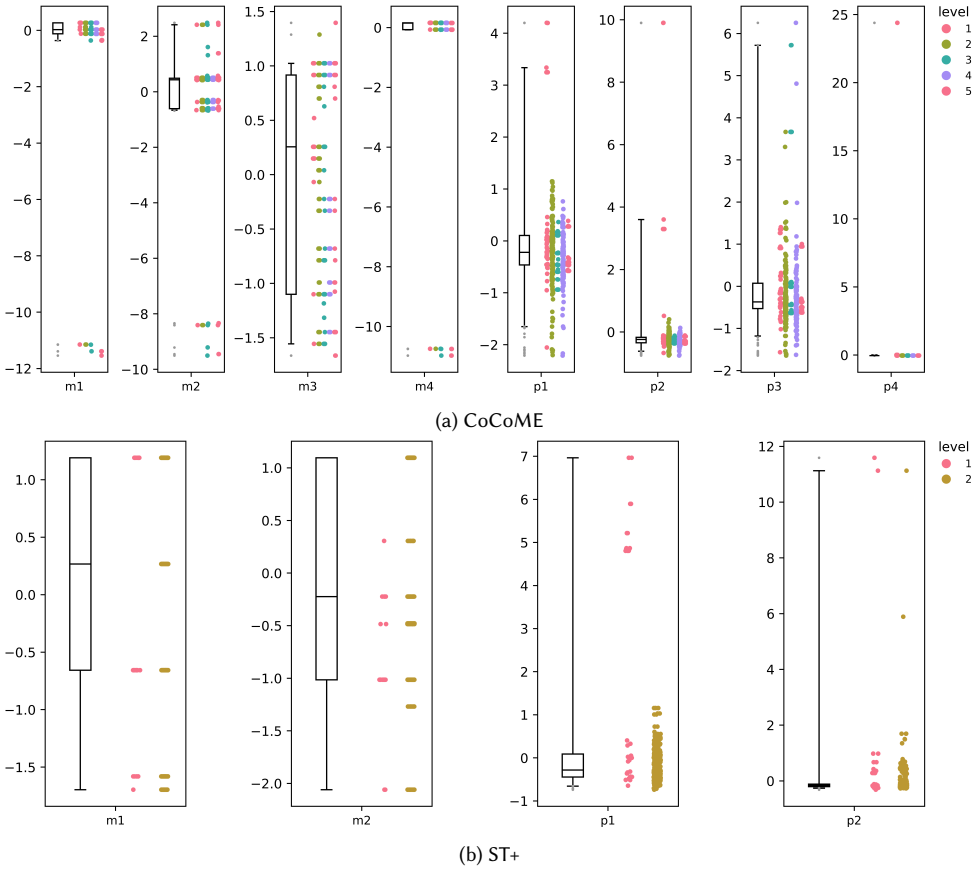


Fig. 6. Distribution of (standardized) objective values grouped by tree level during the search for two systems.

can also be applied to unseen architecture instances. Further implementation details are available in [46]. Details of how the representations were computed are available here [46].

Because the original PCM format is highly detailed and not directly suitable for graph embedding methods, the models were first transformed into a simplified representation provided by *SQuAT-Vis* [22]. This representation preserves only elements relevant to runtime structure (components, connectors, and hardware) organized in a graph-like architectural view. Hardware is represented as resource containers (e.g., servers with CPUs), software elements as components with usage relations, and allocation as explicit links from components to containers (see Fig. 3 for examples).

4.3 Model selection and building

We evaluated regression algorithms from the gradient boosting family, such as `XGBoost` and `CatBoost`, given their strong performance in handling non-linear relationships, robustness to outliers, and support for incremental model training. The regression task [11] was defined to predict all quality-attribute objectives either simultaneously (multi-output) or individually (single-output), using the feature representations of the architecture instances. Based on the comparative results from our earlier work [47], we selected `CatBoost` as the main algorithm for this study. To maintain

the focus on representation and regression strategy rather than on parameter tuning, we did not apply feature selection or hyper-parameter optimization.

The ST+ and CoCoME datasets were split into training and test sets using a 70/30% partitioning scheme. Prediction accuracy was assessed with the Root Mean Square Error (RMSE), which is expressed in the same units as the quality-attribute responses. Since all targets were Z-standardized (see Fig. 6), the responses are centered on zero, with each unit (± 1) corresponding to one standard deviation from the mean.

As a starting point, we established a *naïve baseline* using standard regression. Training and test instances were selected using stratification by tree level (2 levels for ST+ and 5 levels for CoCoME), and split according to a 70/30% partitioning schema. This procedure approximates the natural order in which architecture instances are generated during the optimization search. The *RMSE* values of the baseline were 0.15 and 0.03 for CoCoME and ST+, respetively. These per-level values are averaged for the whole set of regression targets.

The IAL strategy reflects a more realistic setting in which architecture instances are generated incrementally during the iterative search process. As new batches become available, the regression model is updated and evaluated at each batch (or tree level). Unlike the naïve baseline, IAL applies a sampling factor (*SF*) to the training data: at each label, only a fraction of the 70% training split is used for (re)training the surrogate. By default, $SF = 0.5$, meaning the 35% of the total data is effectively used at each iteration. The test set is not affected by this sampling. Using $SF < 1$ simulates realistic constraints in which only part of the available instances can be exploited at each iteration due to time or evaluation costs. While incremental training can lead to overfitting, this risk is not critical in our context, since the surrogate is only used to guide the optimization during its execution and is discarded once the process ends.

For the *sample()* function of IAL, training instances were selected randomly at each iteration according to the sampling factor *SF*. Given that this choice can lead to different subsets of the training set (per level), and therefore can influence the performance of the regression model, we designed a Monte Carlo procedure to quantify the resulting variability. Specifically, the IAL strategy (with a predefined *SF*) is run 100 times with different random seeds, and the average and standard deviation of the prediction errors (*RMSE*) were reported.

5 RESULTS

5.1 RQ#1: Comparison of regression algorithms

Figs. 7 and 8 show the evolution of prediction errors for CoCoME and ST+ respectively, as SQuAT moves through the levels of the search tree. The plots report the average *RMSE* for both the single- and multi- output variants, together with their variance, as estimated through the Monte Carlo procedure. Training times were low in all cases, with $\approx$ 3.5 seconds and $\approx$ 1.5 seconds for the multi-output regression for CoCoME and ST+. For the single-output regression (summing across all objectives), training took $\approx$ 2.5 seconds (CoCoME) and $\approx$ 1 seconds (ST+). Regression was slightly faster for the single-output variant because the multi-output variant optimizes a shared model across all objectives, probably increasing the complexity of the training process compared to fitting independent models. Nonetheless, training times for both surrogate variants are negligible relative to the cost of invoking the solvers.

In CoCoME, the prediction errors for the modifiability objectives started around 10^{-1} and increased to 1 (one std. dev.) across the tree levels for both variants, indicating that the predictions became less accurate over time. However, the prediction variance was wide for certain objectives (e.g., *m1* and *m4*), particularly in the multi-output case, though occasionally low-error cases (as small as 10^{-3}) were observed. For performance objectives, the difference between the two variants was more pronounced: the single-output regression showed a consistent improvement over time

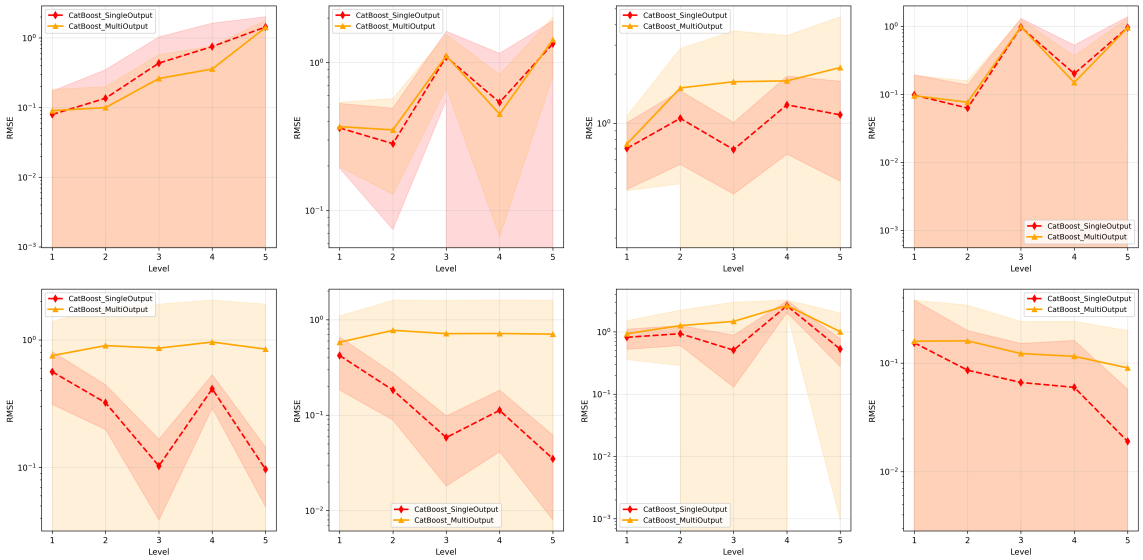


Fig. 7. CoCoME performance results (RMSE) per tree level, using the IAL strategy with single- and multi- output regression with $SF = 0.5$. Objectives in the first row: *m1*, *m2*, *m3* and *m4*. Objectives in the second row: *p1*, *p2*, *p3*, *p4*.

(down to 10^{-2} for *p1* and *p2*) with low variance, whereas the multi-output regression did not improve and showed considerable variability. Aside from *m4* and *p4*, where both approaches showed high variability, the single-output variant outperformed the multi-output one. We conjecture that the number of instances available per level, as characterized in Fig. 5b, may have influenced these results.

In contrast, ST+, had only two levels in the search tree and the average RMSE exhibited a more uniform improvement trend. Errors dropped below 1 (one std. dev.) for *p1* and *p2* and range between 10^{-1} and 1 for *m1* and *m2*. Variance caused by random sampling was bounded and consistent across both regression variants. Unlike the CoCoME case, here the multi-output regression performed better than the single-output regression, likely due to the smaller search space and lower number of objectives, which favored having a shared predictive model.

Complementary data analyses shed light on these findings. Correlation studies showed that while some objectives (e.g., *m2*–*m3* and performance pairs) became strongly related in specific iterations, others remained weakly correlated, suggesting that multi-output models can benefit only when objectives evolve in tandem. Feature–objective correlations further indicated that most features are predictive of a single objective, limiting opportunities for shared learning. Finally, distributional analysis highlighted that some objectives (e.g., *p4* for CoCoME) behaved differently from the rest, with higher skewness and variance, which likely introduced noise into joint modeling. These factors help explain why the single- versus multi-output performance varied across case studies and iterations.

RQ#1. Prediction errors varied substantially and appeared to depend on the characteristics of the objectives (e.g., their distributions), the number and correlation of objectives, and the amount of data available per level. Overall, single-output regression emerged as a more reliable choice when dealing with a larger number of objectives. In addition, the accuracy of predictions proved sensitive to random sampling effects and the depth of the search process, leading to wider variance in some cases.

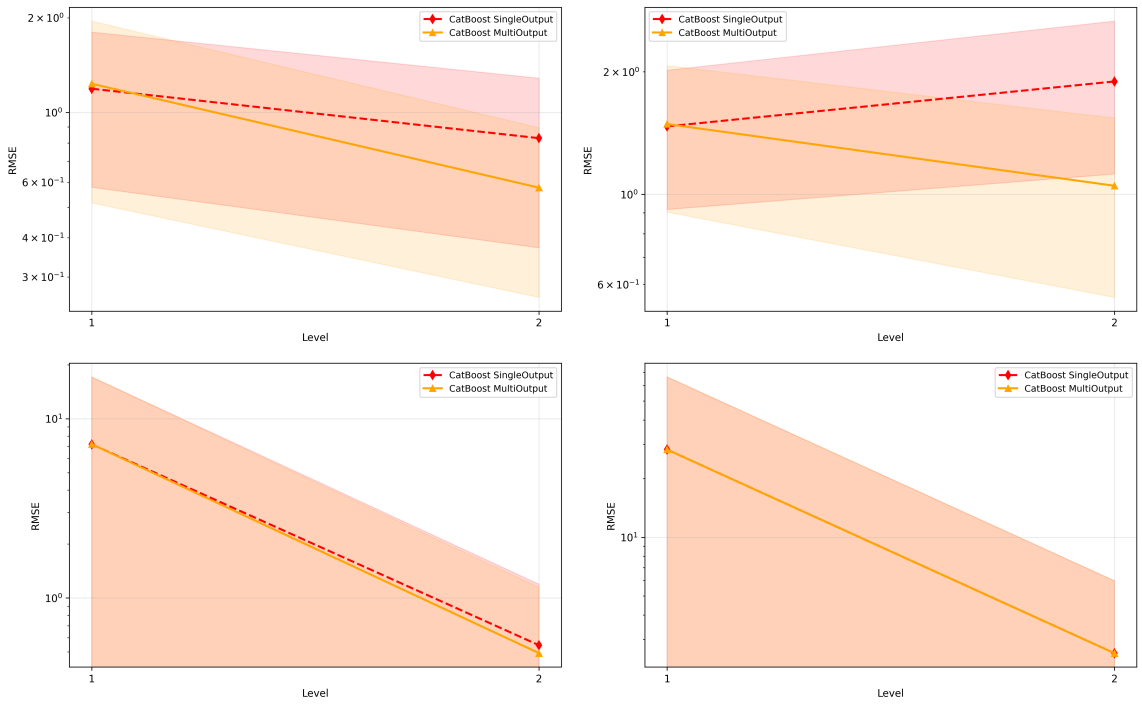


Fig. 8. ST+ performance results (RMSE) per tree level, using the IAL strategy with single- and multi- output regression with $SF = 0.5$. Objectives in the first row: *m1*, *m2*, *m3* and *m4*. Objectives in the second row: *p1*, *p2*, *p3*, *p4*.

5.2 RQ#2: Effect of sampling factor on prediction accuracy

Figs. 9 and 10 present the evolution of *RMSE* as the sampling factor (*SF*) increases, with dashed lines marking the metrics for the naïve baseline. As it can be observed, the sampling factor plays a key role in balancing computational cost against predictive performance. A higher sampling factor means more invocations to the solvers, which should increase the model accuracy (since more "real' values are used to re-train the model) but incur higher computation costs. On the contrary, a lower sampling factor should save costs, but it might cause underfitting and lower model accuracy.

In ST+, prediction errors converge quickly, accuracy seems to stabilize at 0.4, once *SF* reaches approximately 30%, with little improvement beyond that point. By contrast, CoCoME shows a near-linear gain in accuracy as *SF* grows, with the best results at 0.8 ($SF \approx 55\%$) for levels 3 and 4. At levels 2 and 5, accuracy improves up to 0.7 ($SF \approx 50\%$) but degrades slightly afterwards.

These contrasting behaviors can be attributed to the characteristics of the design spaces of the case studies. The CoCoME surrogate benefited from four re-training iterations, enabling progressive refinement, while ST+ had only one. In addition, the irregular shape of the CoCoME search tree likely made predictions more sensitive to sampling compared to the more balanced ST+ tree. We should note that the architecture instances in CoCoME are more complex structurally than the ST+ counterparts.

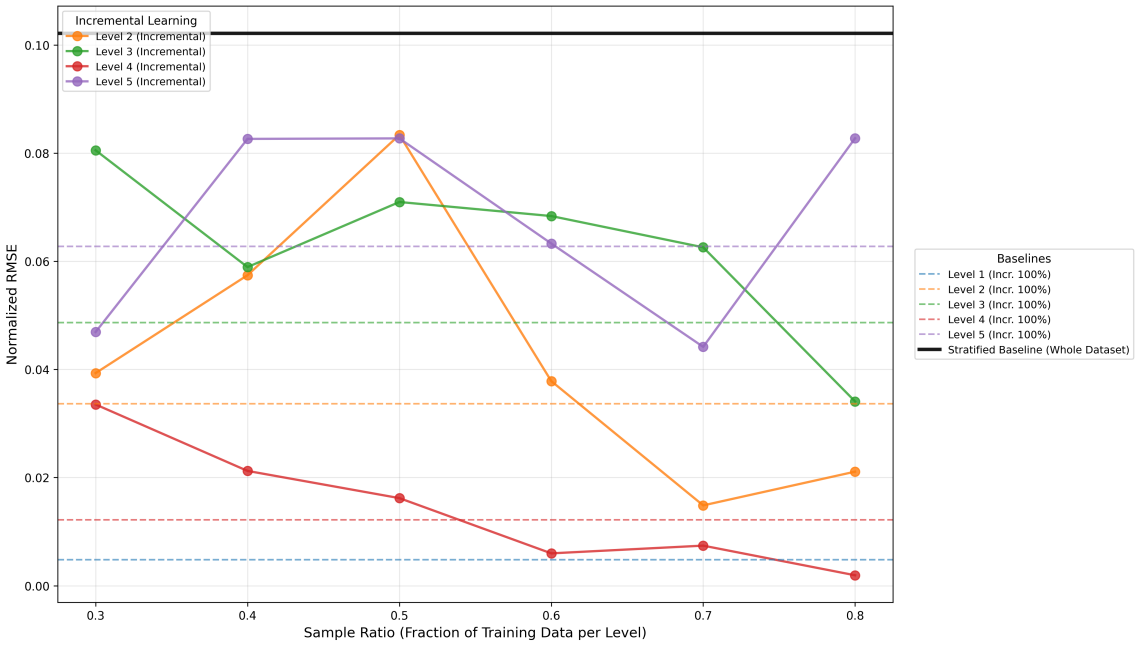


Fig. 9. Evolution of average prediction errors for IAL (single-output regression) per level and sampling factor for CoCoME. Note that $SF = 0.7 * sample\ ratio$. Thick horizontal **black** line corresponds to the (aggregated) error of the baseline, while dotted horizontal lines show the baseline errors per level.

RQ#2. *RMSE* results confirm that it is feasible to construct surrogates and progressively update them during the optimization process. In general, higher sampling factors lead to more accurate models, although this trend does not hold consistently across all objectives. Overall, the findings indicate that surrogate-based optimization can predict reasonable quality-attribute values and its effectiveness depends on the characteristics of the system under study and the specific quality attributes being optimized.

5.3 RQ#3: Tradeoff between computational cost and accuracy

To analyze the accuracy-cost tradeoffs, we related the prediction errors (Figs. 9 and 10) to the total execution time spent by the optimization process[2]. In extreme cases where no surrogate is used, the total execution time can be approximated as the average time for processing an architecture instance with every solver times the number of instances in the dataset. Solver calls typically required 30 seconds for ST+ and up to 1 minute for CoCoME. When using a surrogate, the time consumed by the solvers is modulated by the sampling factor. Since solver invocations dominate the runtime of SQuAT, the execution time grows linearly with the sampling factor. In ST+, the total runtime ranged from 0.3 to 2.5 hours, while in CoCoME it ranged from 2 to 20 hours. For example, using a surrogate in ST+ with $SF = 30\%$ reduced the total runtime to about 1.15 hours. Similarly, applying a surrogate in CoCoME with $SF = 50\%$ required about 10 hours. These are both interesting gains from a cost perspective.

[2]All measurements were taken on commodity hardware: Apple M1 chip with 8GB RAM and 256 GB of disk storage.

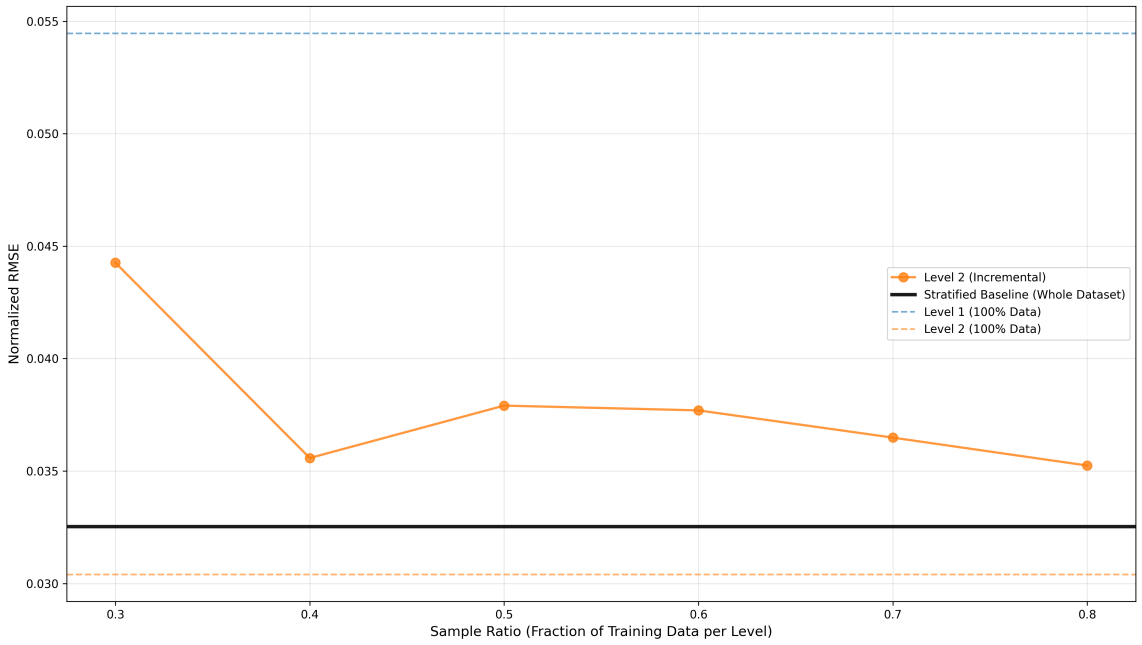


Fig. 10. Evolution of average prediction errors for IAL (single-output regression) per level and sampling factor for ST+. Note that $SF = 0.7 * sample\ ratio$. Thick horizontal **black** line corresponds to the (aggregated) error of the baseline, while dotted horizontal lines show the baseline errors per level.

As mentioned in Section 5.1, processing and training times with `CatBoost` were consistently small for both the naïve and *IAL* strategies. Feature encoding and embedding generation required about 1 minute on average, while training or re-training the models took less than 5 seconds. Although the re-training effort can scale with the sampling factor, it is still very low compared to the time spent by the solvers.

> **RQ#3.** Training the surrogate models incrementally on the case study datasets did not significantly increase computational cost. The sampling factor, however, directly governs the tradeoff between execution time and prediction accuracy. Based on our experiments, sampling factors between 40% and 50% achieved a good balance, reducing total computation time by roughly a half while keeping prediction errors at acceptable levels.

6 THREATS TO VALIDITY

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

Internal validity. Having non-determinism in some algorithms of the ML pipeline is unavoidable. To mitigate this threat, we fixed a random seed that controls dataset shuffling and sample assignment to training and testing subsets. Still, different seeds may lead to different results and affect the conclusions. Moreover, the controlled environment of our solvers and the pre-generated datasets differ from real-world optimization scenarios, particularly regarding the naïve baseline representations. While this allowed us to reduce noise and isolate the effects of surrogate modeling, it might also have biased our regression techniques. Finally, our IAL strategy poses a challenge: a biased sampling in early iterations may lead the surrogate to focus on certain regions of the design space, potentially overlooking others.

Construct validity. Although we relied on widely used error-based metrics(e.g., *RMSE*) [18, 30] for evaluating our approach, the choice of metrics may have influenced the conclusions. Furthermore, while our regression models exploited architectural structure and refactoring tactics, the embeddings were generated from simplified architecture models that disregard parts of the PCM specification (e.g., parameters). We also did not apply feature correlation or feature selection analyses, which may have improved surrogate performance. We measured only the execution times for the solvers, which may underestimate the total overhead in practice. At last, while surrogates approximate solver behavior, solvers themselves are abstractions of reality. This introduces a "double approximation" threat, even perfect surrogate predictions inherit the inaccuracies of the underlying analysis models.

External validity. The generalizability of our results is limited by the datasets and case studies used. Although *ST+* and *CoCoME* are established benchmarks with different system sizes, both are academic systems. Larger or industrial systems, involving different architecture styles and more complex quality-attribute tradeoffs, may yield different results. The datasets are also relatively small by ML standards, so the reported accuracy–cost tradeoffs might change with larger datasets. Our evaluation was limited to `CatBoost` due to their robustness and incremental training support. Other algorithms (e.g., Gaussian Processes, CART, or neural approaches) and alternative sampling criteria might provide better tradeoffs. Differences in hardware and software environments may also affect runtime results. Our feature representations has not been tested with very large architectures, where embeddings may become prohibitively large or sparse to be used in our regression models.

Conclusion validity. Our conclusions may be affected by the choice of hyperparameters for both surrogate models and embeddings. We used default settings for the `CatBoost` and *Feather-N* algorithms. Alternative configurations could have influenced both predictive accuracy and runtime cost. Furthermore, since our evaluation was based on pre-generated datasets, the prediction errors did not influence the search behavior of `SQuAT`, but this could not hold for other optimization approaches (e.g., genetic algorithms), in which the objective values of a given search cycle drive the exploration of alternatives in the next cycle. Thus, in a "live" optimization loop, however, surrogates could steer the search into different regions of the design space, resulting in different architectural alternatives being explored.Surrogate inaccuracies can also affect how architects use and trust the approach in practice, which we did not evaluate in this study.

7 RELATED WORK

Surrogate models [30], which approximate a real-world scenario or a complex behavior using a simplified model, are a promising method for reducing the time and effort to evaluate architectural solutions. Surrogates have already been used in a wide variety of research areas and applications, ranging from traditional engineering disciplines such as aerodynamics [34] and ship design [45], over natural gas liquefaction [21] to complex environmental optimization problems [48]. For instance, surrogate-assisted evolutionary algorithms, like NSGA-II, typically exploit regression-based models to predict objective values of candidate solutions, or classification-based models to decide whether a candidate solution should survive into the next generation. These techniques seek to reduce the number of evaluations required to approximate Pareto fronts in multi-objective optimization problems.

Surrogate-assisted optimization has also gained traction in adjacent domains such as neural architecture search (NAS) and hardware design. Recent approaches combine multiple surrogates to improve sample efficiency and training stability, as in NSGANetV2 [35], while others introduce Pareto rank-preserving surrogates to address multi-objective hardware-aware NAS [8]. Beyond neural networks, surrogate models have been used for co-optimizing neural architectures and

the physical implementation of hardware accelerators, targeting both accuracy and energy efficiency [49]. These results highlight the broader applicability of surrogate modeling for architecture exploration.

In the systems engineering community, surrogate-based optimization has been extensively explored in the context of system architecture optimization (SAO). In this setting, the design of complex systems (e.g., jet engines or satellites) is treated as an optimization problem where candidate architectures are encoded as design vectors with hierarchical, mixed-discrete variables, and evaluated using expensive physics-based or simulation-based models. Strategies for integrating hierarchical information into optimization have been proposed, for example, through Bayesian optimization with specialized Gaussian process kernels and sampling schemes [13]. Bussemaker et al. [14] studied surrogate-based optimization in the presence of hidden constraints, such as solver non-convergence or infeasible geometries, showing that explicitly modeling the probability of viable solutions can improve performance. Empirical studies comparing surrogate-based algorithms with evolutionary approaches show that surrogates can approximate Pareto fronts with significantly fewer function evaluations, although their effectiveness varies depending on the structure of the design space [12]. In contrast, in software architecture optimization, the design space consists of alternative architectural models generated through refactorings and tactics, and evaluations rely on quality-attribute solvers. While such solvers are less costly than physics-based simulations, they still pose efficiency challenges when repeatedly invoked during large-scale multi-objective optimization.

In software engineering, surrogate and other ML approximations have mostly been applied in configuration and code optimization tasks. For configurable systems, regression has been used to predict performance from configuration options, sometimes incorporating additional factors such as workload inputs [33] or causal dependencies across software and hardware layers [28]. Guided sampling and transfer-learning strategies have also been developed to reduce the cost of data collection in performance modeling [29]. These techniques exploit prior knowledge from a similar environment by extracting source characteristics that likely remain stable across environments by progressively shrinking and adaptively concentrating on interesting regions to then take informative samples to inform the sampling of the target. The approach is shown to outperform state-of-the-art performance learning and transfer-learning approaches. The goal of attempting to decrease the cost of learning a black-box performance model by selecting a small but still representative set of samples is similar to our approach of reducing the costly acquisition of samples. However, Jamshidi et al. [29] aim at the configuration space for a user to define a system, while our work focuses on the space of architectural alternatives that an architect has at her disposal during a design activity.

Grebhahn et al. [25] conducted an empirical study to determine the impact of using particular ML algorithms and sampling strategies on the accuracy of performance predictions for binary and numeric configuration options. In their experiments, they found that the error rate of the ML techniques strongly depends on the subject system and the sampling strategy chosen. Since no significant interactions between ML techniques and particular sampling strategies could be observed, it is claimed to be difficult to recommend one ML technique over another independently of the learning set and the subject system under consideration. In contrast to our work, the authors focused on performance prediction and not on multi-objective optimization problems. At the code level, surrogates have been employed to optimize compiler transformations; for example, predictive models embedded in LLVM can estimate the runtime and energy impact of optimization sequences, enabling greener software with reduced evaluation cost [17].

Few works directly address surrogate models in the context of software architecture optimization. Xin et al. [51] assisted an evolutionary algorithm with Random Forest surrogates for performance optimization, reporting execution time reductions of up to 48% compared to using NSGA-II. However, they did not evaluate prediction accuracy, and

their scope was limited to performance as a single objective. Our work, in turn, predicts multiple quality attributes and explicitly examines the tradeoff between prediction errors and computational costs.

In summary, while surrogate models have been widely applied in engineering disciplines, system architecture optimization, configurable systems, compiler optimization, and neural architecture search, their use in software architecture optimization remains scarce. Prior work has largely focused on either configuration tuning or system-level engineering problems, whereas our contribution adapts surrogate-based optimization to software architectures. By leveraging architectural structure and transformations in the surrogate design and incorporating incremental learning, our approach seeks to balance computational costs and predictive accuracy for practical optimization settings.

8 CONCLUSION

We investigated the applicability of surrogate models in the multi-objective quality optimization of software architectures. The goal is to speed up the optimization process by interleaving the use of accurate but resource-intensive solvers with the use of less accurate but faster surrogate models. We proposed an approach that builds on (but is not limited to) the *SQuAT* framework, using architectural information such as tactics and graph embeddings to build a surrogate model, which can be updated as the optimization proceeds. In our experiments, we investigated the performance of single- and multi-output regression techniques, including their prediction errors per level of the search tree.

The results obtained with our surrogate models, either using the single-output or multi-output variants, were encouraging in terms of efficiency savings. The `CatBoost` regressors achieved reasonable *RMSE* values in the ranges of $[0.03 - 0.04]$ and $[0.01 - 0.08]$ for the (standardized) quality-attribute objectives of `ST+` and `CoCoME`, respectively, when sampling between 40% and 50% of the data at each tree level. Based on the Monte Carlo analysis, prediction errors can be smaller for some tree levels and objectives, although variance needs to be considered. In this regard, choosing a single-output regression seemed to provide a more uniform and often bound *RMSE* variance than multi-output regression. We acknowledge, therefore, that the accuracy of the predictions can vary for individual optimization objectives. Furthermore, we estimated savings of 50% in computation time compared to using the solvers alone. In general, the architect should balance accuracy and computation time by setting an adequate sampling factor for the exploration task or architecture domain under consideration.

We see several promising directions for future work. We would like to expand the scope of our approach by incorporating additional tactics and case studies, and investigating more optimization approaches for the search, e.g., engines based on evolutionary algorithms such as *EASIER* [4]. Furthermore, we believe that exploring alternative sampling strategies based on the characteristics (e.g., diversity) of the design space [50, 53] could enhance the decision-making of when to use the solver and when to use (or update) the surrogate model. We will also analyze whether having individual regressors (i.e., surrogates) for each quality-attribute objective works better than the bulk mode implemented by `CatBoost` in the current *IAL* strategy. An interesting extension of this work would be to design a hybrid surrogate-assisted optimization framework that adaptively switches between single-output and multi-output regression strategies as the optimization evolves. The idea is to exploit multi-output surrogates when objectives show stable correlations and joint evolution, while falling back to independent single-output surrogates when objectives become weakly correlated, evolve differently, or exhibit diverging distributions. Such an adaptive mechanism would allow the optimization process to balance efficiency and accuracy more effectively across iterations, dynamically tailoring the surrogate modeling strategy to the characteristics of the predicted objectives as they unfold in the search.

Finally, we plan to integrate our approach into selected optimization tools to apply the *IAL* approach in "live" optimization scenarios, in which the quality-attribute values predicted by a surrogate steer the search in different

directions in the design space. A challenge here is whether the surrogate leads to an improved architecture optimization, in the sense of finding more (or better) solutions in the same time or equally good solutions quicker than the baseline.

ACKNOWLEDGMENTS

J. Andres Diaz-Pace and A. Tommasel were supported by project PICT-2021-00757, Argentina.

REFERENCES

[1] Aldeida Aleti, Stefan Bjornander, Lars Grunske, and Indika Meedeniya. 2009. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In *2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software*. IEEE, 61–71.

[2] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. 2013. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Trans. on Soft. Eng.* 39, 5 (2013), 658–683.

[3] Theodore Wilbur Anderson, Theodore Wilbur Anderson, Theodore Wilbur Anderson, and Theodore Wilbur Anderson. 1958. *An introduction to multivariate statistical analysis*. Vol. 2. Wiley New York.

[4] Davide Arcelli, Vittorio Cortellessa, Mattia D'Emidio, and Daniele Di Pompeo. 2018. EASIER: an Evolutionary Approach for multi-objective Software archItecturE Refactoring. In *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 105–10509.

[5] F. Bachmann, L. Bass, M. Klein, and C. Shelton. 2005. Designing software architectures to achieve quality attribute requirements. *IEE Proceedings - Software* 152 (August 2005), 153–165(12). Issue 4.

[6] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice* (4 ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[7] Steffen Becker, Heiko Koziolek, and Ralf Reussner. 2009. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82, 1 (2009), 3–22.

[8] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. 2022. Pareto rank surrogate model for hardware-aware neural architecture search. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 267–276.

[9] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. 2021. A comparative analysis of gradient boosting algorithms. *Artif. Intell. Rev.* 54, 3 (March 2021), 1937–1967. https://doi.org/10.1007/s10462-020-09896-5

[10] Simona Bernardi, José Merseguer, and Dorina C Petriu. 2011. A dependability profile within MARTE. *Software & Systems Modeling* 10 (2011), 313–336.

[11] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larrañaga. 2015. A survey on multi-output regression. *WIREs Data Mining and Knowledge Discovery* 5, 5 (2015), 216–233. https://doi.org/10.1002/widm.1157 arXiv:https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1157

[12] Jasper H Bussemaker, Nathalie Bartoli, Thierry Lefebvre, Pier Davide Ciampa, and Björn Nagel. 2021. Effectiveness of surrogate-based optimization algorithms for system architecture optimization. In *AIAA Aviation 2021 forum*. 3095.

[13] Jasper H Bussemaker, Paul Saves, Nathalie Bartoli, Thierry Lefebvre, and Rémi Lafage. 2025. System architecture optimization strategies: dealing with expensive hierarchical problems. *Journal of Global Optimization* 91, 4 (2025), 851–895.

[14] Jasper H Bussemaker, Paul Saves, Nathalie Bartoli, Thierry Lefebvre, and Björn Nagel. 2024. Surrogate-based optimization of system architectures subject to hidden constraints. In *AIAA AVIATION FORUM AND ASCEND 2024*. 4401.

[15] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. 2011. *Model-based software performance analysis*. Vol. 980. Springer.

[16] Vittorio Cortellessa, Jorge Andrés Diaz-Pace, Daniele Di Pompeo, Sebastian Frank, Pooyan Jamshidi, Michele Tucci, and André van Hoorn. 2025. Introducing Interactions in Multi-Objective Optimization of Software Architectures. *ACM Trans. Softw. Eng. Methodol.* 34, 6, Article 181 (July 2025), 39 pages. https://doi.org/10.1145/3712185

[17] Juan Carlos de la Torre, Patricia Ruiz, Pedro L Galindo, and Bernabé Dorronsoro. 2023. A surrogate optimization method for automatically generating greener software using LLVM. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE, 1–7.

[18] Jay L Devore. 2011. *Probability and Statistics for Engineering and the Sciences*. Cengage learning.

[19] J Andres Diaz-Pace, Santiago A Vidal, Antonela Tommasel, Sebastian Frank, and Andre van Hoorn. 2023. Can Multi-Agent Consensus Improve Quality Tradeoffs in Software Architecture Optimization?. In *Anais do XXVI Congresso Ibero-Americano em Engenharia de Software*. SBC, 77–91.

[20] B. Efron and C. Stein. 1981. The Jackknife Estimate of Variance. *The Annals of Statistics* 9, 3 (1981), 586 – 596. https://doi.org/10.1214/aos/1176345462

[21] Lucas Francisco dos Santos, Caliane Costa, José Caballero, and Mauro Ravagnani. 2022. Multi-objective simulation optimization via kriging surrogate models applied to natural gas liquefaction process design. *Energy* 262 (09 2022), 125271. https://doi.org/10.1016/j.energy.2022.125271

[22] Sebastian Frank and André van Hoorn. 2020. SQuAT-Vis: Visualization and Interaction in Software Architecture Optimization. In *Software Architecture*. Springer International Publishing, Cham, 107–119. https://doi.org/10.1007/978-3-030-59155-7_9

[23] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, and Salem Derisavi. 2008. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering* 35, 2 (2008), 148–161.

[24] Aurelien Geron. 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed.). O'Reilly Media, Inc.

[25] Alexander Grebhahn, Norbert Siegmund, and Sven Apel. 2019. Predicting performance of software configurations: There is no silver bullet. *arXiv preprint arXiv:1911.12643* (2019). https://doi.org/10.48550/arXiv.1911.12643

[26] Object Management Group. 2005. UML profile for modeling and analysis of real-time and embedded systems (MARTE).

[27] Sebastian Herold, Holger Klus, Yannick Welsch, Constanze Deiters, Andreas Rausch, Ralf Reussner, Klaus Krogmann, Heiko Koziolek, Raffaela Mirandola, Benjamin Hummel, et al. 2008. CoCoME-the common component modeling example. *The Common Component Modeling Example: Comparing Software Component Models* (2008), 16–53.

[28] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. 2022. Unicorn: Reasoning about configurable system performance through the lens of causality. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 199–217.

[29] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 71–82. https://doi.org/10.1145/3236024.3236074

[30] Ping Jiang, Qi Zhou, and Xinyu Shao. 2020. *Surrogate Model-Based Engineering Design and Optimization*. Springer. https://doi.org/10.1007/978-981-15-0731-1

[31] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. 2011. PerOpteryx: Automated application of tactics in multi-objective software architecture optimization. In *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*. 33–42.

[32] Daniel G Krige. 1951. A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy* 52, 6 (1951), 119–139.

[33] Luc Lesoil, Helge Spieker, Arnaud Gotlieb, Mathieu Acher, Paul Temple, Arnaud Blouin, and Jean-Marc Jézéquel. 2024. Learning input-aware performance models of configurable systems: An empirical evaluation. *Journal of Systems and Software* 208 (2024), 111883.

[34] Peng Liao, Wei Song, Peng Du, and Hang Zhao. 2021. Multi-fidelity convolutional neural network surrogate model for aerodynamic optimization based on transfer learning. *Physics of Fluids* 33 (12 2021). https://doi.org/10.1063/5.0076538

[35] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. 2020. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European conference on computer vision*. Springer, 35–51.

[36] Nicholas Metropolis and Stanislaw Ulam. 1949. The monte carlo method. *Journal of the American statistical association* 44, 247 (1949), 335–341.

[37] Tadao Murata. 1989. Petri nets: Properties, analysis and applications. *Proc. IEEE* 77, 4 (1989), 541–580.

[38] Annamalai Narayanan, Chandramohan Mahinthan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning Distributed Representations of Graphs. *arXiv preprint arXiv:1707.05005* (07 2017). https://doi.org/10.48550/arXiv.1707.05005

[39] James R Norris. 1998. *Markov chains*. Number 2. Cambridge university press.

[40] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 6639–6649.

[41] Alejandro Rago, Santiago Vidal, J Andres Diaz-Pace, Sebastian Frank, and André van Hoorn. 2017. Distributed quality-attribute optimization of software architectures. In *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*. New York, NY, USA, 1–10.

[42] Christian P Robert, George Casella, and George Casella. 1999. *Monte Carlo statistical methods*. Vol. 2. Springer.

[43] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 1325–1334. https://doi.org/10.1145/3340531.3411866

[44] Reuven Y Rubinstein and Dirk P Kroese. 2016. *Simulation and the Monte Carlo method*. John Wiley & Sons.

[45] Ari Saptawijaya. 2022. Surrogate Model-based Multi-Objective Optimization in Early Stages of Ship Design. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)* 6 (10 2022), 782–789. https://doi.org/10.29207/resti.v6i5.4248

[46] V. Titov, S. Frank, Informatik und Naturwissenschaften Universität Hamburg Fakultät für Mathematik, and Universität Hamburg Fachbereich Informatik. 2023. *An Empirical Study on the Effectiveness of Surrogate Model Solving for Efficient Archicture-based Quality Optimization*. Universität Hamburg, Universität Hamburg. https://books.google.com.ar/books?id=mU_y0AEACAAJ

[47] Vadim Titov, J. Andres Diaz Pace, Sebastian Frank, and Andre Van Hoorn. 2025. Architecture Optimization using Surrogate-based Incremental Learning for Quality-attribute Analyses . In *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*. IEEE Computer Society, Los Alamitos, CA, USA, 278–288. https://doi.org/10.1109/ICSA65012.2025.00035

[48] Spyridon Tsattalios, Ioannis Tsoukalas, Panagiotis Dimas, Panagiotis Kossieris, Andreas Efstratiadis, and Christos Makropoulos. 2023. Advancing surrogate-based optimization of time-expensive environmental problems through adaptive multi-model search. *Env. Modeling and Software* 162 (04 2023), 105639. https://doi.org/10.1016/j.envsoft.2023.105639

[49] Hendrik Wöhrle, Felix Schneider, Fabian Schlenke, Denis Lebold, Mariela De Lucas Alvarez, Frank Kirchner, and Michael Karagounis. 2022. Multi-objective surrogate-model-based neural architecture and physical design co-optimization of energy efficient neural network hardware accelerators. *IEEE Transactions on Circuits and Systems I: Regular Papers* 70, 1 (2022), 40–53.

[50] Dongrui Wu, Chin-Teng Lin, and Jian Huang. 2019. Active learning for regression using greedy sampling. *Information Sciences* 474 (2019), 90–105. https://doi.org/10.1016/j.ins.2018.09.060

[51] Du Xin, Ni Youcong, Wu Xiaobin, Ye Peng, and Xin Yao. 2017. Surrogate model assisted multi-objective differential evolution algorithm for performance optimization at software architecture level. In *Simulated Evolution and Learning: 11th Int. Conference, SEAL 2017, Shenzhen, China, November 10–13, 2017, Proc. 11*. Springer, Springer International Publishing, Cham, 334–346.

[52] Nooshin Yousefzadeh, My Thai, and Sanjay Ranka. 2023. *A Comprehensive Survey on Multilayered Graph Embedding*. Technical Report. https://doi.org/10.36227/techrxiv.24152793

[53] Marcela Zuluaga, Andreas Krause, and Markus Püschel. 2016. ε-PAL: an active learning approach to the multi-objective optimization problem. *J. Mach. Learn. Res.* 17, 1 (Jan. 2016), 3619–3650.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009