Architecture Optimization using Surrogate-based Incremental Learning for Quality-attribute Analyses

Anonymous Authors

Abstract—**Several approaches have tackled automated architecture exploration using multi-objective optimization techniques. Although these approaches are useful to assist architects when many alternative designs are available, they pose efficiency challenges in large design spaces. This is so because evaluating architecture alternatives often requires transformations to and execution of specialized analysis models (e.g., an LQN solver for performance properties) that are computationally expensive, particularly when invoked repeatedly during the optimization process. In this context, surrogate models, which approximate the behavior of the quality-attribute solvers at a lower computational cost, are a promising approach. In this paper, we adapt a search-based architecture optimization process and combine evaluations of both the solvers and surrogate models using incremental learning techniques. The surrogates are built from the data generated by the optimization via a multi-output regression model. To make quality-attribute predictions, our regression model leverages the structure of the architecture alternatives as well as the refactoring operations being applied to them by the optimization engine. We assessed the approach with two case studies (*ST+* and *CoCoME*) from the literature, particularly investigating the accuracy-cost tradeoff and the effects of our surrogate design on this tradeoff. The experimental simulations of the optimization process showed reductions of up to 50% in computation time, albeit at the cost of inaccuracies (prediction errors) for some quality-attribute metrics in the architecture datasets. Beyond our approach, our results give insights into how architecture exploration approaches can leverage surrogate models to become more efficient and take larger design spaces.**

I. Introduction

To support the exploration of alternative architectures to satisfy a set of quality-attribute requirements, several search-based approaches that employ multi-objective architecture optimization [2] have been proposed, e.g., *ArcheOpterix* [1], *EASIER* [4], *PerOpteryx* [27], and *SQuAT* [36]. The optimization entails an iterative process with certain rules for automatically refactoring an initial architecture and deriving alternatives. Although these approaches are potentially useful to assist architects in navigating a large design space, they usually face efficiency challenges. Specifically, the generation (and evaluation) of new architecture alternatives usually relies on predefined analysis models (or solvers), e.g., using Layered Queuing Networks (LQNs) for performance [19], which transform each architecture to an internal representation and then evaluate it either analytically or via simulations to compute different metrics (e.g., response time for performance, cost for modifiability, probability of failure for availability). These metrics quantify quality-attribute properties of the architectures and serve as the optimization objectives. However, the transform-and-evaluate process above can take considerable computational resources, particularly when executed several times during the optimization process and on non-trivial architecture instances. Thus, expanding the range of explored architectures becomes expensive and limits the applicability of architecture optimization approaches.

This challenge has been faced by other engineering disciplines [26], mostly in cases of multi-objective optimization of numeric variables. In this context, *surrogate models* have proved effective in reducing the computational costs of evaluating complex objective functions. By a surrogate model, we refer to an approximation of the original model or function, which is cheaper to construct — oftentimes following a data-driven strategy. For multi-objective architecture optimization, this means approximating the quality-attribute solvers through Machine Learning (ML) techniques such as multi-output regression [3], [28], [9], based on features for characterizing the architectures. Recently, some works have tackled this direction to learn performance models and make inferences from architectural configurations [41], [25]. In our view, applying a surrogate-based approach to architecture optimization involves three main issues: i) the identification of adequate features for the architectures, ii) the evaluation (and selection) of an appropriate regression technique, and iii) the insertion of the surrogate within the optimization process.

In this paper, we propose a surrogate-based approach for the architecture exploration process followed by the *SQuAT* framework [36] to predict quality-attribute properties of latency, reliability, and cost (of changes). Regarding the feature representation, we investigate the usage of architectural tactics [6] and graph embeddings [42] computed on the architectural configurations. We also propose an incremental and active learning (IAL) strategy, which is depicted in Fig. 1. Based on the workflow of the optimization process, a surrogate model is learned with an initial batch of architectural data, and gets progressively updated as new data (from the optimization rounds) is available. The surrogate can make quality-attribute predictions for architecture alternatives in place of the solvers, except for cases in which the (estimated) prediction error is high and the solvers are preferred instead. Along this line, we evaluated several regression techniques and exemplify how one of the best-performing algorithms (Random Forests) can be integrated into the *IAL* strategy. We perform an experimental evaluation with two case studies from the literature, *ST+* [36] and *CoCoME* [23], to demonstrate the feasibility of the surrogate-based approach, which also exposes a tradeoff between computation time savings and prediction accuracy.

The rest of the paper is organized as follows. Section 2 provides background information about architecture opti-

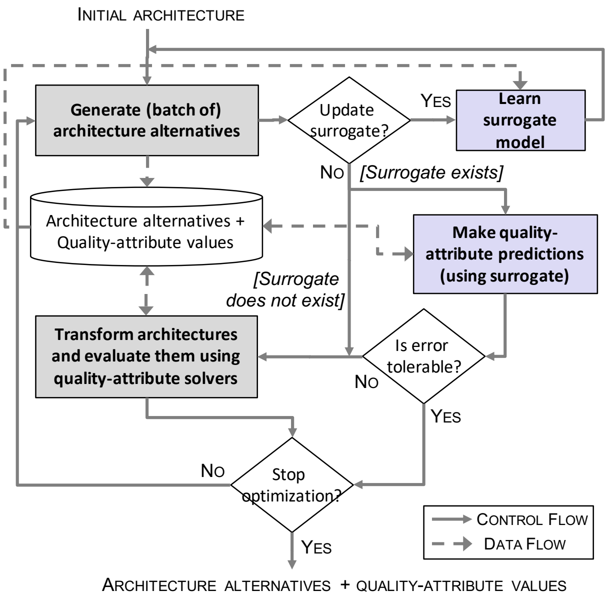

INITIAL ARCHITECTURE
Generate (batch of) architecture alternatives
Update surrogate?
YES
Learn surrogate model
NO
[Surrogate exists]
Architecture alternatives + Quality-attribute values
Make quality-attribute predictions (using surrogate)
[Surrogate does not exist]
Transform architectures and evaluate them using quality-attribute solvers
Is error tolerable?
NO
YES
NO
Stop optimization?
YES
CONTROL FLOW
DATA FLOW
ARCHITECTURE ALTERNATIVES + QUALITY-ATTRIBUTE VALUES

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

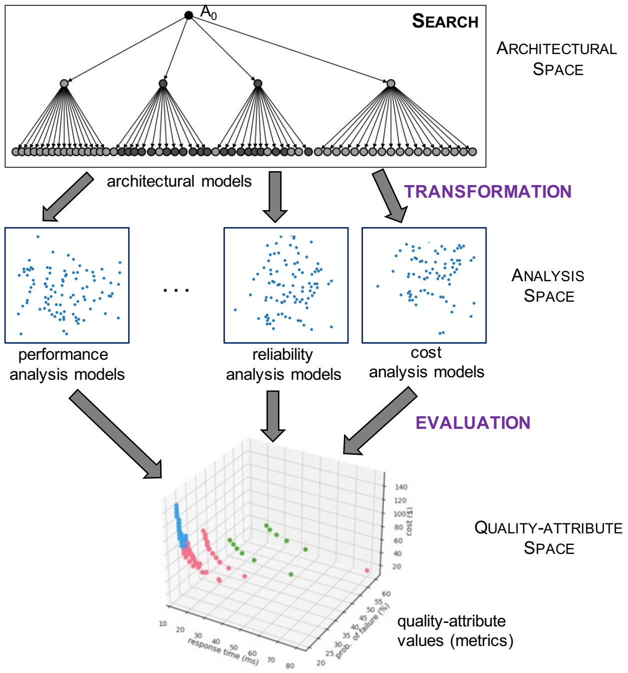


Fig. 2: Search, transformation, and evaluation phases

mization, with an emphasis on the quality-attribute evaluation and transformation of architectures. Section 3 describes how surrogates are modeled, built, and updated using the *IAL* strategy. Section 4 presents the evaluation of the approach via simulated optimization experiments based on case study datasets. Section 5 discusses related work. Finally, Section 6 gives the conclusions and outlines future work.

II. The Architecture Optimization Cycle

A typical architecture optimization process involves three phases, namely: i) search (or exploration), ii) transformation, and iii) evaluation, as illustrated in Fig. 2. The cycle starts with an initial architectural model (A_0) that is fed into an optimization engine. As part of the search phase, the engine generates several children architectures by applying predefined refactoring operations (or architectural tactics [6]) to the initial architecture in the architectural space. Each generated architecture is an alternative for satisfying the quality-attribute objectives. To quantify the satisfaction levels, each architecture alternative is converted to specific analysis models (in the analysis space) to be evaluated by quality-specific solvers as part of the transformation phase. At last, during the evaluation phase, the assessment of the architectures leads to numeric values (according to predefined metrics) in the quality-attribute space. The cycle is repeated for each alternative in order to generate more children, which subsequently go through the transformation and evaluation phases. The process usually finishes once a maximum number of iterations is reached, and all the explored architectures are returned as output. This kind of architecture optimization is implemented by the *SQuAT* framework [36], but other frameworks work similarly.

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 [13]. For instance, performance and reliability estimations can be achieved using UML models annotated according to the MARTE [22] and DAM [8] profiles, respectively. A transformation is then necessary to enable a quantitative analysis. For example, LQNs [19] are a typical formalism for producing performance estimations, while Petri Nets [34] and Markov Chains [35] can be employed for reliability estimations.

To understand how the search phase works, Fig. 3 shows an example of the application of two operations (tactics) to a client-server architecture A_1 to derive alternatives A_2, A_3, A_4 and A_5. Tactic T_1 implies adding a (new) service to an existing device (server), which improves processing time and increases deployment cost. In contrast, tactic T_2 adds a new service but deploys it in a separate device, improving processing time and availability. This example illustrates how the candidate architectures are generated as a tree (from left to right) and how the initial architecture structure is progressively modified.

When it comes to *SQuAT* [36], it seeks 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 (as in Fig. 3) 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, as currently implemented by *SQuAT*, is partial (i.e., the aim is not necessarily to perform a full, combinatorial search in the design space) and is not driven by the evaluation results returned by the solvers.

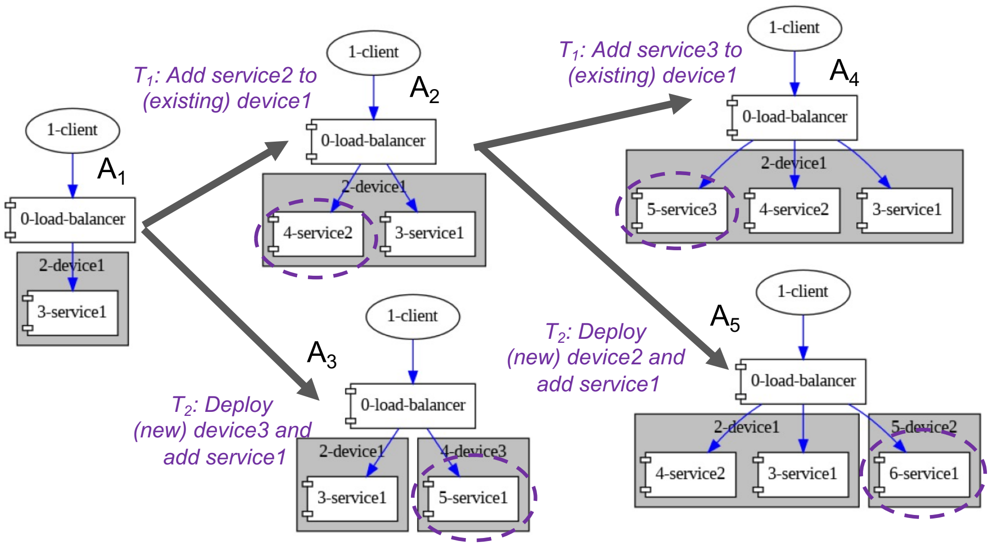


Fig. 3: Example of applying tactics to a client-server architecture A_1 to derive alternatives A_2-A_5

A. Problem formulation

We consider the 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 $d_{ij} = 1$ if the decision was made for configuration A_i or 0 otherwise.

In this work, we restrict the possible decisions d_{ij} to architectural tactics [6]. As mentioned above, a tactic is a design operation 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 applied tactics. AS can be visualized as a directed acyclic graph in which each node represents a configuration, while an edge between two nodes A_i and A_j captures a tactic leading from the former to the latter. 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 $\Re^k$. 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. Precisely, E is the costly function we would replace with a surrogate model when possible.

An automated engine will be responsible for searching the architectural space and generating a (large) graph of configurations. The techniques for populating AS might include specific architectural tools [2], evolutionary algorithms [12], or model checkers [29], among others. Since enumerating all the configurations available in the architecture space is usually computationally unfeasible, only a subset of those configurations will be generated. This approach does not depend on the tool or the search technique as long as it can expose both the configurations and tactics applied to them.

III. Proposed Approach

The optimization cycle can be seen as a relation between architecture instances and a multi-valued response surface, which assigns a numeric score to each instance with respect to the quality-attribute objectives. From a data-driven perspective, the execution of the optimization generates a dataset comprising both architectural instances 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.

A. 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 keep a 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], [28]. In particular, since we deal with multiple objectives, *multi-output regression* [9] can be applied.

B. Feature representation

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

We identify two sources of information for creating these vectors. First, the fact that the optimization engine derives the different architecture alternatives via tactics. Coming back to the example in Fig. 3, there could be a tactic $T = increaseCapacity(?device, ?service)$ that deploys (and activates) a new service on a given device, and then tactic instantiations for specific devices and services. 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 initial node and a particular node in the search graph. For the regression task, the categorical data involved in the tactic sequences can be converted to a numeric format using a one-hot encoding schema [20].

Second, since architectures can naturally be modeled as graphs (e.g., using a component-and-connector view), we can compute *graph embeddings* [42] for them. By embeddings, we refer to the mapping of entities, edges, and subgraphs to numeric vectors in a lower-dimensional, compact space. In particular, we rely on the *Feather-N* graph (FG) approach [37], which allows to efficiently represent the distribution of vertex characteristics at multiple scales, using a flexible notion of characteristic functions defined on graph vertices. The probability weights of those characteristic functions are defined as the transition probabilities of random walks. Alternative graph embedding strategies could also be experimented.

Note that two sequences with similar tactics but applied in a different order, e.g., $< t_1, t_2, ... >$ and $< t_2, t_1, ... >$, might or might not lead to the same architecture instance, depending on whether the tactics are orthogonal in their changes to the base architecture. In fact, in *SQuAT*, we cannot detect duplicate architecture candidates due to how its search process works. With the graph embeddings though, two architecture instances mapped to the same embedding imply that the instances share the same architectural structure. In such cases, the solver results can be reused (e.g., like a cache mechanism) in order to avoid duplicate computations.

C. Interleaving solvers and predictions

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 optimization process, which proceeds iteratively, not all the architecture instances are available at once. A way around this issue is to construct the surrogate incrementally as the optimization process progresses and new architectural information is acquired. Along this line, we have devised an ML strategy that combines incremental and active learning to adapt the surrogate to the nature of the architecture optimization process. This strategy, called *IAL*, is described in Algorithm 1.

Algorithm 1 Incremental Active Learning (IAL) strategy

```
Inputs: $< X_1, X_2, ..., X_T >$: batches
    $SF$ : sampling factor
    $E$ : transformation and evaluation function (solvers)
Outputs: $\widehat{E}$ : surrogate for $E$
    $< \widetilde{Y_1}, \widetilde{Y_2}, ..., \widetilde{Y_T} >$: objective values for batches

$\widetilde{Y_1} \leftarrow E(X_1)$ {Invoke solvers on initial batch}
$\widehat{E} \leftarrow buildSurrogate(X_1, \widetilde{Y_1})$ {Train initial surrogate}
{Iterate over the remaining batches}
foreach $i \in [2..T]$ do
    $\widetilde{Y_i} \leftarrow \widehat{E}(X_i))$ {Predict with current surrogate}
    $X_i^S \leftarrow sample(X_i, \widetilde{Y_i}, SF)$ {Select SF% of random instances or with largest prediction errors}
    $\widetilde{Y_i^S} \leftarrow E(X_i^S)$ {Invoke solvers on sample}
    $\widehat{E} \leftarrow adjustSurrogate(\widehat{E}, X_i^S, \widetilde{Y_i^S})$ {Re-train surrogate}
    $\widetilde{Y_i} \leftarrow \widehat{E}(X_i - X_i^S) \cup \widetilde{Y_i^S}$
end for
return $\widehat{E}, < \widetilde{Y_1}, \widetilde{Y_2}, ..., \widetilde{Y_T} >$ {Note that some Y values are predicted while others are computed by the solvers}
```

Incremental learning permits using (new) input data to train the (current) model further. This way, we can adjust our regression model with information of 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 [20], 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). The active learning process chooses training (labeled) samples based on their likelihood to improve the model performance, what is known as *sub-sampling*. A common sub-sampling criterion is to estimate the error magnitude (or level of uncertainty) of the current predictions and then select those instances with the largest error. In our approach, we slightly adapt the active learning procedure to select training samples based on the (intermediate) results from the architecture optimization process. As new information is available, the model is adjusted based on the cumulative labeled data, and its accuracy is expected to increase.

In addition to the quality-attribute solvers, the *IAL* strategy takes the architectural instances as inputs, assuming that they arrive arranged in batches $X_1, X_2, ..., X_T$. For instance, a batch X_k can be the set of architectures corresponding to a particular level in the search tree. The evaluation of the architectures in X_k, either returned by the solvers or (predicted by) the regression model, is captured by Y_k. The algorithm begins by training a first version of the surrogate on the first batch. Then, the model just built is exercised with instances from the next batch. The *sample()* function implements the sub-sampling criterion, which returns a set of instances X_i^S that serve to re-train the model via the *adjustSurrogate()*

function. After going through all the batches, both the adjusted surrogate and all the quality-attribute values are returned.

The SF (sampling factor) parameter is used to control the number of instances in X_i^S. For instance, if $SF = 0.1$, the $sample()$ function takes 10% of the instances from the current batch, which are then evaluated by executing the solvers. Thus, the SF plays a role in controlling for accuracy and computational costs. Ideally, SF should take a small fraction of the batch for the approach to be cost-effective. In each iteration (or batch), the candidate instances to sample can be chosen either randomly or based on the uncertainty of the current surrogate predictions. One way of estimating prediction uncertainty is via a *jacknife* method [16], which is a type of cross-validation procedure.

IV. Evaluation

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. 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 only affect the results in the quality-attribute space, but it does not affect the optimization process nor the architecture space.

The research questions are the following:

- **RQ#1**: How do the different feature representations (tactic sequences and embeddings) affect the predictive capability of the surrogate models?
- **RQ#2**: What is the accuracy of the surrogate models with respect to the standard optimization (ground truth)?
- **RQ#3**: What is the computational cost of using the surrogate models in combination with the solvers?
- **RQ#4**: In the *IAL* strategy, how does the sampling factor influence accuracy and computational cost?

We achieved the processing of the case studies for constructing the surrogate models using an ML pipeline. Along this line, the case studies were treated as input datasets for the pipeline, which involved several phases, namely: pre-processing, model building, model validation, and model selection. We used the best-performing models to evaluate the *IAL* strategy. The experiments were carried out with a *Jupyter* notebook, which is available for reproducibility[1].

A. Case study datasets

The datasets for *ST+* [36] and *CoCoME* [23], as provided with the *SQuAT-Vis* toolkit [18], were used for the study. The main characteristics of the case studies are summarized in Table I. Figure 4 depicts the search trees resulting from running *SQuAT* on *ST+* with 2 iterations (a) and *CoCoME* with 5 iterations (b). The initial architecture is located in the center (root), and each branch represents a sequence of applied tactics. Each node is an architecture alternative that can be reached through the corresponding path from the root.

TABLE I: Overview of the case study datasets

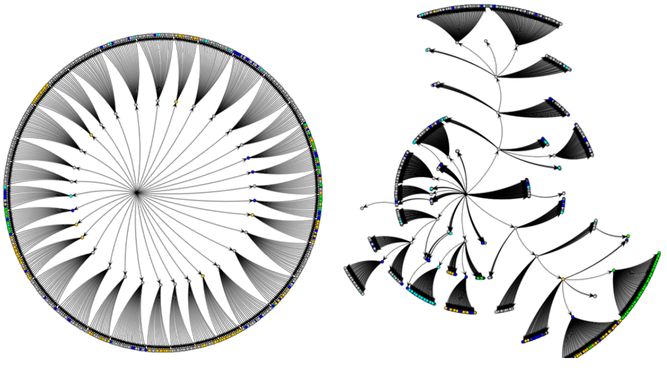
Name	Sequence Length	Objectives	Components	Alternatives
CoCoME	5	8	55	1192
ST+	2	4	9	454

The *ST+* dataset describes a relatively simple trip management system, while the architecture in the *CoCoME* dataset describes a more complex trading system used in a supermarket. In both cases, the modifiability objectives aim to keep the complexity of adding features low. For performance, the objectives aim to keep response times low under certain conditions. 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) redeploy components, iii) scale resources, iv) move distributed components, and v) change resource capacity.

The case studies differ in the structure of their search space (i.e., the *ST+* tree is balanced while the *CoCoME* tree has an irregular shape), but also in the number of alternatives being explored, with 454 and 1192 instances for *ST+* and *CoCoME*, respectively. Each architecture instance is linked to a Palladio (PCM) [7] model in Eclipse-EMF format, which captures structural and dynamic aspects of the architecture.

B. Pre-processing

Both datasets were arranged in a tabular format (`CSV` files) composed of columns for sequences of refactoring actions to be used as ML features, as well as for performance and modifiability metrics for the quality-attribute objectives to be used as regression targets. For *ST+*, the initial encoding has: two modifiability targets (*m1* and *m2*), two performance targets (*p1* and *p2*), and a sequence of at most two tactics (*op1* and *op2*). The *CoCoME* dataset includes: four modifiability targets (*m1* to *m4*), four performance targets (*p1* to *p4*), and



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

Fig. 4: Graphical representations of the search trees (architecture spaces) for the two case studies [15].

[1] Google Colab notebook: https://shorturl.at/ONZW8

a sequence of up to five tactics (*op1* to *op5*). The tactic sequences leading to each architecture instance were taken from [18] and treated with one-hot encoding [20]. Since the targets have different scales, we normalized them via the Z-standardization method.

To enrich the encoding above, we computed graph embeddings on the architectural structure of the instances (i.e., on the PCM models). These embeddings can be appended to the initial encoding or used separately. After testing several embedding techniques, we chose FG [37] because it was fast to compute (less than a minute) and showed reasonable accuracy in the regression tasks. Furthermore, once learned, the FG embeddings can be applied to unseen architecture instances.

Since the original PCM format is quite detailed, and not all its details can be fed into the standard graph embedding techniques, the PCM models are converted to a simplified representation developed as part of *SQuAT-Vis* [18]. The architectures given in Fig. 3 are examples of this language. It only keeps elements related to (runtime) components, connectors, and hardware, which are arranged in a graph-like architectural view. Hardware elements are modeled as resource containers (e.g., servers), which might contain resources (e.g., CPUs), while software elements are modeled as components that make requests to other components via usage relations. The allocation of software to hardware elements is modeled by relations from components to containers.

C. Model building, validation and selection

We defined a multi-output regression task [9], which predicts all the objective values (targets) simultaneously based on a set of features (encoding) for the architectural instances. We evaluated different regression algorithms and chose Random Forests (RF) [10] and XGBoost [11] due to their good performance and ability to deal with non-linearity and outliers. Furthermore, both RF and XGBoost admit incremental model training. We split the *ST+* and *CoCoME* datasets into training and test sets with a $50/50\%$ partitioning schema. This differs from the usual $70/30\%$ schema used in ML, as it is not always realistic in architecture optimization contexts, because a training set considering *70%* of the instances often means a heavy load executing the solvers. In fact, a $50/50\%$ schema is an optimistic case, and a smaller training set is desirable for our surrogates. The accuracy of the predictions was assessed using standard regression metrics such as R^2, RMSE (Root Mean Square Error), and MAE (Mean Absolute Error). A good regression model should often get R^2 values closer to 1.0 and error values towards 0.0.

We initially ran a *naïve baseline* in a standard ML mode, in which the training and test instances were selected using stratification by tree level (2 and 5 levels for *ST+* and *CoCoME* respectively) from the dataset, with a $50/50\%$ partitioning schema. This selection tries to mimic the order in which the architecture instances are generated by the optimization process. Table II summarizes the performance results of the baseline models with respect to the *ground truth*, i.e., using the solvers on all the architectural instances (without surrogates). The metric values are averaged for the whole set of regression targets (4 and 8 objectives for *ST+* and *CoCoME*, respectively). For the RF and XGBoost models, we tested three feature representations: i) using only the tactic sequences, ii) using only the embeddings, and iii) using both tactics and embeddings.

The best predictions were obtained when combining tactic sequences and embeddings for both models with R^2 values in the range $[0.56, 0.81]$ for the test set and small errors (according to the MSE, MAE, and RMSE metrics). In CoCoME, the training times were slightly higher for the combined option (≈ 10 sec) than for the other two options. This result indicates that both feature representations provide complementary information for the prediction, which answers **RQ#1**. Although the embeddings provide a good portion of structural information about the architectures, they do not capture the "links" between those architectures, in the sense that they all derive from the same initial architecture. We argue that the tactic-related features provide such information. As a sanity check, we also computed SHAP values [31] to analyze the key features used by the combined option when making its predictions. A global feature importance chart revealed that both embedding and tactic-related features were among the top-15 contributors to the predictions of both models.

From Table II, we can see that XGBoost outperformed RF in terms of R^2, RMSE and MAE for *ST+* with the three feature representations. However, XGBoost and RF performed similarly for *CoCoME* across all representations. This effect might be due to the differences in the tree levels for each dataset. These experiments provide an initial answer to **RQ#2** by saying that a surrogate model trained with XGBoost can achieve a reasonably good accuracy with respect to the ground truth. Nonetheless, it should be noticed that this baseline disregards the iterative cycle of the optimization process, which prevents one from having the training dataset available all at once. To overcome this issue the instances for (re-)training the model should be picked in an incremental fashion.

D. Experiments with IAL strategy

The *IAL* strategy deals with a realistic scenario of how instances are generated during the optimization process. As new batches of instances are available, the regression model gets progressively adjusted. In this scenario, the model is trained on a given batch (or tree level) and tested on the next one. To compare against the naïve baseline, we configured the *IAL* algorithm with XGBoost and ran it with different sampling factors. For the *sample()* function of *IAL*, both the random and jackknife strategies were tested, but we obtained better accuracy in the predictions with the former strategy. The instances selected via random sampling had more diversity than those based on prediction errors, because the (estimated) errors seemed to be mostly uniformly distributed in the quality-attribute space of our datasets. As we did for the naïve baselines, we also analyzed feature importance for the regression models resulting from the *IAL* strategy via SHAP values. This analysis confirmed that tactic sequence and

TABLE II: Error metrics and time taken for training for performing experiments with feature vectors constructed from refactoring actions and/or architectural embeddings with FG. Colors indicate the best (green) and worst (red) values for each experiment setting within a dataset. Bold values indicate the best values for the comparison between XGBoost and RandomForest when combining tactics and embeddings.

Dataset	Features	Model	Experiment	R^2	RMSE	MAE	train_t [s]
ST+	tactics only	XGBoost	50%/50%	0.6532	0.5888	0.2485	2.88
ST+	tactics only	XGBoost	full set	0.8570	0.3781	0.1385	2.88
ST+	tactics only	RandomForest	50%/50%	0.4944	0.7110	0.2982	3.94
ST+	tactics only	RandomForest	full set	0.7481	0.5918	0.1969	3.94
ST+	embeddings only	XGBoost	50%/50%	0.3321	0.8172	0.4026	2.69
ST+	embeddings only	XGBoost	full set	0.3918	0.7798	0.3945	2.69
ST+	embeddings only	RandomForest	50%/50%	0.3277	0.8199	0.4062	3.51
ST+	embeddings only	RandomForest	full set	0.3897	0.7812	0.3979	3.51
ST+	**tactics & embeddings**	**XGBoost**	50%/50%	**0.8081**	**0.4379**	**0.1517**	**3.47**
ST+	**tactics & embeddings**	**XGBoost**	full set	**0.9327**	**0.2592**	**0.0814**	**3.47**
ST+	tactics & embeddings	RandomForest	50%/50%	0.6983	0.5491	0.1871	3.59
ST+	tactics & embeddings	RandomForest	full set	0.8641	0.3685	0.1174	3.59
CoCoME	tactics only	XGBoost	50%/50%	0.4557	0.7377	0.2804	4.19
CoCoME	tactics only	XGBoost	full set	0.5867	0.6428	0.2181	4.19
CoCoME	tactics only	RandomForest	50%/50%	0.4606	0.7343	0.2799	2.25
CoCoME	tactics only	RandomForest	full set	0.5822	0.6463	0.2214	2.25
CoCoME	embeddings only	XGBoost	50%/50%	0.2954	0.8393	0.2625	7.81
CoCoME	embeddings only	XGBoost	full set	0.3669	0.7956	0.2305	7.81
CoCoME	embeddings only	RandomForest	50%/50%	0.3148	0.8277	0.2601	6.81
CoCoME	embeddings only	RandomForest	full set	0.3759	0.7899	0.2316	6.81
CoCoME	**tactics & embeddings**	**XGBoost**	50%/50%	0.5624	**0.6614**	**0.2092**	11.59
CoCoME	**tactics & embeddings**	**XGBoost**	full set	0.6558	**0.5886**	**0.1594**	11.59
CoCoME	tactics & embeddings	RandomForest	50%/50%	**0.5621**	0.6617	0.2135	**10.34**
CoCoME	tactics & embeddings	RandomForest	full set	**0.6411**	0.5991	0.1691	**10.34**

embedding information was present in the top-15 most relevant features of the best-performing XGBoost and RF models.

Figs. 5 and 6 show the surrogate results with $SF = 0.5$ after training the models with two and five batches for *ST+* and *CoCoME*, respectively. The axes of the charts correspond to the values predicted by the surrogate versus the actual values (i.e., the ground truth). Note that the values are standardized.

At first glance, we can see good accuracy results for the individual objectives in both datasets. In $CoCoME$, the R^2 values were inferior for the performance objectives (and their errors larger) when comparing them to the modifiability objectives. This difference might be attributed to the characteristics of the quality-attribute distributions (e.g., more outliers for the performance objectives). On average, *ST+* achieved an $R^2 = 0.89$ and $RMSE = 0.33$ which are on par with the baseline average metrics using tactics plus embeddings as the feature representation. A similar trend was observed for *CoCoME* with $R^2 = 0.64$ and $RMSE = 0.59$. Thus, coming back to **RQ#2** we can say that the surrogate model, albeit it introduces errors, provides an accuracy that is close to that of the ground truth for most of the quality-attribute objectives.

The performance of the surrogate model varies as a function of the sampling factor. A higher sampling factor means more invocations to the solvers, which should increase the model accuracy (and might even lead to overfitting) but incur higher computation costs. On the contrary, a lower sampling factor should save costs, but it might cause underfitting and lower accuracy in the model. Figs. 7 and 8 show the evolution of R^2 and RMSE for increasing sampling factors. The horizontal dashed lines correspond to the metrics of the naïve baselines. For *ST+*, the accuracy of the predictions seems to stabilize at $SF = 30\%$, whereas for *CoCoME* the predictions improve linearly as the the sampling factor increases. This behavior might be related to the fact that the *CoCoME* surrogate underwent four re-retraining iterations while the *ST+* counterpart took only one re-training iteration. Overall, in the context of **RQ#4**, the metrics of Figs. 7 and 8 indicate that is feasible to build an XGBoost-based surrogate and progressively improve it during the optimization process, as proposed in Fig. 1. Nonetheless, for the surrogate approach we assume that the architect can tolerate some accuracy loss in the quality-attribute analyses performed during the optimization.

The processing and training times using XGBoost for both the naïve and *IAL* strategies were small. Encoding all the features and generating the embeddings took around 1 minute (on average), while training (or re-training) the models took less than 30 seconds on standard hardware. Certainly, the re-training time depends on the choice of the sampling factor. Furthermore, the sampling strategy can add to the *IAL* execution time, when the jackknife method is used for selecting the

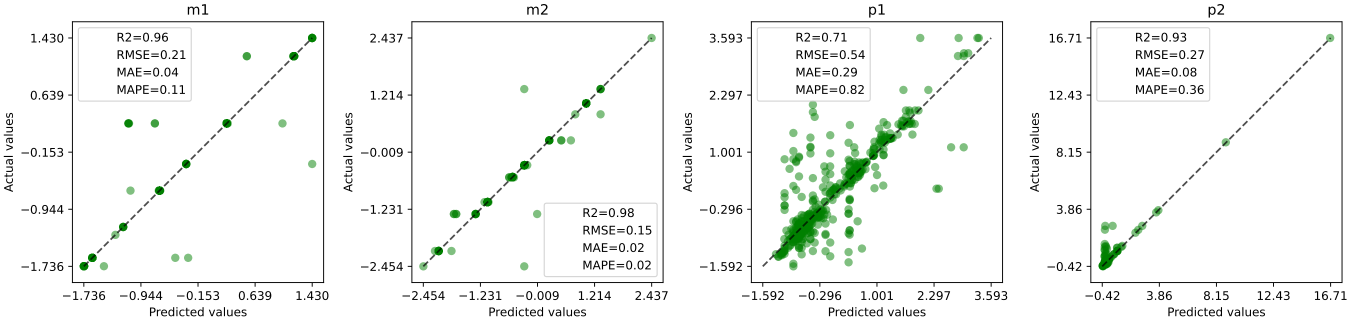


Fig. 5: *ST+* regression results on the whole dataset, using the *IAL* strategy and tactics + embeddings — $SF = 0.5$

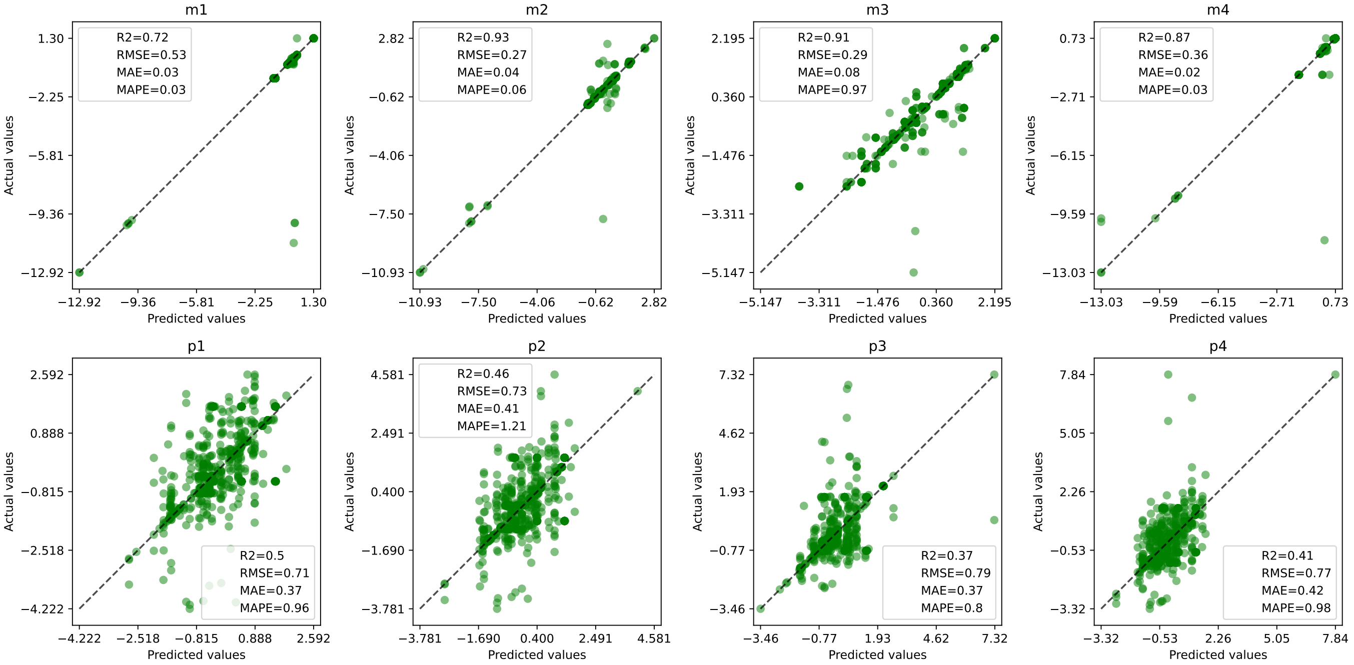


Fig. 6: *CoCoME* regression results on the whole dataset, using the *IAL* strategy and tactics + embeddings — $SF = 0.5$

architecture instances, due to its underlying cross-validation procedure. Anyway, since the transformation and evaluation of an architecture instance often take from 30 seconds up to 1 minute (on average) for *ST+* and *CoCoME*, respectively, we can say that the time reduction offered by the surrogate models was beneficial.

For analyzing the total computation time of the optimization process, we need to consider the time consumed when calling the solvers plus the re-training time for the surrogate model. In the extreme case where no surrogate is used, the total time can be approximated as the average time for processing an architecture with every solver times the number of instances in the dataset. When using a surrogate, the time consumed by the solvers is modulated by the sampling factor. This trend is shown in Fig. 9 for *ST+* and *CoCoME*. The estimated total time increases linearly with a larger sampling factor, while the R^2 values get improved. For instance, with $SF = 0.3$ in *ST+* the optimization would take around 1.15 hours to complete (including the time spent by the solvers), while in *CoCoME* with $SF = 0.5$, it would take around 10 hours of execution time. Thus, we address **RQ#3** by saying that training the surrogate models on the datasets did not have a negative impact on the computational cost.

As expected, the sampling factor introduces a tradeoff between computational time and accuracy of the optimization process. If the R^2 values are judged as good enough (in conjunction with the errors) for a sampling factor between 0.3 and 0.5, then the surrogate models can save half of the computational time approximately in our experimental settings. This result contributes to answering **RQ#4**. Furthermore, we conjecture that having several re-training cycles when sampling instances randomly might hinder the surrogate accuracy, as observed in *CoCoME*. Nonetheless, alternative active learning strategies could be tested in this regard.

E. Threats to validity

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

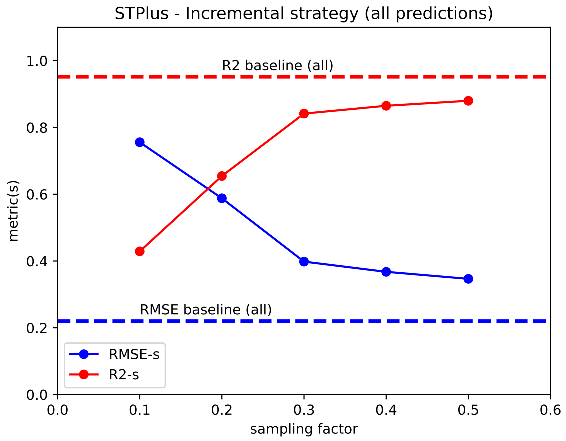


Fig. 7: Variations of R^2 and RMSE metrics for *ST+* based on the sampling fraction (SF parameter in *IAL* strategy).

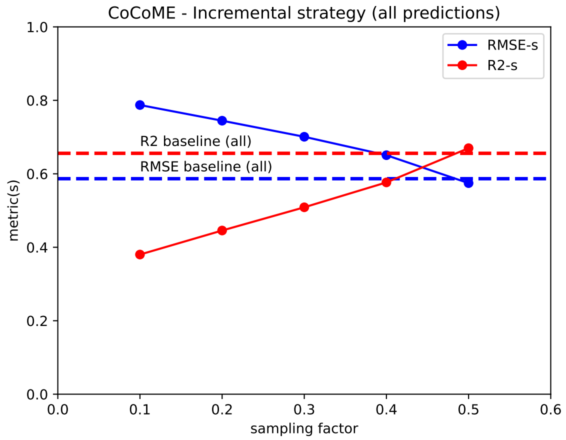


Fig. 8: Variations of R^2 and RMSE metrics for *CoCoME* based on the sampling fraction (SF parameter in *IAL* strategy).

The controlled environment for the simulations and the nature of the data used might differ from a real-world optimization scenario, particularly for the naïve baseline representations. Although this setting allowed us to minimize the influence of noise or model complexity at this stage, it might have also biased our regression techniques. For the feature representation, using tactics and embeddings might not have captured enough architectural information to obtain accurate predictions. For instance, the graph embeddings were based on a simplified architecture structure, which disregards parts of the PCM specification (e.g., parameters of the elements) that might be related to the target objectives. We did not perform analyses, such as feature correlation and selection, which could have removed irrelevant factors from the surrogate representation and potentially boosted performance.

Furthermore, since our experiments were based on pre-generated datasets and *SQuAT* decouples the generation of architecture candidates from their quality-attribute analysis, our evaluation did not assess the behavior of the *IAL* strategy for driving the optimization process. That is, the prediction errors did not affect the architectures being explored.

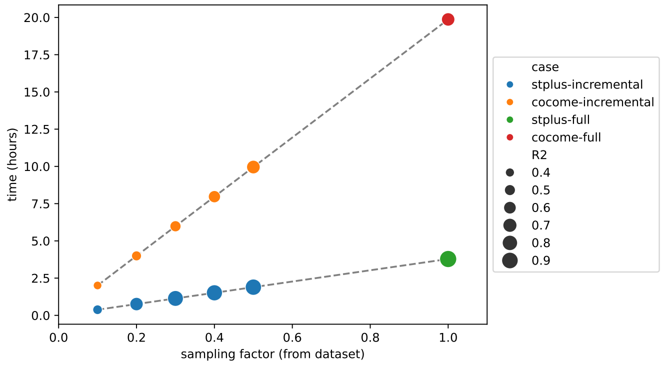


Fig. 9: Tradeoff between total computation time and sampling factor. Average R^2 values are also shown.

The evaluation metrics used in the experiments are a threat to construct validity. While there are plenty of regression metrics, they are all based on assessing the distance of predictions to the ground truth. Along this line, we picked metrics that are commonly used [26], [14]. In addition, the usage of multi-output regression led us to report average metrics for the whole set of objectives instead of individual metrics for each objective, which might have affected our results.

Regarding external validity, the results obtained for the case studies might not generalize to other optimization applications or techniques. While there are various architecture optimization approaches, the available datasets are limited, especially considering the provision of architectural models, sequences of tactics, and quality-attribute values that can be related to every architecture instance. To mitigate this threat, the chosen datasets were taken from other architecture optimization works [36], [15], involving architectural models of different sizes and covering a reasonably rich space of alternatives. We should notice, however, that the size of the datasets is smaller than what is commonly used by ML algorithms. In cases of larger datasets, the reported results might vary.

For selecting the ML techniques, we evaluated a list of well-known regression algorithms for surrogate modeling [21]. We decided to use RF and XGBoost to evaluate the *IAL* strategy due to its robustness to outliers and support for incremental training. Still, there is the possibility we might have missed other promising algorithms for the task. Further investigation of algorithms and sampling strategies is necessary, including hyper-parameter tuning, to see if the performance of the surrogate models can be improved.

V. Related Work

Surrogate-based approaches have already been used in a wide variety of research areas and applications, ranging from traditional engineering disciplines such as aerodynamics [30] and ship design [38], over natural gas liquefaction [17] to complex environmental optimization problems [39]. However, surrogates based on machine learning techniques have been scarcely used in software architecture optimization.

While not explicitly using surrogate models, many works try to leverage ML techniques to solve software (architecture) problems. For example, ML is used to optimize (low-level) code to improve performance [32], mitigate software architecture degradation [24], or assist software architects in proactively adapting systems [33]. However, none of these approaches aims to speed up software architecture optimization.

Xin et al. [41] propose to assist an evolutionary algorithm with surrogate models for performance optimization of software architectures. Random Forests [10] are employed as surrogate models to improve the overall execution time. The results indicate that the surrogate models reduce the execution time by up to 48% compared to NSGA-II and improve the quality of the population significantly. In contrast to our work, Xin et al. [41] do not consider the accuracy of the prediction in their approach and evaluation. Furthermore, they only use surrogates to predict performance, while we predict quality attributes in general.

Jamshidi et al. [25] proposed a guided sampling strategy that can be used on top of any learning mechanism to transfer or make performance extrapolations from a source model to a target model. It exploits 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. [25] aim at the configuration space an end user takes to configure a system, while this work focuses on the search space of architectural design alternatives an architect is concerned with during the software design phase.

Grebhahn et al. [21] investigated the impact of machine-learning algorithms and sampling strategies on the accuracy of performance predictions. In contrast to our work, they disregarded deep learning algorithms and focused on performance prediction and not on multi-objective optimization problems.

VI. Conclusions

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 incrementally updated as the optimization proceeds. In our experiments with two case studies, we investigated the quality of different feature representations and the accuracy of multi-output regression including an incremental learning schema suitable for the optimization problem.

The results obtained with our surrogate models were encouraging in terms of efficiency. The XGBoost regressor achieved very reasonable prediction metrics in the ranges of $[0.65 - 0.85]$ and $[0.35 - 0.55]$ for R^2 and RMSE, respectively, when using between 30% and 50% of the data. We acknowledge, however, that the accuracy of the predictions might 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 domain under consideration.

We see several promising directions for future work. We would like to expand the scope of our approach by incorporating additional 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 [40], [43] could enhance the decision-making of when to use the solver and when to use (or update) the surrogate model. We will also analyze if having individual regressors for each quality-attribute objective works better than the bulk mode implemented by RF and XGBoost in the current *IAL* strategy.

Finally, we plan to integrate our approach into selected optimization tools so as to apply the *IAL* approach in "live" optimization scenarios, in which the quality-attribute values predicted by the surrogate steers the search in different directions in the design space. The challenge here is to assess whether the surrogate models lead to an improved architecture optimization, in the sense of finding more (or better) solutions in the same time or finding equally good solutions quicker than the baselines.

References

[1] Aleti, A., Bjornander, S., Grunske, L., and Meedeniya, I. Archeopterix: An extendable tool for architecture optimization of aadl models. In *2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software* (2009), IEEE, pp. 61–71.

[2] Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., and Meedeniya, I. Software architecture optimization methods: A systematic literature review. *IEEE Trans. on Soft. Eng. 39*, 5 (2013), 658–683.

[3] Anderson, T. W., Anderson, T. W., Anderson, T. W., and Anderson, T. W. *An introduction to multivariate statistical analysis*, vol. 2. Wiley New York, 1958.

[4] Arcelli, D., Cortellessa, V., D'Emidio, M., and Di Pompeo, D. EASIER: an evolutionary approach for multi-objective software architecture refactoring. In *2018 IEEE International Conference on Software Architecture (ICSA)* (2018), IEEE, pp. 105–10509.

[5] Bachmann, F. Designing software architectures to achieve quality attribute requirements. *IEE Proceedings - Software 152* (August 2005), 153–165(12).

[6] Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*, 4 ed. Addison-Wesley Longman Publishing Co., Inc., USA, 2021.

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

[8] Bernardi, S., Merseguer, J., and Petriu, D. C. A dependability profile within MARTE. *Software & Systems Modeling 10* (2011), 313–336.

[9] Borchani, H., Varando, G., Bielza, C., and Larrañaga, P. A survey on multi-output regression. *WIREs Data Mining and Knowledge Discovery 5*, 5 (2015), 216–233.

[10] Breiman, L. Random forests. *Machine learning 45* (2001), 5–32.

[11] Chen, T., and Guestrin, C. Xgboost: A scalable tree boosting system. In *KDD* (2016), B. Krishnapuram, M. Shah, A. J. Smola, C. Aggarwal, D. Shen, and R. Rastogi, Eds., ACM, pp. 785–794.

[12] Coello, C. A. C., Lamont, G. B., and Van Veldhuizen, D. A. Evolutionary algorithms for solving multi-objective problems.

[13] Cortellessa, V., Di Marco, A., and Inverardi, P. *Model-based software performance analysis*, vol. 980. Springer, 2011.

[14] Devore, J. L. *Probability and Statistics for Engineering and the Sciences*. Cengage learning, 2011.

[15] Diaz-Pace, J. A., Vidal, S. A., Tommasel, A., Frank, S., and van Hoorn, A. Can multi-agent consensus improve quality tradeoffs in software architecture optimization? In *Anais do XXVI Congresso Ibero-Americano em Engenharia de Software* (2023), SBC, pp. 77–91.

[16] Efron, B., and Stein, C. The Jackknife Estimate of Variance. *The Annals of Statistics 9*, 3 (1981), 586 – 596.

[17] Francisco dos Santos, L., Costa, C., Caballero, J., and Ravagnani, M. Multi-objective simulation optimization via kriging surrogate models applied to natural gas liquefaction process design. *Energy 262* (09 2022), 125271.

[18] Frank, S., and van Hoorn, A. SQuAT-Vis: Visualization and interaction in software architecture optimization. In *Software Architecture* (Cham, 09 2020), Springer International Publishing, pp. 107–119.

[19] Franks, G., Al-Omari, T., Woodside, M., Das, O., and Derisavi, S. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering 35*, 2 (2008), 148–161.

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

[21] Grebhahn, A., Siegmund, N., and Apel, S. Predicting performance of software configurations: There is no silver bullet. *arXiv preprint arXiv:1911.12643* (2019).

[22] Group, O. M. UML profile for modeling and analysis of real-time and embedded systems (MARTE), 2005.

[23] Herold, S., Klus, H., Welsch, Y., Deiters, C., Rausch, A., Reussner, R., Krogmann, K., Koziolek, H., Mirandola, R., Hummel, B., et al. CoCoME-the common component modeling example. *The Common Component Modeling Example: Comparing Software Component Models* (2008), 16–53.

[24] Herold, S., Knieke, C., Schindler, M., and Rausch, A. Towards improving software architecture degradation mitigation by machine learning. In *The Twelfth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2020), Nice, France, October, 26-29 2020* (2020).

[25] Jamshidi, P., Velez, M., Kästner, C., and Siegmund, N. 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* (2018), pp. 71–82.

[26] Jiang, P., Zhou, Q., and Shao, X. *Surrogate Model-Based Engineering Design and Optimization*. Springer, 01 2020.

[27] Koziolek, A., Koziolek, H., and Reussner, R. 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* (2011), pp. 33–42.

[28] Krige, D. G. 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.

[29] Kwiatkowska, M., Norman, G., and Parker, D. Prism 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23* (2011), Springer, pp. 585–591.

[30] Liao, P., Song, W., Du, P., and Zhao, H. Multi-fidelity convolutional neural network surrogate model for aerodynamic optimization based on transfer learning. *Physics of Fluids 33* (12 2021).

[31] Lundberg, S. M., and Lee, S.-I. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774.

[32] Memeti, S., Pllana, S., Binotto, A., Kołodziej, J., and Brandic, I. Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. *Computing 101* (2019), 893–936.

[33] Muccini, H., and Vaidhyanathan, K. Archlearner: Leveraging machine-learning techniques for proactive architectural adaptation. In *Proceedings of the 13th European Conference on Software Architecture-Volume 2* (2019), pp. 38–41.

[34] Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77*, 4 (1989), 541–580.

[35] Norris, J. R. *Markov chains*. No. 2. Cambridge university press, 1998.

[36] Rago, A., Vidal, S., Diaz-Pace, J. A., Frank, S., and van Hoorn, A. Distributed quality-attribute optimization of software architectures. In *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse* (2017), pp. 1–10.

[37] Rozemberczki, B., and Sarkar, R. 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* (New York, NY, USA, 2020), CIKM '20, Association for Computing Machinery, p. 1325–1334.

[38] Saptawijaya, A. Surrogate model-based multi-objective optimization in early stages of ship design. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi) 6* (10 2022), 782–789.

[39] Tsattalios, S., Tsoukalas, I., Dimas, P., Kossieris, P., Efstratiadis, A., and Makropoulos, C. Advancing surrogate-based optimization of time-expensive environmental problems through adaptive multi-model search. *Environmental Modelling and Software* (04 2023), 105639.

[40] Wu, D., Lin, C.-T., and Huang, J. Active learning for regression using greedy sampling. *Information Sciences 474* (2019), 90–105.

[41] Xin, D., Youcong, N., Xiaobin, W., Peng, Y., and Yao, X. Surrogate model assisted multi-objective differential evolution algorithm for performance optimization at software architecture level. In *Simulated Evolution and Learning: 11th International Conference, SEAL 2017, Shenzhen, China, November 10–13, 2017, Proceedings 11* (2017), Springer, pp. 334–346.

[42] Yousefzadeh, N., Thai, M., and Ranka, S. A comprehensive survey on multilayered graph embedding. Tech. rep., 09 2023.

[43] Zuluaga, M., Krause, A., and Püschel, M. -pal: an active learning approach to the multi-objective optimization problem. *J. Mach. Learn. Res. 17*, 1 (Jan. 2016), 3619–3650.