An Adaptive Application Framework with Customizable Quality Metrics

LIU LIU, Rutgers University, USA
SIBREN ISAACMAN, Loyola University Maryland, USA
ULRICH KREMER, Rutgers University, USA

Many embedded environments require applications to produce outcomes under different, potentially changing, resource constraints. Relaxing application semantics through approximations enables trading off resource usage for outcome quality. Although quality is a highly subjective notion, previous work assumes given, fixed low-level quality metrics that often lack a strong correlation to a user's higher-level quality experience. Users may also change their minds with respect to their quality expectations depending on the resource budgets they are willing to dedicate to an execution. This motivates the need for an adaptive application framework where users provide execution budgets and a customized quality notion. This article presents a novel adaptive program graph representation that enables user-level, customizable quality based on basic quality aspects defined by application developers. Developers also define application configuration spaces, with possible customization to eliminate undesirable configurations. At runtime, the graph enables the dynamic selection of the configuration with maximal customized quality within the user-provided resource budget.

An adaptive application framework based on our novel graph representation has been implemented on Android and Linux platforms and evaluated on eight benchmark programs, four with fully customizable quality. Using custom quality instead of the default quality, users may improve their subjective quality experience value by up to 3.59×, with 1.76× on average under different resource constraints. Developers are able to exploit their application structure knowledge to define configuration spaces that are on average 68.7% smaller as compared to existing, structure-oblivious approaches. The overhead of dynamic reconfiguration averages less than 1.84% of the overall application execution time.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; • **Computer systems organization** → **Availability**; • **Software and its engineering** → **Application specific development environments**; • **General and reference** → *Evaluation*; *Performance;*

Additional Key Words and Phrases: Approximate computing, configuration management, QoS

ACM Reference format:
Liu Liu, Sibren Isaacman, and Ulrich Kremer. 2021. An Adaptive Application Framework with Customizable Quality Metrics. *ACM Trans. Des. Autom. Electron. Syst.* 27, 2, Article 13 (November 2021), 33 pages.
https://doi.org/10.1145/3477428

The presented material is based upon work supported by the National Science Foundation under Grant No. 1617551 and partially supported by the National Science Foundation under Grant No. 1730043.
Authors' addresses: L. Liu and U. Kremer, Rutgers University, 110 Frelinghuysen Rd, Piscataway, NJ 08854; emails: ll557@scarletmail.rutgers.edu, uli@cs.rutgers.edu; S. Isaacman, Loyola University Maryland, 4501 N Charles St, Baltimore, MD 21210; email: snisaacman@loyola.edu.

1084-4309/2021/11-ART13 $15.00
https://doi.org/10.1145/3477428

1 INTRODUCTION

Many applications can adapt mid-execution to give outcomes of different quality with different resource demands. Such adaptive applications allow optimizations in which resource demands are evaluated relative to outcome quality. Optimization may allow successful execution in resource-constrained environments with maximized quality [25, 42] or satisfy minimal quality requirements at minimized cost. Adaptive applications include navigation systems, video/audio players [90, 100], object recognition, face detection, and machine learning [74]. Management of these applications may require dynamic strategies due to the uncertainty of execution environments, particularly regarding available resources.

Adaptive frameworks, also referred to as approximation frameworks, manage program configurations that are mappings of selected application program variables ("knobs") to specific values in their program-defined value ranges. Changing knob values impacts the cost and quality of the produced application outcomes. Configuration management attempts to find a configuration that results in a desired tradeoff between cost and outcome quality. In this article, we introduce an adaptive framework that focuses on maximizing output quality for a given cost budget. Dynamic reconfiguration is implemented by changing knob values during program execution.

A key design decision for any configuration management framework is its choice of problem representation. The representation must be expressive enough to encode all feasible configuration choices and allow efficient algorithms to rank these choices according to different metrics such as execution time, energy consumption, and overall outcome quality. Applications have to be mapped effectively to this problem representation, and metrics have to be expressed and computed as part of or based on the representation. Any selected configuration choice has to be mapped back into applications and their executions. For dynamic optimizations, i.e., optimizations that are performed during application execution, the efficiency of the representation in terms of construction, enabling selection, and final "code generation" becomes an even more important concern as compared to purely static optimizations.

Our work recognizes that effectively dealing with the inherent complexity of managing adaptive applications requires the identification and exploitation of structure, both in the application representation and in the representation of its configuration space. This structure serves as the foundation for scalable, effective, and efficient solutions during different phases of application development, user-defined application execution quality, and adaptive application execution.

The **Knob Dependence Graph** (KDG) is a representation of the configuration space of an application and is used for offline cost/quality model construction and online optimization. Based on the KDG, we designed and implemented Rapids,[1] a new framework that provides customization for both the developer and user. It is efficient (low overhead), portable (short retraining times), and effective (optimal configuration selection). The contributions of this work are:

- The KDG, a graph representing each knob's type, setting range (discrete or continuous), and inter-knob dependencies. This is a compact, flexible way to express the adaptability of these applications.
- A mechanism within the KDG for placeholders for a linear combination of basic quality metrics. Users can use these to provide custom, high-level quality notions that can be traded off during runtime.
- A mechanism for fast and accurate retraining for the weights in the KDG. This makes the system feasible for real deployment across platforms.
- The evaluation of the effectiveness of the KDG based approach within the Rapids adaptive program development and execution framework.

[1]**R**econfiguration, **A**pproximation, **P**references, **I**mplementation, **D**ependencies, and **S**tructure.

The evaluation of Rapids is based on a benchmark of eight applications running on Linux and Android systems. Four of these applications have fully customizable quality notions. Customized quality results in significantly improved user-preferred quality values with 1.76× improvement on average across a range of user-supplied resource budgets, up to 3.59×. An initial user study indicates that higher-level quality metrics can significantly (5×) reduce a user's effort to find a desired quality outcome as compared to using low-level metrics and knobs.

Benchmark applications show an average configuration space reduction of 68.7%. Our two retraining strategies further reduce the configuration space and result in a training time reduction of 87.2% or 92.4% compared to state-of-the-art approaches, while maintaining cost prediction errors of less than 2.5%. The overhead of dynamic reconfiguration for execution time or energy consumption remained below 3.2%, with 1.84% on average.

2 KEY FRAMEWORK DESIGN CHALLENGES

Most existing approaches for adaptive configuration management target applications that have been written without reconfiguration in mind. Automatic techniques identify program variables as "knobs," i.e., entities that, when changed, impact the quality and cost of the application's outcome. This is similar in spirit to the "dusty deck" approach to automatic parallelization and vectorization, which has been only partially successful [52, 99, 103]. Dusty deck programs are legacy programs that are ported to new platforms with the expectation of performance improvements without significant source-code modifications. Prior approaches [47, 70, 96] use automated configuration identification to minimize application developers' effort. However, a "dusty deck" approach does not allow the program developer to express insights into relevant structures and behaviors of an application.

Alternative, approximation-aware approaches use programming language extensions or runtime libraries for configuration definition and management, making approximation part of the semantics of the program. The application developer is responsible for writing code to implement configuration or quality management. There are many language/library-based systems (e.g., [6, 16, 72, 86]) with different abstractions to manage configurations and quality. However, these systems have only limited, if any, support for reconfiguration. Most importantly, the language semantics approach relies solely on the developer to hardcode a configuration management strategy, a challenging and potentially error-prone task. In this section, we discuss key challenges for adaptive application frameworks and motivate the Rapids framework design decisions to address these challenges using a new graph representation of the configuration space, the KDG.

2.1 Managing Large Configuration Spaces

As observed by many projects that address the problem of selecting the best configuration for a particular requirement, a key problem is the size of the configuration space. The configuration space is defined by the number of program variables or program parameters that have impact on the quality of the program's outcome (application program knobs), as well as their particular value ranges. Even a single floating-point valued knob with a bounded value range can conceptually result in an intractably large configuration space, or two knobs with large integer valued ranges may yield configuration spaces that cannot be exhaustively explored in practice.

Large Knob Value Ranges: A sampling strategy may be a solution to this problem. However, although keeping the configuration space at a manageable size, sampling may miss the best knob value settings, resulting in suboptimal outcomes. In addition, a sampling-based solution may only contain configurations that are optimal for a higher or lower resource budget than specified for the execution. In order to fully utilize the specified budget, the execution will need to reconfigure between these two configurations, which is not needed under an optimal configuration that matches the resource budget.

Our Rapids framework allows continuous knob value ranges in addition to discrete ranges. All of our eight benchmark programs have a combination of continuous and discrete knobs as specified by the application developer.

Large Number of Knobs: Even if knobs are integer ranged with only a limited number of settings, the cartesian product of these value ranges can yield impractically large search spaces. The proposed solution for search space management due to the number of knobs is a sensitivity analysis (e.g., [75, 76]) that eliminates knobs that are considered too expensive to change during reconfiguration or only minimally impact the overall quality outcome of the program execution. Therefore, these knobs are excluded from the configuration search space. Our framework based on the KDG allows program developers to specify which knobs to use. Our current Rapids implementation does not perform any automatic knob sensitivity analysis but could be extended to do so.

Dependencies among Knobs and Their Value Ranges: Even with a compact representation of the value ranges of only the most resource- or quality-critical knobs, the search space may still be too large or may allow undesirable knob value combinations. Undesirable knob value combinations may lead to program crashes [47] or may lead to program outcomes with unacceptable user-level quality. For example, in our NavApp benchmark application (see Figure 5), a dark screen showing a detailed map is not useful since the detailed information is not accessible to the user. Our KDG-based framework allows different dependencies among knobs and their value ranges to be expressed explicitly, giving significant flexibility to application developers to manage the configuration search space. This includes customization of the search space for a particular target application user group.

2.2 Customization of User-Level Quality

Existing approaches assume some pre-defined quality notion where each application "comes with" a quality function for its output, e.g., PSNR [100] or SSIM [97] for video processing. However, many applications produce outcomes that have different quality aspects, which we refer to as quality sub-metrics. A family of quality notions can be defined as a weighted combination of these quality sub-metrics. For example, a navigation application may consider a brighter screen, a more detailed map, or a more precise localization service as a desired outcome with the highest quality and best user experience. Therefore, such applications have a highest-quality configuration. The lowest-quality configuration can be defined as the configuration that provides the lowest individual qualities of the different quality aspects. Both the highest- and lowest-quality configurations have a particular resource need, which we refer to as *max* and *min*, respectively. Executing an application with resource budgets greater than *max* will not improve the quality outcome of the application, and assigning a resource budget lower than *min* will result in an unacceptable execution since the lowest-quality outcome cannot be achieved. For any execution budget $user\_budget$, $min \leq user\_budget \leq max$, the best configurations with respect to the individual quality sub-metrics form a Pareto-optimal solution. While the application developer can specify quality sub-metrics, user-level quality customization is needed to select the best configuration within the Pareto-optimal solution space.

In our Rapids framework, users can customize quality notions by providing preferences among the quality sub-metrics. For example, some user may prefer a brighter screen over a more precise localization service, while another user is mostly interested in a detailed map, with the same importance given to the screen brightness and localization accuracy. This quality customization has to be performed just in time before application execution, since it may depend on a particular user need for a specific execution with a specific resource budget.

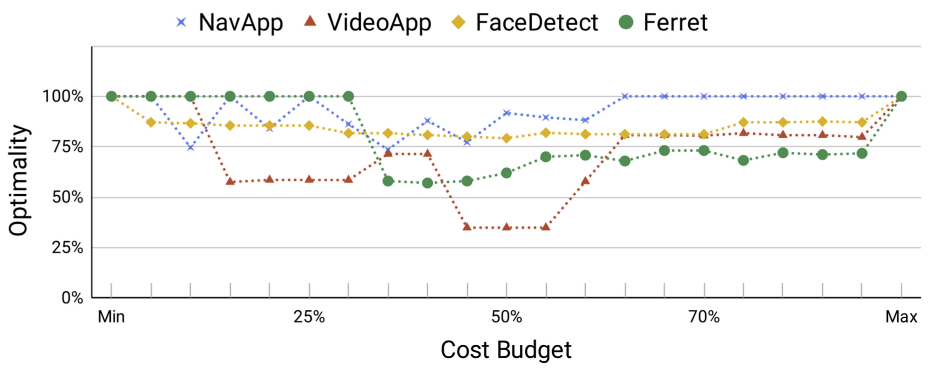


Fig. 1. Maximal quality loss of default configuration *opt* relative to optimal custom quality configuration $\tilde{opt}_i$ across different budgets. X-axis: Budget percentage across *min* and *max* range.

Benefits of Customized Quality: Previous work assumes a single quality model distributed with the application, typically provided by the application developer. We refer to this model as the default quality model. The benefit of a customized quality model has to be assessed relative to this default model. Every model produces a ranking among the possible configurations induced by a numerical value assigned to each configuration. Clearly, this numerical quality value may not directly reflect the subjective quality experience of the end-user. Only a comprehensive user study would be able to establish such a correlation, which is beyond the scope of this article. However, the different rankings among configurations induced by the customized quality models indicate a rich space of quality notions that users are able to express and explore.

In order to allow an initial benefit analysis of customized quality, we defined a family of customized models $\tilde{Q}_i$ where each sub-metric of the default model is given a preference value or weight between 1.0 and 2.0, with increments of 0.1, resulting in 11 possible values. The default model Q is the model where each sub-metric has the preference value of 1.0 with *opt* as its optimal configuration. For example, a default model with two sub-metrics has a family of customized models $\tilde{Q}_i$, $1 \leq i \leq 121$, with the pair of preferences (1.0, 1.0) representing the default model. Note that the preference range of [1.0, 2.0] has been chosen for illustration purposes only. Rapids can support any desired preference value ranges for sub-metrics as defined by the application developer. The intent here is to show that even using custom quality models where the importance of a single sub-metric can at most be doubled as compared to the default model, an optimal customized quality configuration $\tilde{opt}_i$ may have a significantly higher custom quality value than the custom quality value of the optimal default configuration *opt*. In other words, this initial benefit analysis captures how much quality value may be lost if *opt* is used instead of $\tilde{opt}_i$ for a user-customized model $\tilde{Q}_i$.

For four applications with their resource budgets between *min* and *max*, Figure 1 shows the maximal loss in quality value of *opt* under the different possible custom quality models $\tilde{Q}_i$ as compared to custom quality configurations $\tilde{opt}_i$, with $\tilde{Q}_i(opt) \leq \tilde{Q}_i(\tilde{opt}_i)$. This relative magnitude of the quality loss is captured by the following equation:

$$optimality\_loss = 1.0 - \max_i(\tilde{Q}_i(opt)/\tilde{Q}_i(\tilde{opt}_i)). \tag{1}$$

The results show that there is a substantial quality difference due to customization across all four sample applications, in particular under intermediate budgets (30% to 60% of *max* budget) where there are larger Pareto-optimal solution configuration spaces. This indicates that custom quality models are an important tool to allow application users to express their subjective quality preferences.

Quality Sub-metrics: A key challenge for any system that requires user input is to provide enough information and at a level of abstraction that enables the user to make an informed decision. Exposing the program knobs to the end user is therefore not a viable strategy since it would require a deep understanding of the inner workings of the application, which typically only the developer has.

However, the application developer can provide quality sub-metrics or quality dimensions from which an end-user may choose to determine his/her best quality experience. These basic quality notions may not directly map to specific knobs, but may involve more complex, combined knob mappings. This complexity should be hidden from the application user to allow reasoning about quality vs. resource tradeoffs at a level of abstraction that makes sense to the user.

2.3 Direct Problem Formulation vs. Control-Theoretical Approach

Current approaches [45, 47, 48, 89, 90] establish a correlation between configurations and an application's resource cost (performance or energy) and quality outcome through a profiling (training) phase, where all or randomly sub-sampled [70] configurations are executed on the target machine, and their costs and quality are measured and recorded. Based on this training information, reconfiguration is implemented using a control-theoretical approach that adjusts the application's configuration. Control-theoretical approaches (e.g., PID [44]) mainly target applications with dynamic behaviors under continuous operations where configurations are selected based on immediate quality notions and system observations. There is no overall execution resource/cost budget. Carefully designed models with their model parameters allow control-theoretical guarantees, for example, with respect to specific convergence behaviors, but may result in sub-optimal overall configuration selection due to locally optimal solutions [70].

Our Rapids framework targets applications where there is an a priori knowledge about the work that has to be accomplished under a user-provided resource budget. Such problems lend themselves to be formulated as constrained optimization problems where optimal quality solutions can be determined directly under a budget constraint. At any point during program execution, the optimal configuration selection will depend on the remaining work that has to be performed under the remaining resource budget. Uncertainty is introduced due to input dependencies and cost prediction inaccuracies. The goal of the configuration selection is to find a configuration that can successfully finish the execution, yielding the highest output quality while fully utilizing, but not exceeding, the remaining resource budget.

Our Rapids framework employs machine learning strategies to build cost and quality models and uses a quadratic integer programming formulation to compute the highest-quality configuration under the remaining cost budget. Adaptive reconfiguration is performed if the actual and observed resource consumption cross a pre-defined threshold.

2.4 Effective Model Training and Porting

The cost of training is proportional to the size of the configuration space, which may lead to significant training times in practice. For example, [36] reports profiling times of more than 2 weeks for an evaluated application. Further, the ability to port the configuration space management system to new hardware/software platforms is important, but new hardware/software configurations may not be known a priori. The result may be significant porting costs. Current work does not provide effective solutions to this problem.

Our new approach uses structural information to significantly reduce the number of configurations needed to build the cost and quality models for the entire configuration space. This subset of configurations is called the "representative set." Previous work [70] determines the representative set using random sampling of the configuration space, resulting in a fixed number

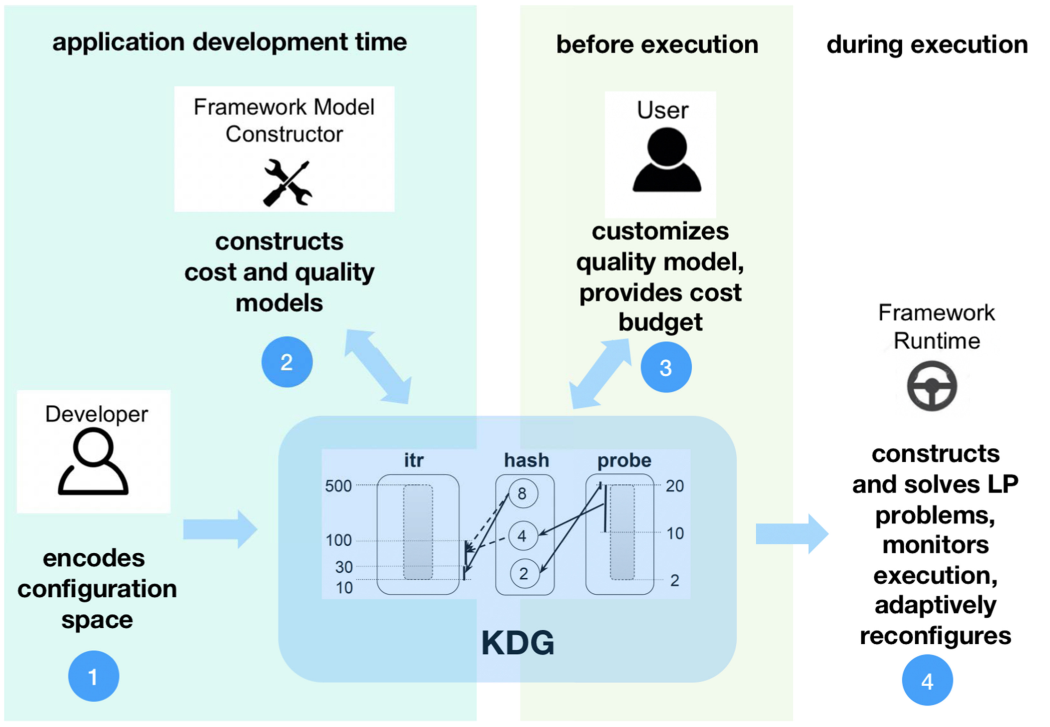

application development time
before execution
during execution
Framework Model Constructor
constructs cost and quality models
2
User
customizes quality model, provides cost budget
3
Framework Runtime
constructs and solves LP problems, monitors execution, adaptively reconfigures
4
Developer
encodes configuration space
1
itr
hash
probe
500
100
30
10
8
4
2
20
10
2
KDG

Fig. 2. High-level workflow of Rapids framework with phases 1 through 4 interacting with the KDG representation.

of configurations. In our Rapids framework, the structural information in the KDG is used to automatically compute a small representative set that allows a model to be automatically reconstructed with a developer-provided error bound.

2.5 The Knob Dependence Graph (KDG) within the Rapids Framework Workflow

The KDG is the central data structure and key new design element in our Rapids framework. It lets application developers specify the desired configuration space, supports automatic cost and quality model construction using machine learning, allows developers to define sub-metrics to enable user-level quality model specification, and serves as the representation from which quadratic integer programming problem instances are automatically generated at runtime to select the optimal configuration under a user-provided cost budget.

A high-level view of the application development and execution workflow is shown in Figure 2. The application developer, Rapids model constructor, application user, and Rapids runtime have their distinct roles with the KDG as their main interface. A detailed discussion of the Rapids framework and its workflow is presented in Section 5.

3 THE KDG - A NEW ADAPTIVE APPLICATION REPRESENTATION

The KDG is a directed graph encoding a developer's insights about the structure of the application through the graph's nodes and edges. The KDG is a representation of the configuration space of an application and is used for offline cost/quality model construction and online optimization. It provides the basis to formulate configuration selection as a constrained optimization problem.

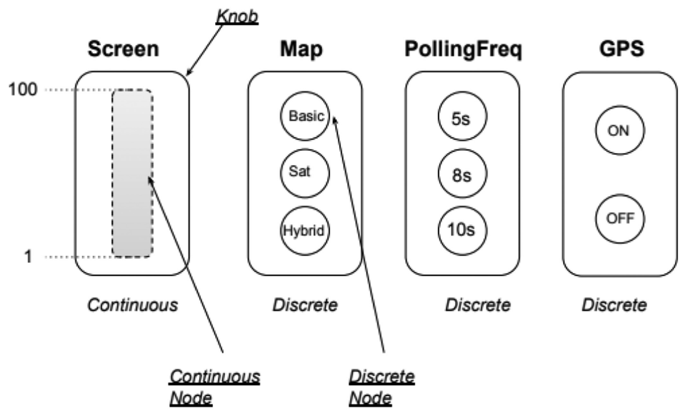


Fig. 3. KDG nodes in NavApp.

3.1 Developers' Insight as Structure

Rather than representing the configurable components (knobs) as a vector, the KDG allows developers to encode information including knob type, per-knob value range, and inter-knob dependency through the graph structure. We use one of our benchmark applications NavApp as an example to illustrate the KDG.

NavApp is a navigation application we designed that runs on the Android platform. The application has four configurable components: (1) **Screen:** controls the brightness of the screen from 1% to 100%; (2) **Map:** controls the map display layout with three options: "basic," "satellite," and "hybrid"; (3) **PollingFreq:** controls the polling frequency in seconds of location with three options: "5s," "8s," and "10s"; and (4) **GPS:** controls whether to use on-board the GPS module or not.

Developer's Insight: To ensure the readability of the map, developers may enforce some constraints: (1) the screen should be brighter if the layout is set to satellite or hybrid because the backgrounds on satellite images are darker and therefore hard or even impossible to interpret on a dark screen, and (2) a more accurate localization is required for users to interpret their current location if a basic layout is rendered because there are fewer landmark references available to users to orient themselves. These constraints reflect the developer's design and assessment of a desirable constrained configuration space based on anticipated user needs and expectations. In current configuration management approaches, such constraints can only be implemented through conditional statements embedded and distributed across the source code. We use the NavApp application to illustrate how developers can use the KDG to encode the constrained configuration space.

Nodes represent knobs and their settings. The KDG supports two types of nodes: Discrete and Continuous. Each knob consists of a collection of *Discrete* nodes or a single *Continuous* node. Each Discrete node within a knob is associated with a specific value setting. A Continuous node represents a possible value range of a setting. Figure 3 shows the knobs in NavApp. There are four knobs: Screen, Map, PollingFreq, and GPS.

Edges represent dependencies between knob settings. Edges are directed with the sink depending on the source. Edges thus encode developers' insight into inter-component dependencies. They enable developers to further manage the configuration space and its size in order to tune system performance or to allow only desirable user experiences. For discrete nodes, dependencies are on

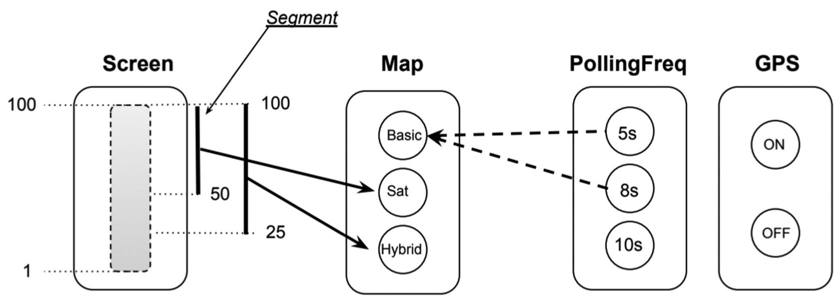


Fig. 4. KDG nodes with edges in NavApp.

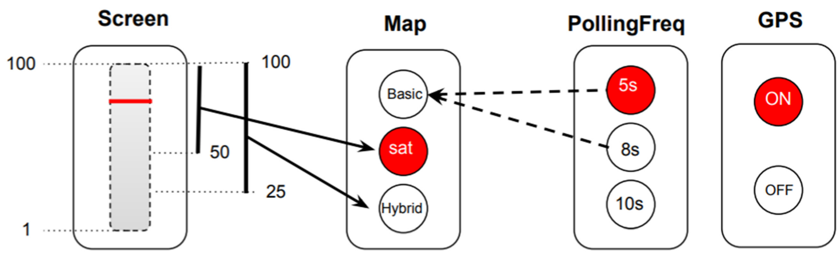


Fig. 5. KDG configuration in NavApp.

the entire node. For a continuous node, a dependency may be specified on only a range of possible values. Such a range is referred to as a *segment*.

The KDG models two types of dependencies, *AND* and *OR*. *AND* dependencies allow a node to be dependent on a set of different nodes that are *all* needed to satisfy the dependence. In contrast, *OR* dependencies allow a node to require *at least one* node in a knob or set of knobs. That is, *OR* dependencies are grouped: at least one source node has to be selected from each *OR* group. A node may *AND*-depend on a set of independent *OR* groups.

In Figure 4, if a satellite image is rendered as the map layout, the screen brightness must be at least 50%. Similarly, "Hybrid" requires at least 25% brightness to ensure readability of additional highlighted road names and information overlays on the satellite image. "Basic" has *OR* dependencies on 5s, 8s in PollingFreq.

Configurations are represented in the KDG as a selection of nodes or values. For knobs with Discrete nodes, a single node will be selected, and for knobs with a Continuous node, a specific value will be selected. Figure 5 demonstrates a particular configuration. The KDG is a compact representation of the entire configuration space where each configuration has to satisfy the edge dependencies. Figure 5 shows a possible configuration of *Screen=75, Map="Sat," PollingFreq="5s," and GPS="ON."*

3.2 Cost Model

The KDG provided by the developer contains only structural information. A full KDG includes the structure, weights (both for cost and quality models), and user preference (as weight augmentation). A training phase is performed to collect a set of data points that map configurations to corresponding costs that are used to build a regression model that in turn provides weights to the KDG. The overall cost C of a specific configuration is calculated as the sum of the contribution from each individual discrete C^c_{dis} knob, each individual continuous C^c_{cont} knob, and the contribution of the combined effects of each pair of knobs C^c_{corr}. We use a second-order linear regression model to derive the first- and second-order parameter values (weights) for discrete and continuous nodes and the pair-wise knob coefficients.

$$C = C^c_{dis} + C^c_{cont} + C^c_{corr} \tag{2}$$

Node Weights in the KDG represent the contribution of a particular node to the overall cost. For a *discrete* node, the weights for each node j in a knob i is represented by single value c^j_i. For a *continuous* node, the weight for a knob i with value range $[min_i, max_i]$ is represented by a function $F^i_c()$ that maps a value v_i within the range to its contribution. Equations (3) and (4) show the cost contribution of the discrete and continuous nodes, respectively. Only the selected discrete nodes ($v^j_i = 1$) will contribute to the overall cost.

$$C^c_{dis} = \sum_i \sum_j c^j_i \times v^j_i, v^j_i \in 0, 1 \tag{3}$$

$$C^c_{cont} = \sum_i F^i_c(v_i), min_i <= v_i <= max_i \tag{4}$$

Correlated Weights model the combined contribution of pairs of knobs. This design captures knob correlations that are more complex than simple addition. For example, in a nested loop with the two knobs representing the loop bounds, the total cost is proportional to $\#outer\_iteration *$ $\#inner\_iteration$. This approach can also represent situations where loop structures are nested deeper than two levels with knobs used as bounds in at most two of the levels. Experimentally, we found that modeling quadratic relations between knobs was sufficient to capture program behaviors. Note that a numerical value $\bar{v}$ is required for discrete nodes with categorical values.

$$C_{corr} = \sum_m \sum_n corr^n_m \times (\bar{v}_m \times \bar{v}_n) \tag{5}$$

3.3 Custom Quality Metrics/Models and Virtual Knobs

Quality and cost metrics rank outcomes of application executions under different configurations. A Quality metric or sub-metric Q *measures* an aspect of observed application outcomes. A model $\tilde{Q}$ for a metric or sub-metric Q *predicts* the expected measurements for a configuration without applying the metric to the observed outcome. In other words, a model $\tilde{Q}$ approximates the measured metric Q for a configuration.

If an application has a single, fixed quality metric, for example, PSNR [100] or SSIM [97] for video playback, the quality model construction process will be identical to the cost model but with cost replaced by the measured quality metric value. However, in general, quality is a subjective metric and therefore needs application users' involvement. If an application has at least two distinct quality sub-metrics, i.e., two distinct ways to rank configuration quality, the application has "customizable quality." These sub-metrics should be easily reasoned about by the application users and may have different importance for different target users. In the presence of customizable quality, two distinct quality preferences may result in different configuration selections under the same budget, each maximizing the distinct subjective overall quality.

Definition. Custom Quality using Sub-Metrics — An application has custom quality if the quality Q can be represented by a function F^Q over several weighted quality metrics $(q_1, \ldots q_n)$, $n \geq 2$. The quality metrics q_1 to q_n are referred to as sub-metrics. Users can provide relative preferences on each sub-metric through preference weights before application execution.

$$Q = F^Q([(w_1, q_1), \ldots (w_n, q_n)]) \tag{6}$$

Table 1. Knobs and Sub-metrics in FaceDetect

Configurable Knobs	Sub-metrics
Neighbour_Pixel, Decomposition_Level, Eye_Detection_Enabled	Precision, Recall

Example. We use another application, FaceDetect, as an example to illustrate the custom quality because the quality metric in FaceDetect is a typical use case where quality is rated differently by different users. In FaceDetect and other classification problems, *F-Score* is widely used as the overall quality metric. F-score represents a family of customized metrics based on sub-metrics precision p and recall r. Users may express the relative importance of sub-metrics p or r by providing different weights w_p for precision and w_r for recall in Equation (7).

$$\begin{aligned} Q_{face} &= F^{Q}_{face}([w_p, p], [w_r, r]) = (1 + \beta^2) \cdot \frac{p \cdot r}{(\beta^2 \cdot p) + r}, \\ with\ \beta &= \frac{w_r}{w_p} \end{aligned} \tag{7}$$

The two high-level sub-metrics "Precision" and "Recall" can be easily reasoned about by application users. Given the same output result, these two sub-metrics will have the same values since they have a fixed evaluation strategy. However, the overall quality might be different if users provide different weights to the sub-metrics.

Custom Quality Models: For single quality metrics Q, the quality model $\tilde{Q}$ can be constructed offline similar to the cost model as discussed in Section 3.2. Each configuration measurement collected from training now includes the measured quality metric in addition to the measured cost metric. The resulting quality model has coefficients for single knobs and pairs of knobs.

Rapids supports customizable quality metrics by applying default weights to sub-metrics at development time, allowing the user to specify sub-metric weights at execution time. For each sub-metric q_i, Rapids builds a model $\tilde{q}_i$ through training in its "offline" phase. Table 1 shows the set of configurable knobs (left column) used to implement models for the recall and precision sub-metrics (right column) for the example FaceDetect application.

Just before application execution, a user may customize his/her quality expectation metric Q by providing weights w_i for each sub-metric q_i as shown in Equation (6). Thus, Rapids must compute the quality model $\tilde{Q}$ "online" based on the known sub-metric models $\tilde{q}_i$ and the user-supplied weights w_i. Unfortunately, there is often no obvious way to use the set of knobs and pairs of knobs coefficients of the individual sub-metric models $\tilde{q}_i$ to effectively compute $\tilde{Q}$. One approach is to express $\tilde{Q}$ as a mapping over weighted individual knob coefficients as illustrated in Equation (8).

$$\tilde{Q}([k_1, \ldots k_n]) => \tilde{Q}([w_1 * k_1, \ldots w_n * k_n]) \tag{8}$$

However, this straightforward extension requires the user to know how to tune the weights on the knobs, which are program variables defined as part of an application's implementation (e.g., #_Neighbour_Pixel, #_Decomposition_Level, and Eye_Detection in FaceDetect). These are application implementation details that typically are non-intuitive to the user and hard to understand. Therefore, a user will not be able to appreciate how these implementation-related knobs will affect the overall quality, making an informed decision about the preference weights impossible. Another approach would be to re-evaluate the execution output with the new Q and reconstruct $\tilde{Q}$. However, this requires the retaining of outputs/results obtained during training and will result in significant space and time overheads. Finally, a full retraining is also a possibility. However, this approach is prohibitively expensive in terms of execution time.

Instead, Rapids predicts all sub-metrics for each trained configuration in the training phase and calculates the overall quality using the developer-provided function (F^Q in Equation (6)) with the quality models $\tilde{q}_i$ instead of the quality metrics q_i. This is more efficient since q_i requires actual application execution results, while $\tilde{q}_i$ only needs the configurations.

The set of calculated quality values yields the overall quality model $\tilde{Q}$ by solving the regression problem as discussed above and described in more detail in Section 4.1. This approach eliminates the overhead of backing up execution results and the re-evaluation process. Experiments show that the overhead of dynamically constructing quality models is negligible, less than half a second for all four of the applications we evaluated with customizable quality. This overhead occurs once, just before application execution.

Virtual Knobs: As discussed above, nodes in the KDG are associated with application-level knobs or objects (e.g., program variables) and their possible value settings, which together define the configuration space. Both cost and quality metrics are defined over this "concrete" configuration space. However, in order to allow application users to customize their quality experience, the knobs of a concrete configuration may be too low level to allow users to make an informed choice. Therefore, Rapids introduces a set of higher-level, "virtual" knobs for the sole purpose of allowing users to reason and manage their quality expectations. The key here is that now users can fine-tune the quality on the level of sub-metrics instead of concrete knob settings and their low-level quality notions.

Each virtual knob corresponds to a specific sub-metric. Virtual knobs are not explicitly represented in the KDG as nodes. Application developers can define and selectively expose these metrics as virtual knobs to users, who in turn will provide relative preferences among the exposed sub-metrics. The idea of virtual knobs is similar to an interface between the users and the KDG. Unlike the configurable knobs in the KDG, virtual knobs do not have quality/cost weights. In the FaceDetect example, users can tune the two virtual knobs, "precision" and "recall," and the Rapids will automatically update the customized quality model accordingly.

4 THE KDG - ENABLING ADAPTIVE APPLICATION MANAGEMENT

The KDG enables solutions to four key problems in adaptive application management: (1) deriving the cost and quality weights that define the models at application development time, (2) efficient model construction at application development and deployment time, (3) calculating the optimal configuration to maximize quality under a cost budget at application execution time, and (4) supporting user-level customizable quality notions just in time before application execution. This final problem was addressed in Section 3.3. The solutions to the other problems are discussed below.

4.1 Weight Derivation for Model Construction

To define cost and quality models, weights are needed for discrete (c_i^j in Equation (3)) and continuous ($F^i()$ in Equation (4)) knobs and their pairwise correlations ($corr_m^n$ in Equation (5)). To compute models with continuous knobs and a desired accuracy, continuous knobs are "discretized," i.e., partitioned into segments, where each segment is represented by its own linear function that maps its knob value range to cost/quality weights (a piecewise linear functions approach). Rapids employs second-order linear regression training to compute the weights for its knobs, coefficients of linear cost functions for knob segments, and correlation parameters based on the observed costs and quality of the corresponding exhaustive configuration space. The exhaustive space consists of all discrete knob settings and continuous knob segments. Rapids trains the application through a number of configurations each with the same training input and records the observed average for each configuration. The regression tries to minimize the error between the predicted and observed cost/quality.

Note that a continuous node is represented as a single variable with its value chosen from the specified range. Rapids uses a piece-wise linear approach to provide finer granularity. To demonstrate this method, consider m breaking points in the range (excluding the upper and lower bound) in a knob. The knob is partitioned to $m + 1$ segments. The regression process calculates a separate set of parameter values for each segment. The breaking points are determined by the configuration being observed.

4.2 Training Sets for Model Reconstruction

The size of the configuration space is exponential in the number of nodes for discrete knobs and number of segments for continuous knobs. In Section 4.1 above, we describe how to derive a discretized, exhaustive configuration space together with accurate cost and quality models represented as weights of the KDG. When porting an application to a new target hardware/software architecture, the model construction process discussed in Section 4.1 needs to be performed again for the new platform. This can be rather expensive. To significantly reduce the cost of retraining the models, Rapids introduces the notion of a ***Representative Set*** **(*RS*)**, a subset of configurations that is sufficient to accurately reconstruct the entire cost/quality model at the potential cost of a slight accuracy loss.

During the construction of the full models, the computation of the representative set is also performed offline on the development platform. The *RS* computation is based on the configuration space of the model construction and its recorded, measured observations for each configuration. This is considered the ground truth. Rapids implements two different *RS* construction methods. Our experiments show that these two strategies are both effective in significantly reducing the training set sizes for our benchmark applications.

Partition-based RS: For each knob, Rapids first considers only its highest and lowest settings. *RS* is initialized with these configurations. Rapids evaluates the prediction accuracy of the model constructed from these configurations against the ground truth. If the developer-defined error bound ϵ is not satisfied, Rapids partitions each knob by a factor of 2, i.e., adding one more setting in the middle of two selected settings per knob. This process iterates until the error bound ϵ is satisfied or a pre-defined maximum partition level is reached.

Selection-based RS: Rapids initializes *RS* with only two configurations, the most and least expensive configuration. Subsequently, Rapids iterates through all unselected configurations and constructs the model using that configuration and the current *RS*. The one configuration that yields the highest prediction accuracy will be added to the *RS*. The termination condition is the same as above.

4.3 Selection Problem Formulation

Given a fully weighted KDG and a user-provided budget at application execution time, Rapids computes the optimal solution by solving a **Mixed-Integer-Quadratic-Constrained-Programming (MIQCP)** [22] as specified by Equation (9). The MIQCP formulation consists of an objective function and several classes of constraints. The constraints enforce the validity of the computed solution. For example, any solution has to respect knob dependencies, can only select a single setting per knob, and has to respect the resource/cost budget.

$$Maximize : \sum_{i} \left(C^{q}_{dis(i)} + C^{q}_{cont(i)}\right) + \sum_{m,n} C^{q}_{corr(m,n)} \tag{9a}$$

$$s.t. \sum_{i} \left(C^{c}_{dis(i)} + C^{c}_{cont(i)}\right) + \sum_{m,n} C^{c}_{corr(m,n)} \leq B \tag{9b}$$

$$\forall AND(s_o - > s_i),\ s_i - s_o \leq 0\ //edges \tag{9c}$$

$$\forall OR(so_1, so_2, \ldots so_n - > si),\ si - \sum_{i} so_i \leq 0 \tag{9d}$$

$$\forall N^{j}_{i}, N^{j}_{i} = 1 \rightarrow v_i = v\left(N^{j}_{i}\right)\ //node \tag{9e}$$

$$\forall S^{j}_{i}, S^{j}_{i} = 1 \rightarrow min_{S^{j}_{i}} \leq v_i \leq max_{S^{j}_{i}}\ //segment \tag{9f}$$

$$\forall i, j, \sum_{j} S^{j}_{i} \leq 1\ //single\ node\ per\ knob \tag{9g}$$

$$\forall i, j, \sum_{j} N^{j}_{i} \leq 1\ //single\ seg\ per\ knob \tag{9h}$$

The optimization problem is formed as in Equation (9), where C^q is calculated with the same approach as cost by solving the weight derivation problem but with customized overall quality based on user-provided relative priorities of quality sub-metrics as discussed in Section 3.3. The second line (Line 9b) ensures that the overall cost does not exceed the provided budget B. The third and fourth lines (Line 9c, 9d) show constraints for *AND* and *OR* edges. The rest encodes the knob values and selection constraints. Rapids solves the optimization problem using the off-the-shelf solver *gurobi* [93].

In this article, we focus on the problem of maximizing the quality output given a user-provided budget. However, the reverse problem, i.e., selecting the configuration with the lowest cost given a lower bound on the quality, could also be solved by our framework by swapping the problem objective on *quality* (Line 9a) with the *cost* constraint (Line 9b) and changing the new objective function to minimizing *cost* and constraining the *quality* to be ≤ *provided quality loss*. However, a discussion of this Rapids capability is beyond the scope of this article.

5 RAPIDS - A KDG-BASED FRAMEWORK FOR ADAPTIVE APPLICATIONS

Rapids provides an end-to-end framework to write and execute reconfigurable applications with the KDG as its key representation. Rapids's workflow consists of four main phases as shown in the high-level Rapids overview in Figure 2 in Section 2.5: (1) application specification and implementation (done by the *Developer*), (2) automatic training and modeling (done by the *Model Constructor*), (3) custom quality model specification and construction and budget specification (done by the *User*), and (4) runtime monitoring and reconfiguration (done by the *Runtime Framework*). The structure of our Rapids framework from the system implementer's point of view is shown in Figure 6.

In the following, we use Ferret as an example application to illustrate the effort required from the developers. Ferret is an image similarity query application returning the top-K similar images for a query image using a Multi-Probe LSH [63] algorithm. Then, it calls a routine that computes the Earth Mover's Distance [84] to rank the images and return the top-K. There are three knobs in Ferret: *hash*: discrete number of hash buckets per table within {2, 4, 8}; *probe*: probing buckets in the multi-probe phase from [2, 20]; and *itr*: maximum iterations from [10, 500]. Through dependencies, we enforce that fewer hash buckets require more probing buckets since more have

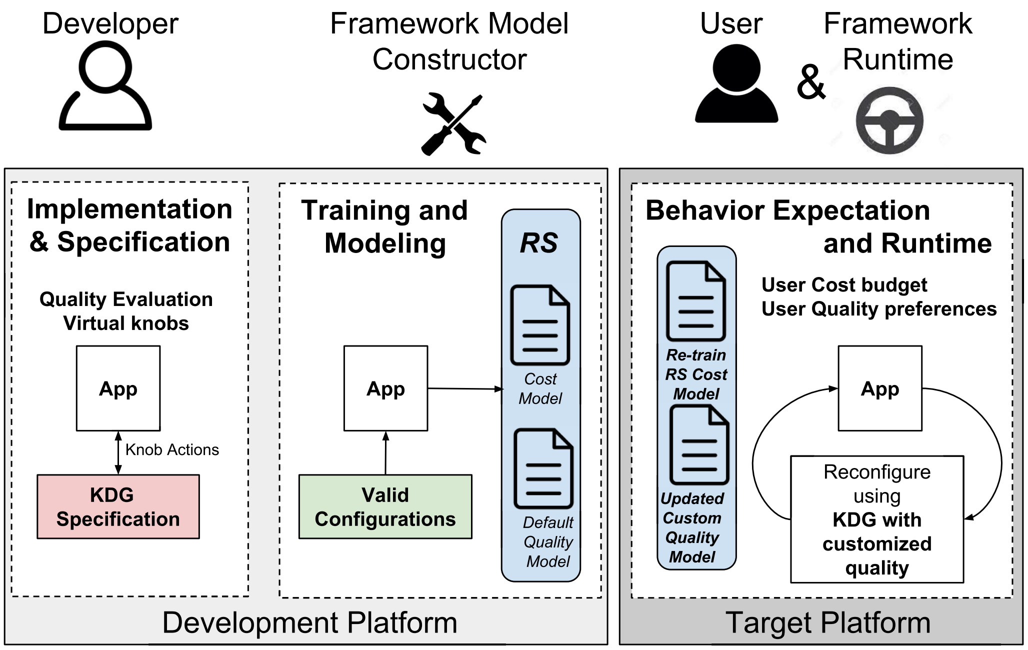


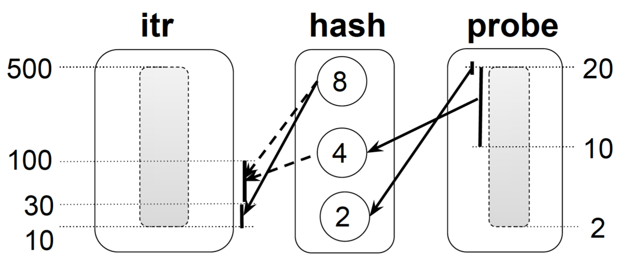
Fig. 6. Overall structure of the Rapids framework.

to be probed if a table has few buckets. Fewer buckets require larger maximum iterations to give meaningful rank scores among poor candidates. Figure 7(a) shows the KDG structure for Ferret.

Developers' Effort: Rapids provides a profiling platform and a runtime library that allows application developers to communicate important application properties to the framework. Figure 7 shows some key components of the developers' effort. To do the profiling, developers need to prepare:

— *KDG specification:* A file to specify insights including knobs and their types, value ranges, and dependencies. Rapids uses the file to generate the KDG structure to represent the configuration space used to train cost and quality models. An example specification for our Ferret application is shown in Figure 7(b). The file represents the KDG of Figure 7(a).
— *Evaluation module:* A Python module that includes basic utility functions for the Rapids profiler to use, including (a) command and command-line arguments for application execution, (b) the optional sub-metric (q_i in Equation (6)) evaluation strategy, and (c) an overall quality function (F^Q in Equation (6)). The module is implemented by extending a Rapids-provided base class.
— *Source code modifications:* The developer inserts library calls into application code as shown in Figure 7(c) to bind actions (e.g., change a variable's value) to knobs and their settings (Lines 5~7) and to inform Rapids about the execution progress (Line 11). *FERRET_CONFIG_FILE* is auto-generated after training and contains model information. It also contains application execution-related information including the budget and customized quality. We believe that the additional demand on the developers is reasonable and well within their expertise.

Training and Model Construction: On the development platform, Rapids generates an exhaustive training set over all knob settings (continuous knobs will be discretized), eliminating all invalid



(a) Ferret KDG

```
<Continuous Knobs>:
itr [10,500] INT;
probe [2,20] INT;
<Discrete Knobs>:
hash {2,4,8} INT ;
<Dependencies>:
AND: itr [10,30] <- hash {8} ;
OR: itr [30,100] <- hash {4}, hash {8};
AND: hash {4} <- probe [10-20]
AND: hash {2} <- probe [20-20]
<Sub-Metrics>:
Coverage, Ranking
```

(b) KDG Specification File for Ferret

```
#include "rapids_mission_c_wrapper.h"
int iteration=500,hash=8,probe=20; // knobs with default values
...
create_rapids_mission(ferretMission, FERRET_CONFIG_FILE);
register_single_knob(ferretMission,"itr", &iteration);
register_single_knob(ferretMission,"hash", &hash);
register_single_knob(ferretMission,"probe", &probe);
while (ent = readdir(pd)) { // pd points to a dir containing images
      if (do_query(ent, hash, probe, itr) != 0) // query for top−K similar images
          return −1;
      finish_one_unit(ferretMission); // report progress
  }
...
```

(c) Ferret Source Code Modification.

Fig. 7. Developer's effort example.

configurations according to the developer's *KDG specification*. The developer-provided *Evaluation Module* is used to automatically build cost models and quality models for sub-metrics (or default metrics), including the representative set *RS* as discussed in Section 4.2.

Users' Effort and Runtime Control: Users express their overall budget and quality preferences through a developer-provided interface. Our current interface design is discussed in Section 7. Just in time before program execution, Rapids constructs the customized quality model in response to the user's preferences. If no customized quality is specified, the developer-defined default model is used. During runtime, Rapids tracks the remaining resource budget relative to the remaining work. If the application is deployed on the same machine on which it is trained, the cost model can be used directly. If not, re-training is needed based on the representative set. The runtime system continuously monitors the application and performs automatic re-configuration if needed to maintain the maximum possible quality while respecting the provided budget even under changing conditions due to system uncertainties.

6 EXPERIMENTAL EVALUATION

To assess the practical, end-to-end effectiveness of Rapids and its use of the KDG, we implemented and evaluated a prototype system. This system includes a preliminary user interface that was used for a limited user study to assess the potential benefits of virtual knobs and customized quality as discussed in Section 7.

6.1 Sample Applications

The evaluation uses eight different sample applications/workloads, three of which are from widely used benchmarks in related works (e.g., [47, 70, 89, 90]) with known quality metrics, namely Swaptions [13], Bodytrack [13], and Ferret [13]. SVM [94], **NeuralNet (**NN**)** [94], and FaceDetect [17] are sample applications we adopted from public libraries. The other two, namely NavApp and VideoApp, are developed using the Rapids framework for use in real-world deployment. The eight workloads cover a wide range of application domains and exhibit a variety of properties (e.g., knob number, type, and dependency) we would expect to see in real-world applications. The applications have been chosen to illustrate the features and benefits of our Rapids framework while allowing others to assess Rapids's effectiveness through applications used by related work as well as highlighting its applicability to more full-featured, real-world-style applications.

Swaptions is a financial analysis application computing iterative simulations. One continuous knob: number of iterations to simulate, within [100,000, 1,000,000]. *Quality metric:* quality loss for each swap computed using the vector distortion [82] described in Equation (10). n and w_i are the total number of elements in the vector and their weights. We use $n = w_i = 1$. $\hat{y}$ is the computed price from the execution and y is the price when executing with the highest setting.

$$\sum_{i=1}^{n} w_i * abs((\hat{y}_i - y_i)/y_i) \tag{10}$$

FaceDetect detects human faces from input images based on Haar Cascade. Optionally, it filters the results by checking the presence of eyes. We use the dataset from FDDB [50] for testing. One continuous knob and two discrete knobs: pyramid levels ranging from [5, 25], neighbor pixels to examine {0, 4, 8}, the minimum number of eyes {0, 1, 2}. *Customization: Precision* and *Recall*. *Quality metric:* We adopt the standard measurement of recognition performance, the F-measure [24] as defined in Equation (7). Users may customize this metric by changing the weights w_p and w_r for virtual knobs *Precision* and *Recall*, respectively. The default quality uses $w_r = w_p = 1$ (F1-score).

SVM and NN are two commonly used supervised learning applications that classify input images using the CIFAR-10 dataset [56]. In our experiments, they run 1,000 iterations on a set of labeled training data and construct a simple **Support Vector Machine (SVM)** and an NN (fully connected Neural Network) model for classification. The source of both applications is adapted from a popular open course project [94]. We only consider the training phase in our experiments where both classifiers iteratively update the model parameters. For NN, we only use the default settings (one hidden layer) for illustration purposes. However, extending the network to deeper settings will not affect the experiment setup since all knobs we use in the experiment are independent of the network structure. The output of the application is a prediction model (classifier), and we treat the overall prediction accuracy on the test data as the quality. Both SVM and NN have one continuous knob and two discrete knobs: learning rate within [1e-7,1e-5], batch size {64, 128, 256, 512, 1024}, regularization rate {5,000, 10,000, 15,000, 20,000, 25,000}. *Quality metric:* prediction accuracy.

Ferret is as described in Section 5. Two continuous knobs and one discrete knob. *Quality metric:* We use a common ranking score measurement G-measure [11] shown in Equation (11):

$$\begin{aligned} err = {} & 2 * (k - z)(k + 1) \\ & + \sum_{i \in Z} |r_1(i) - r_2(i)| - \sum_{i \in S} r_1(i) - \sum_{i \in T} r_2(i), \end{aligned} \tag{11}$$

with $Q = 1 - err/k(k+1)$.

Here, Z is the set of images appearing in both $list_1$ and $list_2$ of size z. S and T are the sets exclusively in $list_1$ and $list_2$ of size k. r_1 and r_2 are the ranks of an image in $list_1$ and $list_2$. *Customization: Coverage* and *Ranking* are exposed as virtual knobs to users. *Coverage* is calculated by the first addend in Equation (11). *Ranking* is the rest of the equation. *Coverage* increases with "correct" results returned, while *Ranking* increases by returning results in the correct order.

Bodytrack is a computer vision application that tracks a set of human body components from a video. One continuous and one discrete knob: number of annealing layers from [1, 5], number of particles to track within [100, 4,000]. *Dependencies:* Lower annealing layers require higher particle numbers. *Quality metric:* The output consists of the position of different body components. We use Equation (10) for quality and w_i to be proportional to the size of the component being tracked.

NavApp as described in Section 3.1 also demonstrates the utility of Rapids under real-world conditions. One continuous knob and three discrete knobs. *Customization:* Different priorities can be given to *brightness*, *localization*, and *information* as described in Section 3.3. *Quality metric:* Weighted sum over three sub-metrics.

VideoApp is a custom-built application also designed to emulate real-world usage. It allows users to watch a high-resolution video locally or stream lower-quality video from a remote server. One continuous knob and three discrete knobs: screen as in NavApp, video frame rate from {15fps, 30fps, 45fps, 60fps}, video resolution from {144P, 240P, 480P, 720P}, network {On, Off}. *Dependencies:* Lower frame rates require lower resolutions to avoid unreasonable situations, e.g., 720P/15fps or 240P/60fps. Also, lower resolutions require the network (high-res content should not be streamed, but low-res saves data). *Customization:* Different priorities can be given to *brightness*, *smoothness*, and *resolution*. *Quality metric:* The weighted sum over resolution, frame rate, and brightness as in NavApp to calculate the overall performance.

Table 2 shows the opportunities for approximation in all our sample applications based on the default metric of each application. *Min Cost* reports the lowest cost in terms of execution time or energy consumption of a configuration relative to the optimal, highest-quality configuration under an unlimited resource budget. *Min Quality* reports the highest-possible-quality degradation of a configuration under the default quality metric. The last two columns summarize the opportunity for both developers and users to participate in Rapids. The table also reflects some empirical decisions made by the authors, i.e., not encoding dependencies or custom quality on certain applications.

Table 2. Approximation Opportunity

	Min Cost	Min Quality	Dependency	Custom Quality
SWAPTIONS	10.08%	57.78%	-	-
FACEDETECT	27.43%	41.23%	-	✓
SVM	57.98%	49.96%	-	-
NN	76.55%	42.8%	-	-
FERRET	37.26%	54.91%	✓	✓
BODYTRACK	7.51%	40.84%	✓	-
NAVAPP	56.12%	22.2%	✓	✓
VIDEOAPP	67.62%	27.7%	✓	✓

In this article, the output from the "default" optimal configuration under an unlimited resource budget is treated as the "ground truth" and is used to evaluate the output quality of other configurations. Therefore, the output from the "default" configuration has always the highest quality. In all our benchmark applications, the default configuration is also the most expensive configuration. Theoretically, there could exist configurations that cost more, i.e., consume more resources than the default configuration. However, a more expensive configuration with lower output quality will never be selected as a solution of our optimization problem.

6.2 Experimental Results

For our sample applications, developers performed the offline training phase on a Linux machine or an Android phone. Application users ran the applications on an embedded Linux board or a separate phone. For all of our experiments, the development and target platforms were distinct. Key specifications are:

LINUX machines $Developer$: 6-cores at 3.7 GHz, — 16 GB RAM at 2,666 MHz; $Target$: Nvidia TX1 [26], 1.9 GHz 64-bit 4-core — 2 MB L2 cache.
Android machines: $Developer$: Nexus-5: 2.26 GHz 4-core processor — 2 GB RAM. 4.95-inch screen, 1080 × 1920 pixels; $Target$: Nexus-6: 2.26 GHz 4-core processor — 3 GB RAM. 5.96-inch screen, 1440 × 2560 pixels.

We evaluate the system with respect to our three main contributions: (1) *Developers:* configuration space reduction from developer-encoded insights, (2) *Machine:* training time reduction and the model accuracy, and (3) *Users:* improvement of user-preferred sub-metrics relative to default metrics. Finally, we evaluate the runtime performance by measuring the overhead and the overall output quality.

Pruned Search Space: Table 3 shows the space pruning and cost prediction error rate for all our applications. $Conf$ reports the number of all possible configurations where every combination of knob settings is considered "valid." *KDG* reports the number of "valid" configurations after developers encode dependencies using #loc lines of specification code to customize the configuration space. *RS_P* and *RS_S* report the size of *RS* with an error threshold ϵ of 5%. To give a meaningful comparison for knobs with continuous settings, we discretized the settings by sampling each continuous node where the number of sampled data points is shown with a † in the third column labeled "$Conf\ Discrete$."

Feasibility for Real-world Applications: In the experiments described above, RAPIDS samples a continuous knob to 10 discrete settings. However, the configuration space size for real-world

Table 3. Search Space Pruning and Specification Effort, Discretization of Continuous Knobs with † Samples (10), RS Calculated with Error Bound $\epsilon = 5\%$

Application	Knob(#)	Conf Discrete(#)	Spec (#loc)	*KDG*(#)	RS_P(#)	RS_S(#)
Swaptions	1	10 ($10^{\dagger}$)	2	-	2	2
FaceDetect	3	90 ($3 \times 3 \times 10^{\dagger}$)	4	-	54	18
SVM	3	250 ($5 \times 5 \times 10^{\dagger}$)	4	-	8	4
NN	3	250 ($5 \times 5 \times 10^{\dagger}$)	4	-	8	4
Ferret	3	300 ($3 \times 10^{\dagger} \times 10^{\dagger}$)	11	162	27	4
Bodytrack	2	50 ($5 \times 10^{\dagger}$)	5	28	15	12
NavApp	4	180 ($10^{\dagger} \times 3 \times 3 \times 2$)	7	40	12	6
VideoApp	4	320 ($10^{\dagger} \times 4 \times 4 \times 2$)	11	24	12	4

Table 4. Search Space Pruning and Specification Effort, Discretization of Continuous Knobs with † Samples (up to 100 Where Possible), RS Calculated with Error Bound $\epsilon = 5\%$

Application	Knob(#)	Conf Discrete(#)	Spec (#loc)	*RSDG*(#)	RS_P(#)	RS_S(#)
Swaptions	1	100 ($100^{\dagger}$)	2	-	2	2
FaceDetect	3	189 ($3 \times 3 \times 21^{\dagger}$)	4	-	54	21
SVM	3	2,500 ($5 \times 5 \times 100^{\dagger}$)	4	-	27	6
NN	3	2,500 ($5 \times 5 \times 100^{\dagger}$)	4	-	27	6
Ferret	3	5,700 ($3 \times 19^{\dagger} \times 100^{\dagger}$)	11	3,040	75	5
Bodytrack	2	500 ($5 \times 100^{\dagger}$)	5	328	25	12

applications with continuous knobs can be theoretically infinite as discussed in Section 2.1. For continuous knobs, Rapids constructs the performance model with piecewise linear functions, covering the entire value range, which cannot be done by discrete-only approaches that rely on sampling. To investigate the impact of the search space size on representative set construction and configuration selection, we increased the sampling granularity for continuous knobs from 10 to 100 settings. The resulting search space sizes and number of configurations in the representative sets are shown in Table 4.

Based on our experience, this corresponds to search space sizes that can be effectively explored for real-world applications. As discussed in Section 2, the KDG configuration search space is based on the number, value ranges, and dependencies of knobs, which are application program variables. In other related work, for instance, in the context of autotuners [6, 7, 98], search spaces include many more different aspects of an application and its structure, resulting in reported search space sizes of up to 10^{3600} [7]. In the context of an adaptive framework with customizable quality metrics such as Rapids, there may be tens of application knobs at best, rather than thousands or even more, making the search space size more dependent on the individual knob ranges than the total number of knobs. Being able to represent continuous knob value ranges in an efficient way enables Rapids to deal with real-world configuration spaces of large sizes.

As shown in Tables 3 and 4, while configuration search space sizes increase by around one order of magnitude, the sizes of RS_P and RS_S do not grow proportionally to the size of the overall configuration spaces. The solution time of the dynamic configuration selection problem depends on the total number of constraints in the problem and the complexity of the objective function and constraints, where both are determined by the number of segments Rapids generated in the piecewise linear approach. The number of segments is directly related to the number of configurations in the representative set. In the case of Ferret, an almost 3× size increase of RS_P

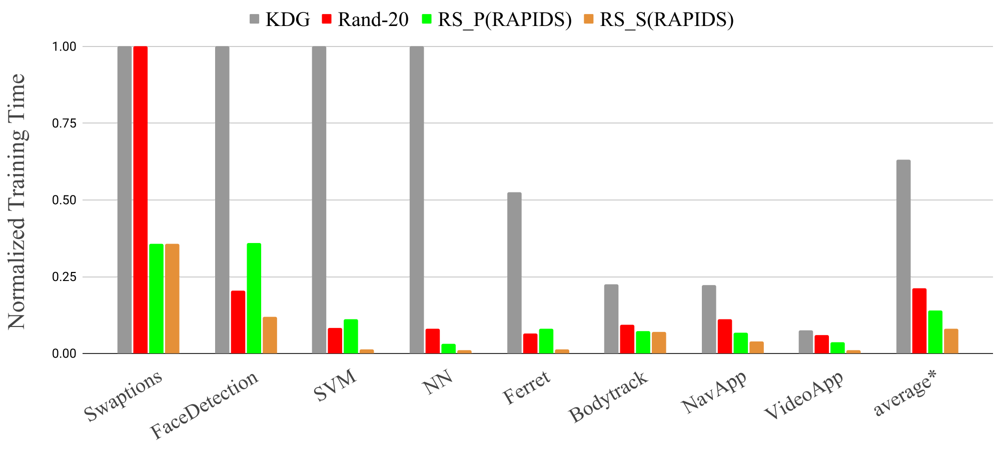


Fig. 8. Required normalized training time. Lower is faster.

occurs. However, no solution time increases were observed in any of our applications, with an average overall solution time of 5.9 ms.

It is important to note that these problems are relatively small for a tool such as gurobi [93], and that for small problems the solution times are often dominated by communicating and/or reading in the problem representation rather than the actual solution time itself. However, solving a quadratic integer programming problem (MIQCP [22]) can have exponential cost. Nevertheless, we did not observe any such increases in the problems Rapids generated. In addition, tools such as gurobi [93] and cplex [14, 27] can be used to generate families of heuristic solutions in cases where solution times are considered too costly. A discussion of such heuristics is beyond the scope of this article.

These results indicate that Rapids is scalable and can handle applications with large number of configurations without significantly increased solution complexity. For simplicity, the results reported in the remainder of this article are based on the configurations shown in Table 3.

Reduced Training Time: Filtering out "invalid" configurations through the KDG or calculating *RS* prunes the configuration space and thus reduces the training time. We report the total training time for constructing the cost model through different approaches normalized by the time required to train configurations without dependencies: (1) **KDG:** configurations with dependencies, (2) **Rand-20:** a heuristic used in CALOREE [70] that constructs the model with 20 random configurations [69], (3) **RS_P(RAPIDS):** partition-based *RS*, and (4) **RS_S(RAPIDS):** selection-based *RS*.

As shown in Figure 8, allowing developers to specify dependencies among knobs reduces the training time by an average of 36.9%. Training 20 random selected samples (**Rand-20**) reduces the training time to an average of 21.24%. The *RS* calculated by Rapids further reduces the training time to 7.9% on average, which translates to up to 13× faster retraining when porting to other devices. The significant reduction in training cost of the RS approach enables Rapids to quickly retrain itself when applications are ported to unknown target devices.

Model Validation and Porting Efficacy: When applications are ported from the developer machine to the user machine, a new cost model must be quickly and accurately constructed. Rapids constructs the cost model based on a pruned space (KDG) and/or *RS*. However, a reduced training set may lead to a higher prediction error. To this end, we evaluate the accuracy of such reconstructed models on our target machines. **KDG** serves as the oracle with *accuracy* = 1.0

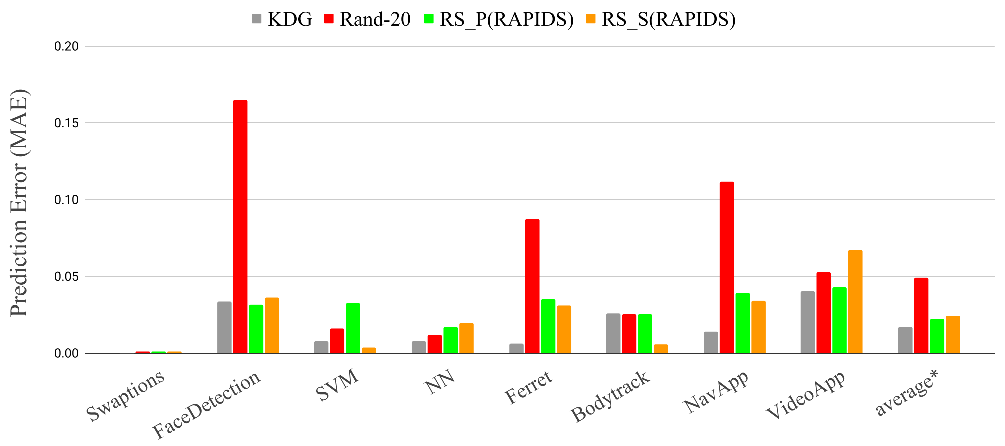


Fig. 9. Model prediction error on target machine. Lower is more accurate.

because the model is built with all observations. When porting to and from machines described as above, we see performance as shown in Figure 9. The figure shows the corresponding prediction errors across all valid configurations. On average, the model constructed from **Rand-20** results in a 6.39% prediction error (averaged over 10 runs). **Rand-20** performs well in simple applications like SWAPTIONS but has high error for complex applications. In FERRET, the inherited randomness in **Rand-20** caused the highest prediction error to be more than 100%, i.e., more than a factor of 2 relative to the observed value. Our two *RS* strategies have a stable average error of 3.5% for *RS_S* and 3.1% for *RS_P*.

Custom Quality: Application developers can provide a customizable, higher-level quality notion through virtual knobs where users can express their preferences for a particular quality outcome at a level of abstraction that makes sense to them. In turn, RAPIDS automatically builds the required quality models to support configuration selection and reconfiguration. We evaluate this benefit on our four applications with custom quality metrics.

User-provided preferences change application behavior when selecting the optimal solution for a budget. Users expect to see improvement on their preferred sub-metrics. For example, Figure 10 reports such behavior for FERRET. We evaluate the *Coverage* and *Ranking* sub-metrics by running the application with budget of *0.5 * (max - min)*. We adjust the preference of one of the metrics from *1.0* to *2.0* in steps of *0.1* and keep the other as *1.0*. Figure 10(a) shows the improvement on "Coverage" as its preference increases. On the other hand, "Ranking" suffers from degradation due to the "Coverage" preference. Figure 10(b) shows similar behavior when "Ranking" is preferred. This is expected since knob settings that benefit one sub-metric may hurt the other.

To show the impact of custom quality selections on the quality values of different sub-metrics, we evaluated different sub-metric preferences under a range of budgets for each of our four applications with customizable quality. A sub-metric is preferred if it is assigned a preference of *2.0* while all other sub-metrics have a preference of *1.0*. For each preferred sub-metric, Table 5 reports the average improvement across all budgets relative to the default quality where all sub-metrics are assigned the preference *1.0*. The recall in FACEDETECT does not change when preference increases because the optimal selection with the default quality metric produces the highest recall. Overall, improvements can be up to 3.59×, with 1.76× on average.

Reconfiguration and Overhead: For each application, RAPIDS constructs the performance model based on a training set of inputs provided with the application. During runtime, the

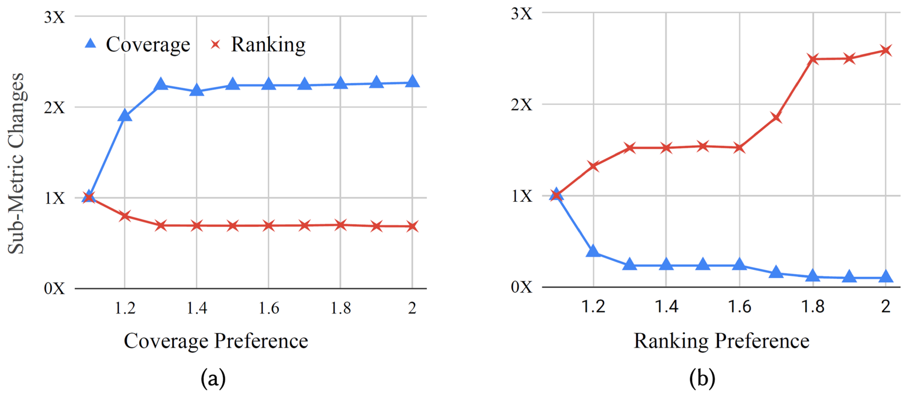


Fig. 10. Sub-metric quality changes in response to preference changes for sample application FERRET under the budget 0.5 ∗ (*max* − *min*). (a) with Coverage ((b) with Ranking) as the preferred sub-metric. X-axis: Preference weight on preferred sub-metrics, y-axis: sub-metric quality changes relative to default quality with 1.0 preference for all sub-metrics.

Table 5. Relative User-Preferred Sub-metrics Value Improvements Averaged across the Min-Max Range of Cost Budgets

	Preferred Sub-metrics and Improvement		
FERRET	coverage 3.2×	ranking 3.59×	
FACEDETECT	precision 1.13×	recall 1×	
NAVAPP	brightness 1.45×	localization 1.5×	information 1.5×
VIDEOAPP	brightness 1.29×	smoothness 1.26×	resolution 1.62×

predicted and the real performance may differ for three main reasons: (1) embedded prediction error in the model, (2) application input dependencies, and (3) dynamic runtime environment.

RAPIDS overcomes these issues by constantly monitoring the resource usage and performing runtime reconfiguration if necessary. The full reconfiguration procedure in RAPIDS has three steps that are common in any adaptive or decision system (e.g., [46]):

(1) *Monitor/Observe* (∼1 ms): Record the current execution progress and actual resource usage for all work units executed so far. The recording frequency is tuneable by developers.

(2) *Problem Solving/Decide*: (a) Calculate the new budget per work unit given the current execution progress and remaining budget (∼1 ms). (b) Generate and solve the new optimization problem and retrieve the new configuration (∼17 ms). If no solver is available on the target device (e.g., gurobi [93] cannot be deployed on ARM), RAPIDS contacts a remote server to determine the optimal configuration based on the remaining work units and remaining budget. The overhead of each remote reconfiguration averages 191 ms/133 ms (ARM/Android). In NAVAPP and VIDEOAPP, we report the overhead as additional energy consumption.

(3) *Result Deployment/Act* (∼1ms): Apply the new configuration.

Overhead Optimization: Ideally, systems like RAPIDS can monitor the budget usage and working progress after every work unit to avoid wasting budget or violating budget constraints. However,

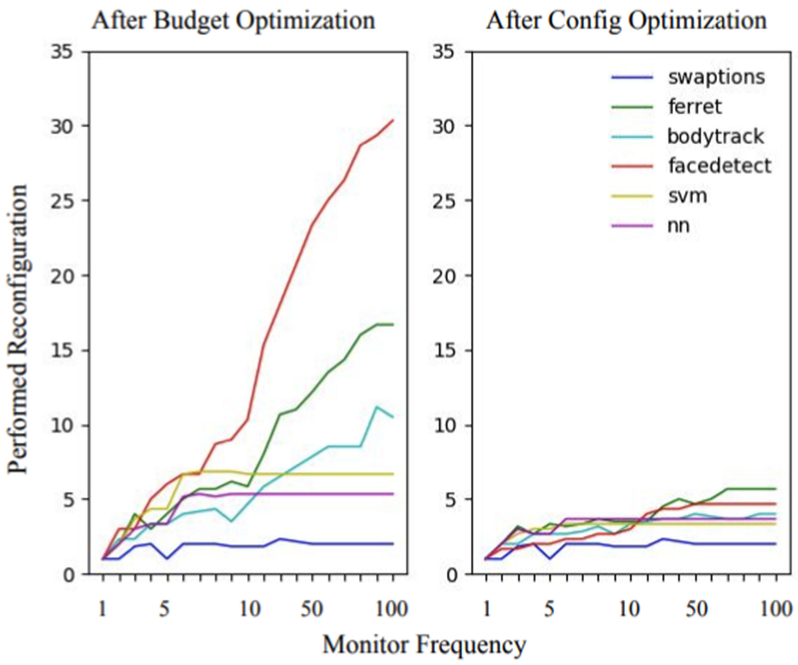


Fig. 11. Reconfiguration overhead.

increased monitoring and reconfiguration leads to higher overhead. In the extreme, the majority of the budget is occupied by problem solving or reconfiguration, leaving little of the budget for application execution. To minimize the overhead, Rapids terminates the reconfiguration procedure by skipping step 2b/3 if the new budget per work unit is within 5% of the previous value (i.e., performing *Budget Optimization*), and skipping step 3 if the new configuration is within 1% of the previous value (i.e., performing *Config Optimization*). These thresholds were selected based on our experimental observations and represent a compromise between successful application outcomes and the number of reconfigurations. Developers can easily change these thresholds.

Figure 11 shows the number of reconfigurations that are actually performed by Rapids given different monitoring frequencies. For each point on each line in the left figure, the difference between the X value (monitor frequency) and the Y value (actual performed reconfiguration) is the number of "Solving+Deployment" (step 2/3) being skipped by the budget optimization. The difference between the left and right figure is the number of "Deployment" (step 3) being skipped by the configuration optimization. Table 6 reports the overall Rapids overhead for all our applications. As shown in the table, the overhead is small.

Budget Constraint Enforcement: Rapids reconfigures applications at runtime by observing the available remaining budget and remaining work units. Rapids's goal is to successfully execute the entire application while not exceeding, but fully utilizing, the user-specified budget. There are several reasons reconfiguration is needed to achieve this goal. Performance/cost models may not be accurate due to discrepancies between training and input data, target system noise due to varying availability of system resources, and configuration space characteristics where the ideal configuration for a given budget is not part of the optimization space and therefore can only be approximated. To verify that Rapids's reconfiguration behavior can correctly honor the budget constraint, we ran each application (six in total) under four different budgets, [0.2, 0.5, 0.8, 1.1] *

Table 6. Overhead in All Applications

	w/ Rapids (seconds)	w/o Rapids	Overhead
Swaptions	461	458	1.09%
FaceDetect	92	89	3.2%
SVM	35.7	34.6	3.17%
NN	23.1	22.5	2.67%
Ferret	551	544	1.22%
Bodytrack	264	260	1.44%
NavApp*	-	-	<0.05%
VideoApp*	-	-	<0.05%

*NavApp and VideoApp each use energy instead of time as a metric and thus have no times reported as overhead.

$(max - min)$. With each budget setting, we ran the application in two different modes, with and without Rapids reconfiguration, with up to 10 reconfiguration points equally spread across the execution. In our experiments, a program execution is considered to be violating its budget if it fails to successfully finish its execution within the user-provided budget with a 5% tolerance, i.e., $execution\_time \geq 1.05 * budget$. A reconfiguration will be skipped if the new budget is within 5% of range of its previous value; i.e., Rapids performs a *Budget Optimization* as discussed above.

When no reconfiguration is performed, 16.6% (8 out of 48) application executions violated the budget. For the remaining successful executions, the average budget utilization was 85.3%. With reconfiguration enabled, Rapids performed on average 4.8 reconfigurations per execution and all executions succeeded with an average budget utilization of 90.3%. These results demonstrate the importance of reconfigurations.

7 APPLICATION USER INTERFACE

The current Rapids implementation includes a module that allows developers to define an application user interface based on a basic user interface layout. The implementation uses the PySide2 [37] software package and communicates with the Rapids runtime environment through an execution configuration file. Developers can customize the interface to match particular characteristics of the supported application, particularly the user-exposed virtual knobs and the quality notion used.

In order to assess the benefits of customized quality through sub-metrics, we have conducted an initial user study based on a single application, Ferret, which has two sub-metrics, namely coverage and ranking (see Section 6.1). Ferret has three knobs, namely hash, probe, and iteration. The goal of this study was to provide an initial insight into the effort needed by an application user to find a configuration that satisfies a specific (subjective) quality requirement under a given resource budget when using either virtual knobs (high-level, customized) or application-level (low-level, program-behavior-specific) knobs. The conducted experiments recorded (1) the steps a test subject needed to reach a knob setting that resulted in an overall configuration that satisfied a provided quality goal, and (2) the quality of that configuration selection relative to the optimal possible configuration.

Each of our 15 study subjects (users) were faced with six challenges, first to be solved using virtual knobs and then application-level knobs. Each challenge asked them to find a configuration meeting ranking and/or coverage sub-metric values under a given execution budget. The results show that the use of virtual knobs substantially reduces the number of steps needed to determine a desired configuration (by 5× on average) and that the selected configurations using virtual knobs were on average 8.3% better than if application-level knobs were used. In 2.3% of application-level knob challenges, the user was unable to find a single valid configuration under the given budget; this never occurred when virtual knobs were used.

8 RELATED WORK

Optimization opportunities of approximations and their use in adaptive frameworks as part of configuration management have been investigated by many research groups. One key aspect is the notion of quality, i.e., the notion that approximation trades off resource usage for result quality, which really implies a change of the semantics of the program. However, there have been works that follow the more the traditional approach to optimizations where the semantics cannot be changed, and therefore consider approximation as a regular optimization transformation, for example, by selecting lower-precision data representation without affecting the overall outcome [10, 18, 19, 59]. However, most research targets approaches that trade quality for resource usage and require different levels of user involvement to make the best tradeoff decisions. All these approaches require metrics for resource usage and quality, which serve as the basis for developer-guided or automatic approximation-level selection, i.e., configuration management. As is common for many optimization problems, proposed solutions range from fully automatic hardware- and software-based techniques to library-based approaches, and new program language abstractions that enable compile-time and runtime optimizations. In the following, we highlight specific important aspects of approximation management as they relate to our proposed framework.

Specific Approximation Techniques: There exists a large family of approximation techniques, and these techniques can be highly application specific. For applications with iterative computations, Loop Perforation [85, 89] gives approximate answers by skipping certain iterations. Compute- and memory-intensive applications with precision tolerance can benefit from precision scaling [4, 32, 88]. The Precimonius [83] system provides recommendations to the developer in terms of type assignments to variables such that the accuracy of the program is within a developer-provided error range. Memoization is also a technique used to speed up floating point calculation by reusing results of similar computation instructions [3, 78]. Dropping inputs through sampling is commonly used as the approximation for applications with a large set of inputs [8, 41, 62]. Similarly, dropping tasks or jobs is common for applications under large multi-task frameworks, e.g., on GPU [20] or Map-Reduce frameworks [41, 49, 58]. Rapids aims to develop a system to seamlessly exploit approximation techniques developed by others, instead of introducing new techniques. Each of the above techniques can potentially be encoded in Rapids as a knob.

Development Support for Approximation: Developing approximate applications requires interactions with the application programmer/developer since configuration management is a semantic issue. An important challenge is to reduce the burden on the developer through new programming abstractions and automatic techniques. Most existing approaches for adaptive configuration management target applications that have been written without approximation in mind. For example, compiler-based automatic techniques [34, 47, 64, 65, 68, 95] identify program variables or functions as "knobs." However, recent work confirms the importance of developers' input in defining the configuration search space and its customization [12, 75, 76].

To help developers implement approximate applications, language/runtime-library-based approaches have received increased attention where approximation is an explicit part of a program's semantics. EnerJ [86] and FlexJava [72] provide data-type annotations to indicate to the compiler when it is safe to use an approximate value. In the case of FlexJava [72], minimal quality requirements can be added. However, it is left to the compiler to what extent the specified approximation is exploited. In Petabricks [6] program developers can provide alternative, approximate implementations of functions that can be selected at runtime. Expressing possible interactions between such function knobs is not directly supported. Uncertain<T> [16] allows developers to use probabilistic value approximations as a form of quality introspection at runtime. The developer can conditionally execute code based on the quality requirements of approximate variables and computations.

All these approaches have in common that configuration management can only be implemented through explicit ad hoc code changes to encode the dependency logic. Expressing correlations between knobs and their desired values has to be done in the form of conditional statements. Therefore, configuration management is cumbersome and potentially error prone. Also, no distinction is made between the application developer and application user, limiting user-level customized quality and higher-level reasoning.

Resource Consumption Prediction: Exploiting application performance degradation has been explored by several groups. Significant research focuses on constructing the cost model. Learning-based models [28, 51, 54, 71, 90, 91, 100] predict performance through either input or execution features, whose accuracy is bound by the richness of the dataset. Additionally, examining each input introduces significant runtime overhead. For example, the rendering logic for a webpage can only be determined after extracting and evaluating the features of the page in Chamelon [31]. Control theoretical approaches [9, 36, 45, 47, 102] aim to deal with runtime disturbance. Recently, Caloree [70] combined the learning-based and the control theory to overcome the shortages in both strategies when used separately. However, these approaches assume that the training phase is free, even though getting the profile for large applications may take weeks [36] due to the size of the configuration space. A full training is required when porting applications to other devices, limiting their applicability. Rapids chooses the learning-based approach for both scalability and efficiency reasons. In Rapids, the piecewise-linear model complexity does not grow proportionally to the number of available configurations because of the modeling. Also, when porting applications to new devices, our notion of representative sets allows the reconstruction of cost and quality models much faster with a few configurations to be retrained. This can also be extended with other re-enforcement learning techniques to refine the model if needed.

Quality Prediction: Probabilistic and approximate programming uses probability variables and their distributions [15, 38–40, 73, 77, 80]. The research focuses on representations of the distributions and operations induced by operations on their associated probabilistic approximate variables. In the database community, approximation has been used to provide statistical error bounds of queries [1, 101], and more recently in the context of Map-Reduce [30] applications [41, 58]. Proving and/or verifying approximation error bounds has also been the topic of ongoing research [21, 23]. However, these approaches cannot effectively enable users to express different quality preferences, which is one on the main contributions of our new KDG-based Rapids framework.

Custom Quality: Existing approaches further assume some pre-defined quality notion where each application comes with a quality function for its output, e.g., PSNR [100] or SSIM [97] for video playback applications. Akturk et al. [2] categorizes the quality metric used in common approximation applications including some of our benchmark applications (Bodytrack, Swaptions). However, many applications have subjective quality notions, so the application user needs to be involved in defining the quality function. For example, a face detection application may use F-score [24] as the quality metric, which is defined as the harmonic mean of the recognition precision and recall. However, the precision can be more important than recall when used in target recognition. On the other hand, higher recall can help the application performance when used in crowd counting. Allowing users to express such preference requires the system to be flexible enough that the quality model can be quickly updated when such preferences change. It is too time consuming to repeat the training process to reflect any changes to the quality notion.

Hardware Architectural Support: Hardware support for probabilistic and approximate computing has also received significant attention in recent years, allowing the use of faulty hardware [33, 43, 53, 57, 66, 67, 79, 87] or to tolerate faulty operations [35, 81]. However, these approaches require the application developers to carefully identify the configurable components but do not allow developers to express more complex correlations between components and their settings. Also, different approximation techniques require different hardware designs, e.g., approximate storage, approximate instruction execution, and so forth. A practical developing framework [5, 29, 55] for configurable applications with end-to-end support [92] for developers has remained an unsolved challenge.

9 CONCLUSION AND FUTURE WORK

The KDG-based Rapids is a novel adaptive application programming framework that supports customizable quality. In such a system, three different stakeholders must be involved in the development and execution of an application. The developer specifies the configuration space, excluding infeasible or undesirable configurations. The generated KDG is a compact representation of the configuration search space. The developer also provides basic quality notions that are exposed to the user as virtual knobs. The user customizes the quality metric using priority weights for the virtual knobs. The offline system produces cost and quality models based on the KDG. The runtime system just in time, i.e., before application execution, creates the customized quality model that is used to select and implement the highest-quality configuration under the user-provided cost/resource budget. Porting Rapids to new platforms requires the reconstruction of the cost/quality models. This process may be optimized through the execution/training of representative configurations that are sufficient to recreate a model for all possible configurations. As in other state-of-the-art comparable approaches that use training for their model construction, Rapids requires a full offline training step, which can take substantial amounts of time. However, Rapids trains the application only once, and only on the reduced set of configurations represented by the KDG. Experimental results on eight application programs show that developers can reduce the size of the configuration space by 68.7% on average by excluding infeasible configurations. The subjective quality experience of the users as represented by a customized quality metric is improved by up to 3.59×, indicating the opportunity and potential benefit of customized quality. The execution time overhead due to program monitoring and reconfiguration is less than 1.84% on average relative to the execution time of the entire application.

Based on our experience with our current prototype implementation of an application user interface, we are designing a Rapids interface tool that will allow application users to easily customize their quality metric, specify the execution budget, and launch an application. In addition, a planning tool will show the expected improvements of quality if alternative quality customizations or cost budgets are chosen. This interface tool will be used as part of a comprehensive user study to quantify the benefits of customizable quality and our adaptive framework in general.

Finally, optimal or close-to-optimal configuration selections across applications that run at the same time on a target platform are an important problem. Single-application configuration systems such as Rapids may use a context-oblivious approach and treat resource usage by other active applications as system noise or model inaccuracy, potentially resulting in dynamic reconfigurations. However, this "local" application view of configuration selection misses its "global" and mutual interaction with other active applications and their configuration selections. We have started investigating global techniques for configuration selections across multiple applications that share the same computing resources [61]. We are planning to further improve our approach to address the cross-application configuration selection problem.

All of the Rapids framework including the profiler, the library, the user interface tool, and all application benchmarks are available online through GitHub [60].

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their insightful comments.

REFERENCES

[1] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. The Aqua approximate query answering system. In *ACM Sigmod Record*, Vol. 28. ACM, 574–576.

[2] Ismail Akturk, Karen Khatamifard, and Ulya R. Karpuzcu. 2015. On quantification of accuracy loss in approximate computing. In *Workshop on Duplicating, Deconstructing and Debunking (WDDD'15)*, Vol. 15.

[3] Carlos Alvarez, Jesus Corbal, and Mateo Valero. 2005. Fuzzy memoization for floating-point multimedia applications. *IEEE Trans. Comput.* 54, 7 (2005), 922–927.

[4] Mohammad Ashraful Anam, Paul N. Whatmough, and Yiannis Andreopoulos. 2013. Precision-energy-throughput scaling of generic matrix multiplication and discrete convolution kernels via linear projections. In *The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*. IEEE, 21–30.

[5] Jesper Andersson, Luciano Baresi, Nelly Bencomo, Rogério de Lemos, Alessandra Gorla, Paola Inverardi, and Thomas Vogel. 2013. Software engineering processes for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*. Springer, 51–75.

[6] Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. 2009. PetaBricks: A language and compiler for algorithmic choice. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'09)*. ACM, New York, NY, 38–49.

[7] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. 2014. OpenTuner: An extensible framework for program autotuning. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT'14)*. 303–315. https://doi.org/10.1145/2628071.2628092

[8] Jason Ansel, Yee Lok Wong, Cy Chan, Marek Olszewski, Alan Edelman, and Saman Amarasinghe. 2011. Language and compiler support for auto-tuning variable-accuracy algorithms. In *International Symposium on Code Generation and Optimization (CGO'11)*. IEEE, 85–96.

[9] Woongki Baek and Trishul M. Chilimbi. 2010. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'10)*. ACM, New York, NY, 198–209. https://doi.org/10.1145/1806596.1806620

[10] David H. Bailey. 2008. Resolving numerical anomalies in scientific computation. Research Gate online publication. http://www.davidhbailey.com/dhbpapers/numerical-bugs.pdf.

[11] Judit Bar-Ilan, Mazlita Mat-Hassan, and Mark Levene. 2006. Methods for comparing rankings of search engine results. *Computer Networks* 50, 10 (2006), 1448–1463.

[12] Saeid Barati, Ferenc A. Bartha, Swarnendu Biswas, Robert Cartwright, Adam Duracz, Donald Fussell, Henry Hoffmann, Connor Imes, Jason Miller, Nikita Mishra, Arvind, Dung Nguyen, Krishna V. Palem, Yan Pei, Keshav Pingali, Ryuichi Sai, Andrew Wright, Yao-Hsiang Yang, and Sizhuo Zhang. 2019. Proteus: Language and runtime support for self-adaptive software development. *IEEE Software* 36, 2 (2019), 73–82.

[13] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation. Princeton University.

[14] R. Bixby. 1992. Implementing the simplex method: The initial basis. *ORSA Journal on Computing* 4, 3 (1992), 267–284.

[15] J. Borgstrom, A. D. Gordon, M. Greenberg, J. Margetson, and J. Van Gael. 2011. Measure transformer semantics for Bayesian machine learning. In *European Symposium on Programming (ESOP'11)*.

[16] J. Bornholt, T. Mytkowicz, and K. S. McKinley. 2014. Uncertain<T>: A first-order type for uncertain data. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. 51–66.

[17] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* 21, 11 (2000), 120–123.

[18] Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julien Langou, Piotr Luszczek, and Stanimire Tomov. 2007. *Exploiting Mixed Precision Floating Point Hardware in Scientific Computations*. Vol. 16.

[19] Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Piotr Luszczek, and Stanimire Tomov. 2008. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. *ACM Transactions on Mathematical Software* 34, 4 (2008), 1–22. https://doi.org/10.1145/1377596.1377597

[20] Surendra Byna, Jiayuan Meng, Anand Raghunathan, Srimat Chakradhar, and Srihari Cadambi. 2010. Best-effort semantic document search on GPUs. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 86–93.

[21] Michael Carbin, Deokhwan Kim, Sasa Misailovic, and Martin C. Rinard. 2012. Proving acceptability properties of relaxed nondeterministic approximate programs. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'12)*. ACM, New York, NY, 169–180. https://doi.org/10.1145/2254064.2254086

[22] IBM Knowledge Center. [n.d.]. *MIQCP: Mixed Integer Programs with Quadratic Terms in the Constraints.* https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.cplex.help/CPLEX/UsrMan/topics/discr_optim/mip_quadratic/03_introMIQCP.html.
Proving programs robust. In European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE.11).

[23] Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara Navidpour. 2011. Proving programs robust. In *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE'11)*.

[24] Nancy Chinchor. 1992. MUC-4 evaluation metrics. In *Proceedings of the 4th Conference on Message Understanding*. Association for Computational Linguistics, 22–29.

[25] Ionut Constandache, Shravan Gaonkar, Matt Sayler, Romit Roy Choudhury, and Landon Cox. 2009. Enloc: Energy-efficient localization for mobile phones. In *IEEE INFOCOM 2009*. IEEE, 2716–2720.

[26] NVIDIA Corporation. 2017. NVIDIA Jetson TX1 developer kit. http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html.

[27] IBM ILOG Cplex. 2009. V12. 1: User's manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.

[28] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. 2017. BOAT: Building auto-tuners with structured Bayesian optimization. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 479–488.

[29] Rogério De Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer, 1–32.

[30] J. Dean and S. Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Symposium on Operating System Design (OSDI'04)*.

[31] Mian Dong and Lin Zhong. 2011. Chameleon: A color-adaptive web browser for mobile OLED displays. In *Proceedings of the 9th International Conference on Mobile System, Applications, and Services (MobiSys'11)*. 85–98.

[32] Peter Düben, Sreelatha Yenugula, John Augustine, K. Palem, Jeremy Schlachter, Christian Enz, T. N. Palmer, et al. 2015. Opportunities for energy efficient computing: A study of inexact general purpose processors for high-performance and big-data applications. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*. IEEE, 764–769.

[33] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. 2012. Architectural support for disciplined approximate programming. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*.

[34] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture support for disciplined approximate programming. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 301–312.

[35] Yuntan Fang, Huawei Li, and Xiaowei Li. 2012. SoftPCM: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write. In *2012 IEEE 21st Asian Test Symposium*. IEEE, 131–136.

[36] Anne Farrell and Henry Hoffmann. 2016. MEANTIME: Achieving both minimal energy and timeliness with approximate computing. In *2016 USENIX Annual Technical Conference (USENIX ATC'16)*. 421–435.

[37] Qt for Python. 2020. *Qt for Python*. https://wiki.qt.io/Qt_for_Python.

[38] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. 1994. A language and program for complex Bayesian modelling.*Journal of the Royal Statistical Society, Series D (The Statistician)* 43, 1 (1994), 169–177.

[39] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. 2008. Church: A language for generative models. In *Conference in Uncertainty in Artificial Intelligence (UAI'08)*. 220–229.

[40] A. Gordon, T. Henzinger, A. V. Nori, and S. K. Rajamani. 2014. Probabilistic programming. In *International Conference on Software Engineering (ICSE'14)*.

[41] I. Gori, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. 2015. ApproxHadoop: Bringing approximations to MapReduce frameworks. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15)*. 383–397.

[42] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. MCDNN: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*. ACM, New York, NY, 123–136.

[43] Rajamohana Hegde and Naresh R. Shanbhag. 1999. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No. 99TH8477)*. IEEE, 30–35.

[44] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. 2004. *Feedback Control of Computing Systems*. John Wiley & Sons, Inc., Hoboken, NJ.

[45] H. Hoffmann. 2015. JouleGuard: Energy guarantees for approximate applications. In *Symposium on Operating Systems Principles (SOSP'15)*.

[46] Henry Hoffmann, Martina Maggio, Marco D. Santambrogio, Alberto Leva, and Anant Agarwal. 2011. *SEEC: A General and Extensible Framework for Self-Aware Computing*. Technical Report MIT-CSAIL-TR-2011-046. Massachusetts Institute of Technology, Cambridge MA.

[47] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic knobs for responsive power-aware computing. In *ASPLOS'11*. Newport Beach, California.

[48] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. 2011. Dynamic knobs for responsive power-aware computing. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. ACM, New York, NY, 199–212.

[49] Guangyan Hu, Sandro Rigo, Desheng Zhang, and Thu Nguyen. 2019. Approximation with error bounds in spark. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'19)*. IEEE, 61–73.

[50] Vidit Jain and Erik Learned-Miller. 2010. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. Technical Report UM-CS-2010-009. University of Massachusetts, Amherst.

[51] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. Transfer learning for improving model predictions in highly configurable software. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'17)*. IEEE, 31–41.

[52] Ken Kennedy and John R. Allen. 2002. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

[53] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke. 2015. Rumba: An online quality management system for approximate computing. In *International Symposium on Computer Architecture (ISCA'15)*. 554–566.

[54] J. C. Knight and N. G. Leveson. 1986. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering* 12, 1 (1986), 96–109.

[55] Jeff Kramer and Jeff Magee. 2007. Self-managed systems: an architectural challenge. In *2007 Future of Software Engineering*. IEEE Computer Society, 259–268.

[56] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. Technical Report, University of Toronto.

[57] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. 2011. Trading accuracy for power with an underdesigned multiplier architecture. In *2011 24th Internatioal Conference on VLSI Design*. IEEE, 346–351.

[58] N. Laptev, K. Zeng, and C. Zaniolo. 2012. Early accurate results for advanced analytics on MapReduce. *Proceedings of the VLDB Endowment* 5, 10 (2012), 1028–1039.

[59] Xiaoye S. Li, James W. Demmel, David H. Bailey, Greg Henry, Yozo Hida, Jimmy Iskandar, William Kahan, Suh Y. Kang, Anil Kapur, Michael C. Martin, Brandon J. Thompson, Teresa Tung, and Daniel J. Yoo. 2002. Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software* 28, 2 (June 2002), 152–205.

[60] Liu Liu. 2019.*RAPIDS Repository on Github*. https://github.com/niuye8911/rapidlib-linux.

[61] Liu Liu, Sibren Isaacman, and Ulrich Kremer. 2020. Global cost/quality management across multiple applications. In *Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20)*.

[62] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. 2012. Flikker: Saving DRAM refresh-power through critical data partitioning. *ACM SIGPLAN Notices* 47, 4 (2012), 213–224.

[63] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 950–961.

[64] Divya Mahajan, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, and Hadi Esmaeilzadeh. 2015. Prediction-based quality control for approximate accelerators. In *2nd Workshop on Approximate Computing across the System Stack (WACAS'15)*.

[65] Lawrence McAfee and Kunle Olukotun. 2015. EMEURO: A framework for generating multi-purpose accelerators via deep learning. In *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO'15)*. IEEE, 125–135.

[66] J. S. Miguel, M. Badr, and N. E. Jerger. 2014. Load value approximation. In *International Symposium on Microarchitectures*. 127–139.

[67] S. Misailovic, M. Carbin, S Achour, Z. Qi, and M. Rinard. 2014. Chisel: Reliability- and accuracy-aware optimizations of approximate computational kernels. In *International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA'14)*. 309–328.

[68] Asit K. Mishra, Rajkishore Barik, and Somnath Paul. 2014. iACT: A software-hardware framework for understanding the scope of approximate computing. In *Workshop on Approximate Computing Across the System Stack (WACAS'14)*. 52.

[69] Nikita Mishra. 2018. Personal communication.

[70] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. 2018. CALOREE: Learning control for predictable latency and low energy. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. Association for Computing Machinery, New York, NY, 184–198.

[71] Nikita Mishra, John D. Lafferty, and Henry Hoffmann. 2017. ESP: A machine learning approach to predicting application interference. In *Proceedings of the International Conference on Autonomic Computing (ICAC'17)*.

[72] Jongse Park, Hadi Esmaeilzadeh, Xin Zhang, Mayur Naik, and William Harris. 2015. Flexjava: Language support for safe and modular approximate programming. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 745–757.

[73] S. Park, F. Pfenning, and S. Thrun. 2005. A probabilistic language based on sampling functions. In *ACM Symposium on Principles of Programming Languages (POPL'05)*. 171–182.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python.*Journal of Machine Learning Research* 12 (2011), 2825–2830.

[75] Yan Pei, Swarnendu Biswas, Donald S. Fussell, and Keshav Pingali. 2019. SLAMBooster: An application-aware online controller for approximation in dense SLAM. In *28th International Conference on Parallel Architectures and Compilation Techniques (PACT'19)*. IEEE, 296–310.

[76] Yan Pei, Swarnendu Biswas, Donald S. Fussell, and Keshav Pingali. 2020. A methodology for principled approximation in visual SLAM. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques (PACT'20)*. Association for Computing Machinery, New York, NY, 373–386.

[77] A. Pfeffer. 2001. IBAL: A probabilistic rational programming language. In *International Joint Conference on Artificial Intelligence (IJCAI'01)*. 733–740.

[78] Abbas Rahimi, Luca Benini, and Rajesh K. Gupta. 2013. Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures. *IEEE Transactions on Circuits and Systems II: Express Briefs* 60, 12 (2013), 847–851.

[79] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K. Gupta. 2015. Approximate associative memristive memory for energy-efficient GPUs. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1497–1502.

[80] N. Ramsey and A. Pfeffer. 2002. Stochastic lambda calculus and monads of probability distributions. In *ACM Symposium on Principles of Programming Languages (POPL'02)*. 154–165.

[81] Ashish Ranjan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. 2015. Approximate storage for energy efficient spintronic memories. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC'15)*. IEEE, 1–6.

[82] Martin Rinard. 2006. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th Annual International Conference on Supercomputing*. ACM, 324–334.

[83] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H. Bailey, Costin Iancu, and David Hough. 2013. Precimonious: Tuning assistant for floating-point precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*. Association for Computing Machinery, New York, NY, Article 27, 12 pages.

[84] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision* 40, 2 (2000), 99–121.

[85] Mehrzad Samadi and Scott Mahlke. 2014. CPU-GPU collaboration for output quality monitoring. In *1st Workshop on Approximate Computing across the System Stack*. 1–3.

[86] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate data types for safe and general low-power computation. In *ACM Conference on Programming Language Design and Implementation (PLDI'11)*.

[87] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. 2014. Approximate storage in solid-state memories. *ACM Transactions on Computer Systems (TOCS)* 32, 3 (2014), 9.

[88] Byonghyo Shim, Srinivasa R. Sridhara, and Naresh R. Shanbhag. 2004. Reliable low-power digital signal processing via reduced precision redundancy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 5 (2004), 497–510.

[89] Stelios Sidiroglou, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. 2011. Managing performance vs. accuracy trade-offs with loop perforation. In *ESEC/FSE'11*.

[90] Xin Sui, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. 2016. Proactive control of approximate programs. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*. ACM, New York, NY, 607–621.

[91] Xin Sui, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. 2016. Proactive control of approximate programs. *ACM SIGOPS Operating Systems Review* 50, 2 (2016), 607–621.

[92] Rasha Tawhid and Dorina Petriu. 2008. Integrating performance analysis in the model driven development of software product lines. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 490–504.

[93] Mixed Integer Programming Tool. 2017. *Gurobi Optimizer 7.25*. http://www.gurobi.com/products/gurobi-optimizer.

[94] Stanford University. [n.d.]. *Convolutional Neural Networks for Visual Recognition*. http://cs231n.stanford.edu/index.html.

[95] Vassilis Vassiliadis, Konstantinos Parasyris, Charalambos Chalios, Christos D. Antonopoulos, Spyros Lalis, Nikolaos Bellas, Hans Vandierendonck, and Dimitrios S. Nikolopoulos. 2015. A programming model and runtime system for significance-aware energy-efficient computing. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 275–276.

[96] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. 2018. Understanding and auto-adjusting performance-sensitive configurations. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 154–168.

[97] Zhou Wang, Ligang Lu, and Alan C. Bovik. 2004. Video quality assessment based on structural distortion measurement. *Signal Processing: Image Communication* 19, 2 (2004), 121–132.

[98] R. Whaley, Antoine Petitet, and Jack Dongarra. 2001. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing* 27 (2001), 3–35. https://doi.org/10.1016/S0167-8191(00)00087-9

[99] Michael Joseph Wolfe. 1990. *Optimizing Supercompilers for Supercomputers*. MIT Press, Cambridge, MA.

[100] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. 2018. Videochef: Efficient approximation for streaming video processing pipelines. In *2018 USENIX Annual Technical Conference (USENIX ATC'18)*. 43–56.

[101] K. Zeng, Shi Gao, B. Mozafari, and C. Zaniolo. 2014. The analytical bootstrap: A new method for fast error estimation in approximate query processing. In *ACM SIGMOD'14*. 277–288.

[102] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In (*ASPLOS'16*).

[103] Hans Zima and Barbara Chapman. 1991. *Supercompilers for Parallel and Vector Computers*. ACM, New York, NY.

Received February 2021; revised July 2021; accepted July 2021