Machine Learning-Guided Design Space Exploration for Instruction-Level Approximate Computing

Gregório K. Neto[1], **Marco Antônio de R. Zárate**[1], **Gislayne G. Damasceno**[1], **Bruno Nogueira**[1], **Daniela Catelan**[1], **Ricardo R. Santos**[1], **Liana Duenha**[1]

[1]Faculty of Computing – Federal University of Mato Grosso do Sul (UFMS)
Campo Grande – MS – Brasil

{gregorio.neto, liana.duenha}@ufms.br

Abstract. *Approximate Computing (AC) trades reduced precision for energy savings in error-tolerant scenarios, but identifying the right instruction-level placement demands extensive, computationally prohibitive Design Space Exploration (DSE), further burdened by costly simulation-based evaluation. To address this, we propose a heuristic that reduces the design space via an energy-accuracy trade-off, combined with machine learning-based models that replace simulation during DSE. We introduced custom analysis passes to extract program features for these models, and trained predictors achieved* $R^2 \geq 0.955$ *for error estimation and* $R^2 \geq 0.986$ *for energy estimation across all benchmarks, with Spearman rank correlations at least* $\rho = 0.966$*, confirming that solution ordering is faithfully preserved against full simulation. Replacing simulation with ML-based prediction yielded mean speedups from* $\approx 23{,}987\times$ *to* $\approx 823{,}693\times$*, demonstrating that instruction-level approximate DSE can be made tractable without meaningful loss of solution quality.*

1. Introduction

The breakdown of Dennard scaling and the slowdown of Moore's Law have repositioned energy efficiency as the primary bottleneck in computing system design. Approximate Computing (AC) addresses this by trading strict precision for energy efficiency in error-tolerant applications [Mittal 2016], such as Machine/Deep Learning [Das et al. 2024] and Digital Signal and Image Processing [Bharti et al. 2022], whose noisy data or human-perceived outputs tolerate relaxed precision [Leon et al. 2025, Rasheed et al. 2025].

While AC has been explored from software techniques [Hoffmann et al. 2009, Akhlaghi et al. 2018] to inexact hardware circuits [Chen et al. 2015], a gap remains in its systematic exploration at the Instruction Set Architecture (ISA) level [Catelan 2025], selectively replacing exact instructions with approximate counterparts in general-purpose processors. This is non-trivial: indiscriminate substitution can degrade output quality to unacceptable levels despite the energy savings achieved.

This work presents a software infrastructure that generates an approximate application's best version from a fault-tolerant application and a set of approximate ISA instructions, meeting designer-specified accuracy or energy criteria. The following contributions have been completed and validated:

- A heuristic for dimensionality reduction of the multi-objective combinatorial optimization problem underlying the Design Space Exploration (DSE),

incorporating a weighting factor formulation that balances accuracy and energy reduction for flexible, designer-driven exploration;
- Machine learning-based predictors that evaluate candidate solutions within the DSE flow, replacing costly simulation and significantly reducing computational overhead;
- A proof-of-concept evaluation in a RISC-V ISA simulation environment extended to support approximate instructions, over the AxBench [Yazdanbakhsh et al. 2017] benchmark suite.

2. Instruction-Level Approximate Computing

2.1. Background

The adoption of instruction-level Approximate Computing requires a software ecosystem capable of reproducing hardware behavior while preserving the integrity of error models [Almurib et al. 2016]. This work adopts a methodology centered on the RISC-V ecosystem and the AxBench benchmark suite [Yazdanbakhsh et al. 2017].

RISC-V was selected as the target ISA due to its open and modular design, which provides dedicated custom opcode spaces that enable the integration of experimental instructions. SPIKE is the official reference simulator (*golden model*) for the RISC-V ISA, modeling architectural behavior with high execution speed. To support AC semantics, SPIKE was extended into a modified version named SPIKEX, which introduces five custom approximate instructions, `ADDX`, `SUBX`, `FADDX`, `FSUBX`, and `FMULX`, derived from low-power approximate arithmetic circuit designs InXA1 and APSC4 validated in [Catelan 2025]. Energy figures in this work are simulator-derived rather than physical measurements: SPIKEX accounts for the exact number of architectural cycles executed by each application variant and multiplies it by the dynamic and static energy costs pre-characterized per instruction (exact or approximate) via gate-level synthesis, as validated in Catelan's work [Catelan 2025], yielding the total energy of each candidate variant in Joules.

2.2. Approximate DSE Problem

The application of approximation techniques to a program generates a vast design space. Consider a program P containing n arithmetic instructions eligible for approximation $(i_1, i_2, ..., i_n)$. For each instruction i_k $(1 \leq k \leq n)$, a binary decision variable $x_k \in \{0, 1\}$ indicates whether the exact $(x_k = 0)$ or approximate $(x_k = 1)$ version of instruction i_k is used, resulting in a combinatorial space of 2^n possible variants of the same application.

The DSE problem addressed in this work is formally defined as a scalarized multi-objective optimization problem: identifying the configuration C, containing a mix of exact and approximate instructions, that maximizes energy efficiency while keeping output quality degradation within acceptable limits, as stated in Equation 1.

$$\begin{array}{ll} \text{Find} & C = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n \\ \text{That minimizes} & cost(C) \\ \text{Subject to} & Er_C \leq \mathcal{T} \end{array} \tag{1}$$

Where:

- C denotes a candidate configuration, in which each binary variable indicates whether instruction i_k ($1 \leq k \leq n$) is executed in its exact ($x_k = 0$) or approximate ($x_k = 1$) version;
- $cost(C)$ is the heuristic cost of candidate configuration C, defined as:

$$cost(C) = \alpha \cdot Er_C + (1 - \alpha) \cdot En_C$$

- $\alpha \in [0, 1]$ is a weighting parameter balancing output error and energy consumption during evaluation;
- Er_C and En_C are functions that quantify the normalized output error and energy consumption of configuration C, respectively;
- $\mathcal{T}$ represents the maximum admissible output error.

Improving energy efficiency typically increases computational error and vice versa [Deb et al. 2000], so candidate configurations represent different trade-offs along a Pareto frontier. By varying α, different regions of this frontier can be explored, while $\mathcal{T}$ prunes candidates whose error exceeds the maximum acceptable value.

2.3. Approximate DSE Framework

An initial offline preprocessing stage identifies the application hotspot, i.e., the function responsible for the largest fraction of the computational workload, by profiling all AxBench applications with the GNU Profiler (`gprof`) [Graham et al. 1982]. The proposed framework is a design-time tool: profiling, training, and exploration are performed offline, and the output is an approximate binary ready for deployment; a new application is handled by re-running the same offline flow.

All explored approximations are restricted to the selected function scope. Users annotate lines suitable for approximation with a `#pragma approx` tag, remaining responsible for excluding critical operations such as memory address computations and control-flow instructions. Annotations are applied at the line level rather than to entire blocks, a design choice regarding the granularity of approximation control that enables each approximated instruction to be controlled independently.

The workflow automates the generation, compilation, simulation, and result collection of application variants, and supports both an exact DSE strategy (exhaustive search, suitable only for low-dimensional spaces) and a heuristic-based exploration for larger search spaces. The workflow is driven by three inputs:

- **Annotated Source Code:** The processed file containing directives identifying instructions eligible for transformation.
- **Error Threshold ($\mathcal{T}$):** $\mathcal{T} \in [0, 1]$ defines the maximum acceptable output error.
- **α Factor:** The weighting coefficient introduced in Equation 1.

The execution flow then proceeds as follows: for an application with n annotated operations, 2^n source-code variants are generated and stored for subsequent processing. A variant queue is then built through a breadth-first traversal of the exploration tree, which can be explored either exhaustively or with threshold-based pruning. Each candidate is evaluated along one of two paths: In the simulation-based path, the variant is executed on SPIKEX to measure its output error and energy consumption. In the ML-based path, trained predictors estimate both metrics directly, bypassing simulation and substantially reducing the computational cost. In both cases, the cost of each solution is computed using the balancing factor α, as defined in Equation 1. Finally, a Pareto frontier is generated and the candidate solutions are ranked by $cost(C)$, allowing the designer to select the configurations that best satisfy the design criteria.

2.4. A Heuristic for Search-Space Pruning

The design space grows prohibitively large for real-world applications, making exhaustive search computationally intractable. Algorithm 1 models the search space as a decision tree, where the root represents the fully exact application and each descendant corresponds to a configuration generated by replacing additional exact instructions with approximate counterparts, pruning infeasible regions of the search space.

The exact application is first simulated to establish reference values for energy and error. Exploration then proceeds via queue-based traversal: for each candidate, normalized energy and error are evaluated by the evaluation function Evaluate($node$), and configurations exceeding the error threshold are marked *PRUNED*, discarding all descendants without further evaluation. Feasible configurations receive a heuristic cost combining normalized error and energy according to α, yielding a filtered tree that enables ranking and Pareto-based trade-off analysis. The level-wise organization of the process also enables parallel evaluation of candidates, improving scalability. The evaluation function is deliberately method-agnostic: it is instantiated by the simulator (Simulate) in the simulation-based path, or by the trained predictors (PredictEnergyAndError) in the ML-based path. Thus, Algorithm 1 is shared by both exploration strategies, which differ only in how each candidate is evaluated.

Algorithm 1: Pruned Search Tree for Instruction-Level DSE

Input: Root of a decision tree, representing the exact solution; Error threshold $\mathcal{T}$; Trade-off weight factor α.
Output: Filtered tree of feasible configurations or variants with associated cost values.

$(Er_{exact}, En_{exact}) \leftarrow$ Simulate($root$);
$F \leftarrow root.children$;
while F *is not empty* **do**
  Obtain batch $level$ by removing all elements from F;
  foreach $node \in level$ ***in parallel*** **do**
    $(Er_{node}, En_{node}) \leftarrow$ Evaluate($node$);
    $En \leftarrow \min(En_{node}/En_{exact}, 1.0)$;
    $Er \leftarrow \min(Er_{node}, 1.0)$;
    if $Er \leq \mathcal{T}$ **then**
      $node.cost \leftarrow (\alpha \times Er) + ((1-\alpha) \times En)$;
      $F.enqueue(node.children)$;
    else
      PruneDescendants($node$);
    end
  end
end
return Filtered tree of feasible configurations or variants;

3. Accuracy Predictor Design

3.1. Static Analysis for Feature Extraction

Part of the dataset features were extracted through static analysis of the application source code using the LLVM-MLIR (Multi-Level Intermediate Representation) infrastructure, with the objective of identifying structural and semantic properties of instructions eligible for approximation that may influence error propagation and the quality of the final application output.

Three data-flow analysis techniques [Aho et al. 2006] were adopted as independent variables to estimate approximation opportunities. *Liveness analysis* identifies computations whose results have limited influence on subsequent operations, indicating reduced impact on final outputs and thus safer approximation candidates. *Error propagation analysis* estimates how approximation errors introduced into a computation amplify or attenuate as they spread across dependent operations [Barros et al. 2021]. *Loop-aware data-flow analysis* [Muchnick 1997] extends these techniques to iterative regions, which typically dominate application runtime and repeatedly propagate arithmetic values, making the behavior of approximation errors within loops a key factor for identifying efficient approximation opportunities.

3.2. Dataset Design

The list below summarizes the feature groups extracted to compose the dataset used for training the error prediction model. These groups describe multiple structural and behavioral aspects of approximate code regions, including dynamic instruction information, MLIR-based static analysis metrics, distances between approximate operations and function returns, criticality and dependency characteristics, and structural properties extracted from expression trees. Together, these features provide a multidimensional representation of candidate approximate configurations, enabling the learning model to estimate the impact of approximation decisions on application output quality and energy consumption.

- **Line Encoding**: One-hot binary representation indicating which source-code lines contain approximated instructions, enabling the model to correlate spatial approximation patterns with output quality and energy impact.
- **Dynamic Features**: Counts of approximate operations by type (additions, subtractions, multiplications), reflecting the computational intensity of the approximation.
- **Region Properties**: MLIR-extracted structural descriptors of the approximate region, including variable usage counts, tainted-input flags, expected static error, loop count, nesting structure, and global variable statistics.
- **Distance to Return**: Average and minimum distance (in basic blocks) from approximate operations to the function return, computed at four abstraction levels: MLIR report, intermediate representation (IR), source code, and control-flow graph (CFG).
- **Depth in the Expression Tree**: Sum, average, and maximum depth of approximate operations in the type and argument trees, indicating how deeply errors may propagate.

For each benchmark, approximate configurations were generated through binary combinations of the annotated operations. For the four smaller benchmarks (512 to 8,192 variants), the design space was enumerated completely, a deliberate choice that provides full ground truth for validating the predictors. For Black-Scholes, whose 26 eligible lines induce a space of $2^{26} \approx 6.7 \times 10^7$ configurations, exhaustive enumeration is infeasible; the dataset instead comprises a structured sample of 35,119 variants of the original code ($\approx 0.05\%$ of the space), covering all single-line variants, nearly all two- and three-line combinations, and the complete enumeration of a 15-line subspace. Each generated variant was simulated to collect application-specific error metrics, such as the

mean absolute relative error (MARE), miss rate, or root mean squared error (RMSE) (following [Yazdanbakhsh et al. 2017]), in addition to runtime-related metrics including execution cycles and energy consumption. These datasets were subsequently used to train and evaluate the error prediction models associated with each application category. Producing the dataset via simulation required substantial time across benchmarks, totaling over 360 hours of simulation.

3.3. Model Architecture

Different regression algorithms were evaluated to predict the error of approximate variants. We restricted our evaluation to tree-based ensemble regressors, which represent the state-of-the-art for tabular regression with mixed numerical and binary features and consistently outperform deep neural networks at the dataset scales considered here (hundreds to tens of thousands of samples per benchmark) [Borisov et al. 2024]. Within this family, three representative ensemble strategies were selected to cover the main approaches to variance control and bias-variance trade-off: gradient boosting (LightGBM [Ke et al. 2017]), bagging with bootstrapped trees (Random Forest [Breiman 2001]), and extremely randomized trees (Extra-Trees [Geurts et al. 2006]).

All models were trained using early stopping to mitigate overfitting. The datasets were divided into 80% for training and 20% for testing, using stratification by failure status whenever applicable. Approximate variants may collapse into a failure state, producing outputs with huge accumulated errors or NaN (Not-a-Number) values; such variants are labeled *failed*. Benchmarks with a bimodal error distribution and a saturation peak (Black-Scholes and Inversek2j) employed a two-stage prediction pipeline: a binary classifier first predicts whether a variant fails, and only non-failed variants are forwarded to a regression model that estimates the continuous error magnitude. Benchmarks with unimodal error distributions (FFT, Jmeint, and K-means) were handled by a single regression model, since no explicit failure boundary separates the target space. Note that one surrogate is trained per application, with the train/test split performed within that application's configuration space.

3.4. Feature Analysis

The feature groups aggregate more than forty descriptors, raising the question of how many are actually required for accurate prediction. We assessed the catalog using pairwise Pearson correlation, permutation importance [Breiman 2001], and an ablation study retraining the predictor on progressively smaller, importance-ranked feature subsets. On Black-Scholes, selected for its largest dataset (35,119 variants), correlation analysis revealed limited redundancy (13 of the 351 pairs among the 27 descriptors with non-constant values on Black-Scholes exhibit $|\rho| > 0.9$), justifying the retention of closely related descriptors that capture complementary structural information. Figure 1(a) shows the top-15 features by permutation importance for Black-Scholes, a ranking stable across validation splits and combining MLIR-derived structural metrics (`depth_t_mean`, `dist_to_return_ir_mean`) with dynamic descriptors of the approximate region (`fmulx_count_variaveis`, `num_approx_operations`); extending this to the top-10 features per application, `uses_global_sum` appears across all five benchmarks, with three other features appearing in four out of five, indicating a small cross-application

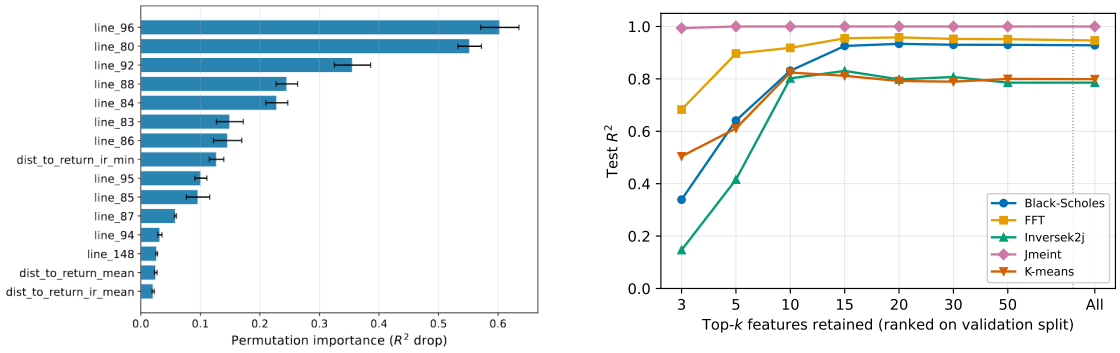

Figure 1. (a) Top-15 features by permutation importance on Black-Scholes. (b) Test R^2 as a function of the top-k features, reported on the untouched test set.

predictive backbone. The ablation in Fig. 1(b) generalizes these findings: Jmeint saturates at $k = 3$–5, K-means and Inversek2j plateau at $k \approx 10$, and FFT and Black-Scholes approach asymptotic accuracy at $k \approx 15$–20 (Black-Scholes top-20 even slightly outperforming the full set, $R^2 = 0.934$ vs. 0.928), indicating that the catalog could be pruned to the top-20 features per benchmark without measurable accuracy loss, although all predictors used in this paper retain the full catalog.

4. Evaluation of the ML-Guided DSE

4.1. Predictive Models Evaluation

Table 1 summarizes the predictive performance of the evaluated regression models for error and energy estimation. Model quality is assessed using the coefficient of determination (R^2) together with the mean absolute error (MAE) for error prediction and the mean absolute percentage error (MAPE) for energy prediction. MAE was adopted for error prediction because the evaluated quality metrics are already relative or normalized quantities. Energy consumption, in contrast, varies by several orders of magnitude across benchmarks and configurations; MAPE expresses the deviation as a percentage, allowing predictive quality to be assessed independently of each application's absolute energy scale. All metrics are computed on the held-out test set (20% of the dataset).

The regression models for both error and energy prediction achieved high coefficients of determination across all benchmarks, demonstrating the effectiveness of the proposed approach (Table 1). For error prediction, the LightGBM models reached R^2 values above 0.95, indicating a strong fit between predicted and actual error values. The *Jmeint* benchmark yielded near-perfect predictions, with $R^2 = 0.999$ and MAE = 0.002, suggesting that its error behavior is highly regular and predictable. The *K-means* benchmark also achieved high predictive accuracy, with $R^2 = 0.955$ and MAE = 0.024. The *Black-Scholes* and *FFT* benchmarks also performed well, achieving R^2 values of 0.962 and 0.957 with low MAE. The *Inversek2j* benchmark, despite its R^2 of 0.956, presented the highest MAE among the group (0.242), which may reflect a greater intrinsic variability in its error response to code transformations. Notably, *Black-Scholes* and *Inversek2j* required a two-stage prediction architecture, while all other benchmarks were handled with a direct model, suggesting that their approximation space exhibits a more complex structure.

For energy prediction, all benchmarks were modeled using a direct architecture targeting energy consumption in μW·s. The results are consistently strong, with R^2 values ranging from 0.986 to 0.9997. *Jmeint* stood out with the highest R^2 (0.99972) and the lowest relative errors (MAPE $= 0.00005\%$). The *Inversek2j* benchmark showed the largest MAPE (2.43%), yet still maintained an R^2 of 0.986, indicating that while the relative prediction error is higher, the model still captures the overall energy trend effectively. The remaining benchmarks (*Black-Scholes*, *FFT*, and *K-means*) achieved MAPE values below 0.30%, reflecting highly accurate energy predictions.

Overall, the results indicate that the regression models generalize well across diverse benchmarks and approximation targets, with only minor variations in predictive performance attributable to the complexity and variability of each application's behavior under approximate computing transformations. Based on these results, LightGBM was selected as the predictor integrated into all DSE experiments reported in Section 4: it achieves the highest average R^2 across applications for error prediction (mean 0.966, against 0.908 for Random Forest and 0.918 for Extra-Trees, dominated by the strong difference on Inversek2j), and competitive accuracy for energy prediction (mean $R^2 = 0.995$). Combined with its $\approx 10\times$ shorter training time relative to the bagging-based alternatives under the same hyperparameter budget, this makes LightGBM the most computationally affordable choice for iterative usage within the DSE loop.

Table 1. Regression models for error and energy prediction. (*) R^2 is reported along with MAE (error) and MAPE (energy) on the held-out 20% test set.

Benchmark		LightGBM		RF		Extra-Trees	
		R^2	Metric (*)	R^2	Metric (*)	R^2	Metric (*)
Black-Scholes	Error	0.962	0.033	0.973	0.014	0.965	0.011
	Energy	0.998	0.058	0.999	0.018	0.998	0.015
FFT	Error	0.957	0.040	0.941	0.041	0.976	0.024
	Energy	0.996	0.292	0.995	0.259	0.997	0.162
Inversek2j	Error	0.956	0.242	0.628	0.591	0.650	0.405
	Energy	0.986	2.434	0.977	2.099	0.997	0.408
Jmeint	Error	0.999	0.002	1.000	0.000	1.000	0.000
	Energy	0.999	0.000	1.000	0.000	1.000	0.000
K-means	Error	0.955	0.024	1.000	0.001	1.000	0.001
	Energy	0.995	0.040	0.992	0.044	0.999	0.014

Since exhaustive simulation was employed here only to enable a ground-truth validation of the predictors, a natural question is how much simulation the method actually requires. To quantify this, we retrained the predictors on randomly sampled fractions of each design space (stratified by failure status, averaged over three seeds) and evaluated them on the entire remaining space. Table 2 summarizes the results. With only 25% of the configurations, the Black-Scholes predictors reach $R^2 = 0.87/0.99$ (error/energy), cutting dataset generation from 234.5 to under 59 hours, and with 50% they approach the accuracy obtained with the full dataset; the failure classifiers exceed 99% accuracy with as little as 2% of the space. In absolute terms, a few thousand simulated samples suffice, a fixed budget that becomes a vanishing fraction of the space as program size grows, precisely the regime targeted by this work. K-means is the exception, reflecting its narrow RMSE range and small design space (512 variants); spaces of this size, however, remain cheap to simulate exhaustively. Residual borderline mispredictions

can be eliminated by re-simulating only the predicted top-k configurations, at negligible cost.

A complementary leave-one-line-out analysis, in which the predictors are trained with all variants containing a given source line removed and tested exclusively on them, indicates that extrapolation to never-approximated lines is feasible in the richer design spaces (median R^2 of 0.90–1.00 for Black-Scholes and Jmeint), as the static features capture the structural contribution of the unseen line, while it remains unreliable for the smaller benchmarks. Extending this property to arbitrary program regions is left as future work.

Table 2. Predictor accuracy (R^2 error/energy) when trained on sampled fractions of the design space and evaluated on the entire remaining space (mean of three random seeds).

Benchmark	10%	25%	50%
Black-Scholes	0.75 / 0.98	0.87 / 0.99	0.93 / 0.99
FFT	0.81 / 0.87	0.88 / 0.98	0.93 / 0.99
Inversek2j	0.57 / 0.85	0.79 / 0.93	0.90 / 0.97
Jmeint	0.99 / 0.99	1.00 / 1.00	1.00 / 1.00
K-means	— / 0.88	0.33 / 0.88	0.47 / 0.95

—: For K-means at the 10% fraction, the error predictor yields a negative R^2 (i.e., worse than predicting the mean), as the training sample (≈51 variants) is too small for a meaningful model. The entry therefore reports energy prediction only.

4.2. Integrating predictors into the DSE flow

Based on the predictor evaluation results (Section 4.1), the LightGBM model was adopted as the most suitable model for integration into the DSE flow. This section presents the results of the experimental evaluation of the proposed machine learning-guided DSE across the five benchmarks of the AxBench suite. All experiments were conducted on a machine equipped with an AMD Ryzen 9 5900XT processor, 32GB of DDR4 RAM, and an NVIDIA GeForce RTX 3060 Ti GPU. For each application, the top-ranked solutions obtained by the prediction-based strategy are compared against those produced by the simulation-based approach, considering four values of the balancing factor α (0.1, 0.3, 0.5, and 0.7) and a fixed error threshold $\mathcal{T} = 0.5$. For the speedup analysis of Section 4.4, the error threshold is additionally swept over $\mathcal{T} \in [0.1, 1.0]$. The comparison aims to assess whether replacing the costly simulation process with ML-based predictors preserves the quality and ordering of the solutions identified by the DSE framework.

4.3. Quality of Prediction-Based DSE Solutions

Before discussing per-application rankings, Table 3 reports the aggregate agreement between predicted and simulated costs on the held-out test set, combining the four values of α. Across all benchmarks, the Spearman rank correlation $\rho \geq 0.966$ confirms that the predictor preserves the simulator-induced ordering, which ultimately drives DSE decisions. The lower R^2 of K-means (0.888) reflects its narrow target range (RMSE in $[5.13, 5.30]$) rather than ranking instability ($\rho = 0.974$), while Inversek2j (0.921) inherits residual variance from the two-stage classifier's NaN-rate boundary. In both cases the absolute MAE remains below 0.025. Table 4 presents the top-3 solutions for all benchmarks, across all four values of α, with solutions identified by the approximated source-code lines and ranked by $cost(C)$.

Table 3. Agreement between predicted and simulated costs on the held-out test set, aggregated over $\alpha \in \{0.1, 0.3, 0.5, 0.7\}$.

Benchmark	n	R^2	MAE	Spearman ρ
Black-Scholes	28,096	0.975	0.007	0.973
FFT	820	0.987	0.007	0.996
Inversek2j	6,504	0.921	0.024	0.966
K-means	412	0.888	0.006	0.974
Jmeint	412	1.000	0.000	0.999

Black-Scholes: The prediction-based and simulation-based strategies produced identical top-3 rankings in every case, with deviations below 0.005, confirming that the predictors faithfully reproduce the simulator-induced ordering. The relative ranking also remained stable across α, with $[89]$ consistently first, followed by $[81]$ and $[81, 89]$, indicating that α affects the magnitude of the cost but not the identification of the best configuration, with line 89 offering the most favorable error-energy trade-off.

FFT: The prediction-based approach consistently identified $[65]$ as the best solution across all α values, matching the simulation-based results. Minor discrepancies appeared in lower positions ($[73]$ and $[65, 73]$ swapped order, for example), but the associated cost differences were negligible, indicating marginal prediction uncertainty rather than meaningful quality gaps.

Inversek2j: Both strategies agreed on the top solution for $\alpha = 0.1$ and 0.3, with only minor position swaps among near-tied costs ($[49]$ and $[64]$ at 0.8964, for example). For $\alpha = 0.5$ and 0.7, positions 2 and 3 were swapped between $[64]$ and $[49, 64]$, again with negligible cost differences. Overall, the predictors reliably identify the top-ranked solution across all α values.

Jmeint: A distinctive trait of this benchmark is that all top-ranked solutions share identical heuristic costs across every α, meaning any individual or combined subset of candidate lines yields equivalent error-energy trade-offs; rankings thus reflect arbitrary ordering among equally optimal solutions. Note that the specific lines selected differ between the strategies: prediction-based favoring multi-line configurations and simulation-based favoring single-line ones. Nevertheless, identical cost values confirm that both strategies identify the same optimal region, with cost differences on the order of 10^{-5}.

K-means: Unlike the other benchmarks, only a single viable solution, $[48]$, was found within $\mathcal{T} = 0.5$ by both strategies, suggesting that K-means is not a favorable candidate for instruction-level approximation under these constraints. The prediction-based strategy identified this same solution across all α values, with cost deviations below 0.001, confirming the predictor generalizes correctly even in highly constrained design spaces.

4.4. Speedup Analysis

We measured the execution time of the DSE flow with prediction against the same flow with simulation for all applications, sweeping the error threshold $\mathcal{T}$ from 0.1 to 1.0. The results demonstrate substantial gains in all cases, ranging from $23{,}987\times$ for

Table 4. Top-3 solutions for all benchmarks ranked by the heuristic cost using different balancing factor α and $\mathcal{T} = 0.5$.

Benchmark	α	Rank	Prediction-based		Simulation-based	
			Approx. Lines	Cost	Approx. Lines	Cost
Black-Scholes	0.3	1	[89]	0.6994	[89]	0.7006
		2	[81]	0.8189	[81]	0.8164
		3	[81,89]	0.8225	[81,89]	0.8168
	0.5	1	[89]	0.4984	[89]	0.5004
		2	[81]	0.7194	[81]	0.7177
		3	[81,89]	0.7246	[81,89]	0.7180
	0.7	1	[89]	0.2975	[89]	0.3002
		2	[81]	0.6200	[81]	0.6190
		3	[81,89]	0.6268	[81,89]	0.6192
FFT	0.3	1	[65]	0.7450	[65]	0.6981
		2	[73]	0.7562	[65,73]	0.7006
		3	[65,73]	0.7589	[73]	0.7008
	0.5	1	[65]	0.5768	[65]	0.4986
		2	[73]	0.5927	[65,73]	0.5004
		3	[65,73]	0.5977	[73]	0.5006
	0.7	1	[65]	0.4087	[65]	0.2992
		2	[73]	0.4292	[65,73]	0.3003
		3	[65,73]	0.4364	[73]	0.3003
Inversek2j	0.3	1	[49]	0.7102	[49]	0.7000
		2	[49,64]	0.7863	[49,64]	0.7521
		3	[64]	0.7883	[64]	0.7521
	0.5	1	[49]	0.5170	[49]	0.5000
		2	[64]	0.6588	[49,64]	0.6078
		3	[49,64]	0.6727	[64]	0.6078
	0.7	1	[49]	0.3239	[49]	0.3000
		2	[64]	0.5293	[49,64]	0.4635
		3	[49,64]	0.5590	[64]	0.4635
Jmeint	0.3	1	[28,29,30,32,33,34]	0.7000	[28]	0.7000
		2	[32,34]	0.7000	[29]	0.7000
		3	[28,29,33]	0.7000	[30]	0.7000
	0.5	1	[28,29,30,32,33,34]	0.5000	[28]	0.5000
		2	[32,34]	0.5000	[29]	0.5000
		3	[28,29,33]	0.5000	[30]	0.5000
	0.7	1	[28,29,30,32,33,34]	0.3000	[28]	0.3000
		2	[32,34]	0.3000	[29]	0.3000
		3	[28,29,33]	0.3000	[30]	0.3000
K-means	0.3	1	[48]	0.7003	[48]	0.6997
	0.5	1	[48]	0.5002	[48]	0.4998
	0.7	1	[48]	0.3001	[48]	0.2999

Inversek2j to $823{,}693\times$ for K-means. Black-Scholes and FFT achieved mean speedups of $67{,}610\times$ and $40{,}827\times$, respectively, while Jmeint reached $417{,}100\times$. These results confirm that replacing simulation with ML-based predictors reduces exploration time by several orders of magnitude, making the proposed approach highly practical for large-scale DSE workflows.

The particularly high speedups observed for Jmeint and K-means are explained by the high approximation errors characteristic of these applications. Since the same threshold range was applied across all experiments, the error-based pruning mechanism discarded a large fraction of candidate configurations early in the exploration, significantly reducing the number of predictor evaluations required. Consequently, applications with fewer viable solutions benefit proportionately from pruning, as the DSE flow terminates earlier and the contrast with the simulation-based approach, which must evaluate each configuration individually, becomes more pronounced. The speedups above refer to the exploration phase. Accounting for the full up-front cost does not change the conclusion: with sampled training (Table 2), dataset generation for Black-Scholes drops to under 59 hours, while a single simulation-based exploration sweep must simulate configurations at the same cost scale as dataset generation itself. The up-front investment is therefore recovered within the first exploration query, and every subsequent query, a different α, $\mathcal{T}$, or Pareto sweep, reuses the trained predictors at negligible cost.

5. Related Work

Chisel [Misailovic et al. 2014] automated the binary selection of exact versus approximate instructions via integer linear programming with formal reliability constraints, targeting hardware with built-in probabilistic unreliability. At the circuit level, autoAx [Mrazek et al. 2019] introduced ML-based surrogates to estimate quality and hardware cost, guiding Pareto search over approximate circuit libraries without exhaustive synthesis, which is the same surrogate-model motivation as our work, but for custom accelerators rather than a general-purpose ISA. ApproxTuner [Sharif et al. 2021] extended this idea to software, using compositional accuracy models to speed up tuning of tensor-based CNN applications by 12–20$\times$ over empirical search, though without addressing ISA-level approximation. Building on our prior work [Neto et al. 2025], which introduced branch-and-bound DSE for RISC-V but required full simulation per candidate, the present framework replaces simulation with LightGBM predictors trained on LLVM/MLIR static features, making instruction-level approximate DSE tractable at previously unreachable scales.

6. Conclusion

This work presented a machine learning-guided DSE framework for instruction-level Approximate Computing on RISC-V processors, combining a pruned heuristic search with ML-based predictors to efficiently navigate exponentially growing configuration spaces. The DSE problem is formulated as a constrained scalarized multi-objective optimization, balancing output quality and energy consumption via a weighting factor α and admissibility threshold $\mathcal{T}$, with a breadth-first pruning heuristic eliminating subtrees whose error exceeds $\mathcal{T}$, and LLVM/MLIR-based static analysis enriching the predictor's feature set with the application structural properties.

LightGBM was selected as the primary predictor for its superior generalization and $\sim$10$\times$ shorter training time. All benchmarks achieved $R^2 \geq 0.955$ for error and 0.986–0.9997 for energy prediction, with MAPE below 2.5%. Integrated into the DSE flow and evaluated against simulation across four values of α (0.3, 0.5, 0.7) with $\mathcal{T} = 0.5$, the predictors achieved Spearman correlations of at least $\rho = 0.966$ in all benchmarks, exactly matching top-ranked solutions in four of five cases (deviations below 0.005), and yielded mean speedups from $\approx$ 23,987$\times$ to $\approx$ 823,693$\times$, reducing exploration from hundreds of hours to seconds without meaningful loss of solution quality.

Future work includes improving predictor robustness for edge cases, investigating adaptive threshold strategies to mitigate over-pruning, training the predictors from small sampled subsets of the design space, as well as quantitatively comparing our predictors against alternative ML-based DSE approaches (e.g., reinforcement learning [Saeedi et al. 2023]) and validating the framework on larger and more complex testbenches beyond AxBench.

References

Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*. Pearson Education, Boston, 2nd edition.

Akhlaghi, V., Yazdanbakhsh, A., Samadi, K., Gupta, R. K., and Esmaeilzadeh, H. (2018). Snapea: Predictive early activation for reducing computation in deep convolutional

neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 662–673. IEEE.

Almurib, H. A., Kumar, T. N., and Lombardi, F. (2016). Inexact designs for approximate low power addition by cell replacement. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 660–665. IEEE.

Barros, G., Akemaru, R., Bezerra, E., Oliveira, R., and Mazzine, M. (2021). The synthesis of computational imprecision: Impact of compilation techniques on the approximate computing paradigm. *ACM Computing Surveys (CSUR)*, 54(8):1–35.

Bharti, S., Kumar, A., and Gupta, P. (2022). Review of approximate computing in image processing applications. In *2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST)*, pages 1–6.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. (2024). Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(6):7499–7519.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Catelan, D. L. (2025). *Approximate computing: Contributions to the design of arithmetic circuits and instruction-set architectures*. Tese de doutorado, Universidade Federal de Mato Grosso do Sul (UFMS).

Chen, L., Han, J., Liu, W., and Lombardi, F. (2015). Design of approximate unsigned integer non-restoring divider for inexact computing. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 51–56.

Das, A., Ghosh, S. K., Raha, A., and Raghunathan, V. (2024). Toward energy-efficient collaborative inference using multisystem approximations. *IEEE Internet of Things Journal*, 11(10):17989–18004.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International Conference on Parallel Problem Solving From Nature*, pages 849–858. Springer.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63(1):3–42.

Graham, S. L., Kessler, P. B., and Mckusick, M. K. (1982). Gprof: A call graph execution profiler. *SIGPLAN Not.*, 17(6):120–126.

Hoffmann, H., Misailovic, S., Sidiroglou, S., Agarwal, A., and Rinard, M. (2009). Using code perforation to improve performance, reduce energy consumption, and respond to failures. Technical report, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Leon, V., Hanif, M. A., Armeniakos, G., Jiao, X., Shafique, M., Pekmestzi, K., and Soudris, D. (2025). Approximate computing survey, part i: Terminology and software & hardware approximation techniques. *ACM Computing Surveys*, 57(7):1–36.

Misailovic, S., Carbin, M., Achour, S., Qi, Z., and Rinard, M. C. (2014). Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels. *ACM Sigplan Notices*, 49(10):309–328.

Mittal, S. (2016). A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):1–33.

Mrazek, V., Hanif, M. A., Vasicek, Z., Sekanina, L., and Shafique, M. (2019). autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 1–6. ACM.

Muchnick, S. S. (1997). *Advanced Compiler Design and Implementation*. Morgan Kaufmann.

Neto, G. K., Sovernigo, F., Catelan, D., Santos, R., and Duenha, L. (2025). Heuristics for energy-efficient instruction-level approximate computing. In *2025 IEEE/SBC 37th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 35–45. IEEE.

Rasheed, Y., Anwar, M. W., Gillani, G. A., and Ottavi, M. (2025). Toward approximate computing for deep learning in embedded systems: A systematic literature review. *IEEE Access*, 13:210863–210891.

Saeedi, S., Savino, A., and Di Carlo, S. (2023). Design space exploration of approximate computing techniques with a reinforcement learning approach. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*.

Sharif, H., Zhao, Y., Kotsifakou, M., Kothari, A., Schreiber, B., Wang, E., Sarita, Y., Zhao, N., Joshi, K., Adve, V. S., et al. (2021). Approxtuner: a compiler and runtime system for adaptive approximations. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 262–277.

Yazdanbakhsh, A., Mahajan, D., Esmaeilzadeh, H., and Lotfi-Kamran, P. (2017). Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design and Test*, 34(2):60–68.