Less Noise, More Signal: the DRR Effect for Better Optimizations of a Range of SE Tasks

Andre Lustosa, Tim Menzies, *Fellow, IEEE*

Abstract— SE analytics problems do not always need complex AI. Better and faster solutions can sometimes be obtained by matching the complexity of the problem to the complexity of the solution. This paper introduces the Dimensionality Reduction Ratio (DRR) effect, a frequently observed empirical effect indicating where effective optimization might be two orders of magnitude faster.
The DRR effect is an empirical observation, not some universal law. With the datasets used in this study, we can comment on SE tasks expressible as classification or regression tasks (where regression may be exploring $N \geq 1$ goals). These tasks include software configuration optimization; cloud resource management; project health prediction (commits, PRs, issues); and process models (effort/defect/schedule estimation). Given the prevalence of the DRR effect in this sample, we conjecture it might hold for other SE tasks, but that is a matter for further research. Hence we recommend practitioners check for high DRR before deploying expensive optimization methods. This simple diagnostic could save orders of magnitude in computational cost.

1 Introduction

When Software Engineering (SE) data grows too large, data mining algorithms can find the signal in the noise. Such algorithms are controlled by "hyperparameters"; e.g.

- When learning k clusters, k is a hyperparameter;
- When learning decision trees, the maximum allowed height of the tree is another hyperparameter.

Finding good hyperparameters is something of a black art. *Hyperparameter optimizers* (HPO) are tools for automating that search. HPO can dramatically improve learner performance [27]–[40]. For example, for code smell detection, Yedida & Menzies found that a decades-old feedforward neural net (which takes seconds to run, so HPO is fast) can be tuned to outperform a state-of-the-art deep learner [40].

But there is a problem. HPO requires running a learner many times. Hence, it can be impractically slow; e.g. Yedida & Menzies could not tune their state-of-the-art deep learner since each run of that learner needed eight hours to complete. Also, there are so many HPO methods [41] that practitioners can get confused about which one to use.

To solve these two problems, we propose

Selecting algorithms via intrinsic problem complexity.

That is, selecting which *HPO method* (e.g., a simple vs. a complex one) to use. Here, "HPO methods" are optimizers that tune a *learner* (e.g., a random forest) to build a better *predictor* (a model trained to make predictions).

We show that *intrinsic complexity* tells us how to speed up HPO for improving predictors for a wide range of SE tasks (such as those listed in Figure 3.2). The *Dimensionality Reduction Ratio* (hereafter, DRR) effect checks if R attributes can be reduced to $I < R$ "intrinsic" underlying attributes. DRR can be easily and quickly calculated from the data used to train a predictor. This means that DRR can offer a signal about new problems without elaborate or expensive algorithms. We show that, often, when

$$(DRR = (1 - I/R)) > 0.35 \quad (1)$$

then very simple methods can (e.g.) find optimal hyperparameters that seek best parameters for SE regression tasks, two orders of magnitude faster than standard state-of-the-art AI optimizers (seconds, as opposed to 20 minutes). This is a significant result since many SE datasets satisfy Equation 1. For example, of the SE data in Figure 3.2, $\frac{38}{50} = 76\%$ of them satisfy Equation 1.

Another important aspect of Equation 1 is that it significantly improves on a recent IEEE TSE publication. Agrawal et al. [34] studied HPO and intrinsic dimensionality. After experimenting with older AI algorithms, Agrawal et al. recommended simpler algorithms when $I < 4$. But our results (which studied more of the state-of-the-art) finds many counter examples to their threshold rule. That is, our Equation 1 fixes numerous errors in prior work.

We note that DRR is not a *universal law* but rather a robust *empirical effect*; i.e. a recurring pattern observed across diverse SE optimization domains. In this paper, we study 50 datasets relating to SE tasks expressible as classification or regression tasks (where regression may be exploring $N \geq 1$ goals). These SE tasks include software configuration optimization; cloud resource management; project health prediction (commits, PRs, issues); and process models (effort/defect/schedule estimation). Given the prevalence of the DRR effect in the datasets we have studied, we conjecture it might hold for other tasks (but that is a matter for further research).

In the philosophy of science, *scientific laws* and *empirical effects* have a different epistemological status. A scientific law describes universal and invariable patterns of natural phenomena, often expressed mathematically, that predict outcomes in a defined domain [42], [43]. In contrast, empirical effects characterize frequently observed regularities that prove useful even when their precise boundaries remain incompletely understood [44].

- *A. Lustosa and T. Menzies are with the Department of Computer Science, North Carolina State University, Raleigh, USA. E-mail:alustos@ncsu.edu, timm@ieee.org*

SE-data
SS-Models (SS-A through SS-X): All of the models with this nomenclature in this repository, including the rs-6d-c3-obj2 dataset were obtained from the software configuration literature [1]. The data was collected via running different software projects configured in different ways (selected at random) and then collecting different performance metrics (runtimes, cpu usage, etc.). The goal of these datasets is to find a configuration of the software that best optimizes the overall software goals for each specific project.
POM3 (A,D): The POM3 model is a tool for exploring the management challenge of agile development balancing idle rates,completion rates and overall cost. More specifically: • In the agile world, projects terminate after achieving a completion rate of $X\%$ ($X < 100$) of its required tasks; • Team members become idle if forced to wait for a yet-to-be finished task from other teams; • To lower the idle rate and improve the completion rate, management can hire staff, but this increases the overall cost. The POM3 model simulates the Boehm and Turner model of agile programming, and has been used before in the literature [2]. In the models used in this study Pom3a is the simpler of the two.
XOMO (Flight, Ground, OSP, OSP2, nasa93dem): XOMO [3] introduced by Menzies et al. is a general framework for Monte Carlo simulations that combines four COCOMO-like software process models from Boehm's group at the University of Southern California. The overall goals for XOMO are to: • Reduce risk; • Reduce effort; • Reduce defects; • Reduce development time. The available XOMO models here all come from NASA's Jet Propulsion Laboratory. Where Flight and Ground are general descriptions of all JPL's flight and ground software. While OSP and OSP2 are two versions of the flight guidance system of the Orbital Space Plane. In terms of complexity we know that nasa93dem is the simpler of the five, followed by OSP and OSP2, which are similar, then Ground being a bit more complex and finally Flight being the most complex of all.
Health (Easy, Hard): The Health datasets show results where random forest regression algorithms were configured to predict for number of (a) commits or (b) closed issues or (c) close pull requests in 12 months time in open source projects hosted on GitHub. The Y values of these datasets show the results of the predictions after certain hyperparameters were applied to the random forests (which, in turn, were applied to the GitHub data). This data was originally used in the niSNEAK [4] study in a hyperparameter optimization context.

Non-SE-data For comparison purposes, our results from the above SE tasks are compared to the following non-SE problems.
Wine Quality: This dataset is well-suited for educational purposes as its subject matter is familiar to a wide audience.
Adult: This dataset [5] is used to predict whether the annual income of an individual exceeds \$50 K / year based on census data. It is also commonly known as the "Census Income" dataset [6].
Default: This dataset [7] collects data related to customers' payment defaults in Taiwan and is used to predict whether or not a certain customer will default on their payment in the following month [8].
German Credit: This dataset [9] classifies people described by a set of attributes as good or bad credit risks [8].
Iris: This dataset [10] is a classic non-Software Engineering dataset from 1936 used to classify flowers based on their external characteristics [11].
Heart Disease: This dataset [12] provides different health metrics of patients with the goal of predicting the presence of heart disease in a given patient [13].
Diabetes: This dataset [14] provides different health metrics of patients across time that can be used as a predictor for the presence or not of the disease in that patient [15].
Bank Marketing: This dataset [16] is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The purpose of this dataset is to predict whether a client will agree to a term deposit [17].
Gamma Telescope: This dataset [18] offers measures of cosmic events directed towards deciding whether the observed event is a signal or just background noise [19].
Power Consumption: This dataset [20] provides measurements of electric power consumption in a household with a one minute sampling rate over a period of almost 4 years. It can be used to predict sub-metering values [21].
Behavioral Data: The datasets, Player Statistics, Student Dropout, Employee Attrition, All players provide datasets to analyze and predict behavioral patterns in multiple scnearios [22]–[25].

TABLE 1: The goal of this paper is to automatically tune a regression-based learner to make predictions about the numeric classes seen in this data. The SE-data (on the left) comes from recent SE papers. The Non-SE data (on the right) comes from the UCI machine learning repository [26] and is used in a large number of machine learning papers.

Dataset :	Original Dimensions
SS-A —— SS-X (23 datasets) :	3-88
iris :	4
Health-Easy :	5
Health-Hard :	5
Power consumption :	6
rs-6d-c3-obj2 :	6
Pom3a :	9
pom3d :	9
Wine Quality :	10
Gamma telescope :	10
SS-T :	12
heart disease :	13
adult :	14
bank marketing :	16
SS-M :	16
german credit :	20
diabetes :	20
SS-U :	21
default :	23
nasa93dem :	25
Xomo (All) :	27
Player Statistics :	27
Student Dropout :	34
Employee Attrition :	35
All Players :	57
SCRUM :	128
FFM-250 :	250

TABLE 2: Table 3.2 data: number of raw dimensions R.

In an ideal world, software engineers and their managers should make their decisions about projects based on universal laws (rather than mere empirical effects). Notable examples of this include the Pareto Principle (the 80/20 rule) [45] and Brooks' Law, which observes that adding personnel to late software projects often delays them further [46], [47]. These effects are now "laws" but they are acknowledged as general observations with known exceptions [48]. Practitioners routinely apply them to prioritize tasks, allocate resources, and manage project risks [49].

That said, setting policy using only SE laws can be inadvisable (or impossible). **First of all**, there are few universally agreed SE "laws". Glass's "Facts and Fallacies of Software Engineering" is a valiant attempt to catalog known laws, but our work shows numerous contradictions to their claimed "laws." For example, Glass lists "Requirements errors are the most expensive to fix during production" as a law. Recently [50], we traced this law's citation chain and found most papers cited prior work without experiments or field data[1]. Moreover, recent papers from Silicon Valley project managers argue that with current continuous deployment methods, change time is constant and very low [51].

1. In fact, of hundreds of papers supporting this "law," only 12 offered detailed empirical evidence (six arguing for it and six against). Within those 12, at least two authors published papers demonstrating the "law" in some projects but not others [50].

Secondly, even if effect are not laws, they can still guide best practice. As shown below, the DRR effect appears frequently enough that practitioners should check for high DRR before deploying expensive optimization methods (such as DEHB). The prevalence of this effect across multiple problem classes suggests that it reflects something systematic about SE optimization landscapes, making it a valuable heuristic for matching solution complexity to problem complexity. As demonstrated here, this simple diagnostic can save orders of magnitude in computational cost while achieving comparable solution quality.

The rest of this paper is structured via guidelines from Wohlin & Runeson et al. [52] on empirical software engineering. We present our methods in Section 3, within which we discuss all algorithms, case studies, experimental design choices and datasets used in this study. In summary, we will

- Apply different kinds of HPO ...
- ... to tune an ensemble learner to improve regression and classification performance of models ...
- ... built from the Table 3.2 data (and where a datasets list many goals, we predict for the first listed goal).

Following this we present results in Section 4. See also Section 5.3 for a discussion on threats to the validity, and Section 6 for our conclusions.

In summary, the **contributions** of this paper are:

1) A new pragmatic and practical diagnostic for SE optimization: We introduce the Dimensionality Reduction Ratio (DRR) and document a concrete and common, empirical threshold ($DRR > 0.35$), above which optimization becomes remarkably easy. This threshold acts as a simple, fast diagnostic to determine when complex, expensive optimization methods are unnecessary for a given SE problem.
2) A correction of a recent TSE paper: We demonstrate that our DRR rule is a more accurate diagnostic than the $I \leq 4$ intrinsic dimensionality threshold proposed by Agrawal et al. We show that the $I \leq 4$ rule makes the wrong recommendation for most of our datasets, while our rule very often identifies the "easy" cases.
3) Evidence of a very large speedup: We show that for the large number of SE tasks with $DRR > 0.35$, a simple optimizer (LITE) achieves statistically identical performance to a complex, state-of-the-art optimizer (DEHB). This allows practitioners to get the same quality of results two orders of magnitude faster (e.g., seconds vs. 20+ minutes), saving significant computational cost.
4) Robust validation of the "DRR Effect": We demonstrate that this is not an isolated finding. After establishing the effect on 24 projects, we doubled the sample size to 50 diverse SE datasets and confirmed the effect still holds. We found 76% (38 out of 50) of our SE datasets exhibited this high-DRR behavior.
5) A Full Reproduction Package: We provide a complete and open artifact so other researchers can repeat, refute, or build upon our work. This package includes all 50 datasets, the implementations for both the simple (LITE[2]) and complex (DEHB[3])) optimizers, and the intrinsic dimensionality calculator.

2. https://github.com/timm/ezr
3. https://github.com/automl/DEHB/

2 Intrinsic Dimensionality

The assumption presented in this paper is that, empirically in many SE data we have seen:

- Problems expressed in R raw attributes can be simplified to a smaller number of I intrinsic underlying dimensions.
- Tasks with $I \ll R$ can be solved very quickly. This effect holds in 50 SE data sets.

This section argues for the first point (and the rest of the paper explores the second point).

2.1 Do We Always Need All That Data?

Centuries of research argues that data can be effectively reduced to a smaller set, without introducing errors into the analysis of that data. In 1901, Pearson argued [53] that equations involving many variables can often be effectively modeled using fewer variables derived from the eigenvectors of their correlation matrix. Such a "principal component analysis" can represent many dimensions using just a handful of components [54], [55].

There is much evidence for this "reduction" hypothesis. If a table of data has (say) $A = 20$ independent attributes, and each attribute has V states then for booleans (where $V = 2$), that table needs $V^A = 2^{20} > 1,000,000$ rows to cover all possible effects in that space. But a repeated result for software analytics is that V^A is a massive overestimation. Models built from $A > 20$ attributes can exhibit high recall, even if that model is trained from just a few hundred rows [56].

The only way to explain this is if some small subset a ($|a| \ll |A|$) of the attributes matter, and the rest can be irrelevant (e.g. they are either noisy or not associated with the target goal). Empirically this is indeed the case. Feature selection research for non-SE data [57], [58] shows that over half the attributes in tabular data can be removed, without loss of signal. For SE data, the reduction ratio can be much higher. Pruning many rows and columns of SE datasets often lead to better models [59]–[63]. The size of the pruning seen in SE data is startling. For example:

- Chen, Kocaguneli, Tu, Peters, and Xu et al. found they could predict for Github issue close time, effort estimation, and defect prediction, even after ignoring labels for 80%, 91%, 97%, 98% (respectively) of their rows [61]–[64].
- Later in this paper Figure 1 and Figure 3 show examples where datasets with dozens to hundreds of attributes can be reduced to half a dozen intrinsic dimensions, or less.

All the above can be summarized as the "manifold assumption" [65]:

> *Real-world higher-dimensional data often lies on a low-dimensional manifold embedded within the high-dimensional space.*

2.2 Calculating Intrinsic Dimensionality

To find the intrinsic dimensions, Agrawal et al. [34] use the fractal-based method of Algorithm 1. In summary, this method leverages the concept of a *fractal dimension*, which quantifies the complexity of a dataset by measuring how things change with the scale of measurement. More specifically, the algorithm checks how many more rows can be found at distance R_x than R_y for $x < y$. For example:

Algorithm 1 Calculating intrinsic dimensionality. From [34].

```
Import data from Testdata.py
Input: sample_num = n, sample_dim = d
Rs_log = start : end : step
Rs = exp(Rs_log)
for R in Rs do
    I = 0
    for (i, j) in combinations(data, 2) do
        d = distance(i, j)
        if d < R then
            I = I + 1
    Cr = 2I / (n(n−1))
    Crs.append(Cr)
for i in step do
    gradient = (Crs[i] − Crs[i−1]) / (Rs[i] − Rs[i−1])
    GR.append(gradient)
Smooth(GR)                                  ▷ smooth the curve
intrinsicD ← max(GR)                        ▷ return the intrinsic dimensionality
```

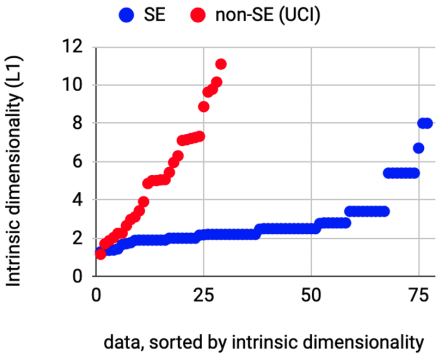

Fig. 1: Differences in the intrinsic dimensionality of SE and non-SE data. From [34].

- If the data lies on the ground all over a football field, then that data lies in a two-dimensional space. Hence, an increase from R_x to R_y will find *polynomially* more examples.
- But if the data lies on a straight road that runs through the middle of the field, then that data is effectively one dimensional and an increase from R_x to R_y will only find *linearly* more examples.

Algorithm 1 returns the maximum gradient of a curve of R_x vs the log of the number of rows found at distance R_x. Agrawal et al. report that this algorithm correctly detects up to $I \leq 20$ intrinsic dimensions within spreadsheets with low-correlated columns[4].

Algorithm 1 offers several advantages, including accuracy, robustness, and scalability related to other intrinsic dimensionality calculators. It provides a more accurate estimation of intrinsic dimensions compared to traditional methods [66] since it is less sensitive to noise and outliers in the dataset. Also, using some stochastic sub-sampling, it can be applied to large datasets efficiently. Its accuracy, robustness, and scalability make it a valuable tool for dimensionality reduction in various applications [67].

In support of the manifold assumption, Figure 1 shows Agrawal's results for SE and non-SE data. For SE data they used data seen for recent SE analytics research publications. For non-SE data, Agrawal et al. used datasets from the UCI data mining repository [26]. This non-SE data has been widely applied in the machine learning community to certify their algorithms [26].

4. To make low-correlated columns, cells had random variables.

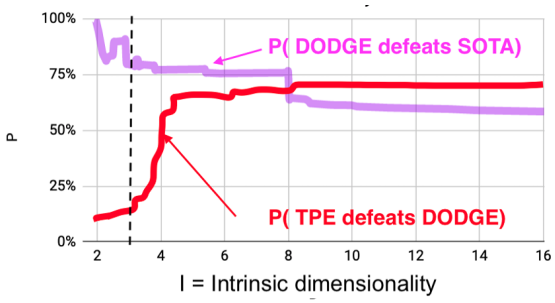

Fig. 2: According to Agrawal et al. [34] different algorithms work best at different intrinsic dimensionalities. Vertical dashed lines shows the median of the SE data from Figure 1.

Figure 1 shows that Algorithm 1 reduces data down to 6 intrinsic dimensions (or less). More importantly, the manifold assumption seems to hold especially true for SE data since, as seen in Figure 1:

- The median SE intrinsic dimensionality is 3.1;
- The median non-SE intrinsic dimensionality is 5;
- That is, non-SE is nearly twice as complex as SE data.

2.3 Implications of Low Intrinsic Dimensionality

Does it matter that SE and non-SE data have different intrinsic dimensionalities? Agrawal et al. argued this difference allows them to recommend when to use simpler or more complex hyperparameter optimizers. They advocated a simple tabu search optimizer called DODGE[5] which they compared against a 2012 optimizer called HYPEROPT/TPE (tree of parzen estimators) [68]. Their results are shown in Figure 2. Agrawal et al. reported that, in their study, that different hyperparameter optimizers work best for different intrinsic dimensionalities. In that figure, for intrinsic dimensionalities less than 4, their simpler tabu method was more likely to outperform TPE in more than half the datasets. We define *Agrawal's threshold* as:

$$I \leq 4 \tag{2}$$

2.4 Issues with the Agrawal Threshold

For many reasons, the Agrawal threshold must now be revisited and revised.

Firstly, the HYPEROPT/TPE algorithm used in the Agrawal study was proposed in 2012. It is no longer state of the art. This paper must repeat the Agrawal et al. analysis, but with more recent algorithms.

Secondly, in the Agrawal et al. data sample, the right-hand-side of Figure 2 was very under-populated. This is to say that their result could have been a conflation of most their data falling to the left of that figure. This paper will study more data at higher intrinsic dimensionalities.

5. Given the input settings S_i to a hyperparameter optimizer, and a resulting output performance score P_i, then if P_i is similar to an older score P_j, then DODGE deprecated settings near S_j.

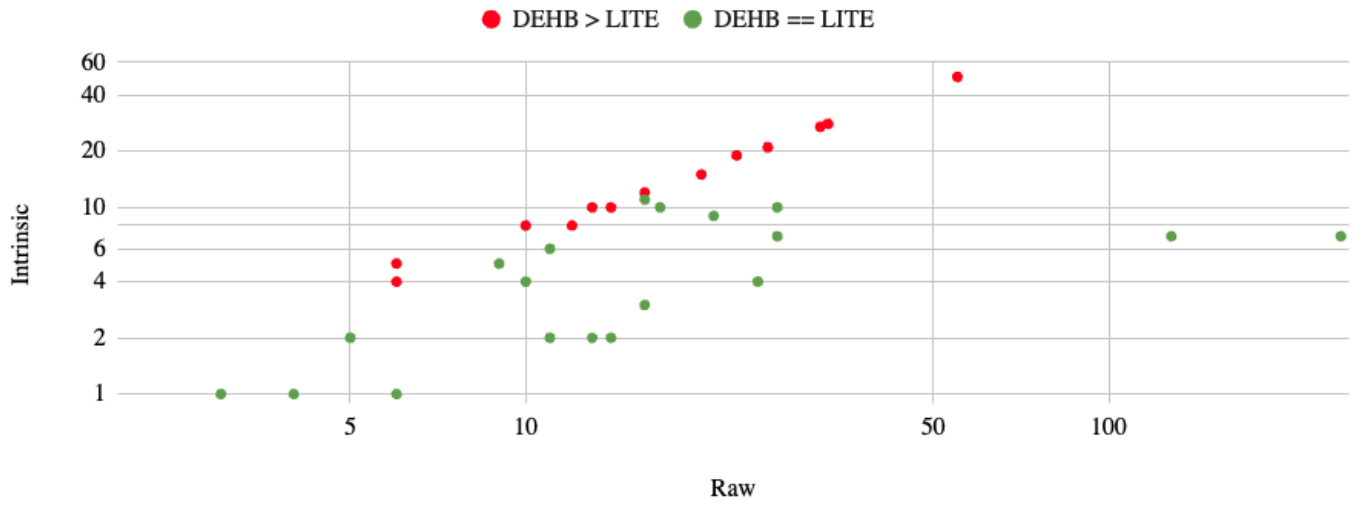

Fig. 3: Intrinsic Dimensionality vs Original Dimensionality of the Table 3.2 data. Red points indicate data sets where complex optimization (that require 3000 samples) defeated simpler methods (that only required 30 samples). Green points are easy cases, all of which come from SE papers.

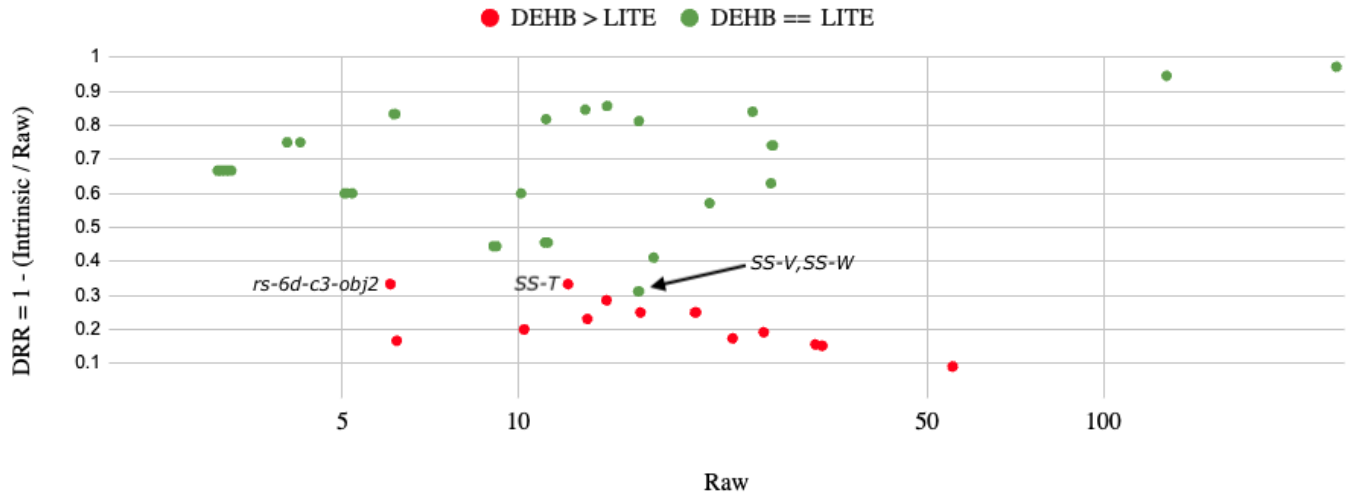

Fig. 4: Shown here are the Table 3.2 datasets, scored on the y-axis by Equation 1. Colors have the same mean as Figure 3. The data sets rs-6d-cs-obj2, SS-R, SS-V, SS-W are discussed in §4.3.

Thirdly, Figure 3 shows the original raw dimensions and the intrinsic dimensionality of the Table 3.2 data. In that figure, the green points come for optimizing results where very simple optimizers preformed as well (or better) than very complex optimizers. It should be noted that for those points, simpler optimization was orders of magnitude faster than more complex methods. Looking at the locations of the SE and non-SE data, it is clear that Agrawal's threshold of $I < 4$ fails to separate the easy from hard cases On the other hand, as seen in Figure 4, Equation 1 is far better at distinguishing these two types of data (nearly everything with DRR > 0.35 is an "easy" case). This separation is important. As shown below, very different algorithms work best for these separate groups.

Fourthly, and most importantly, it turns out that Agrawal's rule gives bad advice. As seen in Figure 3, most of our data has $I \geq 4$; i.e. which Agrawal would advise "use a complex optimizer". Among those datasets, we find many with $DRR > 0.35$ which (as shown by the results later in this paper) can be optimized very simply[6]. To say that another, Agrawal's rule is incorrect most of the time.

The rest of this paper expands on this last point. The next section describes an experiment checking how often state of the art AI methods are needlessly computationally expensive in the region $DRR > 0.35$.

6. Specifically, 30 quick samples will result in optimizations as good as slower methods that require 3000 samples.

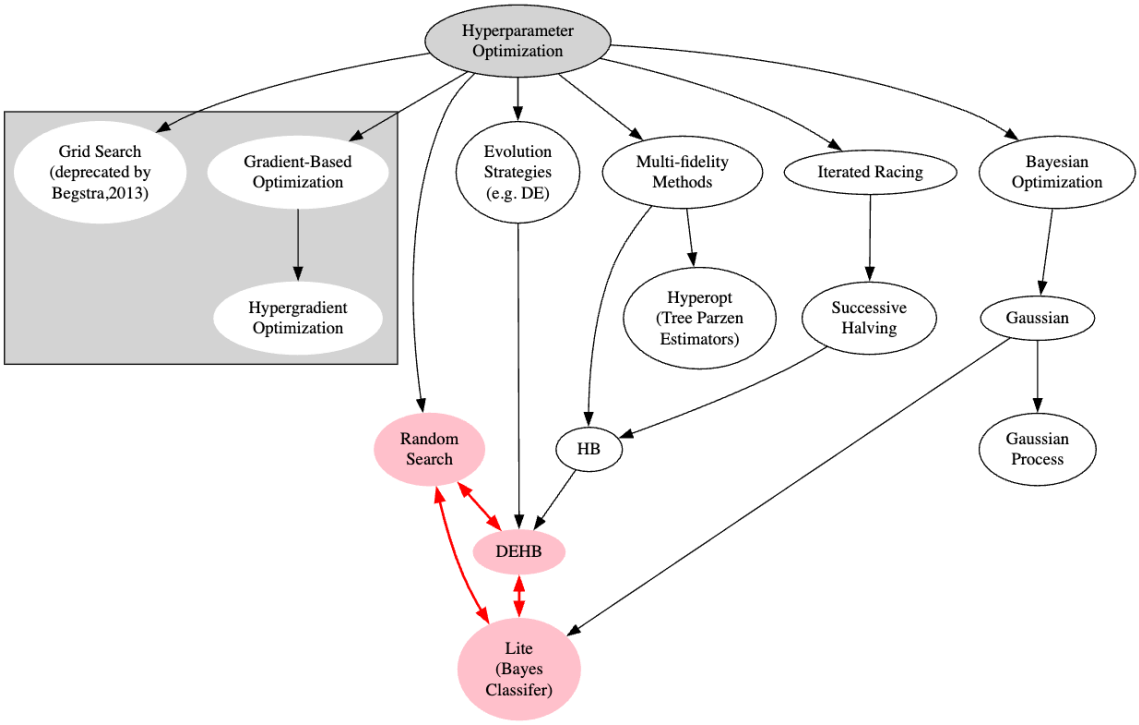

Fig. 5: A range of hyperparameter optimization methods. From [41]. This paper compares the methods shown in red.

3 METHODS

3.1 Research Questions

The methods of this paper aim to answer three questions. Empirically, for the data studied here:

- **RQ1:** Is SE and non-SE data different?
- **RQ2:** Is there an effect where that difference select for better HPO?
- **RQ3:** How common are these effects?

3.2 Data

To answer these questions, we study the data shown in Table 3.2. These datasets have a range of independent values seen in Table 2. The intrinsic dimensionality of those datasets are shown on the x-axis of Figure 3.

. To summarize the kinds of data explored here, we focus on software engineering problems that can be formulated as classification or regression tasks (and the regression can be for $N \geq 1$ goals). Specifically, we examine four major families of SE problems:

Software Configuration Optimization: This includes software systems where the goal is to predict performance outcomes (e.g., runtime, memory usage, energy consumption) or find optimal settings. Examples include compiler configurations (LLVM, x264), database systems (PostgreSQL, HSQLDB, Redis), web servers (Apache), and various software systems (SS-A through SS-X series, representing different configuration spaces with 3-88 tunable parameters). NoT R

Cloud and System Resource Management: Problems involving the optimization of cloud resources and system parameters, including Apache server performance tuning, SQL database optimization, video encoding settings (X264), and miscellaneous configuration tasks across different deployment scenarios.

Software Project Health Prediction: Predicting project health metrics and developer activity patterns using historical data from repositories. This includes predicting commit rates, pull request merge times, and issue closure rates across 35 different project health datasets.

Software Process Models: Classical SE estimation problems including effort prediction, defect forecasting, and schedule estimation. This family includes COCOMO-based models (COC1000), XOMO variants (Flight, Ground, OSP, OSP2) for different project types, POM3 models (A-D) for agile development scenarios balancing cost and completion rates, and the NASA93 dataset for multi-objective optimization of effort, defects, time, and lines of code.

We acknowledge that software engineering encompasses many other important problem types (such as test case generation and minimization, program refactoring, requirements prioritization, and formal verification) that are not naturally expressed as regression or classification tasks. Our work specifically targets the subset of SE problems where the goal is to predict numerical or categorical outcomes based on historical data or to optimize configurations through learned performance models.

3.3 Algorithms

This paper applies *hyperparameter optimization* (HPO) to the control parameters of an *ensemble learner* in order to build better predictors for the dependent attributes in Table 3.2. This section describes ensemble learning, and the HPO methods that might improve them.

3.3.1 Ensemble Learners

Ensemble learning trains multiple learners on somewhat different subsets of the data. This ensemble approach to learning often achieves higher predictive performance than individual models by aggregating predictions from diverse learners [69] [70]. This improvement is particularly significant in complex tasks where single models may struggle. Also, by combining multiple models, ensemble learning helps mitigate overfitting, leading to better generalization on unseen data [71] [72]. Further, ensemble techniques increase model robustness by reducing the impact of noise and outliers in the data [73]. This is useful in applications where data quality may vary.

Random forests are ensembles composed of ensembles of multiple decision trees, each of which is trained on a randomly selected $\sqrt{A}$ sample of the attributes A. Each tree in the ensemble makes a prediction and the final prediction is some aggregation of all the votes.

This study uses scikit-learn's random forest regressors and random forest classifiers [74]. These make predictions using the mean and mode (respectively) of their leaf nodes.

To use these random forests, we will ask our HPOs to make decisions about the following:

- *n_estimators*: number of trees in the forest. Varied from 1 to 200 (in steps of 10).
- *criterion*(The *error estimator*) : when building the tree, how to measure split quality. Options are squared_error, absolute_error, friedman_mse, and poisson.
- *min_samples_leaf*: the minimum number of samples required to be at a leaf node. Varied from 1 to 20.
- *min_impurity_decrease:* A secondary stopping criteria used by random forest. Varies from 1 to 10 (in steps of 0.25)
- *max_depth:* Max depth of tree. Varied from 1 to 20.

In all, this list mentions $20 * 4 * 20 * 40 * 20 = 1,280,000$ different possible configurations.

3.3.2 Hyperparameter Optimizers

The goal of hyperparameter optimization is to explore tables of configuration options containing:

- X columns containing configuration settings and
- the Y columns containing the performance scores seen after running one configuration option.

We assume the existence of function F such that

$$Y = F(X)$$

but we have no direct access to that function.

In this notation, hyperparameter optimization is the selection of some configuration option c_i from a space of possible configurations C. These configurations control all the choices within a learner that returns the model F. For example, the end of the last section offered multiple choices for random forests. The performance score F for the predictions made when a learner used that configuration. This score is computed by observing the performance of

$$Y = F(c_i, X)$$

Figure 5 shows a range of HPO approaches. For reasons explained below, we ignore the boxed methods (shown on the left) and focus on the methods highlighted in red.

Grid Search: The slowest way to perform HPO is *grid search*, which systematically evaluates every possible configuration. Bergstra et al. [68] warn strongly against such grid searches. They note that different learners/datasets need different grid sizes. A grid small enough to catch all nuances in all learners and datasets would be impractically slow to run. In our case, assuming 0.5 seconds to try every configuration (which is our average observed time) and 20 repeated trials (for statistical validity), then our 1.2 millions options need over 19 weeks of CPU time. All these tests would be independent so they could be parallelized to run in the space of a single eight-hour work day (assuming 25 machines with 16 cores). However, and this is our main point, why incur that cost when simpler options are available? This is an important question. When we talk to our industrial colleagues, they point out that reducing the cost of their cloud compute facilities is an increasingly urgent problem. For all these reasons, we seek alternatives to grid search.

Gradient-based Optimization: If exhaustive search is too expensive, a more informed approach might be to study how variables change over time. *Gradient-based optimization techniques*, such as those used in neural network training, adjust hyperparameters by following the gradient of the loss function with respect to the hyperparameters [75]. These methods are highly effective for continuous hyperparameter spaces and are widely adopted in deep learning. Examples include stochastic gradient descent (SGD) and Adaptive Moment Estimation (the ADAM optimizer) [76], [77]. However, these techniques are not typically used in random forests, which are tree-based, non-parametric methods that do not rely on gradients.

Random search: A simple, fast, but potentially incomplete HPO method is *random search* [78], where N random configurations are sampled and evaluated. Random search has been successfully applied in SE, particularly for optimizing models in defect prediction [37]. Simple random search is often defeated by *active learning* (discussed below) that leverages the results of each trial to guide subsequent selections.

Evolutionary methods: Evolutionary methods apply notions from biology to optimization. *Populations* of randomly generated candidates are *mutated* (i.e. changed by a small amount). Better candidates are *selected* for *cross-over* where multiple candidates are combined to create the next *generation* of candidates. Traditional Holland-style mutation [79] is computationally expensive since it mutates $n = 100$ candidates for $G = 100$ generations (i.e. 10^4 evaluations in all). Storn-style mutation [80], as seen in *differential evolution* (DE) needs far fewer evaluations. DE starts like random search and creates a small initial population of randomly selected configuration options (say, 10 vectors of options per feature being configured).

- For each generation, each option is compared to a newly created option that is a mixture of three other options drawn from the population. The new option replaces the old, when it has a better performance score.
- After generation #1, the invariant of DE is that each option in a population in generation g is superior to at least $g - 1$ other options. Hence, as the algorithm runs, it build new items from increasing superior options.

DE has been widely applied [81][7]. Within software engineering, DE has been used for optimization tasks such as Fu et al.'s tuning study on defect prediction [**?**]; Shu et al.'s study on tuning detectors for security issues [82], and Xia et al's study that tuned project health predictors for open-source JAVA systems [38].

Multi-Fidelity Methods do not dictate the optimization method *per se* but offer a meta-principle for their organization. As such they can be an umbrella technique for controlling other methods. For example for neural networks, *coarse-fine evolution* is a multi-fidelity methods [83] that applies low-fidelity methods before exploring more expensive ones. For another example, *successive halving* is a multi-fidelity method introduced by Li et al. [27] in their 2018 Hyperband (HB) algorithm. This approach employs a novel bandit strategy by simultaneously running multiple configurations with varying resource allocations, such as different numbers of training epochs. Resources are allocated by progressively eliminating poorly performing configurations. It starts with many candidates, giving each a small budget (e.g., a few training epochs or a dataset subset), evaluates their performance, and discards the worst-performing half at each iteration while doubling the budget for the remaining ones.

Iterated Racing: This kind of algorithm iteratively evaluates and compares a set of candidate solutions (or configurations) while discarding under-performing ones. Successive halving combines multi-fidelity methods with iterated racing. Other examples of iterated racing include the irace method [84] of López-Ibáñez et al. Irace implements the Iterated F-race algorithm for automatic algorithm configuration which iteratively evaluates and refines candidate configurations to efficiently determine optimal parameter settings. Classic iterated racing reflects on the rankings of different methods. Successive halving, on the other hand, also considers the resources required to collect information about each configuration.

Bayesian Optimization: is an optimization technique used when function evaluations are costly, such as in hyperparameter tuning, experimental design, or engineering simulations [85].

These methods build a *surrogate model* which can very quickly guess the likely outcome of an evaluation [86]. Once a surrogate model is available, the model built so far can be used to guess what example should be studied next. This technique is called *active learning* [87]. This is useful since when learners choose their own training data, they often build better models with fewer labeled examples [87].

One common choice for surrogates are *Gaussian Process Models (GPMs)* which estimate the mean and variance of predictions for unlabeled data. GPMs fit numerous diverse functions to existing labeled data so their computational cost grows with data size, limiting scalability [88].

A faster approach is Tree of Parzen Estimations (TPE) by Bergstra et al. [68] as implemented in the Hyperopt's Python package[8]. Hyperopt/TPE sorts labeled configurations, splits it at some engineer-defined threshold, then builds surrogate models for the "best" and "rest" subsets. For that modeling, it uses a Parzen Estimator kernel density estimator.

7. At the core of this writing (Feb 2025), the original DE paper [80] has over 37,000 citations (in Google Scholar).

8. https://github.com/hyperopt/hyperopt/

Faster than Hyperopt/TPE is LITE [89], a TPE method that uses a Bayes classifier to model "best" and "rest":

1) Given M evaluated configurations and N unexamined configurations, LITE sorts M into $\sqrt{M}$ "best" and the remaining into "rest" examples.
2) These sets train a two-class classifier that reports b, r; i.e. the likelihood of an example being "best" or "rest".
3) The resulting likelihoods are used by an *acquisition function* to select from N the configuration to run next.
4) This results in $M + 1$ labeled examples and $N - 1$ unlabeled examples. Each new labeling decrements the labeling budget B,
5) While $B > 0$, repeat from step #1.

On termination, LITE returns the best labeled example seen so far using the measures on Section 3.5. For its acquisition function, given a budget of B evaluations LITE evaluates the configuration from N that maximizes:

$$\frac{b + rq}{abs(bq - r + \epsilon)}$$

where ϵ is a very small constant (that avoids divide-by-zero errors) and

$$q = \begin{cases} 0 \ if \ \ exploiting \\ 1 \ if \ \ exploring \\ 1 - \frac{M}{B} \ if \ \ adapting \end{cases} \quad (3)$$

Here, *explore*, *exploit*, *adapt* are different search strategies for acquiring new information. When labeled examples are very scarce, it can be best to *explore* regions where oracles are declaring opposite ideas, but with similar weights. Once some more data has arrived, it can be better to switch to *exploiting* that knowledge and just jump to where there are strongest indications of most "best" and least "rest". Finally, "adapt" is a function which, as more M labeled examples M arrive, the acquisition strategy slides from *explore* to *exploit*.

DEHB: DEHB extends Hyperband by running multiple sequential halving brackets with different starting budgets, balancing exploration and exploitation. Within each budget level, DEHB uses DE to generate and evolve configurations, maintaining separate subpopulations for each fidelity level. This allows DE to run independently at each budget while enabling information flow from lower to higher budget subpopulations through a modified DE mutation strategy. The first iteration of DEHB resembles vanilla Hyperband, using random search to initialize the lowest fidelity subpopulation. In subsequent iterations, DEHB reuses and evolves subpopulations from previous brackets, eliminating the need for random sampling.

Our reading of the literature is that, within the conventional AI literature, DEHB [31] is arguably the current state-of-the-art in combining evolutionary methods (using DE's differential evolution), multi-fidelity methods (using HB's Hyperband) and iterated racing (using the sequential halving). DEHB out-performs its predecessor (BOHB [31]) (and BOHB is known to out-perform Hyperopt [32]).

3.4 Experimental Rig

To establish statistical credence, this rig is run 20 times with different random seeds.

Each run optimizes a random forest performing regression (or classification) for a single dependent variable in each of the datasets. The selected dependent was always the first available in each dataset (and, for this analysis, all other dependents are excluded from the data).

From the algorithms in the last section, we select three for comparison purposes:

- DEHB, since it uses a range of state-of-the-art methods (evolutionary methods, multi-fidelity methods, iterated racing).
- Random search, since it is good practice to compare stochastic methods with a completely random baseline[9].
- LITE, since, of the above, it is the fastest and uses the fewest samples. For Equation 3, we use $q = 1$ since that was recommended by the original LITE paper.

For random and LITE, we allow up to 30 evaluations. For DEHB we allow up to 3000 evaluations. These budgets are those recommended in the LITE and DEHB papers.

DEHB, with 3000 evaluations, will serve as our exemplar heavyweight method.

LITE, with 30 evaluations, will serve as our exemplar lightweight method.

For each of these datasets in Table 3.2, we started with 10,000 randomly selected random forest hyperparameter configurations. LITE will explore up to 30 of these (selected at random). DEHB, on the other hand, would sample 100 (at random) then go on to evolve its own set of preferred configurations.

3.5 Evaluation Metrics

When each algorithm was run on the Table 3.2 data, we collected the following evaluation metrics. We selected these since these are widely seen in the literature [4], [90]–[92].

Classification and regression problems need different evaluation criteria. For classification, if A, B, C, D denote the true negatives, false negatives, false positives, and true positives (respectively), then:

$$\begin{aligned} accuracy &= (A+D)/(A+B+C+D) \\ r = recall &= D/(B+D) \\ p = precision &= D/(C+D) \\ F1 &= 2rp/(r+p) \end{aligned}$$

For regression, MRE (magnitude of relative error) is defined as follows:

$$MRE = abs(actual - prediction)/actual \tag{4}$$

MRE measures how far predicted is from actual. PRED40, on the other hand, reports on how often a regression prediction falls close the actual. PRED40 was recommended by Sarro [91] and is defined as:

$$Pred40 = Count(MREs \leq 40)/||MREs|| \tag{5}$$

Shepperd and MacDonell [92] argue that regression measures like the above lack contextual information. Hence they propose Standardized Accuracy (SA) which baselines error measures of some algorithm against the errors seen when applying the simplest reasonable estimator (in our case, mean value of the known actuals). SA is based the Mean Absolute Error (MAE) and is defined as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |prediction_i - actual_i| \tag{6}$$

Here, N is the size of the test set used for evaluating the model performance. SA is then defined as the ratio:

$$SA = (1 - (MAE/MAE_{dumbPrediction})) \times 100 \tag{7}$$

and $MAE_{dumbPrediction}$ is the MAE of a set of guesses.

As per Lustosa et al.'s 2024 TOSEM paper [4], the above values are combined into a composite metric *d2h* (distance to heaven). To explain *d2h*, first we need to explain the Zitzler multi-objective indicator [93]. Zitzler favors example B over A if jumping from B to A loses more than jumping from A to B:

- Let $worse(A, B) = loss(A, B) > loss(B, A)$
- Let $loss(A, B) = \sum_{j=1}^{n} -e^{\Delta(j,A,B,n)}/n$
- Let $\Delta(j, A, B, n) = w_j(o_{j,A} - o_{j,B})/n$

Here $w_i \in \{-1, 1\}$ signifies whether we are minimizing or maximizing goal o_j (respectively).

- In regression, we seek to *maximize* Pred40 and SA and *minimize* MRE so o_j values are $\{1, -1, -1\}$ respectively.
- In classification, we seek to *maximize* accuracy, precision, recall, and F1 so o_j values are $\{1, 1, 1, 1\}$ respectively.

Finally in order to summarize all of these metrics into one, *d2h* takes all evaluated examples from all techniques and asks how close is each example to the best available in the set. It can be defined as:

$$D2H_s^i = i/|Z| \tag{8}$$

where Z is a vector containing all options ranked from best to worst according to Zitzler and i is the index of a sample s in that vector.

3.6 Statistical Methods

In our study, we report median and interquartile ranges (which show 50th percentile and 75th-25th percentile), of the D2H metric for each algorithm on each dataset. We collect median and interquartile range values for each of the datasets.

To make comparisons among all algorithms on a single dataset, we implement the Scott-Knott analysis [94]. In summary, by using Scott-Knott, algorithms are sorted by their performance. After that, they are assigned to different ranks if the performance of the algorithm at position i is significantly different to the algorithm at position $i - 1$.

To be more precise, Scott-Knott sorts the list of experiments (in this paper, LITE, Baseline and DEHB for each of the 16 datasets) by their median score. After the sorting, it then splits the list into two sub-lists. The goal for such a split is to maximize the expected value of differences in the observed performances before and after division [95]. Scott-Knott analysis then declares one of these divisions to be the best split. The best split should maximize the difference $E(\Delta)$ in the expected mean value before and after the split:

$$E(\Delta) = \frac{|l_1|}{|l|} abs(\overline{l_1} - \overline{l})^2 + \frac{|l_2|}{|l|} abs(\overline{l_2} - \overline{l})^2 \tag{9}$$

9. https://github.com/acmsigsoft/EmpiricalStandards/blob/master/docs/standards/OptimizationStudies.md#desirable

where:
- $|l|, |l_1|$, and $|l_2|$: Size of list l, l_1, and l_2.
- $\overline{l}$, $\overline{l_1}$, and $\overline{l_2}$: Mean value of list l, l_1, and l_2.

After the best split is declared by the formula above, Scott-Knott then implements some statistical hypothesis tests to check whether the division is useful or not. Here "useful" means l_1 and l_2 differ significantly by applying hypothesis test H. If the division is checked as a useful split, the Scott-Knott analysis will then run recursively on each half of the best split until no division can be made. In our study, hypothesis test H is the Cliff's delta non-parametric effect size measure. Cliff's delta quantifies the number of differences between two lists of observations beyond p-values interpolation [96]. The division passes the hypothesis test if it is not a "small" effect ($Delta \geq 0.147$).

The cliff's delta non-parametric effect size test explores two lists A and B with size $|A|$ and $|B|$:

$$Delta = \frac{\sum_{x \in A} \sum_{y \in B} \begin{cases} +1, \text{ if } x > y \\ -1, \text{ if } x < y \\ 0, \text{ if } x = y \end{cases}}{|A||B|} \tag{10}$$

In Equation 10, cliff's delta estimates the probability that a value in list A is greater than a value in list B, minus the reverse probability [96]. This hypothesis test and its effect size is supported by Hess and Kromery [97].

4 Results

Empirically, for the data studied here, we offer three results.

4.1 RQ1: is SE and non-SE data different?

Prior work, based on the Agrawal threshold Equation 2, argued that this is indeed the case (see Agrawal et al. [34]). However, that study had preponderance of low dimensional data. Recall from Section 2.4 that when we revisited that conclusion, For the data studied here, Figure 3 showed many cases where Equation 2 would incorrectly select for "hard" datasets half the time ($\frac{34}{48} = 71\%$). Hence, we cannot endorse Equation 2.

That said, when we (a) extend the data analysis to higher dimensional data; and (b) replace Equation 2 with Equation 1, we find ourselves in general agreement with Agrawal. Figure 3 shows that, for at least in the sample tabular data studied here:

> **Answer1:** SE data usually has higher DRRs (i.e. a lower ratio of intrinsic dimensions) than non-SE data.

4.2 RQ2: Does that difference select for better HPO?

To answer this question, we need to look at the median distances to heaven achieved by our different methods.

Figure 6.A shows median *D2H* for SE data:
- The upper curve shows the baseline of the the untreated data (expressed as median *D2H* scores)
- Below that, the other curves fall on top of each other.
- After applying the statistics of Section 3.6, we can confirm the visual impression that the performance of these lower curves are indistinguishable.
- That is, for this data, a state-of-the-art AI optimizer (DEHB) that needs 3000 evaluations does no better than a very lightweight optimizer (LITE) using 30 evaluations.

On the other hand, Figure 7.A shows non-SE data:
- As before, the upper curve shows the baseline (median *D2H* of the the untreated data.
- Below that, we see other curves with more visual separation than before.
- The treatment with the lowest curve (i.e. closest to heaven) is the treatment that uses 3000 evaluations (DEHB).
- That is, for this data, it is not enough to merely do 30 evaluations.

Figure 6.B and Figure 7.B shows us that none of these methods other than the baseline and random selection have a stability problem. Which means that both DEHB and LITE are stable for these datasets (low variance in repeated trials). Figure 6.C and Figure 7.C comment on the runtime cost of running 30 versus 3000 evaluations. In both curves, the LITE method runs two orders of magnitude faster (a few seconds as opposed to 20 minutes, or more).

Recalling Figure 3, most of the SE data ($\frac{38}{50} = 76\%$) have high DRR values. Hence we say:

> **Answer2:** By separating data on DRR, we can find problems that can be solved effectively, two orders of magnitude faster (than using state-of-the-art AI HPO algorithms).

4.3 RQ3: how common are these effects?

To check the generality of this DRR effect, we turned to the MOOT repository of multi-objective optimization tasks [98]. Curated by Tim Menzies and Tao Chen, this repository contains dozens of multi-objective optimization problems collected from recent papers in top SE venues such as the International Conference on Software Engineering [99], Foundations of SE (FSE) conference [100] IEEE Trans. SE [101], the Information Software Technology journal [102], Empirical Softw. Eng. [103], Mining Software Repositories [104], IEEE Access [105], ACM Trans. SE Methodologies [4] and the Automated Software Engineering Journal [106].

By picking randomly from MOOT, we doubled the sample size of this study and ran the experiments described above. This gave rise to results from 50 projects, shown in Figure 4, In that figure, the DRR threshold moved very slightly to

$$(1 - I/R) > 0.35$$

(since , above this threshold, 100% of data sets are easy to optimize). We characterize this new threshold (seen in 50 projects) as a very small change from our original report (as seen in 24 projects). Hence we say:

> **Answer3:** While this larger set of 50 does not guarantee universality over all SE projects, it does show that the DRR effect appears frequently. Practitioners should check for high DRR before deploying expensive optimization

Figure 6.A: SE data sets. D2H median (50th percentile). Lower is better.

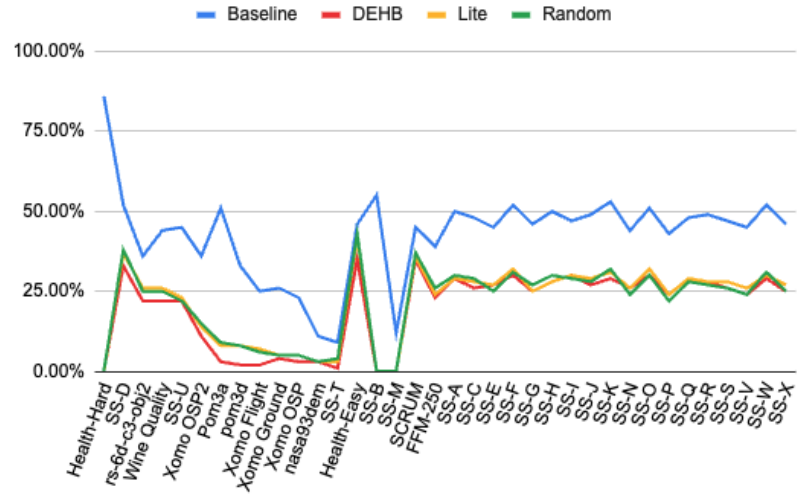

Figure 6.B: SE data sets. D2H IQR ((75th -25th) percentile). Lower is better.

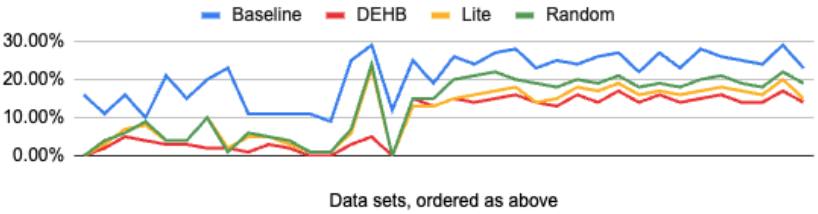

Figure 6.C: mean runtimes (seconds). Lower is better.

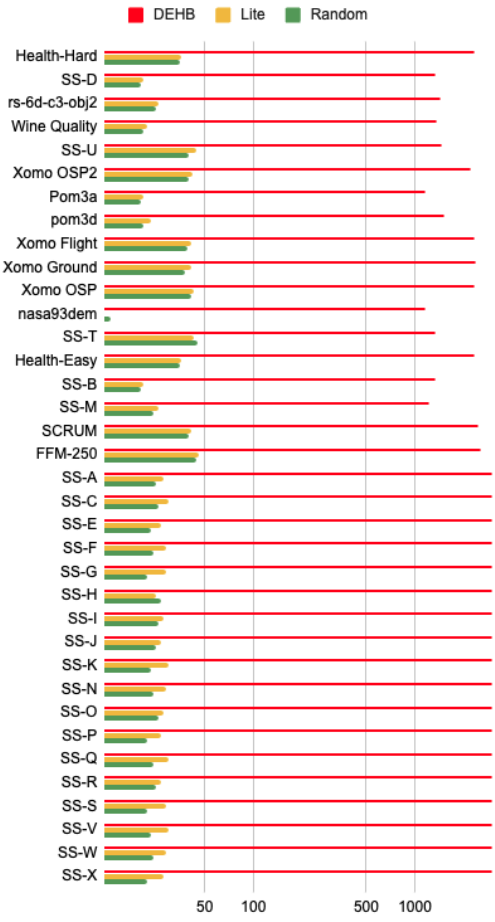

Fig. 6: SE data, results from 20 runs.

Figure 7.A: Non-SE data. D2H median (50th percentile). Lower is better.

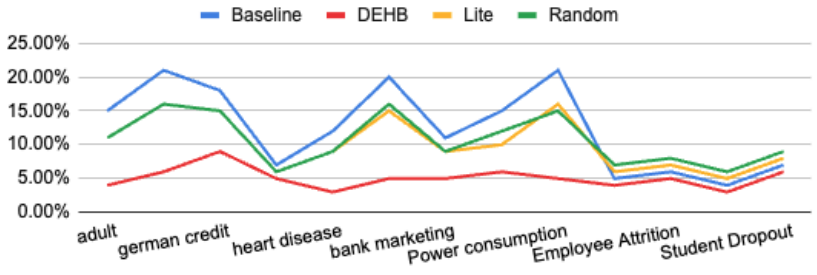

Figure 7.B: Non-SE data. D2H IQR ((75th -25th) percentile). Lower is better.

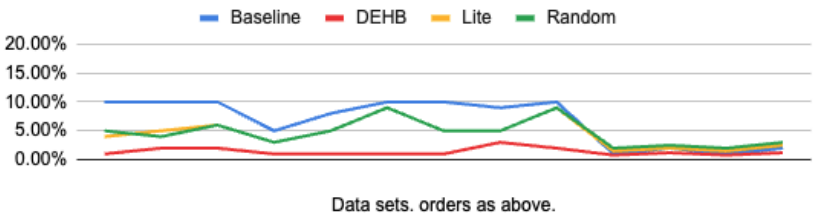

Figure 7.C: Non-SE data. mean runtimes (seconds). Lower is better.

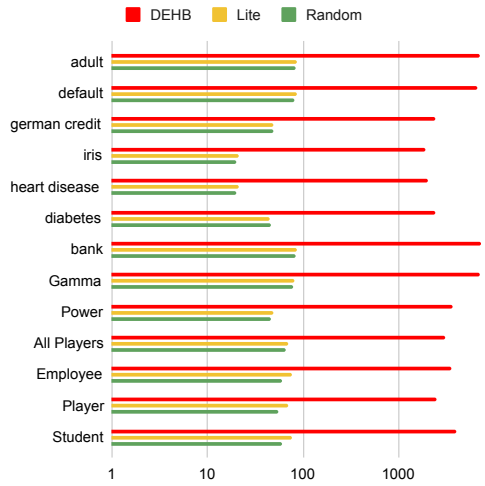

Fig. 7: Non-SE data, results from 20 runs.

> methods. This simple diagnostic could save orders of magnitude in computational cost.

As mentioned in the introduction the DRR effect is an empirical effect and not some binding law of the universe. Hence, there are some exceptions to this effect. Consider in Figure 4:

- SS-V, SS-W, which have a DRR of 0.30
- rs-6d-c2-obj2 and SS-T which have a DRR of 0.33

Note that the latter two data sets, with the larger DRR, are harder to optimize. Given the DRR effect, this is the opposite of what we would expect. Clearly, around the threshold where the effect takes hold, there is some noise. Accordingly, we set the DRR threshold to above 0.35. (and in defense of that new threshold, we note that 100% of the 38 data sets above that threshold exhibit easy optimization).

5 Discussion

5.1 Frequently Asked Questions

When discussing this work with colleagues, we are often asked the following.

FAQ0: Can Equation 1 simplify all SE analytics? No. Generation problems need the complexities of different classes of algorithms. Also, the certification requirements of safety-critical software is not a simple process. That said, we do show here that better and faster results can be obtained selecting algorithms via intrinsic problem complexity. This is an important message since, to our shame, SE researchers rarely benchmark complex methods against simpler approaches[10].

FAQ1: Why does DRR matter? DRR divides data into:

- One group where many attributes have to be explored;
- And another group where many attributes are superflous and can be ignored.

Complex AI algorithms are needed to explore datasets where most of the attributes are important. Otherwise, in SE data, simpler and faster methods may suffice.

FAQ2: Why so many superfluous attributes in SE data? Software construction is a complex combination of tasks. Data collected from software projects is hence a report of many things, not all of which are relevant to particular goals. Hence, we should expect that many attributes from SE data can be ignored.

FAQ3: Why has Equation 1 not been previously reported in the AI literature? Most of our SE datasets ($\frac{38}{50} = 76\%$) have high DRRs but most of our AI datasets have much lower DRRs. Hence, AI researchers have missed our result since they were studying different data.

This observation has a methodological implication. A common practice in SE analytics is to use AI algorithms off-the-shelf. The results of this paper suggest that this practice needs to be deprecated. Other researchers agree:

- Novielli et al. report that sentiment analysis tools perform much better for SE problems when they are specifically trained on SE data [115].
- Binkley et al. [116] warn that off-the-shelf information retrieval tools need to be significantly adjusted before they are applied to SE applications.

FAQ4: When Equation 1 holds, we can recommend something simpler than LITE? Figures 6.A and 6.B showcase that random selection (of 30 examples) performs just as well as LITE on the SE data. Nevertheless, we would still endorse LITE over random. Firstly, while random is simpler than LITE, LITE is hardly a complex algorithm (evidence: see its 30 line implementation [11]). Secondly, it we look into how random would be applied in practice, then "random" starts looking a lot like LITE:

- If new data arises, random search would require 30 more labels to handle that data while LITE could just reuse its existing model (i.e. zero extra labels).
- On the other hand, the random selection results from the old data could be used to build a classifier (for best and rest) and that classifier could be used (without further labels) to predict for the new data.
- However, if anyone asks for validation results from that classifier, statistics would have to be collected over the old data. If it was suggested to collect those statistics incrementally during label collection over the old data, then this random-plus-classifier approach would be almost the same as LITE.

5.2 Related Work

Prior studies connect to this paper through three themes: (a) tuning and optimization, (b) dimensionality reduction, and (c) empirical effects guiding SE practice. Table 3 summarizes key parallels. We also note a recent trend on simplifying large language models (LLMs) for code, which shares the same philosophy of matching solution complexity to problem complexity.

The first group of releated work concerns *hyperparameter optimization (HPO)*. Earlier work such as Agrawal *et al.* [34] and Van Aken *et al.* [117] explored automatic tuning of learner or system parameters. DRR differs by measuring *intrinsic problem complexity* to decide when simple optimizers are sufficient.

The second group involves *feature selection and dimensionality reduction*, including the classical wrappers of Kohavi and John [58] and principal-component methods summarized by Jolliffe [54]. DRR extends these ideas to modern SE datasets, quantifying how many underlying dimensions truly drive behavior.

The third line treats SE "laws" and *empirical effects* as practical heuristics for reasoning about projects [52], [118]–[120]. Our work follows this tradition, proposing the DRR effect as an empirically recurring pattern that links data structure to optimizer choice.

Finally, several recent papers explore how large language models (LLMs) for code can be simplified without losing accuracy. Works such as DietCode [121] and SlimCode [122] focus on the *input side* of model efficiency. DietCode prunes low-information tokens and statements based on attention weights from pre-trained models, showing that 40% of the code can be removed with minimal loss of

10. For example, in a recent systematic review [107] of 229 SE papers using large language models (LLMs), only $13/229 \approx 5\%$ of those papers compared LLMs to other approaches even though other methods can produce results that are better and/or faster [108]–[114].

11. https://github.com/timm/ezr/blob/main/ezr.py#L518C1-L548C68

TABLE 3: Positioning DRR relative to prior and emerging work.

Category	Representative Works	Core Idea	Relation to DRR
Hyperparameter Optimization	Agrawal et al. [34]; Van Aken et al. [117]	Automate tuning of learner parameters to maximize performance.	DRR identifies when simple HPO suffices, avoiding unnecessary complexity.
Reduce Features or Dimensions	Kohavi & John [58]; Jolliffe [54]	Reduce irrelevant or redundant features to improve generalization and runtime efficiency.	DRR quantifies intrinsic vs. observed dimensions, extending classical feature analysis to SE data.
Empirical SE Effects	Boehm [118]; Brooks [119]; Glass [120]; Wohlin & Runeson et al. [52]	Recurring empirical regularities that guide SE practice (e.g., Brooks' Law, Pareto principle).	DRR acts as a similar empirical effect linking data complexity to optimization difficulty.
LLM Input Simplification	DietCode [121]; SlimCode [122]	Prune or compress source-code tokens to lower inference cost and preserve model accuracy.	Conceptually parallel: both seek efficiency by removing redundant information before learning.
LLM Architecture Simplification	Avatar [123]; SimPy [123]	Compress or redesign model grammars and architectures to reduce compute, energy, and tokenization overhead.	Complementary: these simplify the *model*; DRR simplifies the *problem representation*.

performance. SlimCode generalizes this idea, introducing a model-agnostic approach that relies on structural properties of source code rather than model attention, achieving comparable accuracy while reducing training time and inference cost by more than an order of magnitude. Together these studies show that simpler representations can make even large models faster and cheaper to deploy.

Complementary work targets the *model architecture itself*. Avatar [123] formulates the reduction of model size, inference latency, energy consumption, and carbon footprint as a multi-objective optimization problem solved by a satisfiability modulo theory (SMT)–based tuner. The resulting models are up to 160× smaller and 184× more energy efficient than their original counterparts. SimPy [123] takes a different direction by redesigning the Python grammar for AI consumption, removing redundant formatting tokens, shortens keywords, and proposes a dual grammar where humans use Python while models use a condensed variant. Both methods explicitly link computational cost to representational redundancy, an idea that resonates with DRR.

Across these LLM efforts, the unifying principle is that performance and efficiency improve when representation complexity is matched to the task's intrinsic structure. In that sense, DRR and LLM simplification studies belong to the same methodological family: DRR measures redundancy at the *data level* to decide when simpler optimizers suffice, while the LLM community removes redundancy at the *representation or model level* to make heavy architectures more tractable. Both lines of work reflect a broader trend toward empirical diagnostics that quantify when "less" can in fact be "better."

Given the similarities in DRR, DietCode, SlimCode, Simpy and Avatar we conjecture that there is so as-yet-unrealized fusion of research combining tabular data landscape methods (e.g. DRR) with LLMs. We have preliminary ideas on this, but nothing definitive to report at this time.

5.3 Threats to Validity

As with any empirical study, different biases can threaten the final results. Therefore, any conclusions born from this work must be considered with the following concerns in mind.

Parameter Bias: In terms of parameter bias, the LITE algorithm uses one of many proposed acquisition functions towards active learning. There are other options of acquisition functions that may still be explored and that may outperform this current version of LITE. DEHB was also ran with default settings as suggested in their previous work [31], there may yet be other settings that can perform better than LITE in these problems.

Sampling Bias: : We have selected 50 datasets to represent various families of SE problems. These datasets may not be representative all SE problems. As discussed above, the effect we report has survived a "doubling study" (where we examined twice the number of data sets). Still, that coverage does not encompass the entire range seen in software engineering. Looking at the right-hand-side of Figure **[?]** we only have two data sets with very large attribute sets (SCRUM has 128 attributes and FFM-250 has 250 attributes). Notably, these datasets with a large number of raw attributes showed few intrinsic dimensions (for these data sets, both had $I = 7$), raising questions about what is the range of intrinsic dimensions for real-world data. Future work should explore the typical intrinsic dimensions of real-world datasets.

Algorithm Bias: We have only selected DEHB as a baseline, our reasoning behind this is the absolute supremacy this algorithm achieved in the field of hyperparameter optimization when compared to the remaining state-of-the-art algorithms. There are multiple studies that compare DEHB to other state-of-the-art algorithms or use DEHB in problems similar to those tackled here. However, this does not guarantee that there is not a better suited algorithm that could outperform both LITE and DEHB in the case studies provided in this work.

5.4 Future Work

Our findings open several avenues for further research.

First , expanding our dataset collection to include industrial SE projects would help validate whether the observed trends hold beyond controlled benchmarks. This paper has explored the optimization of classification and regression for 50 SE datasets. Future work should explore more data and more tasks.

Second, integrating DRR estimation into automated optimization pipelines could lead to adaptive algorithm selection in real-time.

Thirdly, refining the DRR measurement techniques could improve the accuracy of complexity estimation, reducing reliance on fixed thresholds like Agrawal's (Equation 2).

Fourthly, and perhaps more importantly, we have shown that under certain circumstances (see Equation 1), some SE tasks can be solved very simply. Hence it may be useful to revisit decades of SE analytics to find those tasks that seemed to be very slow, but can now be solved very simply.

6 Conclusion

This study challenges the conventional wisdom that complex AI-based optimizers are always necessary for software engineering (SE) problems. By analyzing intrinsic dimensionality (ID) and dimensionality reduction ratios (DRR), we have shown that many SE datasets contain redundant features that allow for simpler, faster optimization methods. Our results suggest that instead of defaulting to computationally expensive techniques, researchers and practitioners should first assess dataset complexity using DRR before selecting an optimization approach.

Through extensive experiments across multiple SE and AI datasets, we find that traditional hyperparameter optimization strategies can often be replaced by lightweight heuristics like LITE and Successive Halving without loss of effectiveness.

As stated in the introduction, we stress that the DRR effect is a common empirical effect that holds for 50 datasets from a range of SE tasks (see §3.2). As such it is a *limited conclusion* and not some ironclad natural law (as evidence of this, in §4.3 we discuss four counter-examples to this effect).

Nevertheless, the DRR effect is frequently found in our data (and was also observed when we doubled our sample size from 24 to 50 projects). We hence recommend that practitioners check for high DRR before deploying expensive optimization methods. As said above, the DRR effect is a simple diagnostic that could save orders of magnitude in computational cost.

Conflict of Interest Statement

The authors declared that they have no conflict of interest.

References

[1] K. Peng, C. Kaltenecker, N. Siegmund, S. Apel, and T. Menzies, "Veer: Disagreement-free multi-objective configuration," *arXiv preprint arXiv:2106.02716*, 2021.

[2] J. Chen, V. Nair, R. Krishna, and T. Menzies, ""sampling" as a baseline optimizer for search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 597–614, 2018.

[3] T. Menzies and J. Richardson, "Xomo: Understanding development options for autonomy," in *COCOMO forum*, vol. 2005, 2005.

[4] A. Lustosa and T. Menzies, "Learning from very little data: On the value of landscape analysis for predicting software project health," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 3, mar 2024. [Online]. Available: https://doi.org/10.1145/3630252

[5] B. Becker and R. Kohavi, "Adult," UCI Machine Learning Repository, 1996, DOI: https://doi.org/10.24432/C5XW20.

[6] S. Deepajothi and S. Selvarajan, "A comparative study of classification techniques on adult data set," *International Journal of Engineering Research & Technology (IJERT)*, vol. 1, no. 8, 2012.

[7] I.-C. Yeh, "Default of Credit Card Clients," UCI Machine Learning Repository, 2009, DOI: https://doi.org/10.24432/C55S3H.

[8] T. M. Alam, K. Shaukat, I. A. Hameed, S. Luo, M. U. Sarwar, S. Shabbir, J. Li, and M. Khushi, "An investigation of credit card default prediction in the imbalanced datasets," *Ieee Access*, vol. 8, pp. 201 173–201 198, 2020.

[9] H. Hofmann, "Statlog (German Credit Data)," UCI Machine Learning Repository, 1994, DOI: https://doi.org/10.24432/C5NC77.

[10] R. A. Fisher, "Iris," UCI Machine Learning Repository, 1936, DOI: https://doi.org/10.24432/C56C76.

[11] L. Omelina, J. Goga, J. Pavlovicova, M. Oravec, and B. Jansen, "A survey of iris datasets," *Image and Vision Computing*, vol. 108, p. 104109, 2021.

[12] S. Janosi, Andras and William, "Heart Disease," UCI Machine Learning Repository, 1989, DOI: https://doi.org/10.24432/C52P4X.

[13] K. U. Rani, "Analysis of heart diseases dataset using neural network approach," *arXiv preprint arXiv:1110.2626*, 2011.

[14] M. Kahn, "Diabetes," UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C5T59G.

[15] D. Verma and N. Mishra, "Analysis and prediction of breast cancer and diabetes disease datasets using data mining classification techniques," in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*. IEEE, 2017, pp. 533–538.

[16] M. S. R. P. and C. P, "Bank Marketing," UCI Machine Learning Repository, 2014, DOI: https://doi.org/10.24432/C5K306.

[17] M. S. Başarslan and İ. D. Argun, "Classification of a bank data set on various data mining platforms," in *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*. IEEE, 2018, pp. 1–4.

[18] R. Bock, "MAGIC Gamma Telescope," UCI Machine Learning Repository, 2004, DOI: https://doi.org/10.24432/C52C8B.

[19] K. Karthick, S. A. Agnes, S. S. Kumar, S. Alfarhood, and M. Safran, "Identification of high energy gamma particles from the cherenkov gamma telescope data using a deep learning approach," *IEEE Access*, 2024.

[20] G. Hebrail and A. Berard, "Individual Household Electric Power Consumption," UCI Machine Learning Repository, 2006, DOI: https://doi.org/10.24432/C58K54.

[21] R. Chinnaraji and P. Ragupathy, "Accurate electricity consumption prediction using enhanced long short-term memory," *IET Communications*, vol. 16, no. 8, pp. 830–844, 2022.

[22] nyagami, "Ea sports fc 25 database — ratings and stats," https://www.kaggle.com/datasets/nyagami/ea-sports-fc-25-database-ratings-and-stats, 2025, dataset on Kaggle. Accessed: 2025-10-27.

[23] abdullah0a, "Student dropout analysis and prediction dataset," https://www.kaggle.com/datasets/abdullah0a/student-dropout-analysis-and-prediction-dataset, 2025, dataset on Kaggle. Accessed: 2025-10-27.

[24] die9origephit, "Fifa world cup 2022: Complete dataset," https://www.kaggle.com/datasets/die9origephit/fifa-world-cup-2022-complete-dataset, 2025, dataset on Kaggle. Accessed: 2025-10-27.

[25] Pavansubhasht, "Ibm hr analytics employee attrition performance," https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset, 2025, dataset on Kaggle. Accessed: 2025-10-27.

[26] A. Asuncion, D. Newman *et al.*, "Uci machine learning repository," 2007.

[27] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[28] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *International Symposium on Search Based Software Engineering*. Springer, 2011, pp. 33–47.

[29] A. L. Oliveira, P. L. Braga, R. M. Lima, and M. L. Cornélio, "GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation," *information and Software Technology*, vol. 52, no. 11, pp. 1155–1166, 2010.

[30] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 321–332.

[31] N. Awad, N. Mallik, and F. Hutter, "Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed. International Joint Conferences on Artificial Intelligence

Organization, 8 2021, pp. 2147–2153, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2021/296

[32] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: a python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, no. 1, p. 014008, 2015.

[33] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[34] A. Agrawal, X. Yang, R. Agrawal, R. Yedida, X. Shen, and T. Menzies, "Simpler hyperparameter optimization for software analytics: why, how, when," *IEEE Transactions on Software Engineering*, 2021.

[35] Z. Zhou, M. Zhou, Z. Wang, and X. Chen, "Predicting treatment outcome in metastatic melanoma through automated multi-objective model with hyperparameter optimization," in *Medical Imaging 2022: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 12034. SPIE, 2022, pp. 117–121.

[36] A. Mashlakov, V. Tikka, L. Lensu, A. Romanenko, and S. Honkapuro, "Hyper-parameter optimization of multi-attention recurrent neural network for battery state-of-charge forecasting," in *EPIA Conference on Artificial Intelligence*. Springer, 2019, pp. 482–494.

[37] M. Nevendra and P. Singh, "Empirical investigation of hyperparameter optimization for software defect count prediction," *Expert Systems with Applications*, vol. 191, p. 116217, 2022.

[38] T. Xia, W. Fu, R. Shu, R. Agrawal, and T. Menzies, "Predicting health indicators for open source projects (using hyperparameter optimization)," *Empirical Software Engineering*, vol. 27, no. 6, pp. 1–31, 2022.

[39] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Information and Software Technology*, vol. 76, pp. 135–146, 2016.

[40] R. Yedida and T. Menzies, "How to improve deep learning for software analytics: (a case study with code smell detection)," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 156–166.

[41] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, no. 2, p. e1484, 2023.

[42] N. Swartz, *The Concept of Physical Law*, 2nd ed. Cambridge University Press, 2003, available at: https://www.sfu.ca/ swartz/physical-law/.

[43] Wikipedia, "Scientific law," https://en.wikipedia.org/wiki/Scientific_law, 2024, accessed: October 2024.

[44] Encyclopaedia Britannica, "Scientific theory," https://www.britannica.com/science/scientific-theory, 2024, updated: September 20, 2025.

[45] K. Yamashita, S. McIntosh, Y. Kamei, A. E. Hassan, and N. Ubayashi, "Revisiting the applicability of the pareto principle to core development teams in open source software projects," in *Proceedings of the 14th International Workshop on Principles of Software Evolution*, ser. IWPSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 46–55. [Online]. Available: https://doi.org/10.1145/2804360.2804366

[46] F. P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley, 1975.

[47] J. D. Blackburn, M. A. Lapre, and L. N. Van Wassenhove, "Brooks' law revisited: Improving software productivity by managing complexity," 2006, working Paper, Available at SSRN: https://ssrn.com/abstract=922768.

[48] T. Paiva, M. Ferreira, C. de Magalhães, and A. da Silva, "Coordination and productivity issues in free software: the role of Brooks' law," in *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. IEEE Computer Society, 2009, pp. 14–19.

[49] K. W. McCain and L. J. Salvucci, "How influential is Brooks' law? A longitudinal citation context analysis of Frederick Brooks' The Mythical Man-Month," *Journal of Information Science*, vol. 32, no. 3, pp. 277–295, 2006.

[50] T. Menzies, W. Nichols, F. Shull, and L. Layman, "Are delayed issues harder to resolve? revisiting cost-to-fix of defects throughout the lifecycle," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1903–1935, 2017.

[51] C. Parnin, E. Helms, C. Atlee, H. Boughton, M. Ghattas, A. Glover, J. Holman, J. Micco, B. Murphy, T. Savor *et al.*, "The top 10 adages in continuous deployment," *IEEE Software*, vol. 34, no. 3, pp. 86–95, 2017.

[52] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[53] K. Pearson, "Principal components analysis," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 6, no. 2, p. 559, 1901.

[54] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[55] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, P. Yuan, Y. Tang, and T. Zhang, "Software defect prediction based on kernel pca and weighted extreme learning machine," *Information and Software Technology*, vol. 106, pp. 182–200, 2019.

[56] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proceedings of the 4th international workshop on Predictor models in software engineering*, 2008, pp. 47–54.

[57] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437–1447, Nov 2003.

[58] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[59] T. Menzies, "Shockingly simple:" keys" for better ai for se," *IEEE Software*, vol. 38, no. 2, pp. 114–118, 2021.

[60] M. Rees-Jones, M. Martin, and T. Menzies, "Better predictors for issue lifetime," *arXiv preprint arXiv:1702.07735*, 2017.

[61] K. E, M. T., K. J., C. D., and M. R, "Active learning and effort estimation: Finding the essential content of software effort estimation data," *TSE*, vol. 39, no. 8, pp. 1040–1053, 2013.

[62] F. Peters, T. Menzies, and L. Layman, "Lace2: Better privacy-preserving data sharing for cross project defect prediction," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 801–811.

[63] Z. Chen, T. Menzies, D. Port, and D. Boehm, "Finding the right data for software cost modeling," *IEEE software*, vol. 22, no. 6, pp. 38–46, 2005.

[64] H. Tu and T. Menzies, "Frugal: unlocking semi-supervised learning for software analytics," in *ASE'22*. IEEE Press, 2022, p. 394–406.

[65] X. J. Zhu, "Semi-supervised learning literature survey," 2005.

[66] F. Camastra, "Data dimensionality estimation methods: a survey," *Pattern recognition*, vol. 36, no. 12, pp. 2945–2954, 2003.

[67] L. Fragoso, T. Paul, F. Vadan, K. G. Stanley, S. Bell, and N. D. Osgood, "Intrinsic dimensionality of human behavioral activity data," *Plos one*, vol. 14, no. 6, p. e0218966, 2019.

[68] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[69] T. G. Dietterich, "Ensemble methods in machine learning," *Multiple classifier systems*, pp. 1–15, 2000.

[70] Z.-H. Zhou, "Ensemble methods: foundations and algorithms," *Chapman and Hall/CRC*, 2012.

[71] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[72] R. E. Schapire, "A brief introduction to boosting," *IJCAI*, vol. 99, pp. 1401–1406, 1999.

[73] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine learning*, vol. 51, no. 2, pp. 181–207, 2003.

[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, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[75] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," *International Conference on Machine Learning*, pp. 2113–2122, 2015.

[76] O. Shamir and T. Zhang, "Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes," in *Proceedings of the 30th International Conference on*

Machine Learning, ser. Proceedings of Machine Learning Research, vol. 28. PMLR, 2013, pp. 71–79.

[77] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR'15*, 2015.

[78] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[79] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992. [Online]. Available: http://www.jstor.org/stable/24939139

[80] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[81] M. Pant, H. Zaheer, L. Garcia-Hernandez, A. Abraham *et al.*, "Differential evolution: A review of more than two decades of research," *Engineering Applications of Artificial Intelligence*, vol. 90, p. 103479, 2020.

[82] R. Shu, T. Xia, L. Williams, and T. Menzies, "Dazzle: Using optimized generative adversarial networks to address security data class imbalance issue," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 144–155.

[83] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*. PMLR, 2018, pp. 4095–4104.

[84] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[85] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf

[86] A. Deshwal, S. Belakaria, J. R. Doppa, and D. H. Kim, "Bayesian optimization over permutation spaces," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 6, 2022, pp. 6515–6523.

[87] B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009.

[88] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using FLASH," *IEEE Transactions on Software Engineering*, vol. 46, no. 7, pp. 794–811, 2018.

[89] T. Menzies and A. Lustosa, "Streamlining software reviews: Efficient predictive modeling with minimal examples," 2024. [Online]. Available: https://arxiv.org/abs/2405.12920

[90] P. Reddy, K. Sudha, P. R. Sree, and S. Ramesh, "Software effort estimation using radial basis and generalized regression neural networks," *arXiv preprint arXiv:1005.4021*, 2010.

[91] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 619–630. [Online]. Available: https://doi.org/10.1145/2884781.2884830

[92] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[93] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design, Optimisation, and Control*. CIMNE, Barcelona, Spain, 2002, pp. 95–100.

[94] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE Transactions on software engineering*, vol. 39, no. 4, pp. 537–551, 2012.

[95] T. Xia, R. Krishna, J. Chen, G. Mathew, X. Shen, and T. Menzies, "Hyperparameter optimization for effort estimation," *arXiv preprint arXiv:1805.00336*, 2018.

[96] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[97] M. R. Hess and J. D. Kromrey, "Robust confidence intervals for effect sizes: A comparative study of cohen'sd and cliff's delta under non-normality and heterogeneous variances," in *annual meeting of the American Educational Research Association*, 2004, pp. 1–30.

[98] T. Menzies, T. Chen, Y. Ye, K. K. Ganguly, A. Rayegan, S. Srinivasan, and A. Lustosa, "Moot: a repository of many multi-objective optimization tasks," in *MSR 2026*, 2026.

[99] P. Chen and T. Chen, "Promisetune: Unveiling causally promising and explainable configuration tuning," in *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*. IEEE/ACM, 2026.

[100] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Using bad learners to find good configurations," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 257–267.

[101] P. Chen, J. Gong, and T. Chen, "Accuracy can lie: On the impact of surrogate model in configuration tuning," *IEEE Transactions on Software Engineering*, vol. 51, no. 2, pp. 548–580, 2025.

[102] J. Chen, V. Nair, and T. Menzies, "Beyond evolutionary algorithms for search-based software engineering," *Information and Software Technology*, vol. 95, pp. 281–294, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917300757

[103] K. Peng, C. Kaltenecker, N. Siegmund, S. Apel, and T. Menzies, "Veer: enhancing the interpretability of model-based optimizations," *Empirical Software Engineering*, vol. 28, no. 3, p. 61, 2023.

[104] V. Nair, A. Agrawal, J. Chen, W. Fu, G. Mathew, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu, "Data-driven search-based software engineering," in *MSR*, 2018.

[105] A. Lustosa and T. Menzies, "isneak: Partial ordering as heuristics for model- based reasoning in software engineering," *IEEE Access*, vol. 12, pp. 142 915–142 929, 2024.

[106] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Faster discovery of faster system configurations with spectral learning," *Automated Software Engineering*, vol. 25, no. 2, pp. 247–277, 2018.

[107] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3695988

[108] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *NeurIPS'22*, 2022.

[109] S. Somvanshi, S. Das, S. A. Javed, G. Antariksa, and A. Hossain, "A survey on deep tabular learning," *arXiv preprint arXiv:2410.12034*, 2024.

[110] V. Tawosi, R. Moussa, and F. Sarro, "Agile effort estimation: Have we solved the problem yet? insights from a replication study," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2677–2697, 2023.

[111] S. Majumder, N. Balaji, K. Brey, W. Fu, and T. Menzies, "500+ times faster than deep learning," in *Proceedings of the 15th International Conference on Mining Software Repositories. ACM*, 2018.

[112] X. Ling, T. Menzies, C. Hazard, J. Shu, and J. Beel, "Trading off scalability, privacy, and performance in data synthesis," *IEEE Access*, vol. 12, pp. 26 642–26 654, 2024.

[113] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *FSE*, 2017.

[114] B. Johnson and T. Menzies, "Ai over-hype: A dangerous threat (and how to fix it)," *IEEE Software*, vol. 41, no. 6, pp. 131–138, 2024.

[115] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 364–375. [Online]. Available: https://doi.org/10.1145/3196398.3196403

[116] D. Binkley, D. Lawrie, and C. Morrell, "The need for software specific natural language techniques," *Empirical Software Engineering*, vol. 23, pp. 2398–2425, 2018.

[117] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1009–1024. [Online]. Available: https://doi.org/10.1145/3035918.3064029

[118] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall, 1981.

[119] F. P. Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 1/E*. Pearson Education India, 1975.
[120] R. L. Glass, *Facts and fallacies of software engineering*. Addison-Wesley Professional, 2002.
[121] Z. Zhang, H. Zhang, B. Shen, and X. Gu, "Diet code is healthy: simplifying programs for pre-trained models of code," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1073–1084. [Online]. Available: https://doi.org/10.1145/3540250.3549094
[122] Y. Wang, X. Li, T. N. Nguyen, S. Wang, C. Ni, and L. Ding, "Natural is the best: Model-agnostic code simplification for pre-trained large language models," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3643753
[123] J. Shi, Z. Yang, H. J. Kang, B. Xu, J. He, and D. Lo, "Greening large language models of code," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*, ser. ICSE-SEIS'24. New York, NY, USA: Association for Computing Machinery, 2024, p. 142–153. [Online]. Available: https://doi.org/10.1145/3639475.3640097

Andre Lustosa received his PhD in Computer Science from North Carolina State University in 2025. He is a Principal Software Engineer and Team Lead at Red Hat, where he leads efforts to build and package the AI kernel across Red Hat's AI product portfolio. His research interests include software engineering, optimization, and data mining. He is an active open source contributor and recipient of the Red Hat AI Jedi Award (Q4 2025). For more information, please visit http://alustos.us.

Tim Menzies (ACM Fellow, IEEE Fellow, ASE Fellow, Ph.D., UNSW, 1995) is a full Professor in Computer Science at North Carolina State. He is the director of the Irrational Research Lab (mad scientists r'us) and the author of over 300 publications (refereed) with 24,000 citations and an h-index of 74. He has graduated 22 Ph.D. students, and has been a lead researcher on projects for NSF, NIJ, DoD, NASA, USDA and private companoes (total funding of $19+ million). Prof. Menzies is the editor-in-chief of the Automated Software Engineering journal and associate editor of TSE and other leading SE journals. For more, see https://timm.fyi.