MOOT: a Repository of Many Multi-Objective Optimization Tasks

Tim Menzies
Computer Science
NC State, USA
timm@ieee.org

Tao Chen
Yulong Ye
Computer Science
University of Birmingham, UK
t.chen@bham.ac.uk
yxy382@student.bham.ac.uk

Kishan Kumar Ganguly, Amirali Rayegan, Srinath Srinivasan, Andre Lustosa[*]
Computer Science, NC State, USA
{kgangul; arayega; ssrini27; alustos}@ncsu.edu;

Abstract

Software engineers must make decisions that trade off competing goals (faster vs. cheaper, secure vs. usable, accurate vs. interpretable, etc.). Despite MSR's proven techniques for exploring such goals, researchers still struggle with these trade-offs. Similarly, industrial practitioners deliver sub-optimal products since they lack the tools needed to explore these trade-offs.

To enable more research in this important area, we introduce MOOT, a repository of multi-objective optimization tasks taken from recent SE research papers. MOOT's tasks cover software configuration, cloud tuning, project health, process modeling, hyperparameter optimization, and more. Located at github.com/timm/moot, MOOT's current 120+ tasks are freely available under an MIT license (and we invite community contributions). As shown here, this data enables dozens of novel research questions.

Keywords

Optimization, multi-objective, search-based SE

1 Introduction

This paper introduces MOOT [49][1], a repository of SE multi-objective optimization tasks. MOOT currently contains 120+ datasets (see Table1). MOOT is a very active and growing repository: it has doubled in size, just in the last 12 months. Sample scripts for reading MOOT and generating baseline results are available– just clone the repository and run the rqx.sh files in MOOT0[2].

This paper (a) motivates MOOT (§2); (b) shows sample data (§3); (c) offers baseline results (§4); (d)lists enabled research questions enabled by MOOT (Table2); and (e) discusses how we are building a research community around MOOT-style issues.

2 Why MOOT?

MOOT enables data mining research for trading off multiple user goals such as *better, faster, cheaper* (fewer bugs, less time, less cost); *performance vs. sustainability* (faster response, less energy); and *hyperparameter optimization* (finding learner parameters that minimize false alarms and maximize recall). Formally, these are multi-objective *configuration optimization tasks*.

MSR has long recognized configuration optimization as promising for data mining. A 2019 NII Shonan Meeting on Data-driven Search-based SE[3] drew two dozen+ senior MSR researchers who

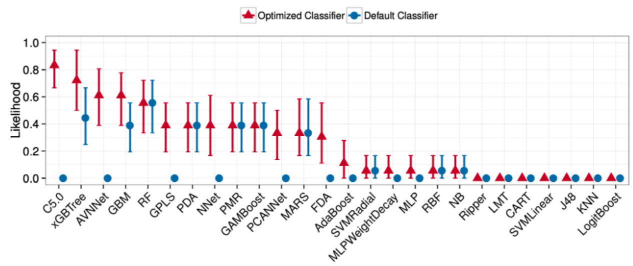


Figure 1: Odds of defect predictor performing best. BLUE = pre-optimization, RED = post-optimization. From [63].

concluded that repository mining can benefit from MSR techniques (e.g. tuning MSR algorithm parameters [24]). Conversely, MSR's data miners help optimization (e.g. summarize data to speed optimization [42]).

Yet this convergence has not gained momentum. In 2025, researchers still use simplistic hyperparameter methods [7] (e.g. grid search with one loop per hyperparameter[4]). Meanwhile, configuration optimization has suffered from external validity since much prior work was tested on very little data[5]. All this amounts to a critical missed opportunity for the MSR community.

We now offer reasons why **MSR should care about MOOT**.

First, automatic configuration dramatically improves learner performance. A staple of MSR research is software analytics, where data miners are applied to software project data. For software analytics, automatic configuration optimization of learner parameters can dramatically transform results [3, 24, 72, 73]. Figure 1 shows an example of defect prediction optimization (BLUE= pre-optimization, RED= post-optimization). When optimized, the C5.0 algorithm (left side) moved from being the *worst* to being the *best* learner. Similar gains have been seen in other SE domains [3, 4, 24, 63]. These improvements mean unoptimized data miner results can be easily refuted by tuning[6]. MSR needs to understand how much optimization changes conclusions from off-the-shelf tools.

Second, industry urgently needs better configuration methods. Modern software has extensive configurability, but tuning parameters critically affect performance [18]. For example, Storm's

[*] All authors contributed equally to this research.
[1] URL: http://github.com/timm/moot;
DOI: https://doi.org/10.5281/zenodo.17354083;
ZENODO: https://zenodo.org/records/17354083
[2] http://tiny.cc/moot0
[3] http://goo.gl/f8D3EC

[4] Grid search is adequate for tuning a few parameters, but much better alternatives exist [9] with public implementations [6, 8, 10].
[5] As reported in Table 2 of https://arxiv.org/pdf/2501.00125, much prior work has tested their optimization methods on five data sets or less.
[6] E.g. [2] showed (a) half of prior results were just learner nondeterminism; (b) optimization can tame that nondeterminism.

Table 1: Summary of the datasets currently existing in the MOOT repository. "x/y" denotes the number of independent and dependent attributes, For an example of how each dataset looks like refer to Figure 2. These datasets come from papers published in top SE venues such as the International Conference on Software Engineering [15, 30, 50, 68], Foundations of SE (FSE) conference [35, 51] IEEE Trans. SE [13, 16, 42, 44, 70], the Information Software Technology journal [12, 24], Empirical Softw. Eng. [33, 38, 57], Mining Software Repositories [52], IEEE Access [46], ACM Trans. SE Methodologies [45] and the Automated Software Engineering Journal [53].

# Datasets	Dataset Type	File Names	Primary Objective	x/y	# Rows	Cited By
25	Specific Software Configurations	SS-A to SS-X, billing10k	Optimize software system settings	3-88/2-3	197–86,059	[25, 46–48, 51–53, 57, 59, 60]
12	PromiseTune Software Configurations	7z, BDBC, HSQLDB, LLVM, PostgreSQL, dconvert, deeparch, exastencils, javagc, redis, storm, x264	Software performance optimization	9-35/1	864-166,975	[15, 16, 26, 28, 71]
1	Cloud	HSMGP num	Hazardous Software, Management Program data	14/1	3,457	[16, 25, 48, 59, 60]
1	Cloud	Apache AllMeasurements	Apache server performance optimization	9/1	192	[16, 25, 48, 59, 60]
1	Cloud	SQL AllMeasurements	SQL database tuning	39/1	4,654	[25, 48, 59, 60]
1	Cloud	X264 AllMeasurements	Video encoding optimization	16/1	1,153	[25, 48, 59, 60]
7	Cloud	(rs—sol—wc)*	misc configuration tasks	3-6/1	196–3,840	[25, 47, 48, 52, 59, 60]
35	Software Project Health	Health-ClosedIssues, -PRs, -Commits	Predict project health and developer activity	5/2-3	10,001	[25, 45, 47, 48, 59, 60]
3	Scrum	Scrum1k, Scrum10k, Scrum100k	Configurations of the scrum feature model	124/3	1,001–100,001	[46–48, 59]
8	Feature Models	FFM-*, FM-*	Optimize number of variables, constraints and Clause/Constraint ratio	128-1,044/3	10,001	[46–48, 59]
1	Software Process Model	nasa93dem	Optimize effort, defects, time and LOC	24/3	93	[45, 47, 48, 60]
1	Software Process Model	COC1000	Optimize risk, effort, analyst experience, etc	20/5	1,001	[12, 45, 48, 59, 60]
4	Software Process Model	POM3 (A–D)	Balancing idle rates, completion rates and cost	9/3	501–20,001	[45–48, 60]
4	Software Process Model	XOMO (Flight, Ground, OSP)	Optimizing risk, effort, defects, and time	27/4	10,001	[12, 45–48, 60]
3	Miscellaneous	auto93, Car_price, Wine_quality	Miscellaneous	5-38/2-5	205–1,600	[25, 45, 47, 48, 59, 60]
4	Behavioral	all_players, student_dropout, HR-employeeAttrition, player_statistics	Analyze and predict behavioral patterns	26-55/1-3	82–17,738	From [1, 22, 55, 56]
4	Financial	BankChurners, home_data, Loan, Telco-Churn	Financial analysis and prediction	19-77/2-5	1,460–20,000	From [11, 19, 29, 76]
3	Human Health Data	COVID19, Life_Expectancy, hospital_Readmissions	Health-related analysis and prediction	20-64/1-3	2,938–25,000	From [20, 31, 58]
2	Reinforcement Learning	A2C_Acrobot, A2C_CartPole	Reinforcement learning tasks	9-11/3-4	224–318	
5	Sales	accessories, dress-up, Marketing_Analytics, socks, wallpaper	Sales analysis and prediction	14-31/1-8	247–2,206	From [21, 36, 62]
2	Software testing	test120, test600	Optimize the class	9/1	5,161	
127	**Total**					

defaults yield 480× performance degradation vs. optimal parameters [34]. Such poor performance is hardly surprising since industrial optimization faces major obstacles:

- Configuration spaces explode exponentially (in 7z: 14 parameters = one million configurations).
- Performance landscapes are rugged and sparse [15, 27, 45], creating local optima traps.
- Evaluation is costly: x264's 11 parameters need 1,536 hours to explore [67], limiting budgets to dozens of evaluations [14, 54].

Krishna et al. report that these obstacles result in the delivery of suboptimal products [43].

Third, MSR algorithms excel at large configuration spaces [37]. Configuration optimization seeks $c^* \in C$ optimizing multiple objectives (e.g., *maximize* database throughput, *minimize* energy). Given $f : C \to \mathbb{R}^M$ mapping configurations to performance metrics:

$$c^* = \text{argmax}_{c \in C}\, f(c)$$

The space of possible configurations seem too large to explore. For example, MySQL's 460 binary options generate 2^{460} configurations—more than the 2^{80} stars in the observable universe [17]. But in the past, MSR researchers have successfully tackled analogous large-space problems. Yu et al. showed vulnerability detection across tens of thousands of reports needed only hundreds of support vectors [74], which is to say that buried inside seemingly complex configuration spaces contain exploitable and simpler structure.

Enter MOOT. To address this critical opportunity, we built MOOT, a curated repository of configuration and optimization data drawn from top SE venues (e.g. ICSE, FSE, TSE, IST, EMSE, TOSEM, ASE, etc. [12, 13, 15, 16, 24, 30, 33, 35, 38, 42, 44, 45, 50, 51, 53, 57, 68, 70]). Using this resource, researchers have already produced new state-of-the-art techniques for SE optimization, software configuration, cloud tuning (Apache, SQL, X264), project health prediction, feature models, process modeling (nasa93dem, COC1000, POM3, XOMO), behavioral analytics, financial risk, churn prediction, health data, reinforcement learning, sales forecasting, testing, text mining, and more. With MOOT, these ideas (previously validated on a few datasets) can now be evaluated at scale, opening the door to more research questions (see end of this article), broader validation, replication, and discovery of more general principles. For an example of this process, see §4 where clustering methods enable very fast optimization.

In summary: researchers and industry have complex configuration problems which MSR can solve with data mining. MOOT is one way to explore such problems.

Figure 2: An example of a MOOT dataset.

```
x = independent values              | y = dependent values
-----------------------------------|---------------------
Spout_wait, Spliters, Counters, | Throughput+, Latency-
   10,         6,        17,      |     23075,    158.68
    8,         6,        17,      |     22887,    172.74
  [Skipped],   ...,       ...,             ...,      ...
10000,         1,        18,      |    402.53,    8797.5
10000,         1,         1,      |    310.06,    9421
```

3 Inside MOOT

Table 1 showed the current 120+ tasks in MOOT. Each task has:

- One to 11 goals (median=3)
- 3 to 10044 input variables (median=11)
- and 100 to over 100,000 instances (median=10,000).

The last few rows of Table 1 show non-SE datasets. These are useful for explaining MOOT to visitors from other fields of research.

To the best of our knowledge, MOOT is the largest and most varied collection of real multi-objective optimization tasks in SE. Earlier resources (e.g., SPLOT) were valuable but narrow and are now offline. Toolkits like Pygmo or Platypus offer synthetic benchmarks like ZDT, DTLZ, Rosenbrock's banana[7]. But it's somewhat bananas to expect synthetic problems to convince hard-nosed business users. MOOT's datasets come from published SE studies, real performance logs, cloud systems, defect predictors, and tuning tasks where bad configurations cost time, money, and credibility.

Fig. 2 shows the typical structure of a MOOT dataset. In Fig. 2, the goal is to tune *Spout_wait, Spliters, Counters* in order to achieve the best *Throughput/Latency*. As seen in that figure,

(1) MOOT datasets are tables with x inputs and y goals.
(2) The first row shows the column names.
(3) The other rows show examples of a $y = f(x)$ relation.
(4) Numerics start with uppercase letters, all else are symbolic.
(5) Goal columns (e.g. Fig. 2's *Throughput+, Latency-*) use +/- to denote maximize and minimize.
(6) Columns with uppercase "X" at the end of their names are to be ignored by the inference.

For a larger example of MOOT data, see the repository itself; e.g https://github.com/timm/moot/blob/master/optimize/config/SS-M.csv. For the purposes of illustration, the rows in Figure 2 are sorted from best to worst based on those goals. During experimentation, row order should initially be randomized.

4 Using MOOT: an Example

MOOT supports many research questions (see Table 2). This section offers baseline results of one of those questions.

Consider the problem of making conclusions via labeled data. Finding authoritative labels remains a major challenge for MSR-style research [5, 32, 39, 64, 65, 69, 75]. **Labeling by human experts** is possible but slow and error-prone when rushed [23], often taking hours for just a few cases [41, 45, 66]. **Historical logs** provide large label sets but are frequently unreliable (e.g., 90% of technical debt "false positives" were incorrect [75]; similar issues occur in security [69], static analysis [40], and defect data [61]).

[7]The banana is a task with a non-convex decision frontier; i.e. minimize $\sum_{i=1}^{n-1}\left[100(x_{i+1}-x_i^2)^2+(x_i-1)^2\right], \quad -2.048 \le x_i \le 2.048, \quad i=1,\ldots,n.$

Automated labeling also faces limits: regex-based heuristics are crude [39]; LLMs are only assistive (but not authoritative). Some domains have naturally occurring oracles (e.g. compile with certain Makefile settings then run the full test suite) but, as noted, these can be extremely slow (e.g. recall the x264 example [67]).

When we cannot trust many labels, we must do what we can with very few labels. Enter data mining. The `rq2.sh` script of http://tiny.cc/moot0 clusters data, labels a few items per cluster, then uses that information to guess the labels of nearby items. We make no claim that this **BASELINE** MOOT optimizer is state-of-the-art. We only show it here as an example of optimize-via-data-mining (in this case, Euclidean distance measures and clustering).

After dividing data 50:50 into *train* and *hold-out*, **BASELINE**:

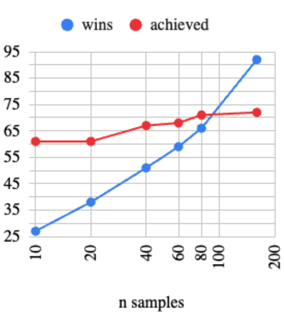
- **TRAINS a model:**
  - Label, say, $n_1 = 30$ randomly selected training examples;
  - Sort them on y
  - Divide sorted data into a $\sqrt{n}$ best set and a $n - \sqrt{n}$ rest set
  - Return $Model(row) = dist(row, C_b) - dist(row, C_r)$ where C_b and C_r are the centroids on best and rest sets; and *dist* is Euclidean distance between x values
- **TESTS a model on hold-outs:**
  - Sort the hold-out on y using *Model*
  - Label the first, say, $n_2 = 10$ items in that sort
  - Return the best of these as the recommendation

Note that this process uses only $n = n_1 + n_2$ labels, which in this case, was $30 + 10 = 40$.

Figure 3 compares using only n labels versus all $N > n$, evaluated with nonparametric tests (Kolmogorov–Smirnov for significance and Cliff's Delta for effect size, see MOOT0's `stats.py`).

The **BLUE** curve shows the usual story; i.e. more labels ($N > n$) wins statistically. But the **RED** curve shows something unexpected: after about 40 labels, there is very little further improvement (from 68 to 73). For mission-critical and safety-critical applications, that improvement may be important. But for many other applications, Figure 3 suggests that a mere 40 labels could suffice.

Note that this result runs contrary to much of the current fashion in AI since it says more data is not always better. If this holds for other optimizers (and not just this simplistic **BASELINE** method) then this would be an important insight for resource-limited processes such e.g., cutting cloud costs, deploying on the edge, or avoiding fatigue in human-in-the-loop labeling.



For the **BLUE** curve, y is the percent frequency where $x < 160$ samples performed statistically as well as 160 samples.

For the **RED** curve, y shows the performance of the row recommended by **BASELINE** after x samples. This number is normalized such that $y = 0$ means "same as the untreated mean goal score" (i.e. no improvement); and $y = 100$ means "best possible row found".

Figure 3: Mean results (20 repeats), BASELINE on 127 datasets.

Table 2: Research questions: a roadmap for future work using MOOT data.

A. Core Optimization Strategies & Performance: How best to find good solutions?

- Sample size
  - **Minimality:** What is the minimum budget for reliable optimization?
  - **Surprising Simplicity:** Preliminary results (e.g. Figure 3) suggest optimization in SE is surprisingly simple. What makes SE problems unique and so simple?
- Optimization:
  - **Algorithms:** Which AI optimization algorithms are best for MOOT data?
  - **Baselines:** If publishing on MOOT, what are the state of the art algorithms against which we should compare?
  - **Strategy:** When is it best to use *"Pool-based"* search of pre-enumerated examples or *"Query-based"* methods that interpolate between existing examples?
  - **Landscape:** Are optimization results controlled by algorithms? Or if controlled by the overall “shape” of the data, do we need different “landscape aware” optimizers?
- Metrics:
  - **x-metrics:** How do distance metrics (Euclidean, Hamming) affect performance, and can we predict the best?
  - **y-metrics:** What measures (e.g., "distance to heaven", Chebyshev, HV, IGD) best identify good solutions?
  - **Surrogates:** If evaluating via quickly built approximations (e.g. random forests), how to certify the surrogates? How good do they need to be?
- **Ensembles:** Can ensemble of algorithms or ensembles built over bagged data outperform single-algorithm approaches?
- **Causality:** Does using causality (rather than just correlation) improve optimization?

B. Human Factors & Interpretability: Making solutions understandable and usable.

- **Explanation:** How can we (visually or otherwise) explain opaque optimization results to help stakeholder decisions?
- **Trade-offs:** What analytics best help humans understand the decision space (especially for 5+ objectives)?
- **Requirements engineering:** Using MOOT, can we test requirements engineering methods to help stakeholders trade off between their different concerns?
- Bias:
  - **Unfairness:** Optimization is always biased towards the stated goals. Does that routinely disadvantage certain social groups?
  - **Fairness Repair:** If unfair, can we fix it (e.g. with more optimization goals)?
- **Knowledge:** How can we best incorporate other knowledge sources (human, ensemble)?
- **Human Acceptance:** Will humans accept recommendations, and do explanations change their minds?
- **Human vs. Machine:** How do human vs. machine explanations for "good" configurations compare?

C. Industrial Deployment & Adoption: Bridging the gap-research and practice.

- **Problem Identification:** How can we identify and validate high-impact industrial problems where MSR optimization techniques would provide the most value?
- **Case Studies:** What are the main barriers to applying these methods to real-world industrial systems, and how can they be overcome?
- **Data Collection:** How can we incentivize and manage the collection of new industrial (perhaps proprietary) optimization datasets for MOOT?
- **Education:** How to introduce newbies (in industry and academia) to these methods?
- **Deployment:** What tools & tutorials are needed for widespread use of all this ?

D. LLMs & Emerging AI Technologies: Foundation models and optimization.

- **Distillation:** Figure 3 suggests that a few labels can replace a large set. Could this be used to guide LLM distillation? (finding small useful models inside larger models?)
- **Configuration Generation:** Can LLMs generate effective initial configurations or suggest promising search directions based on natural language descriptions?
- **Explanation Translation:** Can LLMs translate technical optimization results into domain-specific recommendations practitioners understand?
- **Prompt Engineering as Configuration:** Can MOOT techniques optimize LLM prompts as multi-objective configuration problems?
- **LLM-Generated Surrogates:** Can LLMs learn to approximate expensive evaluation functions and serve as fast surrogates during optimization?
- **Natural Language Constraints:** Can practitioners specify configuration constraints in natural language, with LLMs translating to formal specifications?
- **Meta-Learning:** Can LLMs identify which optimization strategies work best for new problems based on problem descriptions?

E. Solution Quality & Generalizability: Understand solution reliability.

- **Generalization:** Any commonalities in MOOT-generates solutions?
- **Transferability:** Can we transfer knowledge between tasks, or must we start fresh?
- **Stability:** How stable are conclusions across stochastic multi-objective algorithms?
- **Robustness:** Robustness of results to noise, incomplete data, changing conditions?
- **Uncertainty:** Can we learn confidence intervals around our conclusions?
- **Failure Prediction:** Can data features predict optimization failure in advance?
- **Constraints:** How to automatically detect and handle infeasible configurations?

F. Future Directions: Exploring new domains and advanced concepts.

- **Temporal:** Reasoning across time. How to leverage the past? How (and when) to unlearn (to forget some of the past)?
- **Philosophical:** Is solution space "flat" (many "near-maxima") so no single "truth"?
- **Artificial General Intelligence:** If we can learn, optimize, trade off goals, transfer knowledge to new domains– when does this become AGI?

5 Discussion

To summarize the above: (a) industry and researchers need better ways to solve configuration problems; (b) MSR provides scalable data mining methods for this task; and (c) MOOT enables this community to explore those kinds of task, using large numbers of realistic examples In this way, MOOT enables harder and more credible research. MOOT lets us replace studies on toy problems (or just half a dozen hand-picked examples) with case studies on 120+ examples. Such studies could focus on stability, sample efficiency, failure modes, cross-domain generality, or many other questions (see Table 2).

MOOT is open: new data sets are welcome via pull request. We plan (a) annual ICSE research events on empirical optimization in SE, where results must generalize across MOOT data; and (b) tutorials at major SE venues (ICSE, FSE, ASE, MSR) on using MOOT-style tools. We have also arranged for expedited publication of MOOT work in the *Automated Software Engineering* journal (either as full papers or three-page tools/tutorials/registered reports[8]). To qualify, mention “MOOT” in the title or abstract and use MOOT data.

[8] See https://ause-journal.github.io/cfp.html

Depending on how we use it, MOOT could be a *catalyst* or *cage* for future research. Used thoughtfully, it could broaden the questions we ask and the answers we trust. But if used carelessly (obsessively, mindlessly) then MOOT could suffer the same lifecycle as many other repos:

(1) *Rejected* — “Find Data? That will never happen.”
(2) *Respected* — “Fine. . . it helps sometimes.”
(3) *Expected* — “You must compare to this baseline.”
(4) *Exhausted* — the research graveyard where innovation stalls; results become repetitive and derivative.

To avoid the trap of the research graveyard, we should discourage papers that only offer minor percentage gains over existing baselines. Instead, we should encourage work that, for example:

- introduces new tasks or domains; or
- challenges or expands current assumptions; or
- extends MOOT itself (data, tooling, research directions, scripts, organization, or community).

References

[1] Abdullah0a. Student dropout analysis and prediction dataset, 2025. Kaggle.
[2] A. Agrawal, W. Fu, and T. Menzies. What is wrong with topic modeling? and how to fix it using search-based software engineering. *IST*, 98:74–88, 2018.

[3] A. Agrawal, W. Fu, D. Chen, X. Shen, and T. Menzies. How to "dodge" complex software analytics. *IEEE Trans Softw Eng*, 47(10):2182–2194, 2019.
[4] A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu. Better software analytics via "duo": Data mining algorithms using/used-by optimizers. *Empir. Softw. Eng.*, 25:2099–2136, 2020.
[5] T. Ahmed, P. Devanbu, C. Treude, and M. Pradel. Can llms replace manual annotation of software engineering artifacts? In *MSR*, pages 526–538, 2025.
[6] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proc. 25th ACM SIGKDD international conf. on knowledge discovery & data mining*, pages 2623–2631, 2019.
[7] M. Ali et al. Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning. *PeerJ CS*, 10:e1860, 2024.
[8] N. Awad, N. Mallik, and F. Hutter. DEHB: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In *Proc. of the 13th International Joint Conference on Artificial Intelligence*, pages 2147–2153, 2021.
[9] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proc. NeurIPS*, pages 2546–2554, 2011.
[10] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for model selection and hyperparameter optimization. In *Computational Science & Discovery*, volume 8, page 014008. IOP Publishing, 2015.
[11] blastchar. Telco customer churn, 2025. Kaggle.
[12] J. Chen, V. Nair, and T. Menzies. Beyond evolutionary algorithms for search-based software engineering. *Inf. Softw. Technol.y*, 95:281–294, 2018.
[13] J. Chen, V. Nair, R. Krishna, and T. Menzies. "sampling" as a baseline optimizer for search-based software engineering. *IEEE Trans. Softw. Eng.*, 45:597–614, 2019.
[14] J. Chen, N. Xu, P. Chen, and H. Zhang. Efficient compiler autotuning via bayesian optimization. In *43rd IEEE/ACM Int. Conf. Softw. Eng. ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 1198–1209, 2021.
[15] P. Chen and T. Chen. Promisetune: Unveiling causally promising and explainable configuration tuning. In *Proc. of the 48th IEEE/ACM Int. Conf. Softw. Eng.*, 2026.
[16] P. Chen, J. Gong, and T. Chen. Accuracy can lie: On the impact of surrogate model in configuration tuning. *IEEE Trans. Softw. Eng.*, 51(2):548–580, 2025.
[17] T. Chen. Personal communication with author, at icse'25, ottawa, canda, May 2025. Regarding the results of the K-means++ experiment.
[18] T. Chen and M. Li. Do performance aspirations matter for guiding software configuration tuning? an empirical investigation under dual performance objectives. *ACM Trans. Softw. Eng. Methodol.*, 32(3):68:1–68:41, 2023.
[19] dansbecker. Home data for ml course, 2025. Kaggle.
[20] dansbecker. Medical data and hospital readmissions, 2025. Kaggle.
[21] J. Daoud. Marketing analytics – marketing data, 2022. Kaggle.
[22] die9origephit. Fifa world cup 2022: Complete dataset, 2025. Kaggle.
[23] M. Easterby-Smith. The design, analysis and interpretation of repertory grids. *International Journal of Man-Machine Studies*, 13(1):3–24, 1980.
[24] W. Fu, T. Menzies, and X. Shen. Tuning for software analytics: Is it really necessary? *Inf. Softw. Technol.*, 76:135–146, 2016.
[25] K. K. Ganguly and T. Menzies. Bingo! simple optimizers win big if problems collapse to a few buckets. *arXiv preprint arXiv:2506.04509*, 2025.
[26] J. Gong and T. Chen. Predicting configuration performance in multiple environments with sequential meta-learning. *FSE*, 1:359–382, 2024.
[27] J. Gong and T. Chen. Deep configuration performance learning: A systematic survey and taxonomy. *ACM Trans. Softw. Eng. Methodol.*, 2025.
[28] J. Gong, T. Chen, and R. Bahsoon. Dividable configuration performance learning. *IEEE Trans. Softw. Eng.*, 2025.
[29] S. Goyal. Credit card customers, 2020. Kaggle.
[30] H. Ha and H. Zhang. Deepperf: performance prediction for configurable software with deep sparse neural network. 2019.
[31] Hendratno. Covid-19 indonesia dataset, 2022. Kaggle.
[32] S. Herbold, A. Trautsch, F. Trautsch, and B. Ledel. Problems with szz and features: An empirical study of the state of practice of defect prediction data collection. *Empir. Softw. Eng.*, 27(2):42, 2022.
[33] J. Hulse, N. U. Eisty, and T. Menzies. Shaky structures: The wobbly world of causal graphs in software analytics. *Empir. Softw. Eng.*, 2025.
[34] P. Jamshidi and G. Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *Proc. 24th IEEE Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2016.
[35] P. Jamshidi, M. Velez, C. Kästner, and N. Siegmund. Learning to sample: exploiting similarities across environments to learn performance models for configurable systems. *FSE*, 2018.
[36] jessicali9530. Animal crossing new horizons: Nookplaza dataset, 2021. Kaggle.
[37] G. Jianmei, C. Krzysztof, A. Sven, S. Norbert, and W. Andrzej. Why cart works for variability-aware performance prediction? an empirical study on performance distributions. Technical report, GSD Lab, University of Waterloo, 2013.
[38] G. Jianmei, Y. Dingyu, S. Norbert, A. Sven, S. Atrisha, V. Pavel, C. Krzysztof, W. Andrzej, and Y. Huiqun. Data-efficient performance learning for configurable systems. *Empir. Softw. Eng.*, 2018.
[39] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Trans. Softw. Eng.*, 39(6):757–773, 2012.
[40] H. J. Kang, K. L. Aw, and D. Lo. Detecting false alarms from automatic static analysis tools: How far are we? In *44th Int. Conf. Softw. Eng.*, 2022.
[41] A. Kington. *Defining Teachers' Classroom Relationships.* 01 2009.
[42] J. Krall, T. Menzies, and M. Davies. Gale: Geometric active learning for search-based software engineering. *IEEE Trans. Softw. Eng.*, 41(10):1001–1018, 2015.
[43] R. Krishna, Z. Yu, A. Agrawal, M. Dominguez, and D. Wolf. The "bigse" project: lessons learned from validating industrial text mining. In *Workshop on BIG Data SE*, page 65–71, 2016.
[44] R. Krishna, V. Nair, P. Jamshidi, and T. Menzies. Whence to learn? transferring knowledge in configurable systems using beetle. *TSE*, 47(12):2956–2972, 2020.
[45] A. Lustosa and T. Menzies. Learning from very little data: On the value of landscape analysis. *TOSEM*, 33(3):1–22, 2024.
[46] A. Lustosa and T. Menzies. isneak: Partial ordering as heuristics for model- based reasoning in software engineering. *IEEE Access*, 12:142915–142929, 2024.
[47] A. Lustosa and T. Menzies. Less noise, more signal: Drr for better optimizations of se tasks. *arXiv preprint arXiv:2503.21086*, 2025.
[48] T. Menzies. The case for compact ai. *Commun. ACM*, 68(8):6–7, 2025.
[49] T. Menzies, T. Chen, K. Ganguly, A. Rayegan, S. Srinivasan, A. Lustosa, and K. Vilkomir. Moot: a repository of many multi-objective optimization tasks, October 2025. URL https://doi.org/10.5281/zenodo.17354083.
[50] S. Mühlbauer, F. Sattler, C. Kaltenecker, J. Dorn, S. Apel, and N. Siegmund. Analysing the impact of workloads on modeling the performance of configurable software systems. In *ICSE*, pages 2085–2097, 2023.
[51] V. Nair, T. Menzies, N. Siegmund, and S. Apel. Using bad learners to find good configurations. In *FSE*, pages 257–267, 2017.
[52] 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.
[53] V. Nair, T. Menzies, N. Siegmund, and S. Apel. Faster discovery of faster system configurations with spectral learning. *Autom. Softw. Eng.*, 25:247–277, 2018.
[54] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel. Finding faster configurations using FLASH. *IEEE Trans. Software Eng.*, 46(7):794–811, 2020.
[55] nyagami. Ea sports fc 25 database — ratings and stats, 2025. Kaggle.
[56] Pavansubhasht. Ibm hr analytics employee attrition performance, 2025. Kaggle.
[57] K. Peng, C. Kaltenecker, N. Siegmund, S. Apel, and T. Menzies. Veer: enhancing the interpretability of model-based optimizations. *Empir. Softw. Eng.*, 28:61, 2023.
[58] K. A. Rajarshi. Life expectancy (who) dataset, 2025. Kaggle.
[59] A. Rayegan and T. Menzies. Minimal data, maximum clarity: A heuristic for explaining optimization. *arXiv preprint arXiv:2509.08667*, 2025.
[60] L. Senthilkumar and T. Menzies. Can large language models improve se active learning via warm-starts? *arXiv preprint arXiv:2501.00125*, 2024.
[61] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Trans. Softw. Eng.*, 39(9):1208–1215, 2013.
[62] syedfaizanalii. Car price dataset – cleaned, 2025. Kaggle.
[63] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *Proc. of the 38th Int. Conf. Softw. Eng.*, pages 321–332, 2016.
[64] H. Tu and T. Menzies. Better data labelling with emblem (and how that impacts defect prediction). *IEEE TSE*, 2019.
[65] H. Tu and T. Menzies. Frugal: Unlocking semi-supervised learning for software analytics. In *ASE*, pages 394–406, 2021.
[66] R. Valerdi. Heuristics for systems engineering cost estimation. *IEEE Systems Journal*, 5(1):91–98, 2010.
[67] P. Valov, J. Petkovich, J. Guo, S. Fischmeister, and K. Czarnecki. Transferring performance prediction models across different hardware platforms. In *Proc. of the 8th ACM/SPEC on Int. Conf. Perf. Eng.*, pages 39–50, 2017.
[68] M. Weber, C. Kaltenecker, F. Sattler, S. Apel, and N. Siegmund. Twins or false friends? A study on energy consumption and performance of configurable software. In *ICSE*, pages 2098–2110, 2023.
[69] X. Wu, W. Zheng, X. Xia, and D. Lo. Data quality matters: A case study on data label correctness for security bug report prediction. *TSE*, 48(7):2541–2556, 2021.
[70] T. Xia, R. Shu, X. Shen, and T. Menzies. Sequential model optimization for software effort estimation. *IEEE Trans. Softw. Eng.*, 48:1994–2009, 2020.
[71] Z. Xiang, J. Gong, and T. Chen. Dually hierarchical drift adaptation for online configuration performance learning. In *ICSE*, 2026.
[72] R. Yedida and T. Menzies. On the value of oversampling for deep learning in software defect prediction. *IEEE Trans. Softw. Eng.*, 48(8):3103–3116, 2021.
[73] R. Yedida, H. J. Kang, H. Tu, X. Yang, D. Lo, and T. Menzies. How to find actionable static analysis warnings. *IEEE Trans. Softw. Eng.*, 49:2856–2872, 2023.
[74] Z. Yu, C. Theisen, L. Williams, and T. Menzies. Improving vulnerability inspection efficiency using active learning. *IEEE Trans. Softw. Eng.*, 47:2401–2420, 2019.
[75] Z. Yu, F. M. Fahid, H. Tu, and T. Menzies. Identifying self-admitted technical debts with jitterbug: A two-step approach. *IEEE Trans. Softw. Eng.*, 2021.
[76] L. Zoppelletto. Financial risk for loan approval, 2025. Kaggle.

Received 23 Oct 2025