Can Large Language Models Improve SE Active Learning via Warm-Starts?

LOHITH SENTHILKUMAR, TIM MENZIES, NC State, USA

Software engineering now demands the navigation of vast configuration spaces, not only for software systems but also for the processes used to build them. Misconfigurations threaten performance, stability, and security. Worse, our ability to manually reason across large design spaces is limited. Hence we say (a) configurability is a liability without tool support and (b) as a priority, SE needs to explore better configuration tools.

Finding good configurations is a black art, requiring much experience. When that experience is scarce, "active learners" can build configuration models by labeling the most informative examples (so configurations can be generated using very little data).

Active learning can benefit from good initial guesses (known as "warm starts"). This paper tries using Large Language Models (LLMs) for creating warm-starts. Results are compared against Gaussian Process Models and Tree of Parzen Estimators. For 49 SE tasks, LLM-generated warm starts significantly improved the performance of low- and medium-dimensional tasks. However, LLM effectiveness diminishes in high-dimensional problems, where Bayesian methods like Gaussian Process Models perform best.

In order to support open science, our scripts and data are online at github.com/timm/moot and github.com/lohithsowmiyan/lazy-llm.

CCS Concepts: • **Software and its engineering → Active learning**; • **Computing methodologies → Machine learning → Supervised learning / semi-supervised learning (including active learning)**; • **Software and its engineering → Search-based software engineering for system configuration**; • **Computing methodologies → Machine learning → Unsupervised learning / Dimensionality reduction**;

Additional Key Words and Phrases: Software configuration optimization; Multi-objective optimization; Bayesian optimization; Tree of Parzen Estimators (TPE); Warm-start

1 Introduction

Modern software engineering is characterized by a plethora of choices. Given a goal, there are many ways to implementing all the parts that lead to that goal. And across those choices there is a large space of magic configuration parameters that must be set [24].

While configuration flexibility is a strength, it comes at a steep cost. This space of choices can be very large. For example, Chen et al. report over a billion combinations of choices just within the Makefiles of SQLite. Configuration spaces are combinatorially large[1], often poorly documented, and rich in subtle interactions. Defaults are unreliable, manual tuning is unscalable, and human intuition is fallible [82]. As systems and processes grow in complexity, the ability to manage these configurations—both statically and adaptively—becomes very difficult [120]. Part of that difficulty arises from the necessity to trade off between competing constraints; e.g how to deliver *more* code but at *less* cost?. Or how to answer database queries *faster* but use *less* energy? Hence, like other researchers [6, 37], we warn:

Configurability is a liability without tool support.

Accordingly, we explore multi-objective optimization for configuration support. Optimizatiers are tools that try to satisfy numerous competing constraints [44]. Many of these algorithms refine an initial set of guesses [79, 125]. One way to improve such optimization is to improve the initial guess. Such "warm start" policies use some background knowledge to make informed decisions about those initial guesses. For example, warm starts can be generated by asking the opinion of some subject matter expert (SME) [43, 65, 122].

[1]e.g. Table 1 of Agrawal et al. [1] shows one hundred sextillion (10^{46}) ways to configure data miners that using some text pre-processing.

Author's Contact Information: Lohith Senthilkumar, Tim Menzies, lohithsowmiyan1@gmail.com, timm@ieeee.org, NC State, Computer Science, Raleigh, NC, USA.

As described later in this paper (in §2.4.3), SMEs are often in short supply. Real-world SMEs are experts precisely because their expertise is valuable to the organization. This means that SMEs are often called away to other tasks. So, where can we find the expertise needed to generate the warm starts?

Large Language Models (LLMs) have proved useful in many areas of SE [21, 48, 107, 116]. Hence it is timely to ask if LLMs can help the optimization of SE configuration by synthesizing plausible initial guesses. As shown by the literature review of this paper, this is an underexplored area, particularly for multi-objective, tabular SE data where labeling is slow and/or expensive. Accordingly, we explore LLMs and warm starts for the 49 multi-objective SE optimization tasks [25, 41, 68, 75, 77, 83, 87, 92] shown in Table 1. To structure this investigation, we ask: **R0.1a**

- **RQ1**: *Are warm starts useful for active learning?* The premise of this work is that the warm start tactic is worthy of exploration. To check that, this paper compares active learning results, with and without warm starts.

Table 1. Data sets used in this paper. x/y shows the number of independent x vales and dependent y values. The last column show the heuristic categorization of data sets by Di Fiore et al. [32] (data sets with less than 6 or 11 x columns are *low* or *medium* complexity while other data sets are *high* complexity). For further notes on this data, see §3.1.

Groups	**File Name**	|rows|	x / y	**Dims.**
Software Config	SS-A	864	3/2	low
	SS-B	972	3/2	low
	SS-C	1080	3/2	low
	SS-D	2736	3/2	low
	SS-E	3008	3/2	low
	SS-F	3839	3/2	low
	SS-G	5184	3/2	low
	SS-H	4608	4/2	low
	SS-I	6840	5/2	low
	SS-J	65536	6/2	medium
	SS-K	86058	6/2	medium
	SS-L	1023	11/2	low
	SS-M	864	17/3	high
	SS-N	86058	18/2	high
	SS-O	972	11/2	high
	SS-P	1023	11/2	high
	SS-Q	2736	13/3	high
	SS-R	3008	14/2	high
	SS-S	3840	6/2	medium
	SS-T	5184	12/2	high
	SS-U	4608	21/2	high
	SS-V	6840	16/2	high
	SS-W	65536	16/2	high
	SS-X	86058	11/3	high
	Apache AllMeasurements	192	9/1	medium
	SQL AllMeasurements	4653	38/1	high
	X264 AllMeasurements	1152	16/1	high
	rs-6d-c3 obj1	3840	6/1	medium
	rs-6d-c3 obj2	3840	6/1	medium
	sol-6d-c2-ob j1	2866	6/1	medium
	wc-6d-c1-ob j1	2880	6/1	medium
	wc+sol-3d-c4-ob j1	196	3/1	low
	wc+rs-3d-c4-obj1	196	3/1	low
	wc+wc-3d-c4-ob j1	196	3/1	low
Software Process Modeling	pom3a	500	9/3	medium
	pom3b	500	9/3	medium
	pom3c	500	9/3	medium
	pom3d	500	9/3	medium
	xomo_flight	10000	23/4	high
	xomo_ground	10000	23/4	high
	xomo_osp	10000	23/4	high
	xomo_osp2	10000	23/4	high
	coc1000	1000	17/5	high
	nasa93dem	93	22/4	high
Project Health	healthCloseIsses12mths0001-hard	10000	5/1	low
	healthCloseIsses12mths0011-easy.csv	10000	5/1	low
Miscellaneous	auto93	398	5/3	low
	Wine_quality	1599	10/2	medium
	HSMGP num	3456	14/1	high

- **RQ2**: *Are LLMs the best way to generate warm starts?* This is main research question of this paper. Here we will compare LLM-driven warm starts to alternative methods.

As seen in the results of this paper, **RQ1** will demonstrate the value of warm starts; and **RQ2** will show that, sometimes, LLMs are good way to generate those warm starts. LLMs significantly improve outcomes for low-dimensional configuration problems, reducing the need for extensive labeled data. However, for higher-dimensional tasks, LLMs face challenges in synthesizing effective examples, highlighting the continued relevance of Bayesian methods in such scenarios.

This study makes the following contributions:

- A novel method leveraging LLMs to warm-start active learning for SE configuration tasks (see §3.3).
- An empirical comparison of LLM-based methods with alternate approaches across 49 datasets (see §4).
- Insights into the strengths and limitations of LLMs in addressing multi-objective SE problems (see §4.2).
- A reproducible package of data and scripts for benchmarking active learning strategies[2].

The rest of the paper is organized as follows: Section 2 discusses our motivation and related work, Section 3 details our methodology, Section 4 presents experimental results, and Section 5 discusses implications and future directions.

2 Background

2.1 "But is this an SE problem?"

This section motivates the need for this kind of analysis in software engineering. We will say that (a) configuration is a major problem in software engineering and (b) optimization is a useful and necessary tool for selecting better configurations.

2.1.1 The Configuration Crisis. Xu et al. [120] documents the **configuration crisis** in software engineering. As systems mature, developers keep adding more configuration options. But it is difficult to understand and apply this exponentially exploding space of options. Hence humans make less and less use of that configuration space. Xu et al. document this effect in the two plot of Figure 1:

- Figure 1.a show an effect seen in many systems: after a systems is released, the number of configuration parameters grows perpetually large.
- This ever-increasing number of configuration options is a problems since, as seen in Figure 1.b, another troubling effect is that most users ignore most parameters. For example, Xu et al. found that 80% of the parameters are ignored by 70,80,90% of users for Apacke, MySQL, and Storage-A.

Van Aken et al. [3] also reports the same trend as Xu et al. They indicate that, in over 15 years, the number of configuration options of POSTGRES and MYSQL increased by a factor of three and six, respectively. The configuration crisis means that between (a) what software can do, and (b) what it is actually are asked to do, there is a very large *configuration gap* filled with sub-optimal systems.

2.1.2 Misconfiguration as a Primary Risk: Misconfiguration is is a primary source of software failure. In a foundational study, Zhou et al. [37] examined system failure reports and found that *"Configuration errors are among the top contributors to system outages. They are difficult to avoid, hard to detect, and costly to debug"*. Their work demonstrated that in real-world deployments of MySQL, Apache, and Hadoop, over 40% of failures stemmed from configuration errors.

[2]Data:https://github.com/timm/moot. Code: https://github.com/lohithsowmiyan/lazy-llm

Figure 1a: software accumulates config.

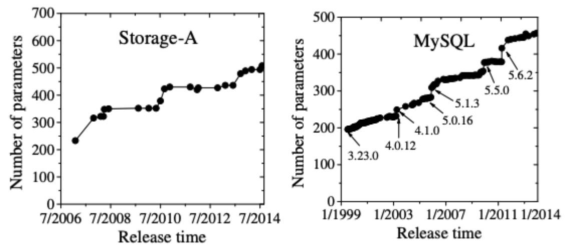


Figure 1b: much config is ignored

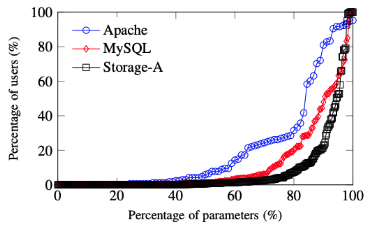


Fig. 1. Number of configuration parameters over time (left). Usage of those parameters (right).

These failures are not simply due to complexity, but also to neglect:

- In the case of MySQL, Van Aken et al. [3] noted that default configurations in 2016 were based on obsolete assumptions—e.g., systems having only 160MB of RAM—resulting in underutilized resources and poor performance.
- Herodotou et al. [45] show how standard settings for text mining applications in Hadoop result in worst-case execution times.
- Jamshidi [50] reports that for text mining applications on Apache Storm, the throughput achieved using the worst configuration is 480 times slower than the throughput achieved by the best configuration.

2.1.3 The Case for Auto-Configuration. Apel et al. [6] discuss design principles for variability-intensive systems, such as software product lines or cloud services. In all these contexts, the configuration space is not only large, but also dynamic. Developers face feature interactions, dependency constraints, and shifting non-functional goals. Apel writes:

> *Without automated techniques to reason about performance and correctness, the complexity of configuration spaces will overwhelm developers and users alike.*

Siegmund et al. [99] demonstrate how intelligent sampling can infer performance across vast configuration spaces with high accuracy from very few samples. This is to say, it is useful to treat configurability as a learning problem.

2.1.4 Optimization for Automated Configuration . According to Siegmund et al. [99] exploring the very large space of possible configurations for a system is an *optimization* problem. Consider a system with a configuration space C, where each configuration $c \in C$ is a unique set of parameter settings. Configuration optimizations seeks the configuration c^* that optimizes a set of objectives, such as performance (e.g., throughput, response time), resource consumption (e.g., memory, CPU), or correctness (e.g., error rate). Let $f : C \to \mathbb{R}^M$ be a multi-objective performance function, where M is the number of objectives, which maps a given configuration to a vector of its performance metrics. The challenge then becomes identifying:

$$c^* = \underset{c \in C}{\operatorname{argmax}}\, f(c)$$

This formulation indicates that we are searching for the configuration c^* that yields the maximal performance across all dimensions defined by $f(c)$.

$|C|$ can be astronomically large. For example, suppose all the 460 choices of the 2014 version of MySql (from Figure 1.a) were simple binary options. This leads to a search space of 2^{460} options. To but that number in perspective, there are only 2^{80} stars in the observable universe [33].

It is impractical to perform an exhaustive search of such a large space of configuration options. Therefore, approaches like intelligent sampling, machine learning, and metaheuristics are employed to efficiently explore this complex space and identify configurations that are optimal or near-optimal.

2.1.5 Configuring Process, Not Just Product. Configuration is not limited to product-level concerns like Makefile options—it is equally critical and useful and possible at the process level.

In our SEESAW project [78], we investigated whether significant improvements in software projects could be achieved by intelligently adjusting internal project parameters, rather than introducing *disruptive changes* (e.g. laying off large numbers of employees). SEESAW employed stochastic sampling to identify impactful internal modifications—such as team skill, documentation practices, or tool maturity. Those modifications were ranked based on their influence on effort, defects, and schedule (those influences were determiend using well-established process models developed by Boehm and others [16, 17, 69, 76]).

The results were compelling. In multiple case studies, SEESAW's internal recommendations performed as well as, or better than, radical interventions like team replacement or platform migration. A key insight was that *combinations of internal changes frequently match or exceed the benefits of drastic changes.* This demonstrates that process improvement opportunities are abundant— provided we apply the right configuration tools.

Equally important, SEESAW delivered *stability reports*, identifying recommendations that consistently improved outcomes across a range of uncertainties. From this, we conclude that software process decisions can be treated as configuration problems. Encouragingly, these problems respond well to the same modeling, reasoning, and optimization frameworks used for code-level parameter optimization.

2.1.6 Summary: The SE Configuration Problem. Configuration is a unifying principle across both product and process domains in software engineering. Exploring configuration—of any kind— can thus yield transferable insights across a broad array of SE challenges.

The time has come to treat configuration control as essential infrastructure that requires our full attention. It is as fundamental to software engineering as testing, versioning, or debugging. It supports agility without chaos, performance without brittleness, and change without disruption. Configuration should no longer be viewed as a hasty once-only setup task—it is a discipline that may be required again and again throughout the lifetime of a software project.

The rest of this paper discusses solutions to optimization problems that are catheterized by fitting settings to variables under a wide variety of constraints. All these can be characterized as configuration optimization problems since they all address what are the best choices in order to achieve the particular goals required for particular systems.

In keeping with the above discussion, some of configuration tasks will be process-oriented (e.g. what management decisions to make) and some will be product-oriented (e.g. what settings to apply in a Makefile).

2.2 Improving on the Past

This research team has been exploring optimization for product and process configuration optimization in SE for many years. Our usual tool for that exploration was some kind of stochastic method such as simulated annealing [76], genetic algorithms. local search [41], or other randomizes algorithms [26, 38, 72, 86]. Given the random nature of that approach, the question is often asked "would a more informed approach result in better and/or faster optimizations?".

To address that question, at the start of 2024, we set out to measure the effects of an informed start, on a large number of SE optimization problems. To that end, we collected examples from the SE literature where optimizers had automatically selected parameters for (e.g.) cloud services or data miners or software process options. This resulted in dozens of tables of data summarized in Table 1 (and for full details on that data, see §3.1). Those tables of data had

- 93 to 86,000 rows;
- three to 38 independent x variables; and
- one to five dependent y variables.

In practice, for any row in these tables, the dependent variables values are unknown until some optimizers decides to test a particular set of independent values. The process of computing the dependent variables is called *labeling*. As discussed in §2.4.2, labeling can be very slow and/or expensive. Hence, our optimizations should achieve their goal using less than a few dozen labels (for a justification of that labeling budget, see §2.4.3 and §3.6).

It is prudent to ask if these data sets are connected to real-world SE tasks. All our data sets relate to **real, often highly complex, configuration problems** that exemplify the "configuration crisis" (discussed in §2.1) prevalent in modern software engineering. For example, in Table 1:

- All the SS-* data sets were generated by mutating Makefiles then measuring properties of the system compiled with those mutations, representing tangible challenges in system performance and reliability.
- Other datasets, such as those from the "Software Process Modeling" group, deal with optimizing crucial **organizational and management decisions**, directly addressing the need for tool support in navigating dynamic and large configuration spaces for both product and process as identified by Apel et al. [6].

As further (partial) evidence for the value and relevance of these data sets, we note we collected these from the last five years of the SE literature from papers from TSE, ICSE, etc.

2.3 A Gap in the Literature

Having assembled test data, and defined the optimization task, the literature was searched (in September 2024) for work trying an informed approach for the better generation of initial candidate solutions. After some initial reading, it was seen that “active learning” and “warm start” returned relevant papers. Google Scholar was queried for papers from the last ten years containing those two terms.

To ensure a focused yet comprehensive review, we adopted a structured approach to identifying seminal and highly influential works. After performing an initial broad search, we ranked the retrieved papers by their citation counts. We then identified the ’knee’ of the citation curve (see Figure 2), a common heuristic in bibliometric analysis [23, 110], which marks the point where the rate of citation accumulation significantly decreases. Papers above this threshold typically represent core contributions and foundational research that have significantly shaped the field [93]. This pragmatic stopping rule allowed us to efficiently prioritize the most impactful literature, ensuring that

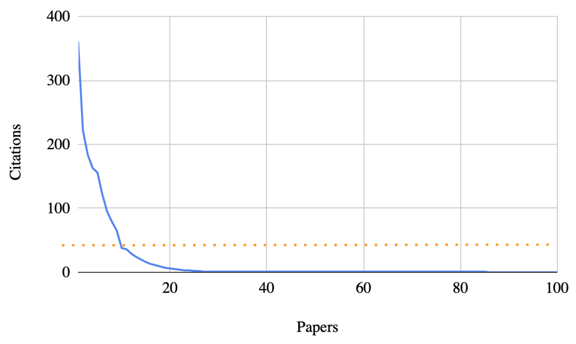
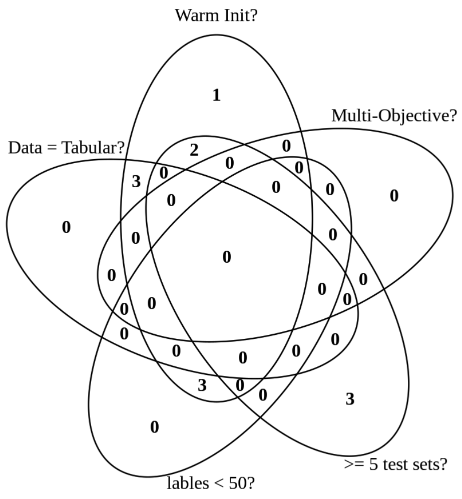


Fig. 2. Citation curve of the first 100 papers returned by the query `"active learning" "warm starts"` on `scholar.google.com` in 2024. The line indicates the “knee” of the curve (computed as the furthest point to a line connecting the first and last points).

Table 2. Literature review results. Cells marked as "-" denotes no information on that point in that paper. The center of the right-hand-side Venn diagram is the gap in the current literature, explored by this paper.

Ref	Citations	Year	Warm Init?	Data= Tabular?	labels < 50?	≥ 5 test sets?	Multi-Objective?
[56]	361	2017	✓	✓	-	✗	✗
[98]	222	2018	✗	✗	-	✗	✗
[124]	184	2020	✗	✗	✗	✗	✗
[34]	163	2020	✗	✗	-	-	✗
[8]	156	2020	✗	✗	-	-	✗
[67]	122	2018	✓	✗	✗	✓	✗
[64]	95	2018	✗	✗	✗	✗	✗
[94]	79	2021	✗	✗	✗	✓	✗
[49]	65	2018	✓	✓	✗	✗	✗
[70]	38	2021	✓	✗	✗	✓	✗
[57]	36	2018	✗	✗	-	✓	✗
[27]	16	2024	✓	✗	✗	✗	✗
[10]	8	2022	✓	✓	-	✗	✗
[30]	2	2023	✗	✗	✗	✓	✗
[117]	1	2024	✓	✗	✓	✗	✗
[43]	96	2022	✓	✗	✓	✗	✗
[122]	37	2022	✓	✗	✓	✗	✗
		This Paper	✓	✓	✓	✓	✓



our review covers the primary intellectual currents relevant to warm starts and active learning, while managing the vast volume of available publications (this approach akin to reaching a form of *conceptual saturation* in qualitative data analysis [40]).

After reading the titles and abstracts of papers "above the knee" (i.e. 35 cites or more), snowballing[3] was used to find additional relevant work. In all, we found the 17 papers of Table 2. All the papers with less than 35 cites in that table come from the snowballing phase. Subsequent filtering[4] yielded the papers of Table 2. In those papers, we found the following patterns:

- **$\geq$ 5 test sets?**: This paper checks its conclusions on 49 data sets. Related work in Table 2 used less than half a dozen.
- **Multi-objective?**: Not only do we explore more data, we explore more complex problems. While 30 of our data sets have multiple goals, most prior work in this area explored tasks with only a single objective.
- **Warm Init?**: Very few papers in Table 2 used some background knowledge to generate warm starts (e.g. an LLM or results from a prior run).
- **Data = Tabular?**: Several other researchers also explore tabular data.
- **Labels < 50?**: Many papers in Table 2 gave their algorithms access to data from thousands of labeled examples (or even more). However, as discussed in the next section, we have reasons to constrain evaluations to just a few dozen.

When expressed as a Venn diagram in Table 2, this revealed a gap in the literature (see the center of that diagram). In summary, unlike this paper, most prior studies (a) did not handle tabular data; or (b) focused on single-objective tasks where (c) it was practical to find 10^3 labels per data set (and sometimes, much more). Note that (c) violates our assumption that we should label only a few dozen examples.

[3] *Snowballing:* starting with a set of core papers, look for papers commonly cited in their references.
[4] We accepted papers that mentioned (a) warm-start used for label initialization; or (b) testing on tabular data; or (c) mentioned the number of data sets used to evaluate the algorithm; or (d) mentioned the evaluation budget.

2.4 The Problem of Not Enough Data

Why does this paper assume that we should only evaluate a few dozen examples? This section argues that this an important assumption since there are many problem domains in SE where data is limited. Specifically, data can be in short supply for many reasons including *naive or faulty data* collection (see §2.4.1); *quirks* of data collection (see §2.4.2); or *slow data* collection (see §2.4.3).

2.4.1 Naive or Faulty SE Data Collection. The lesson of decades of SE analytics is that even when data seems readily available, much of it may be of dubious quality. For example, defect prediction researchers [22, 46, 52, 54, 80] often label a commit as "bug-fixing" when the commit text uses words like “bug, fix, wrong, error, fail, problem, patch”. Vasilescu et al. [111, 112] warns that this can be somewhat ad hoc, particularly if researchers just peek at a few results, then tinker with regular expressions to combine a few keywords. For another example, Yu et al. [123] explored labels from prior work exploring technical debt, over 90% of the "false positives" were incorrectly labeled.

More generally, there are many reports where errors in data labels have corrupted majority of the examples for security bug labeling [119]; for labeling false alarms in static code analysis [53]; or for software code.

When data collect is naive or faulty, some secondary process must clean up the data; e.g. select a small subset of the data that is not contaminated by quality concerns. If we could reduce the amount of data needed by our learners, then this would reduce how much effort must be allocated to this clean-up process. Hence, we choose to explore methods that only need a few dozen examples.

2.4.2 Quirks of SE Data Collection. Another reason why we may lack data on SE problems is a quirk in the nature of SE data collection. Specifically, in software engineering, the cost of collecting independent and dependent information can be very different. Consider tabular data where rows store examples of generated from some function y=$f(x)$:

- x columns hold data on independent input observables and/or controllables (e.g. lines of code)
- y columns hold data on the dependent values (e.g. bugs per line).

As stated above, we say that:

- *Unlabeled* data has x values, with no associated y values
- *Labeling* means using the x values to find associated y values;

A repeated effect in SE is that

- We can access copious amounts of unlabeled x data (e.g. from open source projects at Github);
- But it is much harder to collect high quality labeled y data.

For example, while it is easy to find software on Github. It is much harder to determine how much a team spent to create that software.

Table 3 lists other examples were y values are much more expensive to collect than x values. For SE tasks like those of Table 3, we need methods that can operate, despite a shortage in training data about the y values.

2.4.3 Slow Data Collection from SE Experts. Sometimes, data is missing, since it is fundamentally slow to collect. Requirements engineering researchers in software engineering report the rate at which subject matter experts (SMEs) can generate high quality labels; i.e. labels that would be acceptable to a panel of other experts. Numerous sources report that high quality labels can be collected from SMEs at the rate of 10-20 items/hour:

- Valerdi [108] worked with *a panel of experts* labeling 60 software project effort estimation examples, described using 20 attributes. He needed 3*three hour sessions, spread out over a week.

Table 3. For $y = f(x)$, it is often cheaper to collect x values than the associated y values.

Find X	Find associated Y values
It can be very quick to ...	It is a much slower task to...
Mine GitHub to find all the distributions of code size, number of dependencies per function, etc.	Discover (a)how much that software could be sold on the market or (b) what is the time required to build this kind of software
Count the number of classes in a system.	Negotiate with an organization permission to find how much human effort was required to build and maintain that code.
List design options; e.g. 20 binary choices is $2^{20} > 1,000,000$ options.	Check all those options with a group of human stakeholders.
List the configuration parameters for some piece of software.	Generate a separate executable for each one of those parameter settings, then run those executable through some test suite.
List the controls of a data miners used in software analytics (e.g. how many neighbors to use in a k-th nearest neighbor classifier).	Run a grid search looking for the best settings for some local data.
Generate test case inputs (e.g.) using some grammar-based fuzzing.	Run all all those tests. It can be even slower for a human to check through all those results looking for anomalous behavior.

- In iSBSE (*interactive search-based software engineering*) humans can serve as one of the oracles to guide the search. Research on the SNEAK iSBSE tool [68] reported that when SNEAK worked with humans, it collected human insight at the rates suggested above (15-20 labels per hour).
- *Repertory grids* researchers conduct interviews where humans justify attributes and attribute settings for random subsets of 3 examples, drawn from a larger set. Easterby-Smith [36] advises "keep the (repertory) grid small. A grid containing ten elements and ten constructs may take two hours to complete. Larger grids may take substantially more time". Kington [55] agrees, saying that it takes humans an hour to reflect over 16 examples with 16 attributes using repertory grids.
- In prior work on *knowledge acquisition for smart software* [73, 74], we found that SME experts may only be accessible for only a few hours per week. Humans quickly grow exhausted as they struggle to explain their expertise (so it is advisable to run short knowledge acquisition sessions [36, 55]).

Just to say the obvious, when data is collected at 10 to 20 items per hour (for only a few hours per week), then there will be many cases where new problems will lack training data for the task at hand.

2.5 Methods for Handling Data Shortages

The data shortage problem in SE has been widely studied. Various solutions have been proposed including label-less learning (see §2.5.1), metamorphic learning, semi-supervised learning (see §2.5.2), few-shot learning with LLMs (see §2.5.3), and active learning (see §2.6.1).

2.5.1 Label-less Learning. Zero-shot learning and unsupervised learning are two classes of algorithms that can execute, despite a lack of y labels. Zero-shot learners use the background knowledge of a LLM to make decisions without needing new labels [4]. Zero-shot learners works in domains where there exists an appropriate large language model, which is not always the case.

Unsupervised learners, on the other hand, use some domain heuristic to classify examples by peeking at the independent x variables. For example, Nam et al. successfully predicted for defective modules by looking for classes that are unusually large on multiple dimensions [88]. For another example, there may be some general condition under which an example can be unequivocally labeled as "fail" such as:

- A test case generates a core dump;
- A metamorphic predicate [28] reports a problem. For an example of such a predicate, consider "small changes to inputs should not cause large changes to output".

The problem here is that useful domain heuristics may not be available for all domains. For example, in the case studies shown below, we test for a very wide variety of user-supplied local goals such as "how to reduce energy requirements" and "if we implement this requirement next, then some other team should not stand idle waiting for some other function we were meant to implement". We are unaware of (say) metamorphic predicates that apply to such tasks.

2.5.2 Semi-Supervised Learning. Yet another approach finds labels for just a few examples, then propagates those values to other near-by examples. For example, Yehida et al. [121] recursively bi-clustered N examples down to leaf clusters of size $N^{0.25}$. All examples in a cluster are then given a label computed from the cluster centroid. .

Such semi-supervised learners have successfully reasoned over 10,000s of records, after labeling just 1 to 2.5% percent of the examples [5, 71]. While a useful approach, in the studies we have seen [5, 71], 1 to 2.5% of the data still means 100s to 1000s of labels. Given the numbers seen in §2.4.3 (10 to 20 items per hour, for few hours per week), we would hope to have methods that work using just a few dozen labels.

2.5.3 Few-shot Learning with LLMs. In turns out that LLMs need a few examples to activate the latent space of possible solutions within their networks. Hence, before asking a query, it is best to offer some "warm-up prompts" related to the task at hand. In this pre-query process, analysts offer to the LLM a few examples of desired inputs/outputs. Using this approach, a surprisingly small number of examples (sometimes as few as ten) can convert a general large language models into some specific tool; e.g. parsing test case output [59] or translating functions into English [2].

One problem with few-shot learning is how to find a few good examples. Tawosi et al. [106] uses genetic algorithms to select their few shot examples . But that approach has scalability issues (due to the overheads of the GA). When we applied it to the 100,000s of examples in our optimization data sets, it took hours complete the pre-query process. Without guidance on which example was most informative, these GAs were taking hours to process all these examples. Hence, for this work, we turned to methods that pruned away most of the examples before collecting labels. Specifically, we use active learning. In stark contrast to the Tawosi et al. [106] method, our approach can terminate in just a few sections (in particular, the "explore" and "exploit" TPE methods discussed below).

2.6 Successive Halving and DEHB

2.5a Successive halving is a method for prioritizing limited budgets in automatic configuration by focusing on regions of most potential benefit. The *Successive Halving* (SH) algorithm, introduced by Li et al. [60] in their *Hyperband* (HB) framework, is a prominent algorithm. SH starts with a large number of randomly chosen candidate configurations, each given a minimal budget, such as a few training epochs. After evaluating their performance, it discards the worst-performing half of the candidates and doubles the budget for the remaining ones. This process of elimination and resource doubling is repeated iteratively, ensuring that only the most promising configurations receive significant computational resources. SH is a form of *iterated racing* [66] because it continuously evaluates and compares candidate solutions, discarding under-performing ones. This systematic approach allows for an efficient exploration of the search space, saving considerable computational time and effort compared to evaluating every configuration to completion.

DEHB [9] extends Hyperband's successive halving with *Differential Evolution* (DE). In DE [103] new configurations are generated by mixing three others from an archive of previously successful solutions. A new, superior configuration replaces an old one, ensuring that the population improves over time. DEHB maintains separate subpopulations for each fidelity level, allowing the evolutionary process to run independently at each stage of the halving process. Information can flow from lower to higher fidelity subpopulations, further enhancing the search.

Unlike standard Hyperband, which uses random search [89] to initialize each bracket, DEHB reuses and evolves subpopulations from previous brackets in subsequent iterations. This eliminates the need for repeated random sampling, allowing for a more focused and efficient search. This paper uses DEHB as on its comparison algorithms since DEHB is known to outperform its predecessors, such as BOHB (which, in turn, is superior to Hyperopt [12]).

2.6.1 *Active Learning.*

Active learning is a technique that (a) reflects on a model built so far to (b) determine which examples to label next. A repeated result is that this approach learns good models, using very little data [20].

Active learner can begin via a *cold start* that label a few examples via a random or unsupervised process; e.g.

(1) *Knowledge-based sample*: Using some knowledge source, ask it to guess good initial values. This paper explores the use of LLMs for that task.
(2) *Random sampling*: This approach just selects items at random. While certainly the simplest and fastest method, the experiments of this paper show that other approaches give better results.
(3) *Similarity sampling*: Similarity sampling selects subsets of the examples where, within each subset, there is much similarity [63].
(4) *Diversity sampling*: Diversity sampling selects data points that are different from those already chosen or labeled [114]. Many diversity sampling methods are clustering based [43, 122] while other methods like [19] introduces diversity sampling methods using maximum distant points from the projections of the hyper planes.

There is some evidence that better results come from *warm starts* that use some background knowledge to select the initial labels (see [20, 68] and the experiments of this paper). Methods for generating warm starts include (a) reusing the results from a prior session;(b) use a diversity sampler (K-Means Clustering) to pick initial samples [43, 122] or (b) using the knowledge inside an LLMs are select the warm start examples. Once some examples have been labeled, then some *acquisition function* is applied to decide where to sample next.

Srinivas et al. in [101, 102] use Gaussian Process Models for their sequential model optimization problems. Research from other domains like materials discovery [62] and electrical engineering [35] claim GPMs are better active learners compared to other models like Random Forest learners. However, recent literature from hyperparameter optimization [115] claims that Gaussian Process Models (see §2.6.1) are less efficient than faster methods like Tree of Parzen Estimators (see §2.6.1) for active learning and multi-objective optimization. Usage of TPE as an active learner is vastly limited to the hyperparameter optimization space, and we claim that an extensive comparison of the performance between GPMs vs. TPE for diverse tasks is warranted.

GPM = Gaussian Process Models

GPM compute the mean μ and standard deviation σ of an estimate by fitting the data to a wide range of possible functions. These μ, σ values are then used by acquisition functions to decide where to sample next [20, 118].

Table 4 shows the connection of active learning, acquisition functions, and GPM. This paper explores three GPM-based acquisition functions: *UCB, PI* and *EI*.

Table 4. Active Learning with Gaussian Process Models

(1) Evaluate the labels of some rows (using a cold or warm start);
(2) Fit a large space of possible functions to the labeled data.
(3) From that function space, estimate the mean $mu(x)$ and standard deviation $\sigma(x)$ for different x values;
(4) Using those estimates, apply an acquisition function to select the row with the most promising x values to label.
(5) Evaluate a label at that x point;
(6) If evaluation budget exhausted, return row with best label. Else, go to step 2.

UCB = Upper Confidence Bound: Introduced in 1992 by [29], UCB is an adaptive acquisition function that recommends sampling the next example that maximizes;

$$UCB(x) = \mu(x) + \kappa\sigma(x) \tag{1}$$

- μ is the mean predicted at point x.
- $\sigma(x)$ is the predicted standard deviation at point x.
- κ is the parameter that balances the exploration and exploitation.

As active learning progresses, the variance in the predictions decreases so $\kappa\sigma(x)$ term grows smaller w.r.t. the $\mu(x)$ term. This means that UCB is *adaptive* to the amount of data collected; i.e. initially, UCB explores regions of high variance but, as data collection continues, it adapts to exploit regions of best predictions.

PI = Probability Improvement: Introduced first by [58], Probability Improvement (PI) is an acquisition function used to approximate the best configuration over the parameter space of a noisy distribution. PI calculates the probability of improvement the next incumbent sample brings in to the distribution.

$$PI(x) = \phi\left(\frac{\mu(x) - f(x^*) - \epsilon}{\sigma(x)}\right) \tag{2}$$

- μ is the predicted mean at point x.
- $f(x^*)$ is the current best point
- ϵ is the parameter balancing explore and exploit.
- $\sigma(x)$ is the predicted standard deviation at point x.
- ϕ is the PDF of the distribution.

EI = Expected Improvement (introduced in [51]): Expected improvement measures the expected gain from sampling a new point based on the priors.

$$EI(x) = \begin{cases} (\mu(x) - f(x^*) - \epsilon)\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \qquad Z = \begin{cases} \frac{\mu(x) - f(x^*) - \epsilon}{\sigma(x)} & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases}$$

- μ is the predicted mean at point x.
- $f(x^*)$ is the current best point
- ϵ is the parameter balancing explore and exploit.
- $\Phi(Z)$ is the Cumulative Distribution Function (CDF) of the standard normal distribution
- $\sigma(x)$ is the predicted standard deviation at point x.
- $\phi(Z)$ is the PDF of the standard normal distribution.

Expected improvement is a adaptive acquisition function that balances between exploration and exploitation, exploration is high when the surrogate variance is higher and exploitation is high when the surrogate mean is higher.

In this experiment, we use ‘Scikit-learn‘ open source python library to train the Gaussian Process Models for modelling the above acquisition functions.

TPE = Tree of Parzen Estimators

One alterative to using Gaussian process models are acquisition functions that use Tree of Parzen Estimators (TPE) [11, 90]. While Gaussian process-based approaches model $p(y|x)$ directly (i.e., $y = f(x)$), TPE [11] models $p(x|y)$ and $p(y)$ separately. To do this, TPE divides the sorted observations $D = \langle x, f(x)\rangle$ according to the label values into two sets using a threshold y^*. Thus, TPE defines $p(x|y)$ using two distributions:

Table 5. Active Learning with Tree of Parzen Estimators

(1) Evaluate the labels of some rows (using a cold or warm start);
(2) Sort the labeled rows into a small *best* set and a larger *rest* set.
(3) Using that binary division, build a classifier that reports the likelihood B, R of an example being *best* or *rest*.
(4) Using B, R apply an acquisition function to select the row with the most promising x value to label.
(5) Evaluate a row at that x point;
(6) If evaluation budget exhausted, return row with best label. Else, go to step 2.

$$p(x|y) = \begin{cases} best(x) & \text{if } y < y^*, \\ rest(x) & \text{if } y \geq y^*, \end{cases}$$

where $best(x)$ is the model formed by observations that performed well and $rest(x)$ is the model formed by observations that performed poorly. As shown in Figure 3, these two models let us explore several regions of interest. For example:

- Where $B - R \approx 0$, it is unclear if some new example is "best" or "rest". Some acquisition functions target this "zone of uncertainty" since it is here we can learn where an example flips from one class to another.
- Where B/R is large, our two models are in agreement that something is probably "best". Some acquisition functions target this "zone of certainty" since it is here we can be most certain that some is "best".
- Where $B - R \approx 0$ and $B + R$ is large. Here, our two models are strongly arguing for different conclusions. Some acquisition functions target this "zone of dispute" since it is here we can most learn what features lead to very different conclusions.

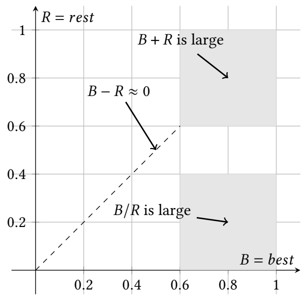


Fig. 3. Our TPE active learner explores the decision space of a two-class classifier.

Table 5, shows the connection of active learning and acquisition functions. This paper explores two TPE-based acquisition functions: *explore* and *exploit*.

Explore: Explore favors the zone of dispute; i.e. where two models are both loudly proclaiming similar strength, but opposite, prediction. When examples fall close to the hyperspace boundary that separates examples, that are most uncertain with respect to currently available data. Our formula for Explore is given by:

$$Explore = \frac{B + R}{abs(B - R) + \epsilon} \tag{3}$$

Here, ϵ is a very small number added to avoid divide-by-zero errors.

Exploit: Exploit targets the zone of certainty; i.e. where the best and rest models are in agreement that an example belongs to "best". The formula for Exploit is given by:

$$Exploit = \frac{B}{R} \tag{4}$$

In this study we use two versions of TPE

- 'Explore' and 'Exploit' acquisition function that uses a Bayes classifier to model the posterior distribution.
- 'Hyperopt ' (an open-source Python-based TPE library) [14].

Hyperopt Hyperopt is an open source python library that uses TPE to navigate through awkward search spaces consisting of real-valued, discrete and conditional dimensions. Hyperopt uses Gaussian Mixture Models (GMM) to model both B and R separately as informed in (Bergestra et.al) [13].

3 Methods

2.3a To compare the above methods, we ran *simulation studies* that emulate an engineer incrementally exploring some software device. Such studies are common practice in active learning research as well as configurations studies in the SE literature [42, 81, 84, 85]

- Specifically, we begin with datasets that have been fully and carefully labeled by prior work. To mimic the situation faced by engineers exploring a new domain, we then "hide" these labels from the inference procedure. Only learners are then given access only to the unlabeled information and must decide which subset of instances to query for labels.
- As the experiment proceeds, parts of the hidden ground truth labels are incrementally revealed, according to the learner's queries.
- The worth of different methods can then be assessed in a two-fold manner:
  (1) **Cost:** How many labels were used to build a model? What was the runtime?
  (2) **Benefit:** How effective was that model at improving the behavior of the device being studied?

This setup allows us to systematically measure how efficiently different methods select informative instances, while preserving full control over the costs (number of labels requested, runtime) and benefits (improvements in optimization outcomes). In this way, the simulation faithfully emulates the real-world process of engineers incrementally labeling data during early exploration of a new software engineering task.

The experiments of this paper use the data of §3.1 to compare the performance of LLM-aided warm start method for

- The state of the art GPM methods using the UCB, PI and EI acquisition functions.
- Against the TPE based acquisition functions Explore, Exploit;

This conclusions of this paper were achieved using the methods described in this section.

3.1 Data

This paper applies the acquisition functions described above to data shown in Table 1 from the MOOT repository (https://github.com/timm/moot). MOOT, short Multi Objective Optimization Testing is a collection of 49 optimization/configuration problems taken from the SE multi-objective optimization literature [25, 41, 68, 75, 77, 83, 87, 92]. For some details on the MOOT data sets, see Table 1.

MOOT stores tabular data. Each row contain many x columns (for input independent variables) and one or more y columns (for the output dependent variables). As seen in Table 1, most of MOOT's data has $|y| > 1$; i.e. they are multi-objective problems. To the best of our knowledge, our use of MOOT makes this paper one of the largest multi-objective SE optimization results reported in the literature. Having worked in this field since 2002 [38], we can assert that resources like MOOT are very rare. As shown in our literature review (§2.3), most research papers in this field certify their methods using fare less data than the 49 current entries within MOOT (usually papers use five data sets, or less).

MOOT's data is mostly generated by exercising some SE model (e.g. the "pom*" models discussed below) or logs of the the operation of project artifacts. As an example of the latter:

- All the “SS-*” data sets were generated by mutating the control parameters of a Makefile, then building some project, then running a test suite through that piece of software.

Note that MOOT stores data rather than models. This was a deliberate choice made for reproducibility reasons. Two different research teams working on (e.g.) “SS-A” will be exploring exactly the same set of values. The same is not true if each researcher codes up their own version of some model like (e.g.) DTLZ3 [31] then explores it using different random number generators.

For the purposes of experimentation, all the rows in MOOT’s data shows values for the y columns. But recalling §2.4.2, all our experiments will assume that accessing a row’s y values is a very expensive process. Hence, our algorithms will strive to find the best rows in these data sets after looking at as few y values as possible.

Prior work by Di Fiore et al. [32] has argued that active learning experimental results should be divided according to the number of independent variables seen in each optimization problem. Accordingly, Di Fiore et al., categorize datasets into 3 categories based on their input (x) dimensionality:

- *Low:* 12 datasets with $|x| < 6$ independent features.
- *Medium:* 14 datasets with $6 \leq |x| \leq 11$ independent features.
- *High:* 19 datasets with $|x| > 11$ independent features.

Other papers define “high dimensionality” in different ways to the above. For example, a common distinction is to refer to text mining as a lower dimensional problem than image processing. Yet this paper would call both “high dimensional”. But in defense of our definitions, we note that this paper (as well as Di Fiore et al.) find important differences in the behavior of acquisition functions over data sets divided according to the *low, medium, high* definitions listed above.

As to the application areas of MOOT, it is divided into four subcategories:

(1) Datasets under the *Config* directory included datasets with the group names "SS-*" that comes from the software engineering literature [83].This data was exhaustive collection of running configuration of different tasks including video encoding with the goals mostly being (Query time, Run times, etc). Config also includes other datasets with database configurations including Apache_AllMeasurements.csv, SQL_ALLMeasurements.csv, X264_AllMeasurements.csv.

(2) The *HPO* directory contains the datasets from the hyperparameter optimization literature [68] . These datasets show the results of random forest regression algorithms trained to predict a) commits, b) closed issues c) close pull requests in 12 months time on Open source projects hosted at Github. The Y values of these datasets shows the error and accuracy of the model for different hyperparameter configurations.

(3) The *Process* datasets comes from the software process modeling literature [41, 76, 77, 92]. The data set named "pom*" shows data from agile development by [18]. POM3 models requirements as a tree of dependencies that (metaphorically) rises out of pool of water. At any time, developers can only see the dependencies above the waterline. Hence they can be surprised by unexpected dependencies that emerge at a later time. POM3 reports the completion rates, idle times, and development effort seen when teams try navigate this space of changing tasks. The "nasdem" dataset contains real world data and "osp2" show data in the format of the USC Cocomo models that predict for development effort, defects, risk in waterfall- style software projects [76].

(4) The *Misc* directory contains some non SE datasets (auto93 and WineQuality). We use these to demonstration MOOT to a non-SE audience.

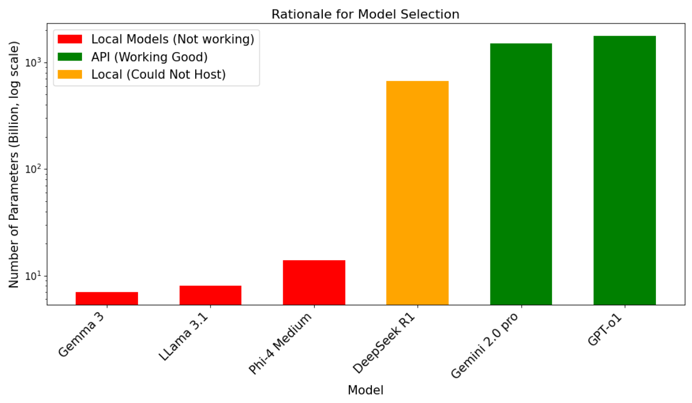


Fig. 4. Rationale for Model Selection: In the accompanying figure, the X-axis represents the model names, while the Y-axis shows the number of parameters (in billions) on a logarithmic scale. Parameter counts for open-source models are publicly available. For proprietary models like Gemini and GPT, the parameter sizes are not officially disclosed; the values presented are estimates gathered from various publicly available sources on the internet.

3.2 Models

Selecting the most suitable large language model (LLM) for a given task requires careful consideration of multiple factors. At the time of this writing, among the publicly available models (see Figure 4), the most prominent families included GPT, Gemini, and DeepSeek.

We use GPT here since our prior research [97] indicates that GPT outperforms its competitors on software engineering benchmarks.

DeepSeek R1 has demonstrated advanced reasoning capabilities across various benchmark datasets, we were unable to host it due to its large size (671B parameters). Next we collected quotes for services that offered to host DeepSeek and run our experiments. All our experiments require an LLM to reflect over all the rows of Table 1 (and, in our rig, this reflection is repeated 20 times to study the effects of different random seeds on the outcome). Hence, the quotes for DeepSeek hosting were prohibitively expensive.

We also tried several local models, including LLaMA 3.1 (8B), Gemma 3 (7B), and Phi-4 Medium (14B). These models consistently crashed during our experiments due to the structured output constraints enforced in our prompts.

Studies such as [47] suggest that Gemini models excel in precise factual reasoning, particularly in tasks requiring high contextual comprehension. For this reason, this study used Gemini 2.0 pro and GPT-o1.

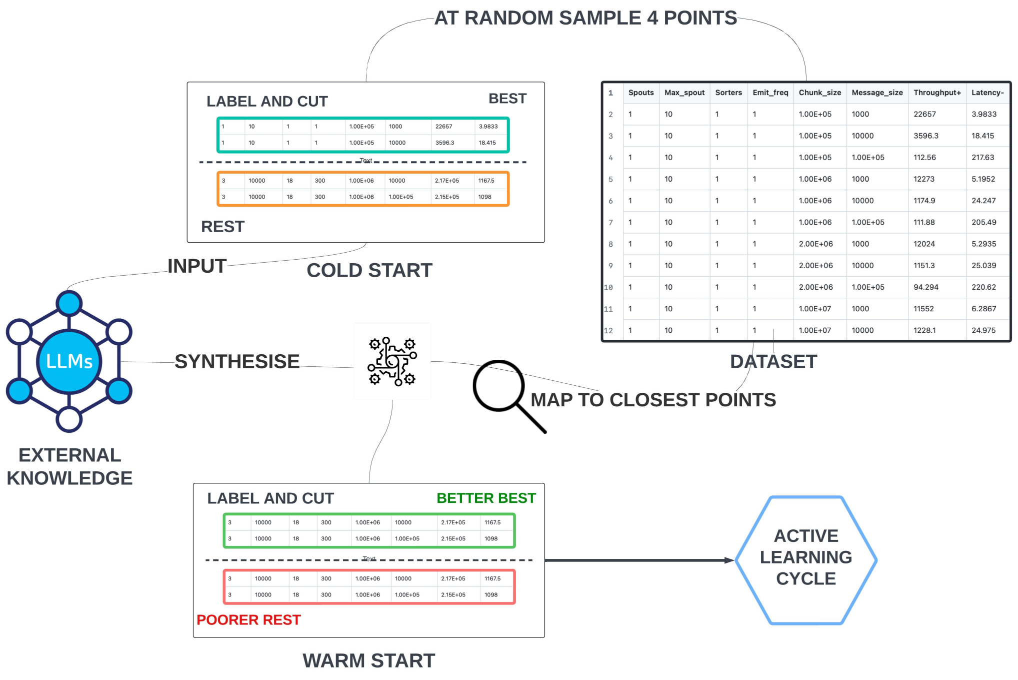
3.3 Algorithms

3.3.1 Active Learners. The active learners used in this study were described in Tables 4 and 5.

3.3.2 Acquisition Functions. All the acquisition functions used in this work were described in §2.6.1 and §2.6.1.

3.3.3 Warm Starts with LLMs. LLMs were used to generate warm starts as per the method illustrated in Figure 5:

(1) Initial labels were assigned to rows selected at random.
(2) These labeled rows, called E_0 were then sorted best to rest using the multi-objective evaluation criteria of §3.4.
(3) Few-shot learning [104] was then applied. The E_0 examples were given as prompts to the LLM, with comments saying "this is a good example", "this is a bad example".



1	Spouts	Max_spout	Sorters	Emit_freq	Chunk_size	Message_size	Throughput+	Latency-
2	1	10	1	1	1.00E+05	1000	22657	3.9833
3	1	10	1	1	1.00E+05	10000	3596.3	18.415
4	1	10	1	1	1.00E+05	1.00E+05	112.56	217.63
5	1	10	1	1	1.00E+06	1000	12273	5.1952
6	1	10	1	1	1.00E+06	10000	1174.9	24.247
7	1	10	1	1	1.00E+06	1.00E+05	111.88	205.49
8	1	10	1	1	2.00E+06	1000	12024	5.2935
9	1	10	1	1	2.00E+06	10000	1151.3	25.039
10	1	10	1	1	2.00E+06	1.00E+05	94.294	220.62
11	1	10	1	1	1.00E+07	1000	11552	6.2867
12	1	10	1	1	1.00E+07	10000	1228.1	24.975

1	10	1	1	1.00E+05	1000	22657	3.9833
1	10	1	1	1.00E+05	10000	3596.3	18.415

3	10000	18	300	1.00E+06	10000	2.17E+05	1167.5
3	10000	18	300	1.00E+06	1.00E+05	2.15E+05	1098

3	10000	18	300	1.00E+06	10000	2.17E+05	1167.5
3	10000	18	300	1.00E+06	1.00E+05	2.15E+05	1098

3	10000	18	300	1.00E+06	10000	2.17E+05	1167.5
3	10000	18	300	1.00E+06	1.00E+05	2.15E+05	1098

Fig. 5. Warm-starting with LLM Synthesized examples in cycle 0 of active learning

Table 6. Prompt template for synthetic data generation, here the two variables are meta which is a table containing the meta data of the dataset and table is set of rows that were randomly picked at the zero'th iteration.

System Message:

```
You are given a dataset with several features. The rows have been categorized into "Best" and "Rest" examples based on
their overall performance. Below are the key features and their descriptions from the dataset:
…
{rows_to_markdown(meta)}
```

Human Message:

```
Given Examples:

{rows_to_markdown(table)}
```

Task:

```
1. Generate Two New Examples that are Better:
These should outperform the given "Best" examples by optimizing the relevant features to better combinations.

2. Generate Two New Examples that are Poorer:
These should under perform the given "Rest" examples by modifying the relevant features to worse combinations.

Consider the inter-dependencies between features, and ensure that the generated examples follow logical consistency
within the dataset's context.
Return the output in the same markdown structure:
```

(4) The LLM was then asked to generate the x values of new examples, called E_1 that were, in the opinion of the LLM, better or worse that all the E_0 examples.
(5) All the items in E_1 are invented by the LLM. Hence, they lack y column labels. To find their labels, we go back to the training data and for each row $r \in E_1$, we found its nearest neighbor from E_0 (where "near" was measured using the Euclidean distance of the independent x column values). This formed the set $E_2 \subset E_0$.
(6) E_2 was then used to warm-start the active learners.

Table 6 offers more details of the prompting used in few-short learner used in step #3. All our prompts, are provide in markdown format for easier reading by the LLM (a technique recommended by [104]). Note that our prompt template was divided into three parts:

- *System message*: Define the role of our LLM, along with the meta data, where we provide information about the individual attributes of the dataset including, type, median value, standard deviation, high & low values if NUM or mode and frequencies if SYM. Note that in this section, we mention attribute names. This has the effect of "waking up" all the LLM's background knowledge of those terms.
- *Few Shot Examples*: Random samples from the data with the features along with their labels (Best / Rest).
- *Task*: in this case we ask the LLM to generate 2 better and 2 poorer samples while maintaining the same markdown format for better post-processing.

3.3.4 Diversity Sampling.

For experimentation purposes, we compare the knowledge-rich warm-start method (in the last section) with a more syntactic approach. *Diversity sampling* finds k very different points [96]. To find those points, we use the centroid finder of K-means++.

K-means++ is an initialization algorithm for K-means that selects initial centroids to be well-separated, significantly improving the clustering outcome [7]. Unlike random initialization, it aims to reduce the likelihood of poor local optima. The algorithm proceeds as follows:

- *Step 1: First Centroid Selection.* The first centroid C_1 is chosen randomly from the data points.
- *Step 2: Subsequent Centroid Selection.* For each subsequent centroid C_k, a data point x_i is selected with a probability proportional to its squared distance $D(x_i)^2$ to the closest already chosen centroid. This distance is defined as $D(x_i) = \min_{j<k} \|x_i - C_j\|^2$.
- *Step 3: Iteration.* Steps 1 and 2 are repeated until K initial centroids have been chosen.

This finds new centroids are likely to be in regions not already well-covered by existing centroids (leading to a better diversity-based initialization).

3.4 Evaluation Criteria

2.3b As per §3, our evaluation criteria should measure the **benefits** of our methods (i.e. what were the optimization results) as well as mentioning the associated **costs** (runtime in seconds, number of required labels). Our ideal method should:

- Achieve good benefits; i.e. find individuals which achieve most of the optimization goals;
- Without incurring a heavy cost; i.e. without long runtimes, or requiring a large number of labeled data items.

In the our results, we will separate what was achieved within the *low*, *medium*, and *high* dimensional data sets. For different labeling budgets we will run all our active learners. Our *results* section reports performance scores and runtimes for different labeling policies and algorithms.

To assess optimization results, the above methods need some way to recognize “best” and “rest” rows. Following the precedent set by the multi-objective research community [126], our rows are judged best or rest using the Chebyshev distance [39] between a row’s y-values and the theoretical ideal y-values:

- For dependent attributes that we wish to minimum or maximize, the *ideal* y-values contain the minimum or maximum vale (respectively) for that column.
- The Chebyshev distance is the maximum distance between any y_i value and its the ideal l_i value:

$$d_{\text{Chebyshev}}(y, o) = \max_{i=1,\ldots,n} |y_i - l_i| \tag{5}$$

- Note that for Chebyshev, *smaller* scores are *better* since this means that the row is closer to the ideal values. Hence, given a set of N labeled row, we sort them ascending via Chebyshev, then declare the first $\sqrt{N}$ rows to be “best” and the remaining $N - \sqrt{N}$ rows to be “rest”.

We use Chebyshev since:

- It is also used by other prominent multi-objective algorithms [39, 126];
- It is a “cruel critique” that punishes failure for any optimization goal. Optimizers that “win” as measured by Chebyshev are successful at simultaneously reducing all goals (at the same time).
- We have run all our results with other evaluation measures (e.g. average distance of row goals to the ideal). In those other studies, the general form of our results was not changed.

In summary, out evaluation measures are

- *Chebyshev*, which we aim to minimize since *smaller* values are *closer* to the ideal optimization;
- *Number of labels* we need to minimize;
- Runtimes (in seconds), which we aim to minimize;

(Aside: A recent IEEE TSE article by Chen et al. [61] reviewed various multi-objective optimization performance measures like Hypervolume (HV), Spread (Δ), Generational Distance (GD), and Inverted Generational Distance (IGD). We contacted those authors, asking their opinion of the appropriateness of those measures for this work. They commented [33] that their measures are inappropriate to the work of this paper due to our focus on minimal labeling. For instance, our approach would inherently result in low hypervolume since we generate very few solutions.)

3.5 Statistical Methods

We rank our results using the Scott-Knott method [95]. The Scott-Knott method is a recursive bi-clustering approach. It recursively divides the treatments (if they are statistically distinguishable from one other) to returns with a list of sets of treatments each ranked from [0..n]. Scott-Knott is recommended over other multiple comparison tests, since it does not produce overlapping groups like other post-hoc tests (e.g., Nemenyi’s test). Also, Scott-Knott tests for both effect size and significance (using the Cliff’s Delta and bootstrapping methods described below); it works well with overlapping distributions; N samples are ranked with just $log_2(N)$ statistical comparisons.

In this work, all treatments (acquisition methods, budget) are sorted in order of their median Chebyshev distances. Scott-Knott then divides a list l into two lists l_1, l_2 at the point where there is maximum expected difference between the mean before and after division.

$$E(\Delta) = \frac{|l_1|}{|l|}(E(l_1) - E(l))^2 + \frac{|l_2|}{|l|}(E(l_2) - E(l))^2 + \tag{6}$$

Cliff's Delta is a non-parametric effect size measure that quantifies the difference between two distributions by calculating the proportion of pairwise comparisons. It evaluates how often values from one distribution are larger or smaller than those from another. The result ranges from -1 to 1, where values closer to 0 indicate little difference, and values near -1 or 1 signify strong separation between groups. Cliff's Delta does not assume normality, making it ideal for non-parametric data. A threshold is used to classify the strength of the observed effect (small, medium, or large).

Bootstrapping is a resampling method. By repeatedly drawing samples with replacement from the original data, bootstrapping creates distributions of statistics (like means or medians) to approximate confidence intervals. This technique tests if the observed data is significantly different from random variation, making it useful for hypothesis testing and validating model performance, especially with limited data.If the results of these statistical tests confirms distinguishably of the groups then Scott-Knott ranks these groups and recursively repeats the process for both these clusters.

Table 7. Scott-Knott Rankings of the SS-A dataset along with treatments and budgets.

		Evaluation	Chebyshev		Visualization:	
Rank	Start	Acquire	Budget	Median	Std.	"o" = median
0	LLM	exploit	20	0.07	0.01	o-
0	LLM	exploit	15	0.07	0.02	o-
0	random	exploit	25	0.08	0.03	o ––
0	LLM	exploit	25	0.08	0.01	o-
0	random	exploit	20	0.08	0.03	- o–
0	random	exploit	15	0.09	0.03	- o –
1	random	PI_GPM	20	0.09	0.00	o
2	random	UCB_GPM	25	0.09	0.00	o
3	LLM	explore	20	0.10	0.02	- o-
4	LLM	explore	25	0.10	0.05	- o –––
4		random	20	0.10	0.02	o-
4	random	EI_GPM	15	0.10	0.00	o
5		random	25	0.11	0.02	– o-
5		random	15	0.12	0.03	– o
5	LLM	explore	15	0.12	0.05	- o –––-
5	random	PI_GPM	25	0.12	0.00	o
5	random	explore	15	0.13	0.04	- o––-
6	random	UCB_GPM	20	0.13	0.00	o
6	random	explore	20	0.13	0.04	- o–––-
6	random	UCB_GPM	15	0.13	0.00	o
6	random	EI_GPM	25	0.13	0.00	o
6	random	PI_GPM	15	0.13	0.00	o
6	random	explore	25	0.14	0.04	- o–––
7		baseline	1512	0.18	0.09	– o –
8	random	EI_GPM	20	0.36	0.00	o

Table 7 shows the output of the recursive bi-clustering ranking procedure of the Scott-Knott Method with treatments under the statistical rank 0 being the best of performing treatment for the dataset.

In Table 7, the statistical rankings found via Scott-Knott are colored alternatively gray or white. In that table, the initial results are the "baseline" set shown at the bottom. Our convention is to report the baseline budget as the number of rows in that data sets. All other rows have an evaluation budget set from *{20,25,30}*. Anything below the baseline is doing worse than the original data set.

In Table 7, for this data set, EI_GPM has failed to optimize this data set. The top ranked results are shown at the top of the table and labeled rank "0".

3.6 Experimental Rig

We ran all our active learners with B_0 evaluations for the warm starts and B_1 total evaluations where $B_0 = 4$ and $B_1 \in \{10, 15, 20, 25, 30\}$. This was repeated 20 times for statistical validity. Budgets up to 30 were chosen since:

- It fits the constraint of §2.4.3;
- There are many results in the recent active learning literature were the knee in the performance results occurs near 15 evaluations (e.g. see Figures 4,5,7 of [15]).
- In our active learning experiments, most of the improvement is witnesses with $B_1 \leq 30$. This "30 is enough" effect is seen in both (a) the visualization of Figure 6; and (b) a detailed statistical analysis. That statistical

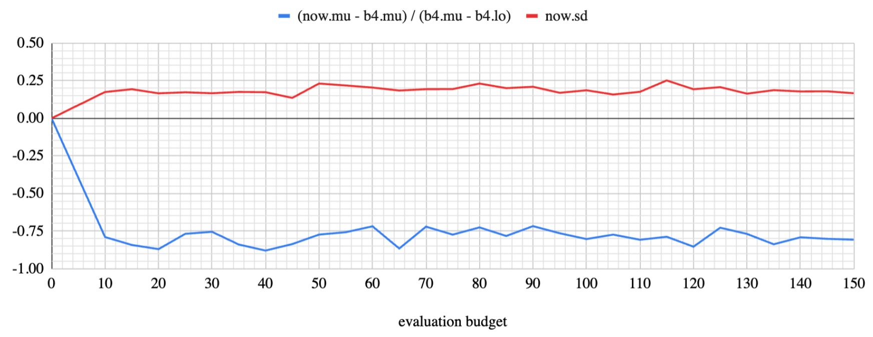


Fig. 6. Average optimizations seen across all data sets. Most improvement were after a few dozens samples The blue plot shows $I = \frac{now.mu - b4.mu}{b4.mu - b4.lo}$. These terms are explain in our text in §3.6. In summary, they are the improvement seen by optimization, normalized by the maximum possible improvement (and for this blue line in this plot, *lower* values are *better*).

analysis built one plot like Figure 6 for each data set. Then, for each plot, the statistical methods of §3.5 search for better optimizations occurred for $B_1 > 30$. None were found.

To summarize all 49 data sets, we generated one table like Table 7 for each data set. Next, we report the percentage of times a treatment appears at different ranks. For example, Table 9 shows that in 100% of our low dimensional data sets, LLM/Exploit achieves top rank.

R0.1b Our overall optimization improvements, across all treatments, are shown in Figure 6. That figure calculates the optimization improvement using some special terms:

- For any data set in Table 1, all the rows have a Chebyshev distance to heaven.
- Let *b4.mu* and *b4.lo* be the mean and smallest Chebyshev distances seen for each data set.
- Let *now.mu* and *now.sd* be the mean and standard deviation of the best Chebyshevs in the rows found in 20 repeats of our active learning experiments.
- To measure the optimization improvement, we normalize the observed change in Chebyshevs:

$$I = \frac{now.mu - b4.mu}{b4.mu - b4.lo}$$

Table 8. Scott-Knott rankings comparison for number of evaluations needed

Scott-Knott Rankings, for number of evaluations.

Start	Acquire	0	1	2	3	4	5	6
Random	UCB_GPM	21	21	21	0	20	0	20
Random	EI_GPM	20	22	23	20	23	0	20
Random	PI_GPM	21	22	20	25	0	0	0
Random	TPE	22	27	26	28	25	24	23
GPT	Exploit	21	20	24	20	20	20	20
Gemini	Exploit	21	25	20	20	20	20	20
Random	Exploit	21	24	26	23	20	30	23
Random	DEHB	91	100	100	23	100	48	
K-Means	Exploit	21	23	15	19	21	17	20
GPT	Explore	20	22	23	23	20	21	
Gemini	Explore 23	23	25	24	23	25		
	Random	20	21	23	25	23	27	25
Random	Explore	23	22	21	21	22	20	20

Here, *lower* values are better since the best results is when we only return the bext rows; i.e. $now.mu == b4.lo$, at which point $I = -1$.

As shown in Figure 6, after 30 labels, we can improve on the baseline by around 75% (measured by I as the normalized distance between the untreated means and the known optimal minimum rows).

The variances across treatments is $\approx \sigma = \pm 20\%$ (see the red line in Figure 6). Hence, all our subsequent results will take care to use the statistics of §3.5 in order to declare that (e.g.) one method is better than another.

Also, Table 8 shows, on average, how many evaluations were required for a treatment to achieve a particular rank (averaged across all data sets). As shown in the rank "0" column of that table, rank0 was achieved after 16 to 22 evaluations. In this table, smaller evaluations are better (fewer demands on data labeling).

$\boxed{\text{2.5b}}$ While this is not the focus of this paper, we note that standard optimization algorithms such as TPE and DEHB performed poorly:

- As seen in Table 8, DEHB needed more data than anything else (in terms of actual evaluations, DEHB always needed more that four time as many labels as anything else).

Table 9. Scott-Knott rankings. Results from 49 data sets and 20 repeats. Calculated using the Chebyshev equation of Equation 5. Blue cells show how frequently (in 49 data sets) did a treatment arrive at that cell. For example, the "100" top left, says that 100% of the time, "GPT Exploit" was ranked 1st (by our statistical measures) for the low dimensional data.

Low dimensional data where $|x| \leq 5$ (best = rank0)

Start	Acquire	0	1	2	3	4	5	6
GPT	Exploit	**92**				8		
Gemini	Exploit	**92**			8			
K-Means	Exploit	**75**	8		15			
Random	Exploit	**69**	23				8	
Random	UCB_GPM	**46**	15		31		8	
Random	EI_GPM	**38**	23	8	15			8
Random	PI_GPM	46	23	8	15		8	
Random	TPE	15	38		8		15	23
GPT	Explore	8	23	8	23	8		
Gemini	Explore	23	8	8	15	8		23
	random	15	8	23	23			23
Random	DEHB	8	8	15		23	8	15
Random	Explore	15		15	15		8	
	Baseline				8	8	8	15

Medium dimensional data where $6 \leq |x| \leq 10$ (best = rank0)

Start	Acquire	0	1	2	3	4	5	6
Random	EI_GPM	**64**	27				9	
Gemini	Exploit	**64**	9		18	9		
Random	UCB_GPM	**64**	27				9	
Random	PI_GPM	64	18	18				
GPT	Exploit	**64**		18		9	9	
Random	DEHB	64	18	9	9			
Random	Exploit	**55**	9		27		9	
K-Means	Exploit	**45**	18	27		9		
Random	TPE	36	18	18	18			9
Gemini	Explore	27	27	9	9	18		
GPT	Explore	18	27	9	27			
	Random	18	9	18	27	27		
Random	Explore	18	18	9	36	9	9	
	Baseline	9			9	9	9	27

High dimensional data where $|x| > 10$ (best = rank0)

Start	Acquire	0	1	2	3	4	5	6
Random	UCB_GPM	**100**						
Random	EI_GPM	**100**						
Random	PI_GPM	100						
Gemini	Exploit	36	13	7	13			
GPT	Exploit	36	**33**					
K-Means	Exploit	36	9	**27**				
Random	Exploit	36	14	29	7		7	7
Random	DEHB	29		36	21		7	7
Random	TPE	7	21	7	21	21	14	7
	Random		21	14	14	29	21	
Gemini	Explore		21	7	21	21	7	14
GPT	Explore		14	14	14	14	21	14
Random	Explore		7	14	14	36	7	7
	Baseline			7		21	14	

- As seen in Table 9, TPE always appear in the lower half of all results. As to DEHB, it has some value in medium dimensional data sets, but not in high or low dimensional data.

The rows with "Acquire=random" in the Table 9 show the results of just selecting *{10,15,20,25,30}* rows at random, sorting them by their Chebshev distance, then returning the best one. In no case in Table 9 did a purely random method score appear most often in the rank "0" column.

4 Results for Research Questions

4.1 RQ1: *Are warm-starts useful for active learning?*

A frequently asked question of this work is "do warm starts with only $B_0 = 4$ examples improve optimization results?". To answer that question, we can look at our results:

- In these results, "explore" is depreciated. In all cases, "explore" appears in the bottom half of all our results.
- For the remaining "exploit" results:
  - For low dimensionality data sets (the top left table of Table 9) we see that (a) LLM/Exploit (with GPT or Gemini) earns top rank most of the time (92%);
  - For medium dimensionality data sets (the top right table in Table 9) the actual warm start method seems not crucial. Note how in this table, many methods (random and otherwise) tie for top place at 64%.
- For high dimensionality data sets (the bottom row of Table 9), random warm starts perform as well as anything else.

Hence we say:

> **RQ1:** For the low dimensional SE tasks studied here, even if warm starts select a handful of initial examples, then those choices can still result in major improvements to optimization results. These benefits are not apparent in the medium and high dimensional data.

4.2 RQ2: *Are LLMs the best way to generate warm starts?*

For low-dimensional data (the top row of Table 9), LLMS for warm starts (using either Gemini or Chatgpt) is the clear winner.

For medium dimensional data sets, other methods do just as well, and for high-dimensional data, simple random section performs better than anything else. LLM/exploit (using either Gemini or GPT) failed for higher-dimensional problems, at which point randomly selected starts followed by UCB_GPM performed best.

We interpret these results in terms of "informational content". When little is known (not very many independent variables), the informational content of the domain data is very low. In this case, the background knowledge stored in LLMs is clearly and advantage.

However, the more we know about a domain (more independent variables), the informational content of the domain data is larger. In this case, the additional benefits of LLM labeling the warm starts is very low. In fact, for high-dimensional data (where there are many independent variables), LLM's background knowledge seems depreciated .

The success of LLM's warm starts for low dimensional data is remarkable, especially considering what we were asking LLMs to do. Recalling Table 5, our experiments ask LLMs to invent two better rows and two worse rows than an initial set of $B_0 = 4$ rows. Geometrically, this means that LLMs had to:

- Perform a multi-dimensional, almost PCA-like analysis[5] to find the dimensions that matter then most,
- Then perform an extrapolation along those dimensions.

To be sure, that approach failed for larger dimensional problems. However, the fact that this performed so well for low dimensional examples is a testament to the inherent power of these large languages.

Overall, we summarize these results as follows:

> **RQ2:** For the SE tasks studied here, LLM-based warm starts following by exploit-guided acquisition performs best for low dimensional problems. However, LLM effectiveness diminishes in high-dimensional problems, where Bayesian methods like Gaussian Process Models perform best

This result naturally begs the question, if high-dimensionality is the problem, is *dimensionality reduction* the answer? It is known that datasets with much higher dimensionality can be reduced to lower-dimensional spaces using various dimensionality reduction techniques. Techniques for dimensionality reduction include principal component analysis [91] (PCA), t-SNE [109], autoencoders, and others, each suited to specific types of data and objectives.

To understand dimensionality reduction, consider Figure 7. The large black arrow represent the directions of the first principal components, which indicate the axes of maximum variance in the data. An argument could be made that, instead of using two dimensions (x, y), this data might be better modeled using a reduced set of dimensions (specifically, just the large black arrow).

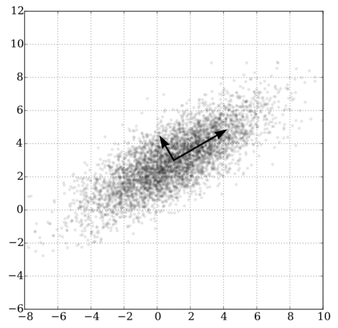


Fig. 7. Scatter plot of a 2D Gaussian-distributed dataset with overlaid principal component vectors. Source: Wikipedia.

To test is dimensionality reduction could help us here, we utilize autoencoders to reduce the dimensionality of our high-dimensional datasets from [32]. By transforming these datasets into a lower-dimensional representation, we aim to test whether this approach improves the performance of LLMs on these tasks. The reason for preferring autoencoders over other dimensionality reduction methods like PCA is that unlike PCA autoencoders can capture nonlinear relationships in the data, thereby proving to be more useful in capturing the underlying structure of the high-dimensional datasets. The structure of our autoencoder is shown in table 10

The results of that autoencoders experiment are shown in Table 12, Here, we used the trained encoder to transform our original high-dimensional dataset into a new, lower-dimensional representation. This reduced dataset is was then used for subsequent analysis and modeling tasks.

After running our experimental rig in Section 3.6 for the newly transformed datasets, we generate a summary table similar to 9. Comparing the results of the original high-dimensional datasets to the reduced-dimensional datasets in Table 11 and 12, we find that dimensionality reduction had little to no impact on attaining rank 0 across the datasets. This suggests that reducing dimensionality did not significantly improve performance in this context.

In future work, it could be insightful to try other dimensionality reduction approaches.

[5] Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms data into a set of uncorrelated variables called principal components, which capture the maximum variance. It simplifies complex datasets by projecting them onto fewer dimensions while preserving essential patterns.

Table 10. Autoencoder Training Process for Dimensionality Reduction

1. Initialize the autoencoder with the specified architecture:
   - Encoder: Input layer (original features) → Hidden layer (2 × encoding dimensions) → Output layer (intended encoding dimensions)
   - Decoder: Input layer (intended encoding dimensions) → Hidden layer (2 × encoding dimensions) → Output layer (original features)
2. Set the maximum number of training epochs to 200.
3. For each epoch:
   - Forward pass the input data through the autoencoder
   - Compute the reconstruction loss
   - Back-propagate the error and update the model parameters
4. Monitor the loss function throughout the training process.
5. If the loss reaches an optimal value or the maximum number of epochs is reached, terminate the training.
6. Use the trained encoder to transform the original high-dimensional dataset into a new, lower-dimensional representation.

Table 11. Frequency of ranks achieved for high-dimensional datasets.

		Scott-Knott Rankings						
Start	Acquire	0	1	2	3	4	5	6
random	UCB_GPM	**50**	22		17	6		
random	EI_GPM	**44**		28	11	6	6	
random	PI_GPM	**39**	28	11	11	6		
LLM	Exploit	**33**	6	22	6	6	6	6
random	Exploit	**17**	6	6		17	11	6
LLM	Explore	17	11	11	6	17	11	
	random	11	6	6	6	6	6	6
random	Explore		11	6			17	17
	Baseline			11	6		6	

Table 12. Frequency of ranks achieved for reduced-dimensional datasets.

		Scott-Knott Rankings (Low-Dimensional)						
Start	Acquire	0	1	2	3	4	5	6
random	UCB_GPM	**47**	18		14	5		
random	EI_GPM	**41**		24	9	5	6	
random	PI_GPM	**35**	25	10	10	5		
LLM	Exploit	**35**	5	20	5	5	5	5
random	Exploit	**15**	5	5		15	10	5
LLM	Explore	15	10	10	5	15	10	
	random	10	5	5	5	5	5	5
random	Explore		10	5			15	15
	Baseline			10	5		5	

4.2.1 *Running Out of Data?*

There is another reason our LLMs have trouble with high-dimensional data: they might not know enough to handle complex optimization problems.

Large language models train from data available "in the commons"; i.e. all the data generated by (say) programmers who store their code in Github. In the commons, there may exist many acceptable solutions for (e.g.) how to build a website in Python. Given a plethora of such solutions, LLMs can offer a useful response to a specific prompt.

However, there are problems that humans rarely address or, if they do, they rarely produce solution that a broad community would find acceptable. For such "uncommon" tasks, LLMs may lack sufficient training data. Many of the optimization tasks in MOOT are "uncommon". For example, our XOMO* data sets come from books discussing process options for software projects. These data sets list 24 parameters, usually discretized into five ranges (very low, low, nominal, high, very high). While many publications mention these choices, we know of none that conclude that one of these $5^{24} \approx 10^{16}$ choices is undeniably better than the rest. Also, several of our models refer to the configuration of cloud-based software systems. We would also call this an "uncommon" problem since there are some few publicly available examples of well-configured cloud environments[6].

We therefore offer the following conjecture. For low- and medium- dimensional SE problems LLMs can make find effective initial candidates, even for uncommon problems. The same is not true for higher-dimensional problems, an observation we attribute to the nature of uncommon problems.

[6]See Table 7 of Tang et al. [105] for a long list of cloud configuration errors. Also see industrial reports such as the 2024 Verizon business data breach report that states 80% of all security breeches on the cloud are configuration-related https://www.verizon.com/business/resources/reports/dbir/.

Looking into the future, we predict that, increasingly, LLMs will be challenged by "uncommon" problems. Recent results strongly suggest that LLMs may stop improving, very soon. In 2024, Villalobos et al. noted that new LLMs require exponentially increasing amounts of training data [113]. In Figure 8, they project that newer and larger models will soon exhaust the available textual data[7]. At that time, research work like this paper will be required to handle all the problems for which LLMs lack sufficient training data.

(Aside: Villlobios et al. warn that methods to extend this data (e.g. using one LLM to generate new data to train another) can actually degrade performance since automatically generated data may lacks the diversity needed for good inference [100].)

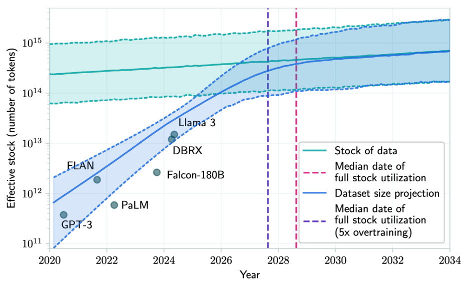


Fig. 8. Median results from the Villalobos et al. model (shown in green) estimate that by 2028, we will run out of new textual data needed to train bigger and better LLMs [113].

4.3 Threats to Validity

Despite the promising results, this study is subject to several threats to validity that must be carefully considered to ensure accurate interpretation of the findings.

Internal Validity. A primary concern is the potential for implementation errors in the active learning algorithms and acquisition functions. Although rigorous validation steps were taken, undetected bugs or implicit assumptions could skew results. Moreover, the selection of hyperparameters for Gaussian Process Models (GPM) and Tree of Parzen Estimators (TPE) may introduce subtle biases. To mitigate this, sensitivity analyses and replication by external researchers are essential.

External Validity. While the datasets span multiple SE tasks, they certainly do encapsulate the diversity of real-world SE problems. As a result, the findings may not generalize to tasks involving vastly different domains, such as next-release planning or SE reinforcement learning. Future work should expand the dataset to encompass a broader variety of SE optimization problems to improve generalizability.

Construct Validity. This study relies on Chebyshev distance to evaluate solution quality, which, while standard in multi-objective optimization, may overlook nuanced trade-offs inherent to SE tasks. Exploring alternative or composite metrics could reveal additional insights and better reflect the complexities of SE decision-making processes.

Conclusion Validity. The use of Scott-Knott clustering and Cliff's Delta provides a statistically grounded analysis. However, the small sample size of labeled data points introduces the risk of both type I and type II errors. Expanding the experimental budget and incorporating complementary statistical methods would enhance the robustness of the conclusions.

Learner Bias. For pragmatic reasons, described in §3.2, this study only used some of the currently available large language models. This study should be repeated, on other models.

[7]One limitation of the Villalobos et al. analysis is that it is only based on text tokens are there are other media that could be explored (e.g. visual). Nevertheless, their general point remains. Learning processes that require exponentially more data will soon exhaust the available training data.

4.4 Future Research

Building on the core research questions outlined in this study, several avenues for further exploration emerge that can enhance the understanding of LLM-based warm starts in software engineering (SE) active learning tasks.

Generalization and Transferability. As discussed in our literature review, most research papers in this arena test their methods on less than half a dozen data sets. Here, we have explored 49 so this work is less susceptible that most papers to criticisms of "lack of generality". That said, it would be wise to continue the testing of this method. Whenever new multi-objective SE data becomes available, we plan to apply the techniques of this paper to that data.

Dimensionality and Complexity. Given the observed decline in LLM effectiveness for high-dimensional problems, future research could explore whether dimensionality reduction techniques can mitigate this limitation. Some results in this vein were offered above but it could be insightful to explore other methods.

Long-Term Performance and Iteration. Future studies may also examine how LLM-generated warm starts evolve over multiple iterations of active learning. For example, can results from previous cycles further improve future warm starts?

Cost and Efficiency. Another critical line of inquiry is the computational cost of using LLMs for warm starts compared to Bayesian methods. Usually, the runtime costs of LLMs are the most prohibitively expensive item in an SE optimization study. Interestingly, here, that it not the case. Gaussian Process Models compute the mean and standard deviation of estimates by running the available data across a wide range of possible kernels. As shown in Figure 9, this process scan be slower even than LLM few shot learning.

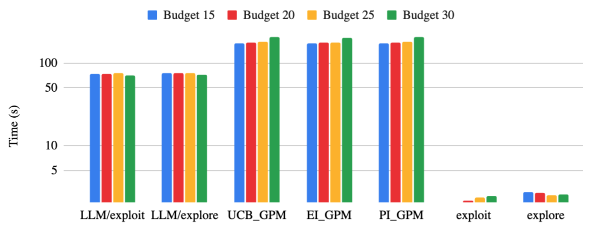


Fig. 9. A sample of runtimes from these experiments.

One other thing to note from Figure 9 is that the TPE methods (explore and exploit) run very much faster than GPM or LLM-based methods. Future research could explore if initial quick "peeks" at the data (with TPE) could inform and improve subsequent reasoning with LLMs or GPM.

Interpretability and Explainability. Given the low number of evaluations used in these studies ($B_1 \leq 30$), then explanations generated from active learning for LLMs could be very simple indeed. For example, perhaps there is some way to use active learning as a post-processor to LLMs to find a small number of easily explain examples. learning.

5 Conclusion

This study presents Ensemble Few-Shot Prompting, a novel approach that surpasses traditional Bayesian methods and random sampling across various SE optimization problems. By leveraging diverse combinations of initial examples, our method effectively utilizes the in-context learning capabilities of LLMs, leading to improved configurations with fewer evaluations. This not only enhances the efficiency of active learning but also demonstrates the potential of LLMs for advancing multi-objective optimization in software engineering. However, the ensemble-based approach requires additional effort in labeling the initial pool of examples.

Our contributions include:

- Introducing an ensemble of LLM few-shot learners to enhance active learning.

- Demonstrating the effectiveness of this approach across 49 SE datasets.
- Providing an empirical comparison of LLMs with alternative methods across 49 datasets.
- Delivering a reproduction package with data and scripts to facilitate open science.

Future work will focus on transferring knowledge from one few-shot learner to subsequent iterations, aiming to further optimize the learning process.

References

[1] A. Agrawal, W. Fu, D. Chen, X. Shen, and T. Menzies. 2019. How to "DODGE" Complex Software Analytics? *arXiv preprint arXiv:1902.01838* (2019).

[2] Toufique Ahmed and Premkumar Devanbu. 2023. Few-shot training LLMs for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (<conf-loc>, <city>Rochester</city>, <state>MI</state>, <country>USA</country>, </conf-loc>) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 177, 5 pages. https://doi.org/10.1145/3551349.3559555

[3] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *SIGMOD*.

[4] Waad Alhoshan, Liping Zhao, Alessio Ferrari, and Keletso J Letsholo. 2022. A zero-shot learning approach to classifying requirements: A preliminary study. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 52–59.

[5] Lauren Alvarez and Tim Menzies. 2023. Don't Lie to Me: Avoiding Malicious Explanations With STEALTH. *IEEE Software* 40, 3 (2023), 43–53. https://doi.org/10.1109/MS.2023.3244713

[6] Sven Apel, Norbert Siegmund, Christian Kästner, and Axel Legay. 2020. A Case for Automated Configuration of Variability-Intensive Systems. *IEEE Software* 37, 3 (2020), 26–33.

[7] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1027–1035.

[8] Jordan Ash and Ryan P Adams. 2020. On warm-starting neural network training. *Advances in neural information processing systems* 33 (2020), 3884–3894.

[9] Noor H. Awad, Neeratyoy Mallik, and Frank Hutter. 2021. DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization. *CoRR* abs/2105.09821 (2021). arXiv:2105.09821 https://arxiv.org/abs/2105.09821

[10] Yikun Ban, Yuheng Zhang, Hanghang Tong, Arindam Banerjee, and Jingrui He. 2022. Improved algorithms for neural active learning. *Advances in Neural Information Processing Systems* 35 (2022), 27497–27509.

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24 (2011).

[12] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. 2015. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery* 8, 1 (2015), 014008.

[13] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*. PMLR, 115–123.

[14] James Bergstra, Dan Yamins, David D Cox, et al. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, Vol. 13. Citeseer, 20.

[15] Muhammad Bilal, Marco Serafini, Marco Canini, and Rodrigo Rodrigues. 2020. Do the best cloud configurations grow on trees? an experimental evaluation of black box algorithms for optimizing cloud workloads. (2020).

[16] Barry Boehm, Chris Abts, A Winsor Brown, Sunita Chulani, Bradford K Clark, Ellis Horowitz, Ray Madachy, Donald J Reifer, and Bert Steece. 2000. Cost estimation with COCOMO II. *ed: Upper Saddle River, NJ: Prentice-Hall* (2000).

[17] Barry W. Boehm. 2002. Software Pioneers. Springer-Verlag, Berlin, Heidelberg, Chapter Software Engineering Economics, 641–686. http://dl.acm.org/citation.cfm?id=944331.944370

[18] Barry W Boehm and Richard Turner. 2004. *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional.

[19] Klaus Brinker. 2003. Incorporating diversity in active learning with support vector machines. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 59–66.

[20] Eric Brochu, Vlad M Cora, and Nando De Freitas. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* (2010).

[21] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[22] Gemma Catolino. 2017. Just-in-time bug prediction in mobile applications: the domain matters!. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 201–202.

[23] Chaomei Chen. 2006. CiteSpace: A new tool for discovering knowledge frontiers. In *Lecture Notes in Computer Science (Lecture Notes in Computer Science, Vol. 3913)*. Springer, 237–248.

[24] Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. 2016. Is Sampling Better Than Evolution for Search-Based Software Engineering? *arXiv preprint* (2016).

[25] Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. 2018. "Sampling" as a baseline optimizer for search-based software engineering. *IEEE Transactions on Software Engineering* 45, 6 (2018), 597–614.

[26] Jianfeng Chen, Vivek Nair, and Tim Menzies. 2017. Beyond Evolutionary Algorithms for Search-based Software Engineering. *Information and Software Technology* 2017 (2017).

[27] Liangyu Chen, Yutong Bai, Siyu Huang, Yongyi Lu, Bihan Wen, Alan Yuille, and Zongwei Zhou. 2024. Making your first choice: to address cold start problem in medical active learning. In *Medical Imaging with Deep Learning*. PMLR, 496–525.

[28] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. 2018. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–27.

[29] Dennis D Cox and Susan John. 1992. A statistical method for global optimization. In *[Proceedings] 1992 IEEE international conference on systems, man, and cybernetics*. IEEE, 1241–1246.

[30] Arnav Mohanty Das, Gantavya Bhatt, Megh Manoj Bhalerao, Vianne R Gao, Rui Yang, and Jeff Bilmes. 2023. Continual Active Learning. (2023).

[31] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. 2002. Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 1. 825–830 vol.1. https://doi.org/10.1109/CEC.2002.1007032

[32] F. Di Fiore, M. Nardelli, and L. Mainini. 2024. Active Learning and Bayesian Optimization: A Unified Perspective to Learn with a Goal. *Archives of Computational Methods in Engineering* (2024), 1–29.

[33] Jane Doe. 2023. Personal Communication with Author, at ICSE'25, Ottawa, Canda. Email to Your Name on May 15, 2023. Regarding the results of the K-means++ experiment..

[34] Manqing Dong, Feng Yuan, Lina Yao, Xiwei Xu, and Liming Zhu. 2020. Mamo: Memory-augmented meta-optimization for cold-start recommendation. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 688–697.

[35] Joseph Duris, Dylan Kennedy, Adi Hanuka, Jane Shtalenkova, Auralee Edelen, P Baxevanis, Adam Egger, T Cope, M McIntire, S Ermon, et al. 2020. Bayesian optimization of a free-electron laser. *Physical review letters* 124, 12 (2020), 124801.

[36] Mark Easterby-Smith. 1980. The design, analysis and interpretation of repertory grids. *International Journal of Man-Machine Studies* 13, 1 (1980), 3–24. https://doi.org/10.1016/S0020-7373(80)80032-0

[37] Yuanyuan Zhou et al. 2011. Understanding and Detecting Software Configuration Errors. *USENIX OSDI* (2011).

[38] Martin S. Feather and Tim Menzies. 2002. Converging on the Optimal Attainment of Requirements. In *10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE 2002), 9-13 September 2002, Essen, Germany*. IEEE Computer Society, 263–272. https://doi.org/10.1109/ICRE.2002.1048537

[39] Ioannis Giagkiozis and Peter J Fleming. 2015. Methods for multi-objective optimization: An analysis. *Information Sciences* 293 (2015), 338–350.

[40] Barney G Glaser and Anselm L Strauss. 1967. *The discovery of grounded theory: Strategies for qualitative research.* Aldine Transaction.

[41] Phillip Green, Tim Menzies, Steven Williams, and Oussama El-Rawas. 2009. Understanding the value of software engineering technologies. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 52–61.

[42] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wąsowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 301–311.

[43] Guy Hacohen, Avihu Dekel, and Daphna Weinshall. 2022. Active learning on a budget: Opposite strategies suit high and low budgets. *arXiv preprint arXiv:2202.02794* (2022).

[44] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 11.

[45] Herodotos Herodotou and Shivnath Babu. 2011. Starfish: A Self-tuning System for Big Data Analytics. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11), Asilomar, CA, USA, January 9-12, 2011*. 261–272. http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper30.pdf

[46] Abram Hindle, Daniel M German, and Ric Holt. 2008. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 99–108.

[47] Hartwig H Hochmair, Levente Juhász, and Takoda Kemp. 2024. Correctness Comparison of ChatGPT-4, Gemini, Claude-3, and Copilot for Spatial Tasks. *Transactions in GIS* (2024).

[48] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. https://doi.org/10.1145/3695988

[49] Peiyun Hu, Zachary C Lipton, Anima Anandkumar, and Deva Ramanan. 2018. Active learning with partial feedback. *arXiv preprint arXiv:1802.07427* (2018).

[50] Pooyan Jamshidi and Giuliano Casale. 2016. Uncertainty-aware self-adaptation in cloud computing. Ph.D. dissertation proposal. Cited by other works for the Storm configuration performance example..

[51] Donald R Jones, Matthias Schonlau, and William J Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization* 13 (1998), 455–492.

[52] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2012. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering* 39, 6 (2012), 757–773.

[53] Hong Jin Kang, Khai Loong Aw, and David Lo. 2022. Detecting False Alarms from Automatic Static Analysis Tools: How Far Are We?. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 698–709. https://doi.org/10.1145/3510003.3510214

[54] Sunghun Kim, E James Whitehead Jr, and Yi Zhang. 2008. Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering* 34, 2 (2008), 181–196.

[55] Alison Kington. 2009. Defining Teachers' Classroom Relationships. (2009).

[56] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. 2017. Learning active learning from data. *Advances in neural information processing systems* 30 (2017).

[57] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. 2018. Discovering general-purpose active learning strategies. *arXiv preprint arXiv:1810.04114* (2018).

[58] Harold J Kushner. 1964. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. (1964).

[59] Van-Hoang Le and Hongyu Zhang. 2023. Log parsing with prompt-based few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2438–2449.

[60] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research* 18, 185 (2018), 1–52.

[61] Miqing Li, Tao Chen, and Xin Yao. 2022. How to Evaluate Solutions in Pareto-Based Search-Based Software Engineering: A Critical Review and Methodological Guidance. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1771–1799. https://doi.org/10.1109/TSE.2020.3036108

[62] Qiaohao Liang, Aldair E Gongora, Zekun Ren, Armi Tiihonen, Zhe Liu, Shijing Sun, James R Deneault, Daniil Bash, Flore Mekki-Berrada, Saif A Khan, et al. 2021. Benchmarking the performance of Bayesian optimization across multiple experimental materials science domains. *npj Computational Materials* 7, 1 (2021), 188.

[63] Yan-Hui Lin, Ze-Qi Ding, and Yan-Fu Li. 2023. Similarity based remaining useful life prediction based on Gaussian Process with active learning. *Reliability Engineering & System Safety* 238 (2023), 109461.

[64] Ming Liu, Wray Buntine, and Gholamreza Haffari. 2018. Learning how to actively learn: A deep imitation learning approach. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1874–1883.

[65] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. 2024. Large language models to enhance bayesian optimization. *arXiv preprint arXiv:2402.03921* (2024).

[66] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58.

[67] David Lowell, Zachary C Lipton, and Byron C Wallace. 2018. Practical obstacles to deploying active learning. *arXiv preprint arXiv:1807.04801* (2018).

[68] Andre Lustosa and Tim Menzies. 2024. Learning from Very Little Data: On the Value of Landscape Analysis for Predicting Software Project Health. *ACM Transactions on Software Engineering and Methodology* 33, 3 (2024), 1–22.

[69] R. Madachy. 1997. Heuristic Risk Assessment Using Cost Factors. *IEEE Software* 14, 3 (May 1997), 51–59.

[70] Rafid Mahmood, Sanja Fidler, and Marc T Law. 2021. Low budget active learning via wasserstein distance: An integer programming approach. *arXiv preprint arXiv:2106.02968* (2021).

[71] Suvodeep Majumder, Joymallya Chakraborty, and Tim Menzies. 2024. When less is more: on the value of "co-training" for semi-supervised software defect predictors. *Empirical Software Engineering* 29, 2 (2024), 1–33.

[72] George Mathew, Tim Menzies, Neil A Ernst, and John Klein. 2017. "SHORT" er Reasoning About Larger Requirements Models. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 154–163.

[73] Tim Menzies. 1999. Critical success metrics: evaluation at the business level. *International journal of human-computer studies* 51, 4 (1999), 783–799.

[74] Tim Menzies, John Black, Joel Fleming, and Murray Dean. 1992. An expert system for raising pigs. In *The first Conference on Practical Applications of Prolog*.

[75] Tim Menzies, Alex Dekhtyar, Justin Distefano, and Jeremy Greenwald. 2007. Problems with Precision. *IEEE Transactions on Software Engineering* (September 2007). http://menzies.us/pdf/07precision.pdf.

[76] Tim Menzies, Oussama Elrawas, Jairus Hihn, Martin Feather, Ray Madachy, and Barry Boehm. 2007. The business case for automated software engineering. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 303–312.

[77] Tim Menzies, Steve Williams, Oussama El-Rawas, Barry Boehm, and Jairus Hihn. 2009. How to avoid drastic software process change (using stochastic stability). In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 540–550.

[78] Tim Menzies, Steve Williams, Oussama El-Rawas, Barry Boehm, and Jairus Hihn. 2009. How to Avoid Drastic Software Process Change (Using Stochastic Stability). In *ICSE*.

[79] Wiem Mkaouer, Marouane Kessentini, Adnan Shaout, Patrice Koligheu, Slim Bechikh, Kalyanmoy Deb, and Ali Ouni. 2015. Many-objective software remodularization using NSGA-III. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 3 (2015), 1–45.

[80] Audris Mockus and Lawrence G Votta. 2000. Identifying Reasons for Software Changes using Historic Databases.. In *icsm*. 120–130.

[81] V. Nair, A. Agrawal, J. Chen, W. Fu, G. Mathew, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu. 2018. Data-Driven Search-based Software Engineering. In *MSR*.

[82] Vivek Nair and Tim Menzies et al. 2018. Finding Faster Configurations Using FLASH. *IEEE TSE* 46, 7 (2018).

[83] Vivek Nair, Tim Menzies, and Jianfeng Chen. 2016. An (accidental) exploration of alternatives to evolutionary algorithms for sbse. In *International Symposium on Search Based Software Engineering*. Springer, 96–111.

[84] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Faster discovery of faster system configurations with spectral learning. *Automated Software Engineering* 99 (2017), 1–31.

[85] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using bad learners to find good configurations. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 257–267.

[86] Vivek Nair, Zhe Yu, and Tim Menzies. 2017. FLASH: A Faster Optimizer for SBSE Tasks. *arXiv preprint arXiv:1705.05018, under review* (2017).

[87] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding faster configurations using flash. *arXiv preprint arXiv:1801.02175* (2018).

[88] Jaechang Nam and Sunghun Kim. 2015. CLAMI: Defect Prediction on Unlabeled Datasets. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*.

[89] Meetesh Nevendra and Pradeep Singh. 2022. Empirical investigation of hyperparameter optimization for software defect count prediction. *Expert Systems with Applications* 191 (2022), 116217.

[90] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. 2020. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 genetic and evolutionary computation conference*. 533–541.

[91] K. Pearson. 1901. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.

[92] Dan Port, Alexy Olkov, and Tim Menzies. 2008. Using simulation to investigate requirements prioritization strategies. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 268–277.

[93] Alan Pritchard. 1969. Statistical bibliography or bibliometrics? *Journal of Documentation* 25, 4 (1969), 340–346.

[94] Christopher Schröder, Andreas Niekler, and Martin Potthast. 2021. Revisiting uncertainty-based query strategies for active learning with transformers. *arXiv preprint arXiv:2107.05687* (2021).

[95] Andrew Jhon Scott and Martin Knott. 1974. A cluster analysis method for grouping means in the analysis of variance. *Biometrics* (1974), 507–512.

[96] B. Settles. 2012. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* (2012).

[97] Md Kamrul Siam, Huanying Gu, and Jerry Q Cheng. 2024. Programming with AI: Evaluating ChatGPT, Gemini, AlphaCode, and GitHub Copilot for Programmers. *arXiv preprint arXiv:2411.09224* (2024).

[98] Aditya Siddhant and Zachary C Lipton. 2018. Deep bayesian active learning for natural language processing: Results of a large-scale empirical study. *arXiv preprint arXiv:1808.05697* (2018).

[99] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-Inference for Highly Configurable Software Systems. In *ESEC/FSE*.

[100] Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. 2023. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585* (2023).

[101] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* (2009).

[102] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias W Seeger. 2012. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE transactions on information theory* 58, 5 (2012), 3250–3265.

[103] Rainer Storn and Kenneth Price. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341–359.

[104] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 645–654.

[105] Lilia Tang, Chaitanya Bhandari, Yongle Zhang, Anna Karanika, Shuyang Ji, Indranil Gupta, and Tianyin Xu. 2023. Fail through the Cracks: Cross-System Interaction Failures in Modern Cloud Systems. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 433–451. https://doi.org/10.1145/3552326.3587448

[106] Vali Tawosi, Salwa Alamir, and Xiaomo Liu. 2023. Search-Based Optimisation of LLM Learning Shots for Story Point Estimation. In *International Symposium on Search Based Software Engineering*. Springer, 123–129.

[107] Vali Tawosi, Rebecca Moussa, and Federica Sarro. 2023. Agile Effort Estimation: Have We Solved the Problem Yet? Insights From a Replication Study. *IEEE TSE* 49, 4 (2023), 2677–2697.

[108] Ricardo Valerdi. 2010. Heuristics for systems engineering cost estimation. *IEEE Systems Journal* 5, 1 (2010), 91–98.

[109] L.J.P. van der Maaten and G.E. Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.

[110] Nees Jan van Eck and Ludo Waltman. 2010. Software survey: VOSviewer, a computer program for bibliometric mapping. *Scientometrics* 84, 2 (2010), 523–538.

[111] B Vasilescu. 2018. Personnel communication at fse'18. *Found. Softw. Eng* (2018).

[112] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 805–816.

[113] Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. [n. d.]. Position: Will we run out of data? Limits of LLM scaling based on human-generated data. In *Forty-first International Conference on Machine Learning*.

[114] Rui Wang, Wubin Ma, Mao Tan, Guohua Wu, Ling Wang, Dunwei Gong, and Jian Xiong. 2021. Preference-inspired coevolutionary algorithm with active diversity strategy for multi-objective multi-modal optimization. *Information Sciences* 546 (2021), 1148–1165.

[115] Shuhei Watanabe, Noor Awad, Masaki Onishi, and Frank Hutter. 2022. Speeding up multi-objective hyperparameter optimization by task similarity-based meta-learning for the tree-structured parzen estimator. *arXiv preprint arXiv:2212.06751* (2022).

[116] Cody Watson, Nathan Cooper, David Nader Palacio, Kevin Moran, and Denys Poshyvanyk. 2022. A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 32 (March 2022), 58 pages. https://doi.org/10.1145/3485275

[117] Jiarong Wei, Yancong Lin, and Holger Caesar. 2024. BaSAL: Size-Balanced Warm Start Active Learning for LiDAR Semantic Segmentation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 18258–18264.

[118] Christopher Williams and Carl Rasmussen. 1995. Gaussian processes for regression. *Advances in neural information processing systems* 8 (1995).

[119] Xiaoxue Wu, Wei Zheng, Xin Xia, and David Lo. 2022. Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction. *IEEE Transactions on Software Engineering* 48, 7 (2022), 2541–2556. https://doi.org/10.1109/TSE.2021.3063727

[120] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, You Have Given Me Too Many Knobs!: Understanding and Dealing with Over-designed Configuration in System Software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. ACM, New York, NY, USA, 307–319. https://doi.org/10.1145/2786805.2786852

[121] Rahul Yedida, Hong Jin Kang, Huy Tu, Xueqi Yang, David Lo, and Tim Menzies. 2023. How to find actionable static analysis warnings: A case study with FindBugs. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2856–2872.

[122] Ofer Yehuda, Avihu Dekel, Guy Hacohen, and Daphna Weinshall. 2022. Active learning through a covering lens. *Advances in Neural Information Processing Systems* 35 (2022), 22354–22367.

[123] Zhe Yu, Fahmid Morshed Fahid, Huy Tu, and Tim Menzies. 2022. Identifying self-admitted technical debts with jitterbug: A two-step approach. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1676–1691.

[124] Michelle Yuan, Hsuan-Tien Lin, and Jordan Boyd-Graber. 2020. Cold-start active learning through self-supervised language modeling. *arXiv preprint arXiv:2010.09535* (2020).

[125] Guofu Zhang, Zhaopin Su, Miqing Li, Feng Yue, Jianguo Jiang, and Xin Yao. 2017. Constraint handling in NSGA-II for solving optimal testing resource allocation problems. *IEEE Transactions on Reliability* 66, 4 (2017), 1193–1212.

[126] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation* 11, 6 (2007), 712–731.

Response to Reviewers for TOSEM-2024-0881.R1

We thank the editors and reviewers for their thoughtful, constructive, and detailed feedback. We have carefully revised the paper in response to all comments. Below, we outline the major changes made, organized by the source of feedback. For each comment, we restate the original point and describe how we addressed it.

Senior Associate Editor . Thank you for submitting your work to TOSEM. After careful evaluation by three reviewers, we agree that the manuscript has been significantly improved. Reviewer 3 is now fully satisfied with the paper and recommends acceptance.

Reviewers 1 and 2 have raised several points that require attention .

Reframing or Removing RQ1: As noted by Reviewer 1, RQ1 (“Is active learning useful for SE tasks?”) currently addresses a well-established conclusion in the literature, which may give an impression of redundancy. Please address this by choosing one of the following paths:

Remove RQ1: Incorporate its conclusion into the background section, citing relevant literature to efficiently establish that active learning is effective for the SE tasks you study. Or **reformulate RQ1** Revise RQ1 to be more specific and innovative, distinguishing it from prior work.

Thank you for this suggestion. We have adopted the first path (Remove RQ1 and work that material into the text see **R0.1a** on page 2 and **R0.1b** on page 21.

Aligning Evaluation Criteria with Research Goals: Reviewer 2 points out a critical disconnect between your stated goal (investigating how LLMs can help active learning for multi-objective optimization) and your current evaluation criteria (finding specific rows in a dataset). The evaluation must directly measure the benefits for the final multi-objective optimization process. Please update your experimental evaluation to include metrics from one or more real multi-objective optimizers.

Thank you for this suggestion. We now include results from two real multi-objective optimizers (TPE and DEHB, see **2.5a** on page 10; DEBH is an algorithm from IJCAI’23). As discussed in **2.5b** on page 22, both these methods perform very badly (e.g. TPE always appears in the bottom half of our results,; DEHB needs over four times as much data as other methods; DEHB’s optimizations are often worse and never better than other methods (see **2.5b** on page 22). So while it was a good methodological point to add these results to this paper, they have no effect on our conclusions.

As to evaluation criteria, this is embarrassing to report, there was a critical paragraph missing in the last drafts about the nature of the simulation studies used in this paper. Without that paragraph, the nature of our evaluation criteria would be very puzzling indeed.

This draft corrects that oversight, see **2.3a** on page 14 and **2.3b** on page 18. With these corrections, we assert that our evaluation methods is now comprehensible.

Strengthening the Baseline Comparison: The choice of TPE as a baseline was not found to be convincing. You must either:

Provide a clear and compelling argument for why TPE, an estimation technique, is an appropriate baseline for your specific task. (Recommended) Select and compare against more conventional baselines, such as typical techniques for generating "warm starts" for optimization problems.

Thank you for this remark. As we report apart, we have now compare against more conventional baselines (DEHB, an algorithm from IJCAI’23; see **2.5a** on page 10 and **2.5b** on page 22).

Improving Clarity of Empirical Results: The presentation of your empirical results is currently difficult for readers to understand. Please elaborate significantly on the experimental details and provide clear, in-depth explanations for the results. Your goal should be to guide the reader through your findings and their implications.

Please see our remark above: prior drafts missed a critical paragraph. This draft corrects that oversight, see 2.3a on page 14 and 2.3b on page 18. With these corrections, we assert that our evaluation methods is now comprehensible.

Minor Revisions These are minor but necessary corrections to improve the paper's quality.

On page 6, line 306: "see Table 2" should be "see Figure 2".

On page 15, line 751: "see Figure 5" should be "see Figure 4".

Fixed. Thank you for the pointers.

*Referee: 1

Recommendation: Needs Minor Revision

Comments: Thank you for the authors' detailed response. The authors state in their response that the purpose of proposing RQ1 ("Is active learning useful for SE tasks?") is to "before anything else, it is good practice to demonstrate some baseline competency in this domain." This rationale is understandable, as it reflects their intent to use RQ1 to establish a foundation for subsequent research and demonstrate their capability in applying active learning to software engineering (SE) tasks. However, the conclusion of this RQ has already been well-supported in prior literature, and thus presenting RQ1 as a standalone research question may lead readers to perceive it as lacking novelty or potentially redundant. If the authors choose to remove RQ1, they could incorporate its conclusion into the background section by citing relevant literature to concisely demonstrate that "For the SE tasks studied here, active learning is useful."

Thank you for this suggestion. We have adopted the first path (Remove RQ1 and work that material into the text see R0.1a on page 2 and R0.1b on page 21.

Minor Page15, line751: see Figure 5->see Figure 4

Page6 line306: see Table 2-> see Figure 2

Fixed. Thank you for the pointers.

Referee: 2

The revised version significantly improves the previous version. In particular, I think my first two comments on the previous version have been addressed in the revised version.

Thank you for that comment.

In the revised version, the background problem has been changed to using active learning for multi-objective optimization, and the paper is about to investigate how LLMs can help active learning multi-objective optimization. Let me examine my last three comments (i.e., 3rd, 4th, and 5th comments) on the previous version one by one.

My 3rd comment on the previous version is about the evaluation criteria. In the revised version, since the target is to investigate LLMs can help active learning multi-objective optimization, the evaluation criteria should be the benefits for the final multi-objective optimization. However, in the current revised version, the criteria seem to be finding some rows in the dataset with certain attribute values.

(So...) in the revised version, the presentation of the empirical results is unclear. I think the authors should elaborate on the details of the empirical results and the explanations of these results. Otherwise, readers can hardly understand this paper.

Thank you for your comments. Now this is embarrassing to report, but there was a critical paragraph missing in the last drafts about the nature of the simulation studies used in this paper. Without that paragraph, the nature of our evaluation criteria would be very puzzling indeed.

This draft corrects that oversight, see 2.3a on page 14 and 2.3b on page 18. With these corrections, we assert that our evaluation methods is now comprehensible.

My 4th comment on the previous version is about baselines. In the revised version, the authors adopt the TPE method as a baseline. Since the TPE method is a technique for estimation, I don't think TPE is an appropriate baseline. At least, the authors should provide convincing arguments to persuade readers to accept TPE as a baseline. Furthermore, some typical techniques for generating warm starts should be considered for baseline selection.

Thank you for this suggestion. We now include results from two real multi-objective optimizers (TPE and DEHB, see 2.5a on page 10; DEBH is an algorithm from IJCAI'23). As discussed in 2.5b on page 22, both these methods perform very badly (e.g. TPE always appears in the bottom half of our results,; DEHB needs over four times as much data as other methods; DEHB's optimizations are often worse and never better than other methods: see 2.5b on page 22).

In summary, while it was a good methodological point to add these results to this paper, they have no effect on our conclusions.

My 5th comment on the previous version does not seem to be addressed yet. This comment is to ask the authors to experiment the warm starts with some state-of-the-art active learners. However, I cannot find clear evidence that the authors have tried to address this comment in the revised version.

Thank you for your valuable feedback. Your comments have helped us to provide a more thorough and explicit explanation of our methodology.

We agree that using the most recent methods is critical. We have now updated our approach to use DEHB (from IJCAI'23), a more recent algorithm than TPE. We believe this satisfies your request for a state-of-the-art method.

Your comment also prompted us to provide you with a more detailed clarification of our warm-start methods. We now outlined five distinct approaches:

- **Random selection.** Instances are drawn uniformly at random. While this is technically a "cold start" rather than a warm start, it provides a simple baseline for comparison.
- **Diversity sampling.** Instances are chosen to maximize coverage of the input space. Here, we employ K-Means clustering to generate representative centroids that capture different clumpings of the data.
- **LLM-based knowledge.** Large language models can be queried for candidate warm starts. This approach is the central focus of this paper.
- **Expert knowledge.** A subject-matter expert can provide initial labeled examples. We do not adopt this approach here, since expert time is a scarce and valuable resource. In our view, experts are best consulted as part of the active learning cycle itself (after some instance selection has identified informative queries) rather than during initial setup.
- **Prior active-learning runs.** Results from a previous active-learning cycle (e.g., interval $i - 1$ in a quarterly workflow) can seed the next cycle. This is an intriguing direction, and we are actively exploring it. However, those results remain preliminary and are outside the scope of this paper.

In this paper, we focus on the first three, as they provide a strong set of baselines and directly support our central hypothesis. For the reasons mentioned above, the other two approaches were determined to be outside the scope of this work.

Referee: 3

Recommendation: Accept

Comments: I thank the authors for their feedback and revision. The revision has successfully addressed my concerns, especially regarding the improvements in experimental evaluation. After the authors add explanations for software configuration optimization, the paper is easier to follow. Therefore, I recommend accept.

Thank you for your guidance in improving this paper.