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

LOHITH SENTHILKUMAR and TIM MENZIES*, NC State, USA

When SE data is scarce, "active learners" use models learned from tiny samples of the data to find the next most informative example to label. In this way, effective models can be generated using very little data. For multi-objective software engineering (SE) tasks, active learning can benefit from an effective set of initial guesses (also known as "warm starts"). This paper explores the use of Large Language Models (LLMs) for creating warm-starts. Those 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 better support open science, all the scripts and data used in this study is available online at https://github.com/timm/moot and https://github.com/lohithsowmiyan/lazy-llm

1 Introduction

Many problems in SE must trade off between competing constraints; e.g

- How to deliver *more* code but at *less* cost?;
- How to answer database queries *faster* but use *less* energy?

Multi-objective optimization algorithms are tools that try to satisfy, as far as possible, these competing constraints [27], Many of these algorithms begin with an initial set of guesses, then refine them iteratively [54, 86]. 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) [26, 43, 83].

As described later in this paper (in §2.3.3), 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 [11, 30, 70, 78]. Hence it is timely to ask if LLMs can help the optimization of SE tasks 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 [13, 25, 45, 51, 53, 56, 58, 61] shown in Table 1. To structure this investigation, we ask:

- **RQ1**: *Is active learning useful for SE tasks?* This is our initial baseline question. By comparing simple random selection to active learners, this first questions asks if active learning is no better than much simpler methods.
- **RQ2**: *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.
- **RQ3**: *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 show the superiority of active learning over random methods; **RQ2** will demonstrate the value of warm starts; and **RQ3** will show that, sometimes, LLMs are good way to generate those warm starts. LLMs significantly improve outcomes for low- and medium-dimensional problems, reducing the need for

Authors' Contact Information: Lohith Senthilkumar, lpanjal@ncsu.edu; Tim Menzies, timm@ieeee.org, NC State, USA.

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. [20] (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

extensive labeled data. However, for high-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 optimization 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 §4.3).
- A reproducible package of data and scripts for benchmarking active learning strategies[1].

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.

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

2 Background

2.1 Motivation

This research team has been exploring optimization for SE for many years usually via stochastic methods [14, 23, 25, 48, 52, 57]. 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 §2.3.2, 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 §2.3.3 and §3.6).

2.2 A Gap in the Literature

Having assembled test data, and defined the optimization task, the literature was searched 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. So in September 2024, Google Scholar was queried for papers from the last ten years containing those two terms. The first 1000 papers from that search were sorted on their citation counts. We observed a knee in that citation curve at 37 citations, above which were 47 papers. Subsequent filtering [2] yielded the papers of Table 2, which we summarize as follows:

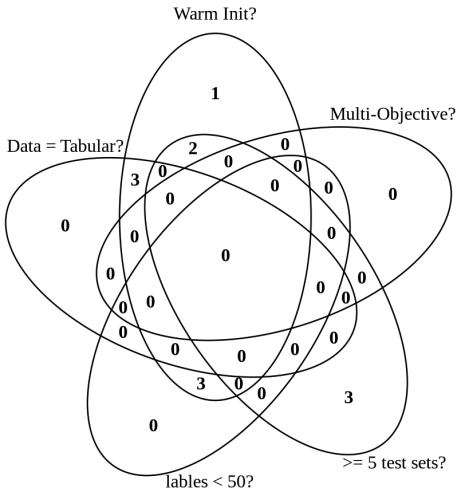
- $\geq$ **5 test sets?**: We check our conclusions on 49 data sets. Many other papers Table 2 use less than half a dozen.
- **Multi-objective?**: Not only do we explore more data, we explore more complex problems. While 30 of our data sets have multiple goals, most prior work in this area explored tasks 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.

[2] 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.

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?
[37]	361	2017	✓	✓	-	✗	✗
[65]	222	2018	✗	✗	-	✗	✗
[85]	184	2020	✗	✗	✗	✗	✗
[21]	163	2020	✗	✗	-	-	✗
[4]	156	2020	✗	✗	-	-	✗
[44]	122	2018	✓	✗	✗	✓	✗
[42]	95	2018	✗	✗	✗	✗	✗
[62]	79	2021	✗	✗	✗	✓	✗
[31]	65	2018	✓	✓	✗	✗	✗
[46]	38	2021	✓	✗	✗	✓	✗
[38]	36	2018	✗	✗	-	✓	✗
[15]	16	2024	✓	✗	✗	✗	✗
[5]	8	2022	✓	✓	-	✗	✗
[18]	2	2023	✗	✗	✗	✓	✗
[79]	1	2024	✓	✗	✓	✗	✗
[26]	96	2022	✓	✗	✓	✗	✗
[83]	37	2022	✓	✗	✓	✗	✗
		This Paper	✓	✓	✓	✓	✓



2.3 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 §2.3.1); *quirks* of data collection (see §2.3.2); or *slow data* collection (see §2.3.3).

2.3.1 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 [12, 28, 33, 35, 55] often label a commit as "bug-fixing" when the commit text uses words like "bug, fix, wrong, error, fail, problem, patch". Vasilescu et al. [73, 74] 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. [84] 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 [81]; for labeling false alarms in static code analysis [34]; 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.

2.3.2 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:

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.

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

- *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.

2.3.3 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 [72] 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.
- 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 [45] 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 [22] 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 [36] 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* [49, 50], 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 [22, 36]).

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.

2.4 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 §2.4.1), metamorphic learning, semi-supervised learning (see §2.4.2), few-shot learning with LLMs (see §2.4.3), and active learning (see §2.4.4).

2.4.1 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 [2]. 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 [59]. 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 [16] 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.

2.4.2 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. [82] 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 [3, 47]. While a useful approach, in the studies we have seen [3, 47], 1 to 2.5% of the data still means 100s to 1000s of labels. Given the numbers seen in §2.3.3 (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.

2.4.3 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 [40] or translating functions into English [1].

One problem with few-shot learning is how to find a few good examples. Tawosi et al. [69] 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. [69] method, our approach can terminate in just a few sections (in particular, the "explore" and "exploit" TPE methods discussed below).

2.4.4 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 [10].

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

(1) *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.
(2) *Similarity sampling*: Similarity sampling selects subsets of the examples where, within each subset, there is much similarity [41].
(3) *Diversity sampling*: Diversity sampling selects data points that are different from those already chosen or labeled [76]. Many diversity sampling methods are clustering based [26, 83] while other methods like [9] 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 [10, 45] and the experiments of this paper). Methods for generating warm starts include (a) reusing the results from a prior session; 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. [77] claim that Gaussian Process Models (see §2.4.5) are less efficient that Tree of Parzen Estimators (see §2.4.6) for active learning and multi-objective optimization. We will explore both approaches

2.4.5 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 [10, 80].

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

UCB = Upper Confidence Bound: Introduced in 1992 by [17], 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 [39], 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

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.

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 [32]): Expected improvement measures the expected gain from sampling a new point based on the priors.

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

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

- 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.

2.4.6 TPE = Tree of Parzen Estimators. One alterative to using Gaussian process models are acquisition functions that use Tree of Parzen Estimators (TPE) [6, 60]. While Gaussian process-based approaches model $p(y|x)$ directly (i.e., $y = f(x)$), TPE [6] 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:

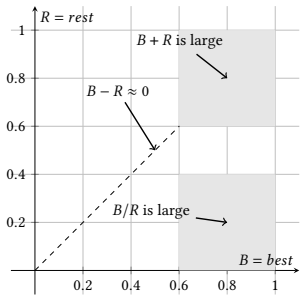


Fig. 1. Our TPE active learner explores the decision space of a two-class classifier.

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

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 1, 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.

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.

- 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.

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:

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

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:

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

3 Methods

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 [13, 25, 45, 51, 53, 56, 58, 61]. For some details on the MOOT data sets, see Table 6.

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 [23], we can assert that resources like MOOT are very rare. As shown in our literature review (§2.2), 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 [19] 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 §2.3.2, 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. [20] 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.

Table 6. Overview of MOOT's data. For notes on the structure of these data sets, see Table 1.

Dataset Type	**Features**	**Primary Objective**
SS-Models	Runtimes, query times, and usage data from software configured in various manners, selected randomly	Identify configurations that optimally meet the software goals of each project
Specific Software Configurations	Configurations of specific software systems—Apache servers, Hazardous Software Management Growth Program (HSMGP), SQL databases, and X264 video encoding	Optimize particular settings that enhance the performance and efficiency of these well-known applications
General SS-Datasets (RS, SOL, WC)	Runtimes, query times, and usage data from software configured in various manners, selected randomly	Collect various performance metrics from randomly configured software systems tailored toward specific software engineering challenges
Health Data Sets	Random forest regression to predict future GitHub project metrics such as commits, closed issues, and pull requests	Predictive measure of project health and developer activity over a 12-month period
Debug Datasets (Auto93, Wine Quality)	Generalizable data for software engineering newcomers	Auto93 optimizes for car performance metrics, whereas Wine Quality focuses on the attributes influencing the quality of wine
COC1000 and NASA93	Based on the Constructive Cost Model (COCOMO) approach	Emphasizes the reduction of risk, effort
POM3 (A-D)	Agile project management dynamics	Simulate scenarios where team idle rates, task completion rates, and overall costs are in flux
XOMO (Flight, Ground, OSP, OSP2)	Software process optimizations at NASA's Jet Propulsion Laboratory	Addresses the complexity and challenges in aerospace software development projects

- *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 [56].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 [45] . 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 [25, 52, 53, 61]. The data set named "pom*" shows data from agile development by [8]. 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 [52].
(4) The *Misc* directory contains some non SE datasets (auto93 and WineQuality). We use these to demonstration MOOT to a non-SE audience.

3.2 Models

Selecting the most suitable large language model for a given task requires careful consideration of various factors. Among the multitude of publicly available models, the most prominent families include GPT, Gemini, and Claude. While prior research [64] indicates that GPT outperforms its competitors in software engineering benchmarks. However, our specific task requires a deeper contextual understanding of the dimensions and configurations of the data.

Previous studies [29] suggest that Gemini models excel in precise factual reasoning, particularly for tasks demanding high contextual comprehension. Furthermore, Gemini offers a 1-million-token context window [71], significantly surpassing GPT-4o’s 128k tokens. This extended context window allows the inclusion of extensive metadata descriptions and examples, which is especially advantageous for high-dimensional datasets. Additionally, Gemini’s competitive per-token pricing compared to GPT was a decisive factor.

For all the reasons pf the last paragraph, this study used Gemini 1.5 Pro.

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 §2.4.5 and §2.4.6.

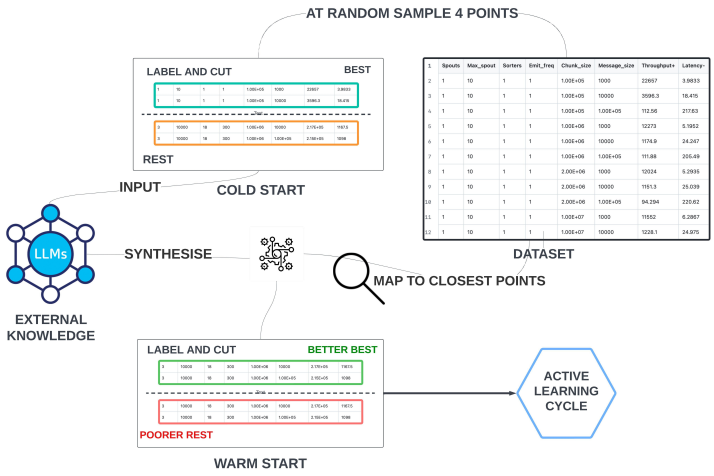


Fig. 2. Warm-starting with LLM Synthesized examples in cycle 0 of active learning

Table 7. 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:
```

3.3.3 Warm Starts with LLMs. LLMs were used to generate warm starts as per the method illustrated in Figure 2:

(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 [67] 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".
(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) Since all the items in E_1 were invented by the LLM, they have no y column labels. To generate labeled examples, we then went 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 7 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 [67]). Note that our prompt template was divided into three parts:

- *System message*: where we define the role of our LLM which is a general prompting strategy, 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.

3.4 Evaluation Criteria

The above methods need some way to recognize "best" and "rest" rows. Following the precedent set by the multi-objective research community [87], our rows are judged best or rest using the Chebyshev distance [24] 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:

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

- 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 there is precedence for its use in the multi-objective research literature [24, 87]. Also it is a "cruel critic"; i.e. it punishes and demotes a row if any of its y values are bad, even in all the others are performing admirably.

3.5 Statistical Methods

We rank our results using the Scott-Knott method [63]. 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 the list into two halves at the point where there is maximum expected difference between the mean before and after division.

$$E(\Delta) = \frac{|l_1|}{l} abs(E(l_1) - E(l))^2 + \frac{|l_2|}{l} abs(E(l_2) - E(l))^2 + \quad (5)$$

Here. $|l_1|$ is the size of list 1 and $|l_2|$is the size of list 2

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.

Table 8. 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

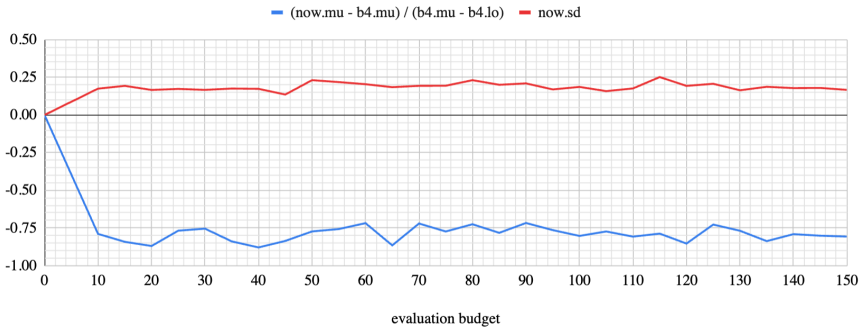
If the results of these statistical tests confirms distinguishably of the groups then Scott-Knott ranks these groups and recursively repeats the process for both these clusters.

Table 8 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 8, the statistical rankings found via Scott-Knott are colored alternatively gray or white. In that table:

- 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 8, that means that for this data set, EI_GPM has failed to optimize this data set.
- Top ranked results are shown at the top of the table and labeled rank "0".

Fig. 3. Average optimizations seen across all data sets. Most improvement were after a few dozens samples Let *b4.mu* and *b4.lo* be the mean and smallest Chebyshev distances seen in the original data. Let *now.mu* and *now.sd* be the mean and standard deviation of the best Chebyshevs seen in 20 repeats of our active learning experiments-(in this case, LLM warm starts followed by exploit). The blue plot shows *(now.mu - b4.lo) / (b4.mu - b4.lo)*; i.e. the improvement seen by optimization, normalized by the maximum possible improvement (and for the blue line in this plot, *lower* values are *better*).



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 §2.3.3;
- 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 [7]).
- 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 3; and (b) a detailed statistical analysis. That statistical analysis built one plot like Figure 3 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 8 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.

		Scott-Knott Rankings						
Start	Acquire	0	1	2	3	4	5	6
LLM	Exploit	**100**						
random	UCB_GPM	**45**	9	9		18	9	9
random	EI_GPM	**45**	18	18		9		9
random	Exploit	**27**	9	18	27	9		9
random	PI_GPM	9	36	18	18	9		
LLM	Explore	9	9	18	18	9	9	9
	random	36	9		9	9	9	
random	Explore		18		9	27	9	
	Baseline					9	18	

Table 9. Percent frequency for some treatment appearing at some rank. e.g. "50" means that a treatment achieved a rank in half of our 49 data sets. Results from 12 low dimensional data sets (i.e. with $x < 6$ independent features).

Scott-Knott Rankings								
Start	Acquire	0	1	2	3	4	5	6
LLM	Exploit	**50**		7	14	7	7	
random	UCB_GPM	**36**	36	14	14			
random	PI_GPM	**36**	14	14	29		7	
random	EI_GPM	**36**	21	14	7	7	14	
random	Exploit	**21**			7		7	7
LLM	Explore	14	7			21	7	7
	random	14	7				14	21
random	Explore	14	7				7	7
	Baseline	7		7				

Table 10. Frequency of ranks achieved. Same format as Table 9. For 16 medium dimensional datasets (with $5 < x < 11$ features).

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 11. Frequency of ranks achieved. Same format as Table 9. For 19 high dimensional datasets (with $x > 10$ independent features).

4 Results for Research Questions

4.1 RQ1: *Is active learning useful for SE tasks?*

The point of active learning is that AI tools should be able to learn a domain, after seeing very few labeled examples. Hence, one way to assess active learning is to ask "how any labels does it require to find good optimizations of the MOOT data?"

As illustrated in Figure 3, the steep decline in the blue curve (representing normalized improvements in performance) shows that most of the optimization benefit is achieved with fewer than 30 labeled samples. Also, Table 12 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.

Skott-Knott Rankings								
Start	Acquire	0	1	2	3	4	5	6
random	UCB_GPM	19	21	22	21	22	15	15
random	EI_GPM	19	17	23	23	18	18	18
random	PI_GPM	17	18	22	21	15	15	
LLM	Exploit	17	15	18	18	18	20	25
random	Exploit	19	23	15	19	21	17	20
LLM	Explore	16	15	18	18	17	16	
	random	18	19	20	20	18	19	19
random	Explore	22	19	15	15	22	18	21

Table 12. Scott-Knott rankings comparison for number of evaluations needed

Another way to assess active learning is to ask "how good is it versus just performing random selection"? The rows with "Acquire=random" in the Tables 9,10,11 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, 10 or 9 did a purely random method score appear most often in the rank "0" column.

In summary, we say:

> **RQ1:** For the SE tasks studied here, active learning is useful since it can explore thousands to tens of thousands of rows of data after just a few dozen labeling.

4.2 RQ2: *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:

- For low dimensionality data sets, in Table 9 we see that (a) LLM/Exploit earns top rank 100% of the time while (b) random/Exploit only earns top rank 27% of the time.
- For medium dimensionality data sets, in Table 10 we see that (a) LLM/Exploit earns top rank 50% of the time while (b) random/Exploit only earns top rank 21% of the time.

Hence we say:

> **RQ2:** For the 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.

4.3 RQ3: *Are LLMs the best way to generate warm starts?*

The success of LLM's warm starts are remarkable, especially considering what we were asking LLMs to do. Recalling Table 2, 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[3] 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 and medium dimensional examples is a testament to the inherent power of these large languages.

One caveat to the last paragraph is that there are some nuances that need to be discussed on the effectiveness of LLMs for warm starts. Observe that LLM/exploit performed very well in Table 10 and better than anything else is Table 9. However, LLM paired with other acquisition functions did not perform well. Specifically, LLM/Explore performed much worse than LLM/Exploit. This results suggests that LLMs has thoroughly searched through the zone of doubt described §2.4.6, thus making further exploration superfluous. Once again, this result speaks to the power of LLM-based inference.

All that said, LLM/exploit failed for higher-dimensional problems, at which point randomly selected starts followed by UCB_GPM performed best. Clearly, further research is required in order to extend LLM-based reasoning to more complex tasks.

Overall, we summarize these results as follows:

> **RQ3:** For the SE tasks studied here, LLM-based warm starts following by exploit-guided acquisition performs best for low and medium dimensional problems. However, LLM effectiveness diminishes in high-dimensional problems, where Bayesian methods like Gaussian Process Models perform best

[3] 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.

5 Discussion

5.1 Why Did LLMs Fail for Higher-dimensional Data?

We applaud the success (reported above) of LLMs for low and medium complexity problems. But why does this approach fail for higher-dimensional data?

Perhaps the answer lies in the data used to train LLMs. 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.

Outside the commons, 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.
- 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[4].

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:

- The more complex the problem ...
- ... and the fewer examples of good solutions in the commons ...
- ... then the harder it becomes for LLMs to learn a model that works for that problem.

For these harder uncommon problems, we will recommend other methods (GPM or TPE).

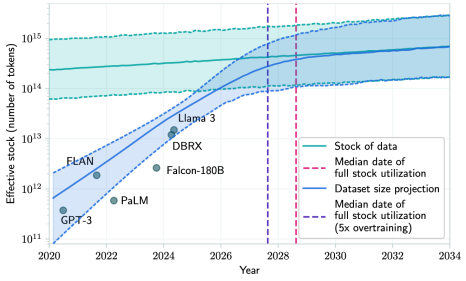


Fig. 4. Median results from the Villalobos et al. model (shown in green) estimate that by 2028, we will run out of new textual data needed to train bigger and better LLMs [75].

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 [75]. In Figure 4, they project that newer and larger models will soon exhaust the available textual data[5]. At that time, research work like this paper will be required to handle all the problems for which LLMs lack sufficient training data.

[4]See Table 7 of Tang et al. [68] 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/.
[5]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.

(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 [66].)

5.2 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. We have used Google Gemini 1.5 pro as our model for generating synthetic warm starts, The kind model used can significantly affect the quality of the generated samples and it is impossible to compare the performances of various models due to restrictions.

5.3 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 (e.g., PCA) can mitigate this limitation.

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 5, this process scan be slower even than LLM few shot learning.

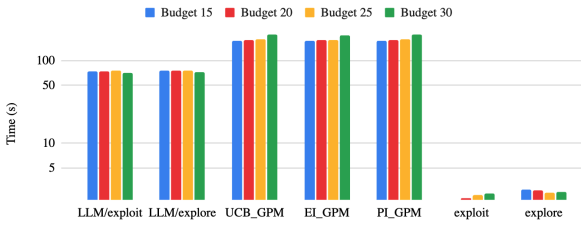


Fig. 5. A sample of runtimes from these experiments.

One other thing to note from Figure 5 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.

6 Conclusion

This study demonstrates that Large Language Models (LLMs) can effectively warm-start active learning for software engineering (SE) tasks, significantly improving performance for low- and medium-dimensional problems. By generating plausible initial guesses, LLMs reduce the labeling effort required, outperforming random starts and matching or exceeding Bayesian methods in many cases.

However, for high-dimensional tasks, LLMs face limitations, with Gaussian Process Models (GPM) continuing to deliver superior results. This highlights the complementary roles of LLMs and traditional Bayesian approaches in SE optimization.

Our contributions include:

- A novel application of LLMs for warm-starting SE active learners.
- Empirical validation across 49 multi-objective SE tasks.
- Open-source release of data and scripts to promote reproducibility.

Future work will focus on extending LLM effectiveness to higher dimensions, exploring hybrid models, and validating results on broader SE tasks. This study underscores the potential of LLMs to enhance optimization in data-scarce SE environments.

References

[1] 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

[2] 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.

[3] 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

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

[5] 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.

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

[7] 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).

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

[9] 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.

[10] 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).

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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

[20] 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.

[21] 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.

[22] 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

[23] 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

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

[25] 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.

[26] 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).

[27] 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.

[28] 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.

[29] 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).

[30] 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

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

[32] 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.

[33] 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.

[34] 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

[35] 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.

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

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

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

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

[40] 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.

[41] 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.

[42] 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.

[43] 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).

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

[45] 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.

[46] 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).

[47] 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.

[48] 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.

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

[50] 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*.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

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

[56] 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.

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

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

[59] 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)*.

[60] 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.

[61] 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.

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

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

[64] 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).

[65] 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).

[66] 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).

[67] 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.

[68] 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

[69] 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.

[70] 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.

[71] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).

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

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

[74] 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.

[75] 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*.

[76] 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.

[77] 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).

[78] 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

[79] 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.

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

[81] 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

[82] 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.

[83] 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.

[84] 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.

[85] 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).

[86] 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.

[87] 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.